R for Basic Biostatistics in Medical Research

Anand Srinivasan
Archana Mishra
Praveen Kumar-M
Editors

Springer

R for Basic Biostatistics in Medical Research

Anand Srinivasan • Archana Mishra
Praveen Kumar-M
Editors

R for Basic Biostatistics in Medical Research

Springer

Editors
Anand Srinivasan
Department of Pharmacology
All India Institute of Medical Sciences
Bhubaneswar, Odisha, India

Archana Mishra
Department of Pharmacology
All India Institute of Medical Sciences
Bhubaneswar, Odisha, India

Praveen Kumar-M
Clinical Sciences
Nference
Bengaluru, Karnataka, India

ISBN 978-981-97-6979-7 ISBN 978-981-97-6980-3 (eBook)
https://doi.org/10.1007/978-981-97-6980-3

This Springer imprint is published by the registered company Springer Nature Singapore Pte Ltd.
The registered company address is: 152 Beach Road, #21-01/04 Gateway East, Singapore 189721, Singapore

Preface

A thorough introduction to using R for statistical analysis in the medical profession can be found in this book, *R for Basic Biostatistics in Medical Research*. It is intended for healthcare professionals, students, and researchers who have little to no familiarity with R. This book's layout is intended to be both practical and user-friendly, offering a step-by-step method for becoming proficient with R's medical data analysis features.

Starting with the basics of R, we go over data processing, basic commands, installation, and descriptive statistics. You don't need any prior programming experience to use R confidently in your own medical research, because we steadily increase your grasp of the language.

Focusing on real-world applicability is one of the book's main characteristics. Every chapter is painstakingly organized, which aids readers in comprehending how R is used in practical situations. Hypothesis testing, regression analysis, and survival analysis are just a few of the many statistical techniques and analytical philosophies that are frequently applied in medical research and are covered by examples.

You will come across several practical examples with succinct, understandable explanations and annotated code samples throughout this book. The focus is not on deep dives into R's programming syntax but rather on assisting you in conducting the required analysis. We offer helpful advice and recommendations on typical problems that arise when performing statistical analysis.

By bridging the knowledge gap between statistical theory and medical practice, this book will enable you to utilize R with confidence to address your research issues. With this book, we hope you will be able to perform a more thorough and intelligent statistical analysis of your medical data.

We take this opportunity to thank all the authors, especially Dr. Inderjeet Singh and Dr. Guruprasad Padmanaban, for their immense contributions in shaping this book. We are open to criticism and ideas for development. Kindly send an email to anandsrinivasan@aiimsbhubaneswar.edu.in with your ideas and remarks.

Sincerely,
The Editorial Team

Bhubaneswar, Odisha, India — Anand Srinivasan
Bhubaneswar, Odisha, India — Archana Mishra
Bengaluru, Karnataka, India — Praveen Kumar

Contents

About the Editors

Anand Srinivasan did his MD in Pharmacology and DM in Clinical Pharmacology at the Postgraduate Institute of Medical Education and Research (PGIMER), Chandigarh, India. He is currently working as an Additional Professor in the Department of Pharmacology at the All India Institute of Medical Sciences (AIIMS), Bhubaneswar. He prefers working in R due to its flexibility and vast potential in data science. Moreover, R is also an open-source tool. He has used R for statistical analysis of data generated from various clinical studies and meta-analyses over the past ten years. He also uses it for simulating various situations to answer a myriad of clinical and research questions and for analyzing the data from population pharmacokinetic–pharmacodynamic studies. He has developed artificial intelligence models on healthcare data using artificial neural networks in R. He has been instrumental in organizing R workshops at AIIMS, Bhubaneswar, for healthcare professionals every year since 2017.

Archana Mishra is a Senior Resident in the Department of Pharmacology at the All India Institute of Medical Sciences (AIIMS), Bhubaneswar. She completed her MD from the same organization and her DM in Clinical Pharmacology from AIIMS, New Delhi. Her journey with R started in 2017 when she participated in the first National Workshop on R for Basic Biostatistics conducted at AIIMS, Bhubaneswar. She was gradually drawn to R's powerful yet intuitive statistical capabilities. Since then, she has been regularly using R for biostatistical analysis of healthcare data. She is also using R to conduct meta-analysis, network meta-analysis, pharmacokinetic modeling, clinical trial simulations, and to develop machine learning models. As she gained experience in R, she is now a part of conducting the annual workshop at AIIMS, Bhubaneswar, in R, as a resource person. She explains the journey from all red errors after every line of R code in the initial days to publishing with R, an ever-rewarding and satisfying one.

Praveen Kumar pursued his MD (Pharmacology) and DM (Clinical Pharmacology) from the Postgraduate Institute of Medical Education and Research (PGIMER). His interest in R began on the day when he realized the versatility, elegance, and impact that R provides for statistical analysis. During his tenure at

PGIMER, his work using R, R shiny, and machine learning models has been published in reputable medical journals. His interest lies in designing graphical charts with R and in developing interactive web interfaces with R Shiny Studio. In addition to R, he is comfortable with Python, SQL, and MATLAB. In his current position as the Head of Clinical Sciences at Nference, Bengaluru, he is proficient in solving multiple real-world data (RWD) based research questions for the pharmaceutical industry using large electronic medical records (EMRs) from top academic medical centers (AMCs) like Mayo and Duke. He has been actively involved in the annual R workshop for healthcare professionals since its inception in 2017 at AIIMS, Bhubaneswar.

Chapter 1
Why R Is Essential? What Are the Prospects of Learning R?

Anand Srinivasan, Archana Mishra, and Debasish Hota

Ross Ihaka and **R**obert Gentleman developed a statistical analysis and graphics system known as R, which is a dialect of the language S [1]. R is regarded as both a language and software. It is a powerful open-source programming language and environment for statistical computing, and graphics offers a versatile toolkit that resonates particularly well with the complex and data-intensive nature of medical research [2]. The application of R language in medical research has become indispensable, addressing the growing need for sophisticated data analysis and statistical modeling in the healthcare domain.

Let us quickly review the advantages of R:

1. It is open-source and runs on Windows, Linux, and iOS
2. The syntax is easy and intuitive
3. It has excellent graphic capabilities
4. It has a relatively better community support
5. It has many built-in statistical functions
6. R allows for combining several functions together to perform complex analyses
7. Since it is an open-source tool, anyone can build packages to enhance the ease of complicated analysis.

R facilitates the creation of reproducible and transparent analyses, which is critical for ensuring the reliability of findings. Its graphics and visualization capabilities enable researchers to present complex data in a comprehensible manner, aiding in the communication of results to both scientific audiences and healthcare practitioners. The collaborative nature of R's open-source community further enhances its appeal in medical research. Researchers can access a vast repository of packages

A. Srinivasan (✉) · A. Mishra · D. Hota
Department of Pharmacology, All India Institute of Medical Sciences, Bhubaneswar, Odisha, India
e-mail: anandsrinivasan@aiimsbhubaneswar.edu.in

A. Srinivasan et al. (eds.), *R for Basic Biostatistics in Medical Research*,
https://doi.org/10.1007/978-981-97-6980-3_1

and scripts developed by the global community, accelerating the implementation of cutting-edge methodologies and ensuring that the latest statistical techniques are readily available for analysis in the healthcare domain. In a nutshell, learning R can be essential for several reasons, particularly for individuals involved in data analysis, statistics, and programming. We have tried to enumerate a few reasons why learning R can be beneficial for healthcare professionals:

1. Planning and designing clinical studies:
   Packages in R allow for sample size calculation of various types of studies. Randomization for clinical trials can easily be performed using R.
2. Data handling and cleaning:
   R is well-suited for data manipulation, cleaning, and transformation tasks. It provides powerful tools like the *dplyr* and *tidyr* packages, making it efficient for handling messy and raw data.
3. Statistical Computing and Data Analysis:
   R is designed specifically for statistical computing and data analysis. It provides a wide range of statistical and mathematical techniques, making it a powerful tool for researchers, statisticians, and data scientists. Apart from analyzing data from individual studies, R can also be used for secondary research such as meta-analysis and network meta-analysis.
4. Adaptive designs:
   Boundaries for adaptive designs like group sequential designs can be easily fixed using packages in R. Planning, designing, and analyzing adaptive trials (both frequentist and Bayesian) can be easily done in R.
5. Data Visualization via various plots:
   R has excellent data visualization capabilities. It offers various packages (e.g., ggplot2) that allow users to create high-quality plots and charts, aiding in the interpretation and communication of data insights.
6. Community and Packages:
   R has a vibrant and active community of users and developers. There is a vast repository of packages available in the Comprehensive R Archive Network (CRAN), covering diverse areas such as machine learning, time series analysis, and bioinformatics.
7. Reproducibility and Documentation:
   R encourages good coding practices, making it easier to document and reproduce analyses. This is crucial for collaboration, sharing results, and ensuring that analyses can be replicated by others.
8. Flexibility with other tools:
   R can be integrated with other programming languages, databases, and tools. This flexibility allows users to combine the strengths of R with other tools and technologies.
9. Academic and Research Use:
   R is widely used in academia and research. Many scientific publications and research studies utilize R for statistical analysis and data visualization. Learning R can enhance your ability to understand and contribute to academic work.

10. Machine Learning and Predictive Modeling:
    R provides various packages and tools for machine learning and predictive modeling, allowing users to build and deploy models for a wide range of applications.
11. Modeling and simulations:
    R allows the creation of simulated clinical data that enables researchers to optimize the clinical trial's design and validate that the suggested statistical analyses are reliable and resilient to challenges.
12. Open Source and Cost-Free:
    R is an open-source language, and its vast ecosystem of packages is freely available. This makes it an attractive choice for individuals, organizations, and educational institutions looking for cost-effective solutions.
13. Career Opportunities:
    Proficiency in R is highly valued in industries that rely on data analysis, such as finance, healthcare, marketing, and technology. Learning R can enhance your employability and open up career opportunities in data-related roles.

In addition to making our life as a healthcare researcher easier, R widens our spectrum of job opportunities. We have tried to compile a few prospects by learning R [3, 4]:

1. Biostatistician and healthcare analyst
   R is widely used in biostatistics and healthcare analytics for analyzing clinical data, conducting epidemiological studies, and developing statistical models. Professionals in healthcare analytics, epidemiologists, and bioinformaticians often use R in their work.
2. Systems biologist and pharmacometrician
   R is pivotal for systems biologists and pharmacometricians. In systems biology, R aids in data analysis, visualization, and network analysis. In pharmacometrics, R is used for population pharmacokinetic-pharmacodynamic modeling, drug interaction modeling, and clinical trial simulations. This can facilitate drug development and individualized dosing in clinical settings.
3. Pharmacoeconomics analyst
   R is employed in the finance industry for quantitative analysis, risk management, and financial modeling. Learning R can be advantageous for roles such as financial analyst, quantitative analyst, or data scientist in the finance sector.
4. Bioinformatician
   R plays a crucial role in bioinformatics by providing a versatile and powerful platform for data analysis, visualization, and interpretation of proteomics, genomics, and metabolomic data analysis. Its extensive ecosystem of packages and active community support make it a preferred choice for researchers working with biological data. Bioinformatics tasks often require custom scripting and automation. R's scripting capabilities allow researchers to develop custom solutions for specific analysis needs and to automate repetitive tasks.
5. Academician

Proficiency in R is highly valued in academia and research. Researchers and academics in various fields, including social sciences, biology, and economics, often use R for statistical analysis and data visualization.

6. Decision-makers

   R can be used for analyzing survey data, conducting statistical tests, modelling and simulations, and creating visualizations. Learning R can be beneficial for individuals pursuing a career in regulatory agencies and data-driven decision-making.

7. Programmer and Software Developer

   R is a programming language, and learning it can provide a strong foundation in programming concepts and data manipulation. This knowledge can be advantageous for individuals interested in software development, especially in roles that involve data-centric applications.

8. Freelancing and Consulting Services

   With expertise in R, individuals can explore freelance or consulting opportunities. Organizations often seek experts to help with specific data analysis projects or to provide training in R and statistical methods.

9. Artificial Intelligence (AI) Research Scientist

   Learning R will help you create AI-based models, which may be helpful in various clinical and experimental settings. The successful AI models created can be deployed for clinical care, such as precision pharmacotherapy.

Learning R makes learning other languages easier, which may widen the use of computational tools for research and patient care. Let's get started and learn the most essential of these aspects throughout various chapters of this book. We wish you a pleasant learning experience.

References

1. Ihaka R, Gentleman R. R: a language for data analysis and graphics. J Comput Graph Stat. 1996;5(3):16.
2. Paradis E. R for beginners: Institut des Sciences de l'Evolution. Université Montpellier II; 2005.
3. Childs DZ, Bethan J. Hindle and Philip H. Warren. Introductory Biostatistics with R. 2022. https://tuos-bio-data-skills.github.io/intro-stats-book/.
4. Giorgi FM, Ceraolo C, Mercatelli D. The R language: an engine for bioinformatics and data science. Life (Basel). 2022;12(5):648.

Chapter 2
An Overview of Statistical Analysis Plan for Clinical Studies

Archana Mishra, Anand Srinivasan, and Rituparna Maiti

In the realm of medical research, data analysis serves as the linchpin that bridges the gap between raw information and actionable insights. As technology continues to evolve, the healthcare industry is witnessing an unprecedented surge in the volume and complexity of data generated through various sources, such as electronic health records, genomic sequencing, and clinical trials. In this landscape, the art and science of data analysis play a pivotal role in unraveling patterns, extracting meaningful information, and ultimately shaping the future of medical science. The exponential growth of data in the medical field has transformed the research landscape, opening new avenues for exploration and discovery. Traditional research methods are being complemented, if not supplanted, by sophisticated data-driven approaches that capitalize on the wealth of information available. From epidemiological studies to personalized medicine initiatives, data analysis is at the forefront, driving innovations and fostering a deeper understanding of diseases, treatment modalities, and overall healthcare dynamics.

In experimental studies, data analysis assumes a paramount role in ensuring the validity and reliability of research findings. Researchers meticulously analyze data from randomized controlled trials and observational studies to draw evidence-based conclusions about the safety and efficacy of interventions. Statistical methods guide the interpretation of trial results, helping researchers discern whether observed differences between groups are statistically significant or merely due to chance. The rigorous scrutiny of data is not only essential for regulatory approval but also for informing clinical guidelines and shaping the

A. Mishra · A. Srinivasan · R. Maiti (✉)
Department of Pharmacology, All India Institute of Medical Sciences, Bhubaneswar, Odisha, India
e-mail: pharm_rituparna@aiimsbhubaneswar.edu.in

A. Srinivasan et al. (eds.), *R for Basic Biostatistics in Medical Research*,
https://doi.org/10.1007/978-981-97-6980-3_2

standard of care. Ethical considerations loom large in the era of big data and data analytics in medical research. The responsible handling of patient data, safeguarding privacy, and ensuring informed consent are paramount concerns. Researchers and institutions must navigate a delicate balance between extracting valuable insights from data and upholding the principles of patient confidentiality and consent.

Collaboration between interdisciplinary teams is essential in harnessing the full potential of data analysis in medical research. Researchers, data scientists, clinicians, and statisticians must work in tandem to design robust study protocols, analyze data effectively, and translate findings into meaningful clinical applications. The synergy between these diverse expertise ensures that the analytical methods applied are not only statistically rigorous but also clinically relevant.

Learning statistical analysis of medical data offers a multitude of benefits, empowering researchers, healthcare professionals, and policymakers to derive meaningful insights from complex datasets. In the rapidly evolving landscape of healthcare, statistical analysis serves as a crucial tool for evidence-based decision-making, contributing to advancements in medical research and improving patient outcomes. Here are some key benefits of mastering statistical analysis in the context of medical data:

1. Evidence-Based Decision-Making
2. Identifying Patterns and Trends
3. Clinical Trial Design and Analysis
4. Epidemiological Studies and Public Health Research
5. Resource Allocation and Cost-Effectiveness Analysis
6. Predictive Modeling and Risk Assessment
7. Communication of Research Findings

The benefits of learning statistical analysis in the context of medical data are far-reaching. From shaping evidence-based medical practices to driving advancements in personalized medicine and public health, statistical analysis is an indispensable tool for those navigating the complex landscape of healthcare research and decision-making. As technology continues to generate vast amounts of data, the ability to harness and interpret this information through statistical analysis becomes increasingly crucial for the advancement of medical science and the improvement of patient care. Learning statistical tools for data analysis is a valuable skill in today's data-driven world. These tools empower individuals to extract meaningful insights, identify patterns, and make informed decisions based on data. Whether you are a student, a researcher, or a professional in any field, acquiring proficiency in statistical tools enhances your ability to derive actionable knowledge from datasets.

Before moving on to exploring the analysis of the datasets, we will briefly revise the core statistical concepts in this chapter.

Statistics and Statistical Analysis Plan [1]

Statistics is the science of collecting, organizing, analyzing, interpreting, presenting, and making inferences from data. Any research should predefine a plan of statistical analysis in its protocol outlining the planned process of achieving laid down objectives. Once the required data has been collected (study completion or interim analysis), a systematic approach should be followed to interpret data effectively. A generic flow diagram for the plan of statistical analysis is given in Fig. 2.1.

Though the general idea behind analysis remains the same, the steps may appear different based on the study design (Fig. 2.2).

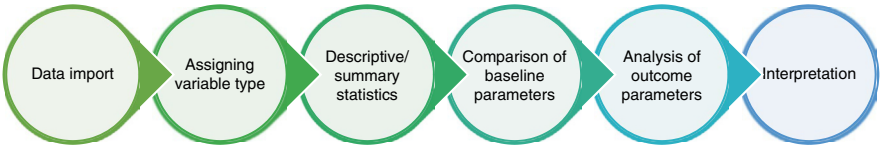

Fig. 2.1 A typical plan for statistical analysis

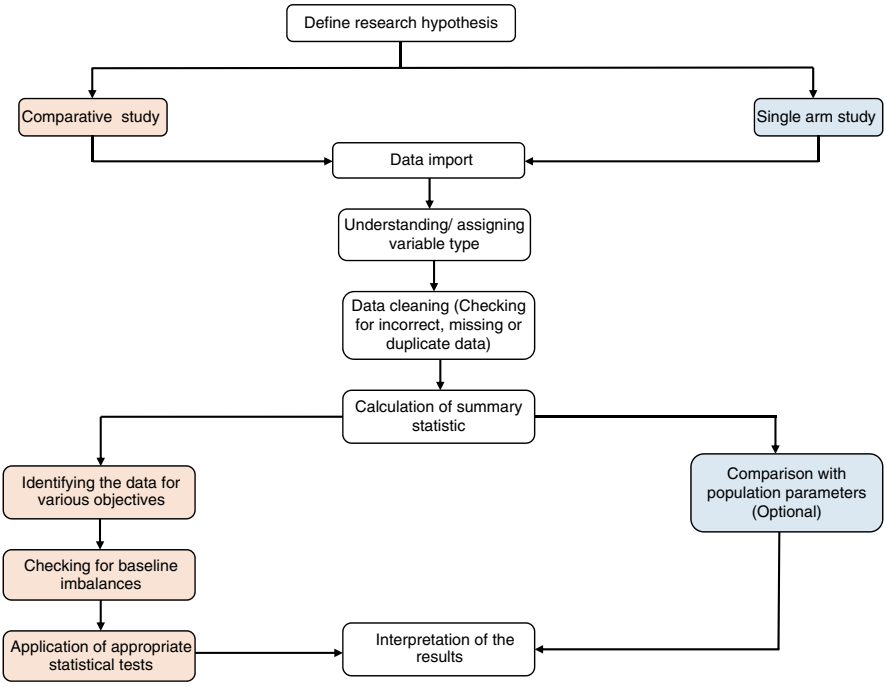

Fig. 2.2 Statistical analysis plan based on type of study (single-arm or comparative)

Data [2, 3]

Data is the lifeblood of statistics, and its importance cannot be overstated in the field. The first step before plotting or analyzing data is to understand the type of data obtained or to be obtained from a study. In statistical terms, a variable is a characteristic or attribute that can vary, and data are the values or observations associated with those variables. There are various methods of classifying variables/data, but understanding biostatistics is based on the typology provided below.

Types of Variables [4]

Variables can be classified into two broad categories: first, which can be considering which category (categorical), and second, which can be measured by questioning how much (quantitative) (Fig. 2.3).

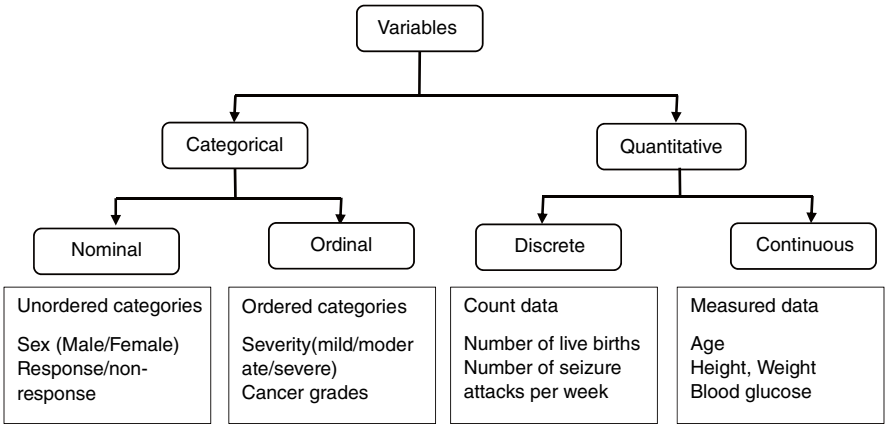

Fig. 2.3 Types of variables

Summary Statistics for Data [5]

Raw data sets can be large and complex, making it challenging to gain meaningful insights. The use of summary statistics is essential for simplifying, communicating, and extracting meaningful information from complex datasets, making them more accessible and manageable for analysis and decision-making.

Mean: Mean is the sum of the values for that variable divided by the number of variables.

Standard deviation: It denotes the spread of data around the mean or average value.

Median: It is a data point for which half the values for a given variable are above, and the other half are below it when the data has been arranged in ascending or descending order.

Interquartile range: One may present the interquartile range (IQR) beside the median. The IQR includes the middle 1/2 of the sample since the first quartile point has 1/4 of the data below it, and the third quartile has 3/4 of the sample below it when the data is arranged in ascending or descending order.

Mode: Mode is the commonest value observed for a variable.

Percentage: This describes a proportion out of 100.

Continuous data is generally expressed as mean and standard deviation if normally distributed and as median (Interquartile range) when non-normally distributed. Normal distribution means that the spread of data around the mid-point is similar and presents a bell-shaped distribution.

Reporting Results

While reporting results, we provide a point estimate along with a measure of dispersion and then draw conclusions regarding the formulated hypothesis.

Point Estimates [6]

Mean difference: This is the difference in the given mean values and is calculated by subtracting one from the other when the data from two groups are compared.

Odds ratio: The number of times an event occurs divided by the number of times it does not occur yields the odds of an event within a group. An odds ratio is computed by dividing the odds in the experimental group by the odds in the control group. This is used commonly in case-control studies.

Risk ratio/Relative risk: Risk is the probability of occurrence of an event. It is calculated by dividing the number of events that occurred by the number of people at risk. The risk ratio is calculated by dividing the risk in the control group by the risk in the experimental group. This is used commonly in cohort/prospective studies.

Apart from the point estimate, inferential statistics requires reporting indicators of the significance of results, viz.

p-value: This is the probability of occurrence of a result by chance. P-value <0.05 is considered statistically significant by convention.

Confidence interval: A range of values with a given degree of confidence that is likely to contain a population value is called the confidence interval (CI). The 95% confidence interval is a range of values that we can be 95% confident contains the true mean of the population.

Hypothesis Testing [7]

A hypothesis is a specific, testable statement or prediction about the outcome of a scientific study, and hypothesis testing is a statistical method used to draw inferences about a hypothesis generated in a population based on a sample of data from that population. The hypothesis can be null or alternate. We intend to reject the null hypothesis for decision-making, and thus, the alternate hypothesis represents the intentions we have for the research.

Hypothesis testing provides information regarding the plausibility of the null hypothesis. The choice of tests depends on the type of variables of concern and the distribution of data (normal or non-normal). The distribution of a given data can be ascertained either by graphical methods (histogram, Q-Q plot) or statistical tests (Shapiro-Wilk test, Kolmogorov Smirnov test).

Case 1 When we want to compare the pre-post exposure value (two time-points; most commonly, baseline and one follow-up) of a variable within the same group (Fig. 2.4).

Case 2 When we want to compare a variable between two different groups (Fig. 2.5).

Case 3 When we want to compare values at more than two-time points (most commonly, baseline and more than one follow-up visit) within the same group, i.e., multiple measurements have been taken from the same participant (Fig. 2.6).

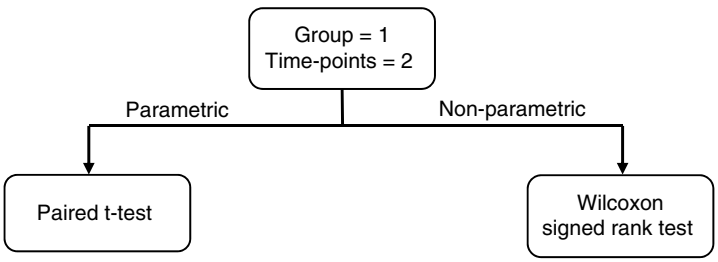

Fig. 2.4 Statistical test for paired comparison

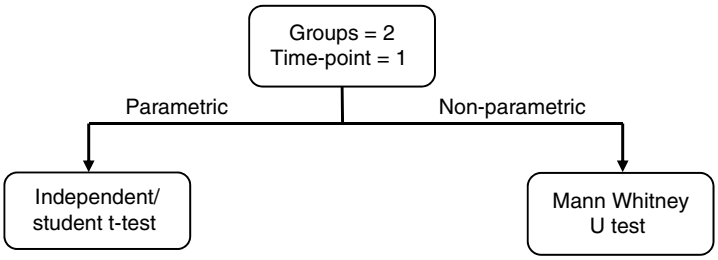

Fig. 2.5 Statistical test for comparison between two groups

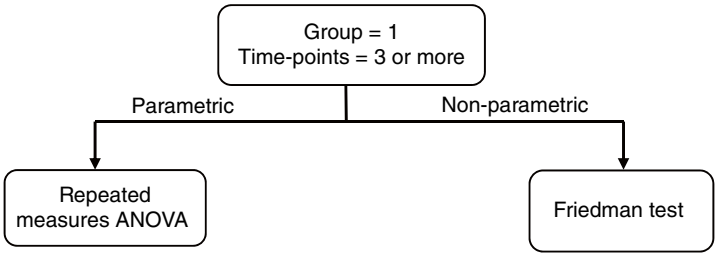

Fig. 2.6 Statistical test for comparison for repeated measures (>2 time-points) within a group

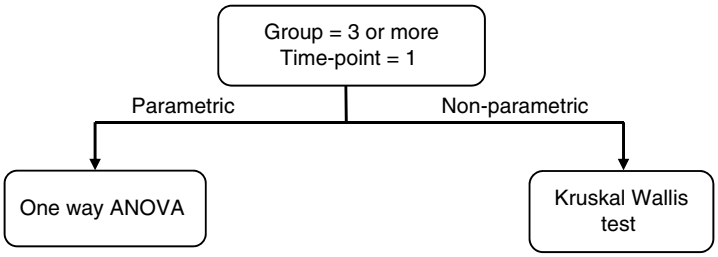

Fig. 2.7 Statistical test for comparison between more than two groups

Case 4 When we want to compare values of a variable between three or more different groups (Fig. 2.7).

Case 5 When we want to compare three or more groups at two or more time points, we should use mixed model ANOVA. This can also be used when we want to compare two groups at three or more time points.

Case 6 When we want to comment on the correlation between two continuous variables (Fig. 2.8).

Case 7 When we want to test the association between two independent categorical variables, the **chi-square test** is used (Fisher exact test is used when more than 20% of cells have expected frequencies <5).

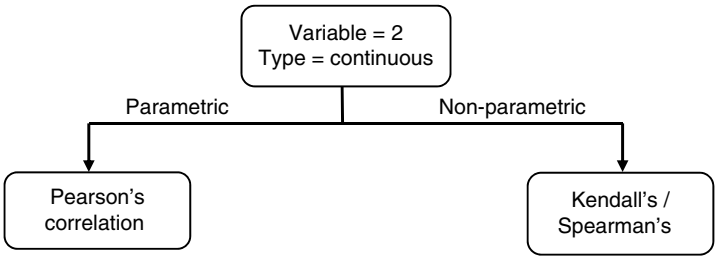

Fig. 2.8 Statistical test for understanding the strength of correlation between continuous variables

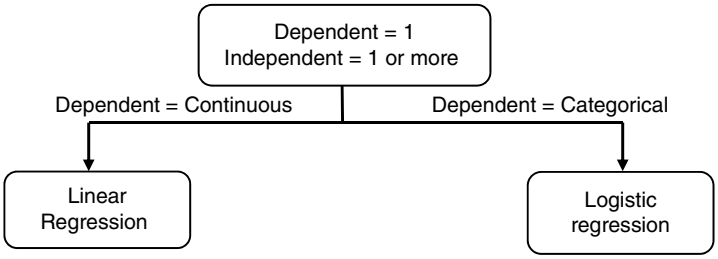

Fig. 2.9 Type of regression based on outcome variable

Case 8 When we want to mathematically describe the quantitative relationship between variables, regression analysis is performed. There is one dependent or response variable that we want to predict, and there is one or more independent variable(s) (Fig. 2.9).

Case 9 When the data has comparisons for time to an event, we perform survival analysis. The tests for survival analysis are semi-parametric. We perform log-rank tests and Cox proportional hazard tests and calculate hazard ratios for survival analysis.

Case 10 When we want to calculate the cut-off for sensitivity and specificity of a new diagnostic test that gives results on a continuous or ordinal scale, we can use receiver operator characteristic curve analysis.

These cases form the basis of the chapters of this book in your hand. The explanation of derivations for these hypothesis tests is beyond the scope of this book. However, we have tried to explain the step-by-step methodology and interpretations of results from these tests in detail in the forthcoming chapters of this book.

References

1. Simpson SH. Creating a data analysis plan: what to consider when choosing statistics for a study. Can J Hosp Pharm. 2015;68(4):311–7.
2. Olson K. What are data? Qual Health Res. 2021;31(9):1567–9.
3. Hazra A, Gogtay N. Biostatistics series module 1: basics of biostatistics. Indian J Dermatol. 2016;61(1):10–20.
4. Kaliyadan F, Kulkarni V. Types of variables, descriptive statistics, and sample size. Indian Dermatol Online J. 2019;10(1):82–6.
5. Nick TG. Descriptive statistics. Methods Mol Biol. 2007;404:33–52.
6. Braitman LE. Statistical estimates and clinical trials. J Biopharm Stat. 1993;3(2):249–56.
7. Yarandi HN. Hypothesis testing. Clin Nurse Spec. 1996;10(4):186–8.

Chapter 3
Introduction to R Environment and Basic Commands

Guruprasad Padmanabhan, Archana Mishra, and Anand Srinivasan

As we described in the previous chapters regarding the usefulness of R in the field of data science for life science professionals, we shall proceed with downloading the latest version of R from the website https://cran.r-project.org/and installing it [1]. Following the installation of R, the integrated development environment (IDE) for R, RStudio, should be installed. RStudio can be downloaded from https://posit.co/downloads/ [2]. We will proceed with writing codes in R using RStudio.

Rstudio Graphical User Interface (GUI) Introduction

There are 4 main windows

1. Source—Upper left. Where you can write and save your R code.
2. Console—Lower left. This is where you primarily code and evaluate it.
3. Environment and History—Upper right. This is where you can see all your objects and tables.
4. Files, plots, packages, and help—Lower right. Useful for navigating and finding files.

We can visualize the three panels (console, environment and files/plots) as we open R studio (Fig. 3.1).

So let's start coding in the R console.

G. Padmanabhan
Metadata Management for Clinical Operations, Novartis, Hyderabad, India

A. Mishra · A. Srinivasan (✉)
Department of Pharmacology, All India Institute of Medical Sciences, Bhubaneswar, Odisha, India
e-mail: anandsrinivasan@aiimsbhubaneswar.edu.in

A. Srinivasan et al. (eds.), *R for Basic Biostatistics in Medical Research*,
https://doi.org/10.1007/978-981-97-6980-3_3

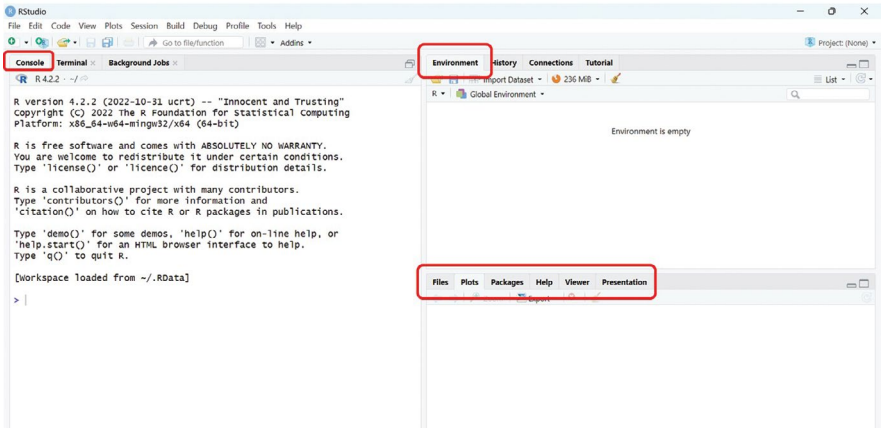

Fig. 3.1 The three panels in RSudio IDE on boot

Prompts in R Console

">"prompt: This means that it is waiting for you to pass on a command.

"+" prompt: This means the prompt is expecting more from the line of code already entered, a sort of continuation.

When this happens, you can exit using the "Esc" key.

Coding in R

Tap on the console so that you see that the cursor is blinking. Now, type any number in it.

```
9
## [1] 9
67.4
## [1] 67.4
```

Now press enter. R will return the number you entered.

Now type some simple arithmetic operations in the console and press enter. R will return the answer.

```
10+5
## [1] 15
66/2
## [1] 33
7-5
## [1] 2
```

R will evaluate whatever expression we enter in the console as long as it is a valid expression. The only disadvantage of typing our codes in the console is that the codes cannot be edited and saved. If we need to make modifications to the codes, we need to type again.

Both these disadvantages could be overcome by typing the codes in the script/source panel of R studio.

Script panel can be opened by clicking on File –> New File –> R Script (Fig. 3.2), and now the screen should appear like this (Fig. 3.3).

The codes from the script can be run using CTRL+ENTER. CTRL+ENTER transfers the code to the console for execution and jumps the cursor to the next line. Additionally, codes could also be run by selecting the desired part of codes and clicking on Run at the right corner of the script window. We can edit and modify codes here and store the codes written in the source window as a plain text file for the reproducibility of results for future use.

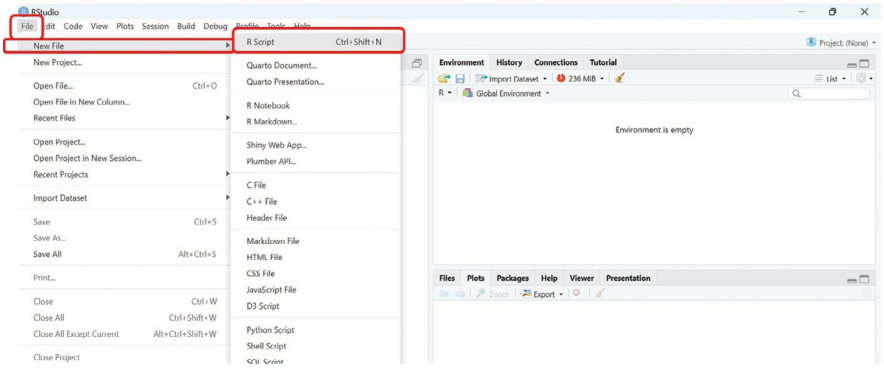

Fig. 3.2 Opening the script panel in RStudio

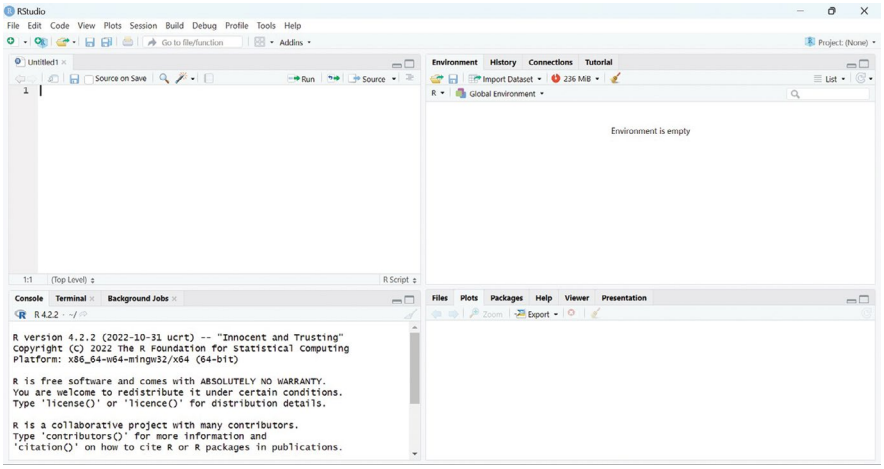

Fig. 3.3 The four-panel RStudio IDE

Data Types

There are five major types of data identified by R language:

1. Numeric/float—Number with decimals.
2. Integer—Number with no decimals.
3. String—Sequences of characters.
4. Logical/Boolean—TRUE or FALSE
5. Factors—Data objects with levels

Numeric/Float

These are numbers with decimals.

```
10.5
## [1] 10.5
689.9999
## [1] 689.9999
```

R also considers numbers without explicit decimals as numeric. Why they are not considered as integers and how to represent integers will be discussed subsequently.

```
67
## [1] 67
100
## [1] 100
```

In order to understand the data type of an object in R, there is a function called "typeof" and another function called "class". Let's try these.

```
typeof(10.5)
## [1] "double"
class(10.5)
## [1] "numeric"
typeof(100)
## [1] "double"
class(100)
## [1] "numeric"
```

Integers

These are numbers without decimals. They are …-3, −2, −1, 0, 1, 2, 3, … from -infinity to +infinity. To make R understand that a given numeral is an integer, a number without decimals should be followed by "L". Lets see.

```
39L
## [1] 39
1000L
## [1] 1000
```

Lets verify its type.

```
typeof(39L)
## [1] "integer"
class(1000L)
## [1] "integer"
```

Characters and Strings

Character or sequence of characters (strings). For example, names. However, we have to write the characters within single(') or double(") quotes.

```
"donald trump"
## [1] "donald trump"
"abdul kalam"
## [1] "abdul kalam"
```

Characters without quotes are used for object assignment. We will discuss objects later in this chapter. Now lets confirm the data type of these strings.

```
typeof("donald trump")
## [1] "character"
typeof("abdul kalam")
## [1] "character"
class("simple")
## [1] "character"
```

Logical/Boolean

This is a special data type. There are only 2 entries for this data type - TRUE and FALSE. These should not be put under quotes.

```
TRUE
## [1] TRUE
FALSE
## [1] FALSE
typeof(TRUE)
## [1] "logical"
class(FALSE)
## [1] "logical"
```

Factors

Factors are used to categorize the data and store it as levels. They store categorical data in the form of either “strings” or integers.

```
sex = factor(c("Male", "Female", "Male", "Male", "Female"))

##If we want to check the class of the vector "sex"
class(sex)
## [1] "factor"
#A vector of the type factor may have 2 or more levels
# To check levels of the variable in the vector sex
levels(sex)
## [1] "Female" "Male"
```

There are 2 other data types - raw and complex. However, we will not be discussing these as they are not useful for our purposes in medical research.

Data Structures

Generally, when we collect data from our research projects, we have data for each variable from more than one participant, animal, or observation. Data structures refer to a specific method for setting up computer data (which is collections of individual objects) such that it can be utilized efficiently.

Data structures in R can be classified based on the dimensionality and whether elements are identical.

1. Homogenous—contains objects of the same data type.
   (a) Vector—1 dimension
   (b) Matrix—2 dimensions
   (c) Array—n dimensions
2. Heterogenous—Contain objects of arbitrary data types.
   (a) List—Can contain any object.
   (b) Dataframe—This is a list containing two or more vectors of equal length.

Vectors

Vectors are ordered collections of objects of identical data type of a given length. To create a vector, we use the "c()" function also called concatenation. The function 'c' is used to combine the arguments. This coerces all the arguments into a single data type to form a vector.

```
c(1, 2, 3, 4, 5)
## [1] 1 2 3 4 5
c("ram", "lakshman", "a")
## [1] "ram" "lakshman" "a"
```

As you just saw, a vector can be created from any of the four datatypes, provided that the data type of the objects of the vector is the same. Hence, we can create a numeric vector (containing only numeric objects), integer vector, character vector, or boolean vector.

Let us use "typeof", and "class" on vectors and see what occurs.

```
typeof(c(4L, 54L, 65L))
## [1] "integer"
class(c(45.3, 65, 100.5958))
## [1] "numeric"
```

```
c(4L, 54L, 65L) # integer vector
## [1]  4 54 65
c(45.3, 65, 100.5958) # numeric vector
## [1]  45.3000  65.0000 100.5958
c("a", "b", "c", "e") # character vector
## [1] "a" "b" "c" "e"
c(TRUE, FALSE, FALSE) # logical vector
## [1]  TRUE FALSE FALSE
```

No surprises. These functions return the data type of the constituents of the vector. However, there is a function to test whether a given object is a vector, "is.vector()".

```
is.vector(c("a", "b", "c", "e"))
## [1] TRUE
is.vector(c(TRUE, FALSE, FALSE))
## [1] TRUE
is.vector(1)
## [1] TRUE
```

A single object can be considered as a vector containing only a single entry. Hence, the output of the previous line.

Let us see what will happen if we try to create a vector containing objects of different data types.

```
c(456.77, 10L, TRUE, "hello")
## [1] "456.77" "10" "TRUE" "hello"
```

As you can see, R converts all objects to the most feasible data type, i.e., a character, in this context. Similarly, if we try to concatenate numeric, integer, and logical objects, all will be converted to numeric. TRUE will be interpreted as 1, and FALSE as 0.

```
c(456.77, 10L, TRUE, FALSE)
## [1] 456.77  10.00   1.00   0.00
```

Try for yourself - concatenate integers and logical objects.

Simple Functions on Vectors

Let us try some simple functions. Such as sum, mean, etc. Let us look at the heart rate of 5 healthy subjects.

```
c(80, 76, 85, 90, 94)
## [1] 80 76 85 90 94
sum( c(80, 76, 85, 90, 94))
## [1] 425
mean( c(80, 76, 85, 90, 94))
## [1] 85
```

Since it is laborious to enter the entire vector every time we want to perform any operation on it, we can assign the vector a name. Now, we can perform any operation on the vector by performing on the name assigned to it. This name assigned is called an **R object**. We can perform assignments using "<–" or "=", and these are called assignment operators.

```
vector_1 <- c(80, 76, 85, 90, 94)
vector_2 = c("a", "b", "c")
vector_1
## [1] 80 76 85 90 94
vector_2
## [1] "a" "b" "c"
is.vector(vector_1)
## [1] TRUE
mean(vector_1)
## [1] 85
length(vector_2) # length() gives the number of elements in the vector.
## [1] 3
```

Let us do some operations on vectors.

```
vector_3 <- 1:10 # another way to create a vector. Will return
consecutive numbers from 1 to 10.
vector_3 # let us look at vector_3
##  [1]  1  2  3  4  5  6  7  8  9 10
sum(vector_3)
## [1] 55
mean(vector_3)
## [1] 5.5
sd(vector_3) # standard deviation
## [1] 3.02765
```

Subsetting Vectors

If we want to pick up the eighth element of the vector or all elements from the third to the seventh, we can do this by using the "[]" operator.

```
vector_3
##  [1]  1  2  3  4  5  6  7  8  9 10
vector_3[8]
## [1] 8
vector_3[3:7]
## [1] 3 4 5 6 7
# picking the 2nd, 4th and 10th element
vector_3[c(2, 4, 10)]
## [1]  2  4 10
vector_3[4] <- 29 # reassigning a value
vector_3
##  [1]  1  2  3 29  5  6  7  8  9 10
```

Matrix

You can think of matrices as two-dimensional "vectors" containing only elements of a single data type. "matrix()" function can be used to create a matrix. You have to send a vector of values, which will be populated as a matrix. Let us see how.

```
matrix_1 <- matrix(data = c(1:100), nrow = 10, ncol = 10) # matrix with
numeric values
matrix_1
##       [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
##  [1,]    1   11   21   31   41   51   61   71   81    91
##  [2,]    2   12   22   32   42   52   62   72   82    92
##  [3,]    3   13   23   33   43   53   63   73   83    93
##  [4,]    4   14   24   34   44   54   64   74   84    94
##  [5,]    5   15   25   35   45   55   65   75   85    95
##  [6,]    6   16   26   36   46   56   66   76   86    96
##  [7,]    7   17   27   37   47   57   67   77   87    97
##  [8,]    8   18   28   38   48   58   68   78   88    98
##  [9,]    9   19   29   39   49   59   69   79   89    99
## [10,]   10   20   30   40   50   60   70   80   90   100
letters # vector containing all the english alphabet.
##  [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q"
"r" "s"
## [20] "t" "u" "v" "w" "x" "y" "z"
matrix_2 <- matrix(data=letters[1:20], nrow=5) # It is enough to specify
either the number of rows or columns. R will figure out the rest
```

Let us try to subset a matrix. Here, since there are 2 dimensions, we have to specify 2 indexes: a row followed by a column number separated by a comma. The subsetting operator is the same '[]'.

```
matrix_1[1, 1]
## [1] 1
matrix_1[3:5, 5:8] # subsetting multiple elements at once.
##      [,1] [,2] [,3] [,4]
## [1,]   43   53   63   73
## [2,]   44   54   64   74
## [3,]   45   55   65   75
matrix_1_sub <- matrix_1[3:5, 5:8] # assigning the subsetted matrix to new
object
matrix_1_sub
##      [,1] [,2] [,3] [,4]
## [1,]   43   53   63   73
## [2,]   44   54   64   74
## [3,]   45   55   65   75
```

Lists

Lists are highly versatile containers. You can put different R objects in the same list. Thus, lists can be identified as one-dimensional heterogeneous data structures.

```
# Let us put a couple of vectors in a new list
list_1 <- list(vec_1 = vector_1, vec_2 = vector_2)
# let us put a matrix in it
list_1[["mat_1"]] <- matrix_1 # subsetting and assigning at the same time.
list_1[["test"]] <- 1
list_1
## $vec_1
## [1] 80 76 85 90 94
##
## $vec_2
## [1] "a" "b" "c"
##
## $mat_1
##       [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
##  [1,]    1   11   21   31   41   51   61   71   81    91
##  [2,]    2   12   22   32   42   52   62   72   82    92
##  [3,]    3   13   23   33   43   53   63   73   83    93
##  [4,]    4   14   24   34   44   54   64   74   84    94
##  [5,]    5   15   25   35   45   55   65   75   85    95
##  [6,]    6   16   26   36   46   56   66   76   86    96
##  [7,]    7   17   27   37   47   57   67   77   87    97
##  [8,]    8   18   28   38   48   58   68   78   88    98
##  [9,]    9   19   29   39   49   59   69   79   89    99
## [10,]   10   20   30   40   50   60   70   80   90   100
##
## $test
## [1] 1
```

As you saw, subsetting a list is done with a double "[[]] "operator. There is another way too: the "$" operator. Lets look at both of them.

```
list_1
## $vec_1
## [1] 80 76 85 90 94
##
## $vec_2
## [1] "a" "b" "c"
##
## $mat_1
##       [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
##  [1,]    1   11   21   31   41   51   61   71   81    91
##  [2,]    2   12   22   32   42   52   62   72   82    92
##  [3,]    3   13   23   33   43   53   63   73   83    93
##  [4,]    4   14   24   34   44   54   64   74   84    94
##  [5,]    5   15   25   35   45   55   65   75   85    95
##  [6,]    6   16   26   36   46   56   66   76   86    96
##  [7,]    7   17   27   37   47   57   67   77   87    97
##  [8,]    8   18   28   38   48   58   68   78   88    98
##  [9,]    9   19   29   39   49   59   69   79   89    99
## [10,]   10   20   30   40   50   60   70   80   90   100
##
## $test
## [1] 1
list_1[[3]] # indexing by position
##       [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
##  [1,]    1   11   21   31   41   51   61   71   81    91
##  [2,]    2   12   22   32   42   52   62   72   82    92
##  [3,]    3   13   23   33   43   53   63   73   83    93
##  [4,]    4   14   24   34   44   54   64   74   84    94
##  [5,]    5   15   25   35   45   55   65   75   85    95
##  [6,]    6   16   26   36   46   56   66   76   86    96
##  [7,]    7   17   27   37   47   57   67   77   87    97
##  [8,]    8   18   28   38   48   58   68   78   88    98
##  [9,]    9   19   29   39   49   59   69   79   89    99
## [10,]   10   20   30   40   50   60   70   80   90   100
list_1[["mat_1"]] # indexing by name
##       [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
##  [1,]    1   11   21   31   41   51   61   71   81    91
##  [2,]    2   12   22   32   42   52   62   72   82    92
##  [3,]    3   13   23   33   43   53   63   73   83    93
##  [4,]    4   14   24   34   44   54   64   74   84    94
##  [5,]    5   15   25   35   45   55   65   75   85    95
##  [6,]    6   16   26   36   46   56   66   76   86    96
##  [7,]    7   17   27   37   47   57   67   77   87    97
##  [8,]    8   18   28   38   48   58   68   78   88    98
##  [9,]    9   19   29   39   49   59   69   79   89    99
## [10,]   10   20   30   40   50   60   70   80   90   100
list_1$vec_1 # indexing by name with "$" operator.
## [1] 80 76 85 90 94
```

Dataframes

A dataframe is a special form of a list with a structure such that each entry in a dataframe needs to be a vector of the same length. A dataframe is a 2-dimensional heterogeneous data structure. A dataframe is the most common data structure as it is used to store our research data in tabular form. A data frame could be created using function "data.frame".

```
names <- c("guru", "anand", "inderjeet", "praveen", "someone")
ages <- c(75, 45, 53, 55, 90)
gender <- c("M", "M", "M", "M", "F")
diabetes <- c(TRUE, FALSE, F, F, T)
# now lets combine all these vectors into a single data frame
data_1 <- data.frame(names, ages, gender, diabetes)
data_1
##       names ages gender diabetes
## 1      guru   75      M     TRUE
## 2     anand   45      M    FALSE
## 3 inderjeet   53      M    FALSE
## 4   praveen   55      M    FALSE
## 5   someone   90      F     TRUE
View(data_1)
```

As shown above, we can understand that TRUE and FALSE can be alternatively represented as T and F, respectively.

Operations on dataframes will be dealt with in the subsequent chapter. Before we begin to code for analysis, we should understand how does R function. A typical code in R will appear as:

Object <− function(arguments).

Variables, data, functions, outcomes, etc., are kept in the computer's active memory as named objects when we are running codes in R. With operators (arithmetic, logical, comparison, etc.) and functions, we can perform operations on these objects. Objects are named by the user, whereas functions are mostly predefined. Some functions may not need us to put any argument, and the function will use default arguments.

References

1. R Core Team. R: a language and environment for statistical computing. Vienna, Austria: R Foundation for Statistical Computing; 2021. https://www.R-project.org/
2. RStudio Team. RStudio: integrated development for R. Boston, MA: RStudio, PBC; 2020. http://www.rstudio.com/

Chapter 4
Data Handling and Manipulation in R with Descriptive Statistics

Archana Mishra, Inderjeet Singh, and Anand Srinivasan

We have already revised the basics of biostatistics and why R is essential to us, and we can now visualize the R and RStudio interface. So, it is time for us to perform some mathematical and statistical operations on the data collected from our research. Before we start getting some useful insights from our data, we need to make sure that we bundle all the files for one project together in one place and keep track of them. R studio gives us the convenience of organizing our files through the creation of a working directory. Additionally, we may enjoy the ease of sharing it with co-investigators. The working directory is the place where the files can be stored, saved, and retrieved from. This is the place where R communicates with. It is a good practice to keep one working directory for each R project. A project in RStudio is simply a file that keeps track of the environment, workspace, and other things related to RStudio. A working directory in R studio can be created in a few simple steps. We can locate the working directory at the top right corner of the console. In the screenshot given below, we can see that no working directory [i.e., project (none)] is available for the operations at this point in time.

Supplementary Information The online version contains supplementary material available at https://doi.org/10.1007/978-981-97-6980-3_4.

A. Mishra · A. Srinivasan (✉)
Department of Pharmacology, All India Institute of Medical Sciences, Bhubaneswar, Odisha, India
e-mail: anandsrinivasan@aiimsbhubaneswar.edu.in

I. Singh
Experimental Drug Development Centre, Singapore, Singapore

A. Srinivasan et al. (eds.), *R for Basic Biostatistics in Medical Research*,
https://doi.org/10.1007/978-981-97-6980-3_4

Creation of an R project

Step 1. If we want to create a new R project, we just need to click the downward arrow beside Project (None), and then a list pops up. For the creation of a new project, we select the first option, i.e., "New Project…" (Fig. 4.1).

Step 2. Selecting the working directory: We can choose a new directory or existing directory based on the organization of our data files in the system. If we have already saved the data files in a folder, it is advised to create an R project in that existing directory folder; otherwise, a new directory may be created (Fig. 4.2).

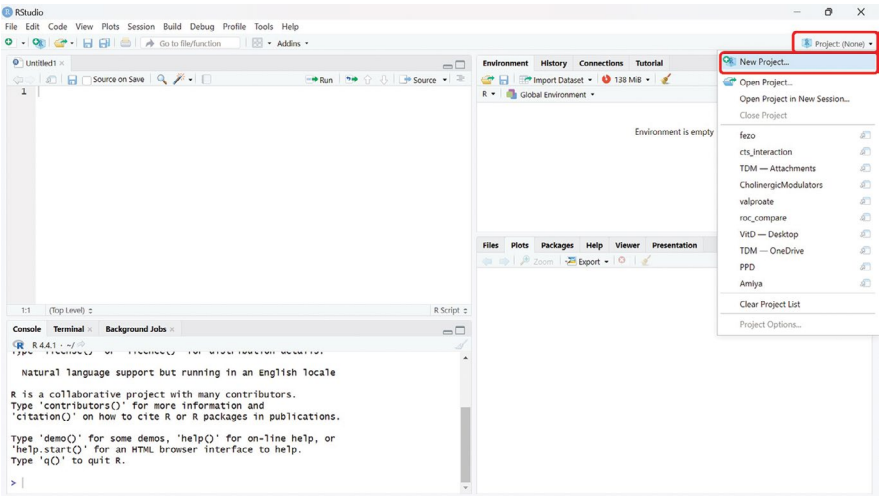

Fig. 4.1 Step1: Creation of new project

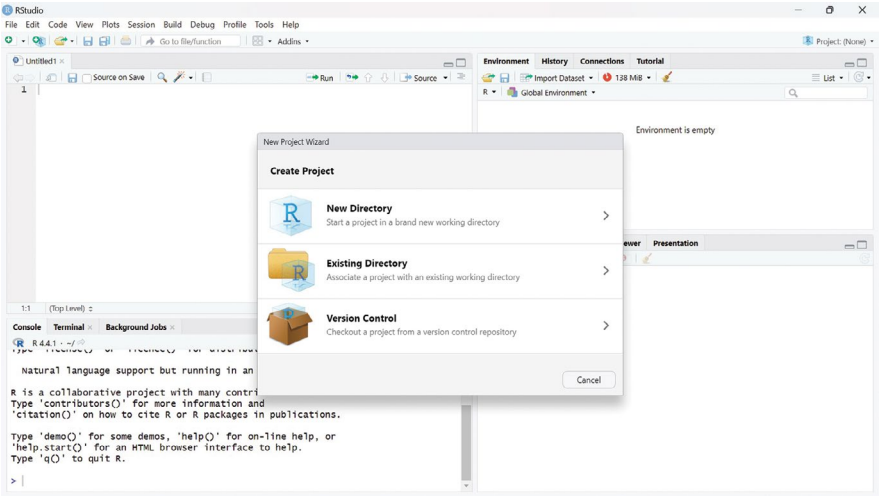

Fig. 4.2 Selecting new/existing directory

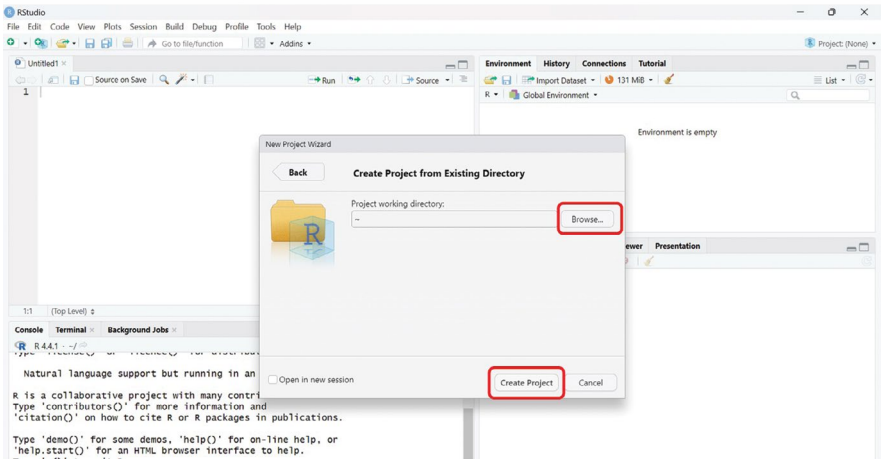

Fig. 4.3 Selecting the folder for the creation of a new R project

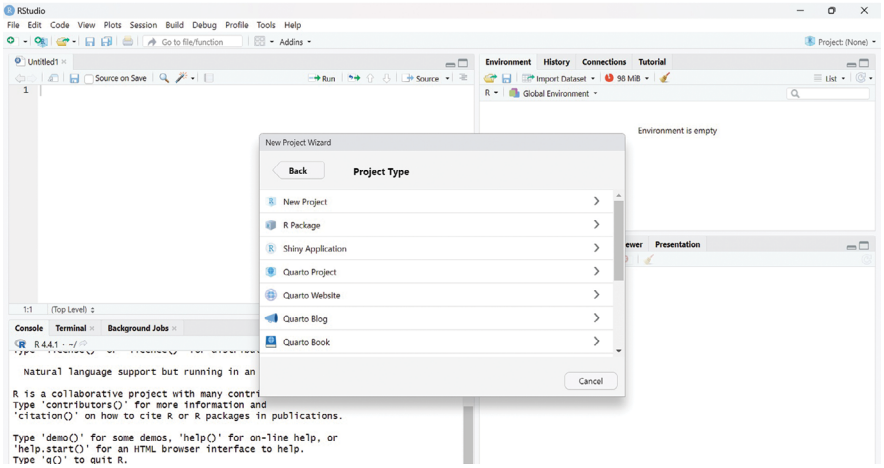

Fig. 4.4 Selecting a new project

Step 3a: When we have all our data files and/or scripts already stored in one folder, we select the "Existing directory" option in the pop-up window. Thereafter, we browse through the system to select the desired folder and select "Create Project" (Fig. 4.3).

Step 3b: If we want to create a new directory, we should select that we want to create a new project in the next window and then select the name and path of the directory to create a project in the subsequent window (Fig. 4.4).

Similarly, a new project can be created from the "File" drop-down menu on the top left side of the GUI.

We will all agree that we collect our data in a tabular form where the first row is assigned to the name of the variables for which data are collected, and the first column contains patient IDs or names as suitable. We already know from the previous chapter that the representation described is called a dataframe. Here, each column is a vector containing identical data objects, and each row contains data from a single participant for all the measured variables. We may save this dataframe as an Excel file, a comma-separated values (.csv) file, or any other format that may be suitable.

Importing data in R is the first step before analyzing the data.

Steps for Importing Data in R

1. Identify the file format where the data has been stored and understand the data.
2. Choose a function in R depending on the file types
   (a) *"read.csv" for data stored as* "csv" *files.*
   (b) *"read.table" for data stored as a table in* "txt" *files.*

Functions from dedicated packages (Chap. 5) for reading Excel sheets ("readxl" package), SAS data files, or other forms of data files can be used to import the data. The "Import Dataset" option from the "File" dropdown menu on the top of the RStudio GUI can also be used to import data.

Understanding "read.csv" Function

Has the following attributes

1. *file*
   (a) Provide a file name (if the file is stored in the working directory).
   (b) Provide the file address (file storage location in the system).

(c) Use "file.choose()".

Assuming we have the file dm.csv stored in the working directory.

2. *header*

```
     dm = read.csv("dm.csv")
head(dm,n=2)
##   group age sex weight ldl fasting_before fasting_after difference
response
## 1     1  46   1    101 104            155           111         44
1
## 2     1  36   0     97 100            153           133         20
1
##   hba1c
## 1   6.4
## 2   5.4
```

It is either "header = T" or "header = F". If "T", this indicates that the first row of the dataset contains the names of the columns (names of the variables in the dataset). If "F" then the columns are numbered automatically like V.1, V.2 …

```
dm = read.csv("dm.csv", header=F)
head(dm,n=2)
##      V1  V2  V3     V4  V5             V6            V7         V8
V9
## 1 group age sex weight ldl fasting_before fasting_after difference
response
## 2     1  46   1    101 104            155           111         44
1
##     V10
## 1 hba1c
## 2   6.4
```

When the column names are not specified in the first row, one can specify the header as "F," and the names of the columns (variable names) can be provided using the *"col.names"* argument.

3. *sep*

```
dm = read.csv("dm.csv", header=F, col.names = c("Group","Age","Sex","weight",
"ldl","fasting_b","fasting_a","diff","response","hbA1c"))
head(dm,n=2)
##   Group Age Sex weight ldl      fasting_b     fasting_a       diff
response
## 1 group age sex weight ldl fasting_before fasting_after difference
response
## 2     1  46   1    101 104            155           111         44
1
##   hbA1c
## 1 hba1c
## 2   6.4
```

This is used to provide a symbol that separates two values in the dataset. Usually, a "," in a csv file. Sometimes, it could be a "tab" or ";".

4. *dec*
   Use this argument to specify the symbol used for decimal points. Usually, this is ".", but for some data sets, this could be ",".
5. *row.names*
   This can be used to specify names to the rows. Similar to the *col.names* argument.
6. *na.strings*
   It should be used to specify characters in the dataset that indicate a missing value.

```
dm.m <- read.csv("dm_missing.csv", na.strings = c(" ","","-"))
head(dm.m)
##   group age sex weight ldl fasting_before fasting_after difference
response
## 1     1  46   1    101 104            155           125         30
1
## 2     1  36   0     97 100            153           163         -9
1
## 3     1  53   0    119  81            130           100         30
1
## 4     1  63   0     NA  86            147           126         21
0
## 5     1  62   1     88  75            130           101         29
1
## 6     1  65   1     58 115            171           146         25
1
##   hba1c
## 1   6.4
## 2   5.4
## 3   6.3
## 4   6.7
## 5   6.4
## 6   6.2
```

```
dm = read.csv("dm.csv")
str(dm)
## 'data.frame':    40 obs. of  10 variables:
##  $ group         : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ age           : int  46 36 53 63 62 65 33 46 61 38 ...
##  $ sex           : int  1 0 0 0 1 1 1 1 1 1 ...
##  $ weight        : int  101 97 119 48 88 58 40 71 41 62 ...
##  $ ldl           : int  104 100 81 86 75 115 146 132 138 141 ...
##  $ fasting_before: int  155 153 130 147 130 171 197 201 194 209 ...
##  $ fasting_after : int  111 133 100 126 101 146 147 125 192 113 ...
##  $ difference    : int  44 20 30 21 29 25 50 76 2 96 ...
##  $ response      : int  1 1 1 0 1 1 1 1 1 1 ...
##  $ hba1c         : num  6.4 5.4 6.3 6.7 6.4 6.2 6.2 6.4 6.4 6.2 ...
```

By default, the empty cells in a data file will be considered as missing values (NA). So, if the data file only has missing cells (not any other notation like "-"), the inclusion of *"na.strings"* is not required in the code. After we have imported the data, the next step is to check the data structure to ensure that the column variables are correctly classified/labeled. The function used to understand the type of the column variables is *"str"*.

"colClasses" function is used to define the classes to columns i.e. to variables representing as columns like numeric, integer, string, logical, etc.

If not specified, default labeling occurs during the initial reading of the dataset into R.

```
dm = read.csv("dm.csv", na.strings=" ", colClasses
=c("factor","numeric","factor","numeric","numeric","numeric","numeric",
"numeric","numeric","numeric"))
str(dm)
## 'data.frame':    40 obs. of  10 variables:
##  $ group         : Factor w/ 2 levels "1","2": 1 1 1 1 1 1 1 1 1 1 ...
##  $ age           : num  46 36 53 63 62 65 33 46 61 38 ...
##  $ sex           : Factor w/ 2 levels "0","1": 2 1 1 1 2 2 2 2 2 2 ...
##  $ weight        : num  101 97 119 48 88 58 40 71 41 62 ...
##  $ ldl           : num  104 100 81 86 75 115 146 132 138 141 ...
##  $ fasting_before: num  155 153 130 147 130 171 197 201 194 209 ...
##  $ fasting_after : num  111 133 100 126 101 146 147 125 192 113 ...
##  $ difference    : num  44 20 30 21 29 25 50 76 2 96 ...
##  $ response      : num  1 1 1 0 1 1 1 1 1 1 ...
##  $ hba1c         : num  6.4 5.4 6.3 6.7 6.4 6.2 6.2 6.4 6.4 6.2 ...
```

Packages to Read Data from Other Formats

"read.csv" is included in the base package of R and thus becomes the default for reading data for further analysis, while there are packages to read other formats. To name a few: *readr*: enables a faster and friendly reading of datasets.

readxl: enables R to read an Excel file.

haven: enables R to read and write dataset formats created in other statistical tools like SPSS, SAS, STATA, etc.

Rstudio "Import Dataset"

This is a dropdown menu on the top of the RStudio GUI that can also be used to import data. The appropriate option is selected based on the data format. The required packages will be installed based on the selected option if they are not installed in the system.

Writing Data in R

Just as a data file can be read using *"read.csv"*, any dataframe created in R can be written to a "csv" file using *"write.csv"* function. The file generated will be saved in the working directory. The arguments for this function are:

x: This is the dataframe to write, i.e., the data that needs to be written.
file: This is the name of the new data file that will be generated in the working directory.
na: This defines the characters used to be written on the data file if there are any missing values.
sep: The character symbol defining the separation between two values.
dec: Symbol to be taken as the decimal point.

```
write.csv(dm,file="myfile.csv", na="NA")
```

Operations on a Dataframe

Now, we will learn some basic operations on dataframe.

As we already discussed, importing the data and assigning it to an R object shall be the first step for starting any sort of operation.

```
dm = read.csv("dm.csv")
```

Data Visualization (Excluding Graphs)

We can visualize the initial and final parts of the dataframe by using the following codes.

```
head(dm,2) #The number in the code is the number of rows to be displayed in
the dataframe, and its default value is 6.
##   group age sex weight ldl fasting_before fasting_after difference
response
## 1     1  46   1    101 104            155           111         44
1
## 2     1  36   0     97 100            153           133         20
1
##   hba1c
## 1   6.4
## 2   5.4
tail(dm,2)
##    group age sex weight ldl fasting_before fasting_after difference
response
## 39     2  59   0     84 170            236           183         53
1
## 40     2  60   0     39  98            163           103         60
0
##    hba1c
## 39   5.9
## 40   7.3
```

We can also check the data by clicking on the "dm" object in the environment tab (upper right quadrant).

We can identify the names of the columns (variables) by the following code.

```
names(dm)
##  [1] "group"          "age"            "sex"            "weight"
##  [5] "ldl"            "fasting_before" "fasting_after"  "difference"
##  [9] "response"       "hba1c"
```

We can also obtain the names of rows and columns by the following code.

```
rownames(dm)
##  [1] "1"  "2"  "3"  "4"  "5"  "6"  "7"  "8"  "9"  "10" "11" "12" "13" "14"
"15"
## [16] "16" "17" "18" "19" "20" "21" "22" "23" "24" "25" "26" "27" "28" "29"
"30"
## [31] "31" "32" "33" "34" "35" "36" "37" "38" "39" "40"
colnames(dm)
##  [1] "group"          "age"            "sex"            "weight"
##  [5] "ldl"            "fasting_before" "fasting_after"  "difference"
##  [9] "response"       "hba1c"
```

The total number of columns in a dataframe can be calculated by the following command.

```
length(dm)
## [1] 10
```

The total number of rows and columns can be obtained by the following commands.

```
nrow(dm)
## [1] 40
ncol(dm)
## [1] 10
dim(dm)
## [1] 40 10
```

The first number always denotes the number of rows (observations or participants), and the second number denotes the number of columns (variables). This pattern is always followed in R (like while indexing using []).

Indexing in a Dataframe

If we need to pick an element from a specific location in a dataframe, we need to enter the row followed by column numbers (not in reverse order) within a square bracket ("[]") separated by a comma (","). For example, if we need to extract the element in the fifteenth row and sixth column, we need to run

```
dm[15,6]
## [1] 179
```

If we need to extract all the elements from the fifteenth row, we must keep the column entry empty.

```
dm[15,]
##    group age sex weight ldl fasting_before fasting_after difference
response
## 15     1  40   1     55 121            179           157         22
0
##    hba1c
## 15   6.6
```

If both the row and column entries are kept empty, as shown below, the entire dataframe will be extracted.

```
dm[,]
##    group age sex weight ldl fasting_before fasting_after difference
response
## 1      1  46   1    101 104            155           125         30
1
## 2      1  36   0     97 100            153           163         -9
1
## 3      1  53   0    119  81            130           100         30
1
## 4      1  63   0     48  86            147           126         21
0
## 5      1  62   1     88  75            130           101         29
1
## 6      1  65   1     58 115            171           146         25
1
## 7      1  33   1     40 146            197            97        100
1
## 8      1  46   1     71 132            201           125         76
1
## 9      1  61   1     41 138            194           192          2
1
## 10     1  38   1     62 141            209           113         96
1
## 11     1  41   1     65 112            178            89         88
1
## 12     1  35   1     69 158            215           100        115
1
## 13     1  34   0     89 209            276           150        126
1
## 14     1  38   1     83 198            256           155        101
0
## 15     1  40   1     55 121            179           157         22
0
## 16     1  34   0     93 125            189           104         85
0
## 17     1  42   1    105 138            202           122         80
1
## 18     1  44   0     47 174            233           117        115
0
## 19     1  51   0     96  85            144           132         12
1
## 20     1  45   0     70 111            165           170         -5
1
## 21     2  42   0     61 122            192           133         59
1
## 22     2  35   0     72 128            180           173          7
1
## 23     2  56   1     75 144            211           188         23
0
## 24     2  59   0     40 119            184           183          1
0
## 25     2  56   0     96  83            136           114         22
1
```

```
## 26     2  42   0     78 130           199           115          84
0
## 27     2  34   1     54 148           215           127          88
0
## 28     2  43   1     86 118           180           146          34
1
## 29     2  48   0     79 146           200           179          20
0
## 30     2  44   0     87 114           172           126          45
1
## 31     2  47   0     85 138           191           166          26
1
## 32     2  51   0     72  91           155           139          16
0
## 33     2  50   0     65 119           187           156          31
1
## 34     2  48   1    101  94           155           133          22
1
## 35     2  49   1    102 147           210           192          18
0
## 36     2  66   0     72  92           146           128          18
1
## 37     2  51   0     48 148           214           178          36
0
## 38     2  53   1     72 147           136           110          26
0
## 39     2  59   0     84 170           236           183          53
1
## 40     2  60   0     39  98           163           123          40
0
##    hba1c
## 1    6.4
## 2    5.4
## 3    6.3
## 4    6.7
## 5    6.4
## 6    6.2
## 7    6.2
## 8    6.4
## 9    6.4
## 10   6.2
## 11   6.2
## 12   6.2
## 13   5.6
## 14   6.9
## 15   6.6
## 16   6.7
## 17   6.3
## 18   6.6
## 19   5.9
## 20   5.9
## 21   6.0
## 22   5.9
## 23   6.9
## 24   6.6
## 25   6.4
## 26   7.2
## 27   7.3
```

```
## 28   5.8
## 29   6.6
## 30   6.1
## 31   6.1
## 32   6.7
## 33   6.1
## 34   5.8
## 35   6.7
## 36   6.4
## 37   7.9
## 38   6.7
## 39   5.9
## 40   7.3
```

We can create a vector by using all the elements in a column of a dataframe using the following codes. For example, if we need to create a vector with all the elements from the last column(hba1c), we must run the following code.

```
dm[,length(dm)]
##  [1] 6.4 5.4 6.3 6.7 6.4 6.2 6.2 6.4 6.4 6.2 6.2 6.2 5.6 6.9 6.6 6.7
6.3 6.6 5.9
## [20] 5.9 6.0 5.9 6.9 6.6 6.4 7.2 7.3 5.8 6.6 6.1 6.1 6.7 6.1 5.8 6.7
6.4 7.9 6.7
## [39] 5.9 7.3
```

The command extracts all rows of the last column, which has been ensured by the argument “length(dm)”. There is another way of creating a vector by using the “$” symbol to specify the column.

```
dm$hba1c
##  [1] 6.4 5.4 6.3 6.7 6.4 6.2 6.2 6.4 6.4 6.2 6.2 6.2 5.6 6.9 6.6 6.7
6.3 6.6 5.9
## [20] 5.9 6.0 5.9 6.9 6.6 6.4 7.2 7.3 5.8 6.6 6.1 6.1 6.7 6.1 5.8 6.7
6.4 7.9 6.7
## [39] 5.9 7.3
dm_hba1c = dm$hba1c #This command stores all the "hba1c" values (the last
column of the dataframe) as a vector to the R object "dm_hba1c.
```

Editing a Dataframe

Replacing an Element

If we want to edit an element in the dataframe, we need to index that specific element and assign a value to the position corresponding to the index. For example, if we need to edit the value to 58 in the age variable of the fifth participant, we can run either of the following codes.

```
dm$age[5] = 58 #or
dm[5,2] = 58
```

Deleting the Contents of a Dataframe

If we need to delete only a cell, we need to assign "NA" to the variable whose location is indexed, as shown below.

```
dm$age[1] = NA
dm$age[1] = 46
```

In the second line above, we are just restoring the value.

If we need to delete all the contents in a column of a dataframe, we need to assign "NA" to the entire column. For example, if we want to remove all the contents of the "difference" column, we have to run the following code.

```
dm$difference = NA
```

Deleting Columns and Rows

For entirely dropping a column from a dataframe, run the following code.

```
dm = dm[,-c(8)]
head(dm,2)
##   group age sex weight ldl fasting_before fasting_after response hba1c
## 1     1  46   1    101 104            155           111        1   6.4
## 2     1  36   0     97 100            153           133        1   5.4
```

Similarly we can delete rows from a dataframe (for example, rows 1 and 2) by typing `dm = dm[-c(1,2),]`

Adding Columns

If we need to add columns to a dataframe, we need to add the name of the column to be added after "dm$" and assign the values as follows.

```
dm$difference = dm$fasting_before-dm$fasting_after
```

Converting the Type of the Variable of Columns in a Dataframe

We should identify the type of variables in the columns before proceeding with further analysis.

```
str(dm)
## 'data.frame':    40 obs. of  10 variables:
##  $ group        :  int  1 1 1 1 1 1 1 1 1 1 ...
##  $ age          :  num  46 36 53 63 58 65 33 46 61 38 ...
##  $ sex          :  int  1 0 0 0 1 1 1 1 1 1 ...
##  $ weight       :  int  101 97 119 48 88 58 40 71 41 62 ...
##  $ ldl          :  int  104 100 81 86 75 115 146 132 138 141 ...
##  $ fasting_before: int  155 153 130 147 130 171 197 201 194 209 ...
##  $ fasting_after:  int  111 133 100 126 101 146 147 125 192 113 ...
##  $ response     :  int  1 1 1 0 1 1 1 1 1 1 ...
##  $ hba1c        :  num  6.4 5.4 6.3 6.7 6.4 6.2 6.2 6.4 6.4 6.2 ...
##  $ difference   :  int  44 20 30 21 29 25 50 76 2 96 ...
```

We can see that all the columns are "integer" types other than the "hba1c" column. Even though this will not affect calculating the mean or while running statistical tests, it is better to convert them to "numeric" type.

We can convert them individually as follows.

```
dm$weight = as.numeric(dm$weight)
dm[,4] = as.numeric(dm[,4])
```

We can convert many columns together by using “apply” functions.

```
dm[,] = apply(dm[,],2,as.numeric)
```

The “apply” function uses three arguments.

apply(X, MARGIN, FUN).

x is the matrix or array on which the operation will be performed.

MARGIN specifies whether we want to apply the function across rows (1) or columns (2).

FUN is the function we need to carry out on X.

We should also convert “group”, “sex”, and “response” to factors.

```
dm$group = as.factor(dm$group)
dm$sex = as.factor(dm$sex)
dm$response = as.factor(dm$response)
```

We can verify if the process was completed successfully by re-running the “str” command.

```
str(dm)
## 'data.frame':    40 obs. of  10 variables:
##  $ group        :  Factor w/ 2 levels "1","2": 1 1 1 1 1 1 1 1 1 1 ...
##  $ age          :  num  46 36 53 63 58 65 33 46 61 38 ...
##  $ sex          :  Factor w/ 2 levels "0","1": 2 1 1 1 2 2 2 2 2 2 ...
##  $ weight       :  num  101 97 119 48 88 58 40 71 41 62 ...
##  $ ldl          :  num  104 100 81 86 75 115 146 132 138 141 ...
##  $ fasting_before: num  155 153 130 147 130 171 197 201 194 209 ...
##  $ fasting_after:  num  111 133 100 126 101 146 147 125 192 113 ...
##  $ response     :  Factor w/ 2 levels "0","1": 2 2 2 1 2 2 2 2 2 2 ...
##  $ hba1c        :  num  6.4 5.4 6.3 6.7 6.4 6.2 6.2 6.4 6.4 6.2 ...
##  $ difference   :  num  44 20 30 21 29 25 50 76 2 96 ...
```

Labeling the Categorical Variables

Now, we will explore the ways to label the categorical variables.

```
factor(dm$group,labels = c("Drug A","Drug B"))
##  [1] Drug A Drug A Drug A Drug A Drug A Drug A Drug A Drug A Drug A Drug A
## [11] Drug A Drug A Drug A Drug A Drug A Drug A Drug A Drug A Drug A Drug A
## [21] Drug B Drug B Drug B Drug B Drug B Drug B Drug B Drug B Drug B Drug B
## [31] Drug B Drug B Drug B Drug B Drug B Drug B Drug B Drug B Drug B Drug B
## Levels: Drug A Drug B
```

The above code will result in the labeling of factors in the results only. The dataframe still contains "1" and "2". Let us understand the difference between the two codes written below.

```
summary(factor(dm$group,labels = c("Drug A","Drug B")))
## Drug A Drug B
##     20     20
summary(dm$group)
##  1  2
## 20 20
```

If we need to change the labels in the dataframe, we need to run the following code.

```
levels(dm$group) = c("Drug A","Drug B")
summary(dm$group)
## Drug A Drug B
##     20     20
```

Relational Operators

We will shift our discussion to relational operators like "equal to", "greater than" and others.

```
#Equal to operator
dm$age == 34 #Note that the "=" appears twice and not once
##  [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
FALSE
## [13]  TRUE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
FALSE
## [25] FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
FALSE
## [37] FALSE FALSE FALSE FALSE
dm$age[5] == 34
## [1] FALSE
```

If we observe the results by running a relational operator in the code, only "TRUE" or "FALSE" are obtained, creating a logical object (atomic vector, vector, or multidimensional object, depending on the object on which the relational operations are performed). Note that we can index a particular element and raise the query. In R, "TRUE" is considered "1", and "FALSE" is considered "0". So, if we

want to count the number of participants who are aged 34, we need to run the following.

```
sum(dm$age == 34)
## [1] 3
```

Thus, 3 participants are 34 years old.
Similarly, we can try other relational operators which are self-explanatory.

```
#Less than operator
dm$age < 40
##  [1] FALSE  TRUE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE  TRUE FALSE
TRUE
## [13]  TRUE  TRUE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE
FALSE
## [25] FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
FALSE
## [37] FALSE FALSE FALSE FALSE
sum(dm$age < 40)
## [1] 9
dm$age <= 40
##  [1] FALSE  TRUE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE  TRUE FALSE
TRUE
## [13]  TRUE  TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE
FALSE
## [25] FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
FALSE
## [37] FALSE FALSE FALSE FALSE
sum(dm$age <= 40)
## [1] 10
#Greater than operator
dm$age > 40
##  [1]  TRUE FALSE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE FALSE  TRUE
FALSE
## [13] FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE
TRUE
## [25]  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
TRUE
## [37]  TRUE  TRUE  TRUE  TRUE
sum(dm$age > 40)
## [1] 30
dm$age >= 40
##  [1]  TRUE FALSE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE FALSE  TRUE
FALSE
## [13] FALSE FALSE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE
TRUE
## [25]  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
TRUE
## [37]  TRUE  TRUE  TRUE  TRUE
sum(dm$age >= 40)
## [1] 31
```

```
#Not equal to operator
dm$age != 34
##  [1]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
TRUE
## [13] FALSE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
TRUE
## [25]  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
TRUE
## [37]  TRUE  TRUE  TRUE  TRUE
sum(dm$age != 34)
## [1] 37
```

Logical Operators

We will now focus on logical operators. As we all know that the logical operators are “AND”, “OR” and “NOT”. If we want to know the number of male participants whose age is 34, we need to run the following code.

```
dm$age == 34 & dm$sex == 1 #Male participants whose age is 34
##  [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
FALSE
## [13] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
FALSE
## [25] FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
FALSE
## [37] FALSE FALSE FALSE FALSE
sum(dm$age == 34 & dm$sex == 1)#AND operator
## [1] 1
```

Participants whose age is either 34 or 38.

```
dm$age == 34 | dm$age == 38 #OR operator
##  [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE
FALSE FALSE
## [13]  TRUE  TRUE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE
FALSE FALSE
## [25] FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
FALSE FALSE
## [37] FALSE FALSE FALSE FALSE
sum(dm$age == 34 | dm$age == 38)
## [1] 5
```

Participants whose age is not 34.

```
sum(!dm$age == 34) #"!" is the NOT operator
## [1] 37
```

Participants whose age is not 34 and not males.

```
sum(!dm$age == 34 & !dm$sex == 1)
## [1] 20
```

Locating a Variable

If we want to locate a variable based on the condition, then the "which" function is used. The output provides the location of the variable in the R object based on the condition provided in the command.

```
which(dm$age == 34)
## [1] 13 16 27
which(dm$age > 34)
##  [1]  1  2  3  4  5  6  8  9 10 11 12 14 15 17 18 19 20 21 22 23 24 25 26
28 29
## [26] 30 31 32 33 34 35 36 37 38 39 40
```

Locating the Position of a Column

If we want to know the position of a column in a dataframe when the name of the column is known, we need to code as follows.

```
grep("hba1c",colnames(dm))
## [1] 9
```

Subsetting a Dataframe

As the name suggests, there are ways and functions which can help in creating a new dataframe by subsetting or using specific columns and rows from the existing dataframe.

For example, if we want to create a dataframe with only "Drug A" data, we need to run the following code.

```
dmA = dm[which(dm$group == "Drug A"),]
head(dmA)
##    group age sex weight ldl fasting_before fasting_after response hba1c
## 1 Drug A  46   1    101 104            155           111        1   6.4
## 2 Drug A  36   0     97 100            153           133        1   5.4
## 3 Drug A  53   0    119  81            130           100        1   6.3
## 4 Drug A  63   0     48  86            147           126        0   6.7
## 5 Drug A  58   1     88  75            130           101        1   6.4
## 6 Drug A  65   1     58 115            171           146        1   6.2
##   difference
## 1         44
## 2         20
## 3         30
## 4         21
## 5         29
## 6         25
```

Dataframe with only “Drug A” data who are females.

```
dmAF = dm[which(dm$group == "Drug A" & dm$sex == "0"),]
head(dmAF)
##     group age sex weight ldl fasting_before fasting_after response hba1c
## 2  Drug A  36   0     97 100            153           133        1   5.4
## 3  Drug A  53   0    119  81            130           100        1   6.3
## 4  Drug A  63   0     48  86            147           126        0   6.7
## 13 Drug A  34   0     89 209            276           225        1   5.6
## 16 Drug A  34   0     93 125            189           190        0   6.7
## 18 Drug A  44   0     47 174            233           167        0   6.6
##    difference
## 2          20
## 3          30
## 4          21
## 13         51
## 16         -1
## 18         66
```

Dataframe with only “Drug A” data without the “ldl” values.

```
dmA1 = dm[which(dm$group == "Drug A"),-c(5)]
head(dmA1)
##    group age sex weight fasting_before fasting_after response hba1c
difference
## 1 Drug A  46   1    101            155           111        1   6.4
44
## 2 Drug A  36   0     97            153           133        1   5.4
20
## 3 Drug A  53   0    119            130           100        1   6.3
30
## 4 Drug A  63   0     48            147           126        0   6.7
21
## 5 Drug A  58   1     88            130           101        1   6.4
29
## 6 Drug A  65   1     58            171           146        1   6.2
25
```

R has a dedicated function called “subset” for this purpose, as shown below.

```
newdm = subset(dm,group == "Drug A" & sex == "0",select = c(1:7))
```

Binding Functions

As the name suggests, the binding functions are used to bind the vectors or dataframes.

Let us examine the binding functions on dataframes.

We create two new dataframes by subsetting the “dm” dataframe based on groups.

```
dm_drugA = dm[which(dm$group == "Drug A"),]
dm_drugB = dm[which(dm$group == "Drug B"),]
dm1 = rbind(dm_drugA,dm_drugB)
dm2 = cbind(dm_drugA,dm_drugB)
```

We combine the two dataframes “dm_drugA” and “dm_drugB” by columns using *“cbind”*. Here, the dataframes are attached along the columns, finally creating 20 columns. We use *“rbind”* to combine the dataframe along the rows. The function *“rbind”* can be used if both the dataframes have the same column names.

Creation of Cross Tabs (n*m) Using Table Command

```
tab1 = table(dm$group,dm$sex)
tab1
##
##           0  1
##   Drug A  8 12
##   Drug B 14  6
tab2 = table(dm$group,
             factor(dm$sex,labels = c("Female","Male")))
tab2
##
##          Female Male
##   Drug A      8   12
##   Drug B     14    6
```

“tab2” is the same as “tab1” except that the sex of the participants appears as entered in “tab1”, whereas the sex is labeled in “tab2” as shown in the output.

Summary Functions

Let us summarize the entire dataframe using the `summary` function

```
summary(dm)
##     group         age         sex        weight            ldl
##  Drug A:20   Min.   :33.00   0:22   Min.   : 39.00   Min.   : 75.0
##  Drug B:20   1st Qu.:40.75   1:18   1st Qu.: 60.25   1st Qu.:103.0
##              Median :46.50          Median : 72.00   Median :123.5
##              Mean   :47.40          Mean   : 74.12   Mean   :126.1
##              3rd Qu.:53.75          3rd Qu.: 88.25   3rd Qu.:146.0
##              Max.   :66.00          Max.   :119.00   Max.   :209.0
##  fasting_before  fasting_after   response     hba1c         difference
##  Min.   :130.0   Min.   : 89.0   0:15     Min.   :5.400   Min.   :-5.00
##  1st Qu.:155.0   1st Qu.:125.8   1:25     1st Qu.:6.100   1st Qu.:20.75
##  Median :185.5   Median :142.5            Median :6.400   Median :30.50
##  Mean   :184.7   Mean   :147.4            Mean   :6.397   Mean   :37.25
##  3rd Qu.:203.8   3rd Qu.:170.8            3rd Qu.:6.700   3rd Qu.:53.50
##  Max.   :276.0   Max.   :225.0            Max.   :7.900   Max.   :96.00
```

If we want to summarize a single column, we need to specify the column as shown below

```
summary(dm$group)
## Drug A Drug B
##     20     20
summary(dm$age)
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   33.00   40.75   46.50   47.40   53.75   66.00
```

The summary function yields a count for categorical variables. For continuous variables, it provides minimum, first quantile, median, third quantile, and maximum values along with the mean value.

There are also specific functions for various summarizing tasks as follows.

```
#Mean calculation
mean(dm$age)
## [1] 47.4
#Standard deviation calculation
sd(dm$age)
## [1] 9.445661
round(sd(dm$age),2)
## [1] 9.45
#Standard error calculation
se = sd(dm$age)/sqrt(length(dm$age))
se
## [1] 1.49349
```

Note the use of “round” and “length” functions. The “round” function limits the number of digits after the decimal, and the “length” function calculates the number of elements in the vector.

Many times during the analysis, we will have to calculate the mean and SD by groups rather than just the mean and SD of all participants together. We can achieve this either by using the “by” function or by indexing the group of interest.

```
by(data = dm$age,INDICES = dm$group,mean)
## dm$group: Drug A
## [1] 45.15
## ------------------------------------------------------------
## dm$group: Drug B
## [1] 49.65
by(dm$age,dm$group,sd)
## dm$group: Drug A
## [1] 10.17362
## ------------------------------------------------------------
## dm$group: Drug B
## [1] 8.305198
sd(dm$age[dm$group == "Drug A"])
## [1] 10.17362
sd(dm$age[dm$group == "Drug B"])
## [1] 8.305198
```

Similarly, we can code for the median, quantile, and interquartile range.

```
#Median, quantile and interquartile range
median(dm$age)
## [1] 46.5
quantile(dm$age)
##    0%   25%   50%   75%  100%
## 33.00 40.75 46.50 53.75 66.00
quantile(dm$age,0.25)
##   25%
## 40.75
quantile(dm$age,0.75)
##   75%
## 53.75
IQR(dm$age)
## [1] 13
#Median, quantile and interquartile range by group
by(dm$age,dm$group,median)
## dm$group: Drug A
## [1] 43
## ------------------------------------------------------------
## dm$group: Drug B
## [1] 49.5
by(dm$age,dm$group,quantile)
## dm$group: Drug A
##   0%  25%  50%  75% 100%
## 33.0 37.5 43.0 51.5 65.0
## ------------------------------------------------------------
## dm$group: Drug B
##    0%   25%   50%   75%  100%
## 34.00 43.75 49.50 56.00 66.00
by(dm$age,dm$group,IQR)
## dm$group: Drug A
## [1] 14
## ------------------------------------------------------------
## dm$group: Drug B
## [1] 12.25
quantile(dm$age[dm$group == "Drug A"],c(0.25,0.75))
##  25%  75%
## 37.5 51.5
quantile(dm$age[dm$group == "Drug B"],c(0.25,0.75))
##   25%   75%
## 43.75 56.00
```

Dealing with "NA" in the Dataset

The current dataset with which we are working is complete with data. There are no missing data or empty cells. This is not the case with actual practice, where there will be a loss to follow-up or lack of data collection at some points, i.e., the dataset will be punched in randomly with missing data. The presence of missing data will disturb many of the functions unless we specify the function to ignore the missing data. This is demonstrated below on a dataframe with missing data.

```
#Dealing with "NA" data in the dataframe
dm_m = read.csv("dm_missing.csv")
dm_m[,] = apply(dm_m[,],2,as.numeric)
dm_m$group = as.factor(dm_m$group)
dm_m$sex = as.factor(dm_m$sex)
dm_m$response = as.factor(dm_m$response)
str(dm_m) #The dataframe is ready for analysis
## 'data.frame':    40 obs. of  10 variables:
##  $ group         : Factor w/ 2 levels "1","2": 1 1 1 1 1 1 1 1 1 1 ...
##  $ age           : num  46 36 53 63 62 65 NA 46 61 38 ...
##  $ sex           : Factor w/ 2 levels "0","1": 2 1 1 1 2 2 2 2 2 2 ...
##  $ weight        : num  101 97 119 NA 88 58 40 71 41 62 ...
##  $ ldl           : num  104 100 81 86 75 115 146 NA 138 141 ...
##  $ fasting_before: num  155 153 130 147 130 171 197 201 194 209 ...
##  $ fasting_after : num  125 163 100 126 101 146 97 125 192 113 ...
##  $ difference    : num  30 -9 30 21 29 25 100 76 2 96 ...
##  $ response      : Factor w/ 2 levels "0","1": 2 2 2 1 2 2 2 2 2 2 ...
##  $ hbalc         : num  6.4 5.4 6.3 6.7 6.4 6.2 6.2 6.4 6.4 6.2 ...
#mean
mean(dm_m$age) #Note the NA as response
## [1] NA
mean(dm_m$age, na.rm = T)
## [1] 47.94595
#sum
sum(dm_m$age)
## [1] NA
sum(dm_m$age,na.rm = T)
## [1] 1774
```

As seen from the results, an argument called "na.rm" has to be set to "T" or "TRUE" so that the function will ignore the NA terms.

Functions in R

A function is a code that incorporates a set of instructions to be executed by the tool to perform a specific task.

A function has four parts:

1. Name of the function
2. Arguments to the function
3. Body of the function
4. Returning result

R has inbuilt functions like “sum”, “mean”, and others. Functions can also be defined by users.

Let us try building a custom function where the function should provide both mean and sd from a vector.

```
#Function to calculate mean and SD together
mean_sd = function(x){
  mean = mean(x,na.rm = T)
  sd = sd(x,na.rm = T)
  a = data.frame(mean,sd)
  return(a)
}

#Demonstration of the function "mean_sd"
mean_sd(dm$age)
##   mean       sd
## 1 47.4 9.445661
mean_sd(dm_m$age)
##       mean       sd
## 1 47.94595 9.603084
by(dm$age,dm$group,mean_sd)
## dm$group: Drug A
##    mean       sd
## 1 45.15 10.17362
##------------------------------------------------------------
## dm$group: Drug B
##    mean       sd
## 1 49.65 8.305198
by(dm_m$age,dm$group,mean_sd)
## dm$group: Drug A
##   mean       sd
## 1   46 10.33871
##------------------------------------------------------------
## dm$group: Drug B
##   mean       sd
## 1   50 8.568067
```

Efficient Utilization of Data Science Packages

If the task is repetitive, sometimes, instead of coding separately to perform each task, we can automate the repetitive task using a single line of code.

We may have to use certain packages for the same as provided in the code [1].

```
library(tidyverse)
## ── Attaching core tidyverse packages ──────────────────────────
tidyverse 2.0.0 ──
## ✓ dplyr     1.1.3     ✓ readr     2.1.4
## ✓ forcats   1.0.0     ✓ stringr   1.5.0
## ✓ ggplot2   3.4.3     ✓ tibble    3.2.1
## ✓ lubridate 1.9.2     ✓ tidyr     1.3.0
## ✓ purrr     1.0.1
## ── Conflicts ────────────────────────────────────────────────
tidyverse_conflicts() ──
## ✓ dplyr::filter() masks stats::filter()
## ✓ dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all
conflicts to become errors
dm_tab = dm %>% select_if(is.factor) %>%
  map(function(x)(table(x,dm$group)))
dm_tab
## $group
##
## x         Drug A Drug B
##   Drug A     20      0
##   Drug B      0     20
##
## $sex
##
## x   Drug A Drug B
##   0      8     14
##   1     12      6
##
## $response
##
## x   Drug A Drug B
##   0      5     10
##   1     15     10
```

From the result, it can be observed that the command has picked all the factorial variables from the dataframe and constructed a table by treatment groups.

Handling Missing Data: Imputation

In a clinical trial, some of the participants may be lost to follow-up due to adverse effects, inefficacy, immigration, administration of rescue medication or comparator therapy, or any other unknown reason. In those cases, to protect the original

randomization and prevent bias from participant exclusion, we should perform an intention-to-treat analysis. In those cases, we need to impute the data for missing values. Data imputations are methods to retain the majority of information in a dataset by substituting missing data with different values based on other available information. Imputations could be single or multiple. Multiple imputations (imputations are performed many times) are the preferred method of dealing with missing data. Let us learn to impute data in R. There are a number of packages for imputation. Let us learn to do the same using *mice* package [2].

Step 1: Loading the Dataset

We will use the dataset "dm_missing.csv" to understand the handling of missing data. This dataset is similar to "dm.csv", where some data from the "dm.csv" have been omitted.

```
data_miss<- read.csv("dm_missing.csv")
#data_miss
summary (data_miss)
##      group          age             sex             weight
##  Min.   :1.0   Min.   :34.00   Min.   :0.0000   Min.   : 39.00
##  1st Qu.:1.0   1st Qu.:41.00   1st Qu.:0.0000   1st Qu.: 61.00
##  Median :1.5   Median :47.00   Median :0.0000   Median : 72.00
##  Mean   :1.5   Mean   :47.95   Mean   :0.4474   Mean   : 74.62
##  3rd Qu.:2.0   3rd Qu.:56.00   3rd Qu.:1.0000   3rd Qu.: 89.00
##  Max.   :2.0   Max.   :66.00   Max.   :1.0000   Max.   :119.00
##                NA's   :3       NA's   :2        NA's   :3
##       ldl         fasting_before  fasting_after     difference
##  Min.   : 75.0   Min.   :130.0   Min.   : 89.0   Min.   : -9.00
##  1st Qu.:101.0   1st Qu.:155.0   1st Qu.:118.2   1st Qu.: 21.25
##  Median :121.5   Median :182.0   Median :132.5   Median : 30.00
##  Mean   :125.4   Mean   :183.2   Mean   :139.3   Mean   : 44.82
##  3rd Qu.:145.5   3rd Qu.:200.8   3rd Qu.:161.5   3rd Qu.: 79.00
##  Max.   :209.0   Max.   :276.0   Max.   :192.0   Max.   :126.00
##  NA's   :2       NA's   :2       NA's   :2       NA's   :2
##     response         hba1c
##  Min.   :0.000   Min.   :5.400
##  1st Qu.:0.000   1st Qu.:6.100
##  Median :1.000   Median :6.400
##  Mean   :0.625   Mean   :6.397
##  3rd Qu.:1.000   3rd Qu.:6.700
##  Max.   :1.000   Max.   :7.900
##
```

Step 2: Visualizing Missing Frequency in Data

To visualize the frequency of missing values, we need to install *VIM* package [3].

```
library(VIM)
## Warning: package 'VIM' was built under R version 4.2.3
## Loading required package: colorspace
## Warning: package 'colorspace' was built under R version 4.2.3
## Loading required package: grid
## The legacy packages maptools, rgdal, and rgeos, underpinning the sp
package,
## which was just loaded, will retire in October 2023.
## Please refer to R-spatial evolution reports for details, especially
## https://r-spatial.org/r/2023/05/15/evolution4.html.
## It may be desirable to make the sf package available;
## package maintainers should consider adding sf to Suggests:.
## The sp package is now running under evolution status 2
##      (status 2 uses the sf package in place of rgdal)
## VIM is ready to use.
## Suggestions and bug-reports can be submitted at:
https://github.com/statistikat/VIM/issues
##
## Attaching package: 'VIM'
## The following object is masked from 'package:datasets':
##
##     sleep
## Visualizing in numbers
aggr(data_miss, prop = F, numbers = T, sortVars = T,
     labels = names(data_miss), cex.axis = 0.7,
     gap = 3, ylab = c("Missing data", "Pattern"))
```

We can visualize the missing data in the "dm_missing.csv" file with the above function "*aggr*". Aggregation plots help us visualize the variables where data is missing and the number of data points missing through the bar graph and grid pattern (Fig. 4.5). In the aggregate plot on the right-hand side, the grid presents all combinations of missing (red) and observed (blue) values in the data. Vertical bars to the right on the plot on the right-hand side show the frequency of the missing value. We have complete data for 26 participants (as shown in the lowermost grid). There is no missing data in the variables group, hba1c, and response (all blues). Data for 2 participants is missing for each of the following variables: sex, ldl, fasting_before, fasting_after, and difference; data for 3 participants is missing for variables age and weight. The missing data for fasting_after and difference is missing for the same two participants (depicted by the same row in the grid). The bar plot on the left-hand side simply shows the number of missing data points for each variable.

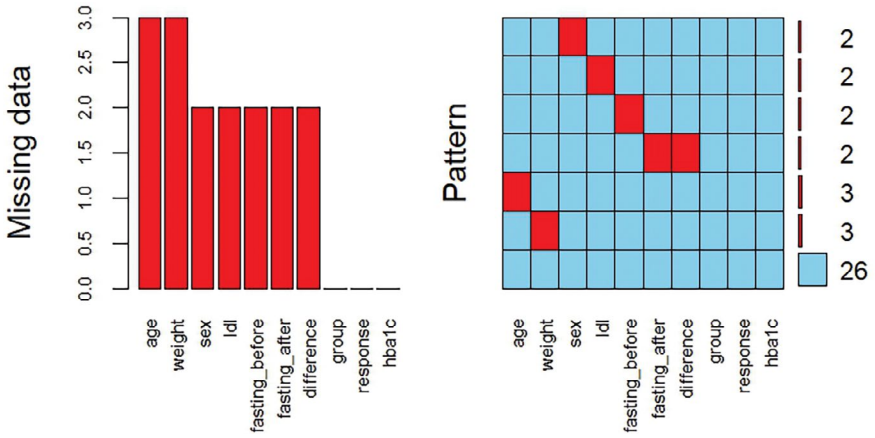

Fig. 4.5 The bar plot (left) and aggregation plot (right) are used to visualize the variables where data are missing and the number of missing data points per variable

```
##
##  Variables sorted by number of missings:
##        Variable Count
##             age     3
##          weight     3
##             sex     2
##             ldl     2
##  fasting_before     2
##   fasting_after     2
##      difference     2
##           group     0
##        response     0
##           hba1c     0
## Visualizing in proportions
aggr(data_miss, prop = T, numbers = T)
```

In combination plot (Fig. 4.6), the vertical bars to the right of the grid show the proportions of missing values in each variable. Complete data was present for 65% of participants, and data was missing for one or more variables for 35%. Age and weight had the highest proportion of missing values, i.e., 0.075, and all the other variables like sex, ldl, fasting_before, fasting_after, and difference variables had 0.050 missing proportions each. The bar plot on the left-hand side simply shows the proportion of missing data points for each variable.

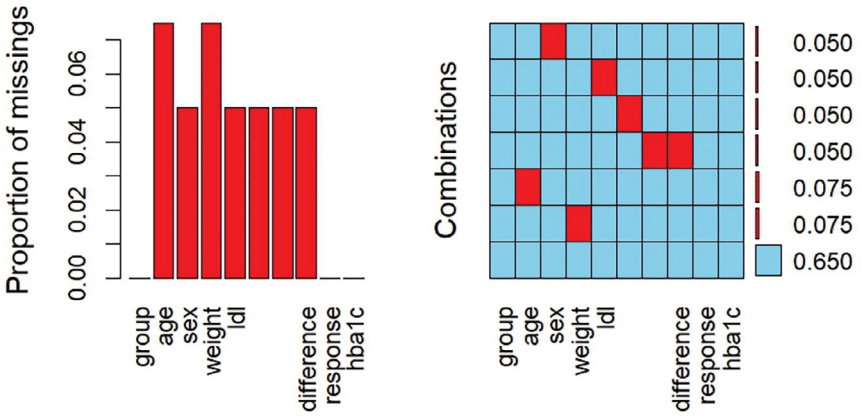

Fig. 4.6 The bar plot and the combination plot show the proportion of missing values for variables

Step 3: Imputing the Missing Values

We will need to install the *mice* package using *"install.packages()"* and then proceed for imputations [2].

```
## Loading the package
library (mice)
## Warning: package 'mice' was built under R version 4.2.3
##
## Attaching package: 'mice'
## The following object is masked from 'package:stats':
##
##     filter
## The following objects are masked from 'package:base':
##
##     cbind, rbind
## Performing imputations
imputed_dm <- mice(data_miss, m = 5, maxit = 5,
                   method = 'pmm', seed = 123)
##
##  iter imp variable
##   1   1  age  sex  weight  ldl  fasting_before  fasting_after  difference
##   1   2  age  sex  weight  ldl  fasting_before  fasting_after  difference
##   1   3  age  sex  weight  ldl  fasting_before  fasting_after  difference
##   1   4  age  sex  weight  ldl  fasting_before  fasting_after  difference
##   1   5  age  sex  weight  ldl  fasting_before  fasting_after  difference
##   2   1  age  sex  weight  ldl  fasting_before  fasting_after  difference
##   2   2  age  sex  weight  ldl  fasting_before  fasting_after  difference
```

```
##   2   3  age  sex  weight  ldl  fasting_before  fasting_after  difference
##   2   4  age  sex  weight  ldl  fasting_before  fasting_after  difference
##   2   5  age  sex  weight  ldl  fasting_before  fasting_after  difference
##   3   1  age  sex  weight  ldl  fasting_before  fasting_after  difference
##   3   2  age  sex  weight  ldl  fasting_before  fasting_after  difference
##   3   3  age  sex  weight  ldl  fasting_before  fasting_after  difference
##   3   4  age  sex  weight  ldl  fasting_before  fasting_after  difference
##   3   5  age  sex  weight  ldl  fasting_before  fasting_after  difference
##   4   1  age  sex  weight  ldl  fasting_before  fasting_after  difference
##   4   2  age  sex  weight  ldl  fasting_before  fasting_after  difference
##   4   3  age  sex  weight  ldl  fasting_before  fasting_after  difference
##   4   4  age  sex  weight  ldl  fasting_before  fasting_after  difference
##   4   5  age  sex  weight  ldl  fasting_before  fasting_after  difference
##   5   1  age  sex  weight  ldl  fasting_before  fasting_after  difference
##   5   2  age  sex  weight  ldl  fasting_before  fasting_after  difference
##   5   3  age  sex  weight  ldl  fasting_before  fasting_after  difference
##   5   4  age  sex  weight  ldl  fasting_before  fasting_after  difference
##   5   5  age  sex  weight  ldl  fasting_before  fasting_after  difference
## Warning: Number of logged events: 9
summary(imputed_dm)
## Class: mids
## Number of multiple imputations:  5
## Imputation methods:
##          group            age            sex         weight            ldl
##             ""          "pmm"          "pmm"          "pmm"          "pmm"
## fasting_before  fasting_after     difference       response          hba1c
##          "pmm"          "pmm"          "pmm"             ""             ""
## PredictorMatrix:
##                group age sex weight ldl fasting_before fasting_after
difference
## group              0   1   1      1   1              1             1
1
## age                1   0   1      1   1              1             1
1
## sex                1   1   0      1   1              1             1
1
## weight             1   1   1      0   1              1             1
1
## ldl                1   1   1      1   0              1             1
1
## fasting_before     1   1   1      1   1              0             1
1
##                response hba1c
## group                 1     1
## age                   1     1
## sex                   1     1
## weight                1     1
## ldl                   1     1
## fasting_before        1     1
## Number of logged events:  9
##   it im    dep meth        out
## 1  3  4    age  pmm difference
## 2  3  5    age  pmm difference
## 3  3  5    sex  pmm difference
## 4  3  5 weight  pmm difference
## 5  3  5    ldl  pmm difference
## 6  4  2    age  pmm difference
```

1. The function *"mice"* is used to perform multiple imputations. The arguments are the data with missing values, number of imputations (m), number of iterations (maxit, method of imputations).
2. Any of the methods *pmm* (predictive mean matching), *logreg* (logistic regression) etc. can be used for generating the imputed values.

Step 4: Getting Complete Data

```
completeData <- complete(imputed_dm)
```

Step 5: Write Imputed Data to a .csv *file*

```
write.csv(completeData, "imputed_data.csv")
```

References

1. Wickham H, Averick M, Bryan J, Chang W, McGowan L, François R, Grolemund G, Hayes A, Henry L, Hester J, Kuhn M, Pedersen T, Miller E, Bache S, Müller K, Ooms J, Robinson D, Seidel D, Spinu V, et al. Welcome to the tidyverse. J Open Source Softw. 2019;4(43):1686.
2. Van Buuren S, Groothuis-Oudshoorn K. mice: multivariate imputation by chained equations in R. J Stat Softw. 2011;45(3):1–6.
3. Kowarik A, Templ M. Imputation with the R package VIM. J Stat Softw. 2016;74(7):1–16.

Chapter 5
Introduction to Packages in R: Installation, Loading, Unloading and Deletion

Praveen Kumar-M

We now have an understanding of the way functions work in R, and we have also observed the way new functions can be generated in R. There may be a need to write many custom functions while undertaking a project. However, building custom functions, especially for complicated tasks, will be exhausting and require expertise. To overcome this difficulty, R has a system to build a collection of R functions, data, and compiled code in a well-defined format. These are called "package(s)" and can be developed by anyone who is an expert in their corresponding fields. These packages can be developed as open-source resources that anyone can use by simply importing them into the working environment. This will spare the individual from writing custom complicated functions for specific tasks. Some preloaded sets of packages come with the standard installation of R. These preloaded sets of packages are referred to as the base packages. Other than the base packages, we can download and install additional packages required for our task in R.

The versatility of R comes from the availability of a wide range of packages created for a wide variety of usages. Installation of additional packages will help save our time and assist in completing our work faster and swifter. You will be surprised to find a domain for which a package has not been developed. This is not an exaggeration but rather a truth. For example, do you know that we can prepare a PowerPoint presentation in R? This is possible using a package known as `ReporteRs` [1].

There are industry-standard visualization packages available in R, such as `ggplot2` [2]. It is possible to make publication-ready tables using a package known as `gtsummary` [3]. We are now quoting some domain-specific packages to give an idea of the different types of packages available in R. We are mentioning the

P. Kumar-M (✉)
Clinical Sciences, Nference, Bengaluru, Karnataka, India

A. Srinivasan et al. (eds.), *R for Basic Biostatistics in Medical Research*,
https://doi.org/10.1007/978-981-97-6980-3_5

utility along with the package name in brackets. The packages are: conduct wide variety of bioinformatics analysis (using `BiocManager` [4]); pharmacokinetic simulation (`mrgsolve` [5]); create machine learning models (`caret` [6]); conduct image processing for information extraction (`magick` [7]); stock analysis (`quantmod` [8]); data restructuring and formatting (`reshape2`, `dplyr`, `plyr` [9–11]); creation of web application (`shiny` [12]); creation of nomogram (`rms` [13]); computation of sample sizes in clinical trial (`TrialSize` [14]); propensity score matching for observational studies (`MatchIt` [15]), importing of data from other statistical programs(`haven` [16]); and the list goes endless. The current number of packages hosted on the official repository can be observed from https://cran.r--project.org/web/packages/.

The chapter contains four package exercise components, namely, installing a package, loading a package, detaching a package, and removing a package. The terms installation and removal are self-explanatory. But what do you mean by the term loading and detaching? After we have installed an R package, we will be able to use the package only after loading (technically, loading involves inputting the package's codes and other materials into the workspace). This requirement of loading the package each time is designed to save RAM and processing memory. Once the work is over, we can detach the package that will save RAM. If you close Rstudio, then the packages will also get detached automatically.

We will be discussing both the code-based and GUI-based methods for performing each of the exercises. The advantage of learning both is to address the varied needs of the readers. By code-based, it will be possible to include all these processes seamlessly into the workflow. The GUI generates the codes on our behalf and performs the function. The advantage of GUI comes if we wish to look up all the available packages before installation and are not entirely sure about the package name.

Installing a Package: Code-Based and GUI

Code-Based

```
install.packages("ggplot2")
```

1. *"install.packages"* function is used for installation of the package. The package which is to be installed is mentioned as a character string to the argument *pkgs*.

GUI

1. We can use GUI to install the package.
2. Select the "Packages" option in the right lower *window (Files, Plot, Packages, Help, Viewer)* of the Rstudio. Then select "Install" (Fig. 5.1). A pop-up window will open (Fig. 5.2).
3. Make sure that Repository(CRAN, CRANextra) is selected in the "Install" from the drop-down menu option at the top (Fig. 5.2).
4. Mention the package name in the "Packages(separate multiple with space or comma)" option in the middle tab (Fig. 5.2).
5. The location for installing the package is to be mentioned in the Install to Library category. The recommendation is to leave the location to default (Fig. 5.2).
6. Check the "Install dependencies" option at last (Fig. 5.2).
7. Press the "Install" button at the end (Fig. 5.2).

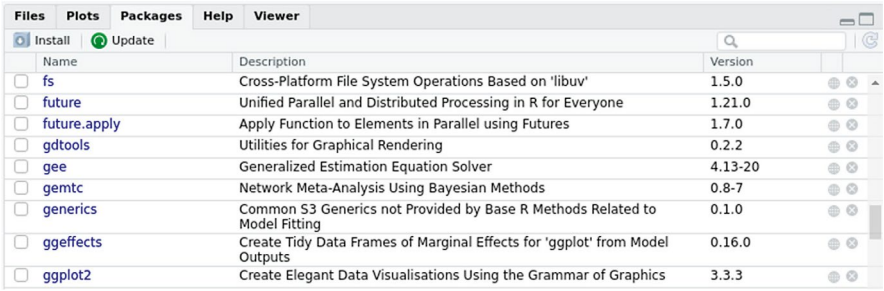

Fig. 5.1 "Packages" in Rstudio

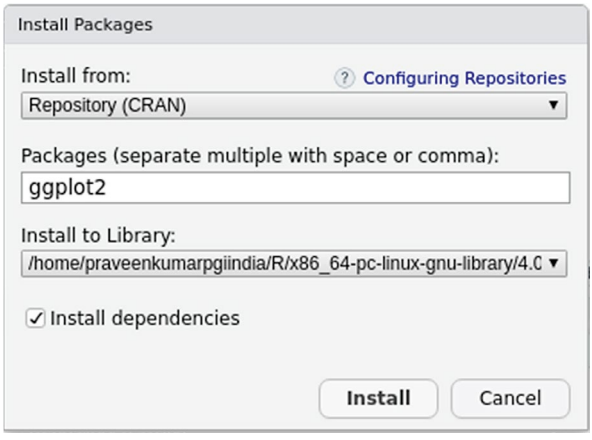

Fig. 5.2 Installation of the required package (in this case, ggplot2)

Messages During Installation

1. The location in the computer where the package is installed (directory—Fig. 5.3).
2. The URL R is trying to contact for downloading the package (Fig. 5.3).
3. The content—is the communicated URL returning (Fig. 5.3).
4. The size of the content which is to be downloaded (Fig. 5.3).
5. The size of the downloaded content after it has been downloaded to your computer (Fig. 5.3).
6. Has the package successfully unpacked the content from the downloaded zip file (Fig. 5.3)?
7. MD5 Sum Checked (R is checking whether the package is correctly downloaded (in a bit technical way)) (Fig. 5.3).
8. The location in which the downloaded package is located in the computer (note that this is different from the installed location) (Fig. 5.3).

- No need to routinely look into all these details. Just search for one single word in the installation message—"successfully". If it is present, then the installation has been successfully undertaken (Fig. 5.3).

```
> install.packages("ggplot2")
Installing package into '/home/praveenkumarpgiindia/R/x86_64-pc-linux-gnu-library/4.0'
(as 'lib' is unspecified)
trying URL 'https://cloud.r-project.org/src/contrib/ggplot2_3.3.3.tar.gz'
Content type 'application/x-gzip' length 3058840 bytes (2.9 MB)
==================================================
downloaded 2.9 MB

* installing *source* package 'ggplot2' ...
** package 'ggplot2' successfully unpacked and MD5 sums checked
** using staged installation
** R
** data
*** moving datasets to lazyload DB
** inst
** byte-compile and prepare package for lazy loading
** help
*** installing help indices
*** copying figures
** building package indices
** installing vignettes
** testing if installed package can be loaded from temporary location
** testing if installed package can be loaded from final location
** testing if installed package keeps a record of temporary installation path
* DONE (ggplot2)

The downloaded source packages are in
        '/tmp/RtmpTfBSJJ/downloaded_packages'
>
```

Fig. 5.3 Confirmation message following installation

What Are Dependencies?

- When you install a package, the functions in the package might require additional functions from other packages for its performance.
- These additional packages are individually a "package" by themselves. Still, the term "dependencies" is used as they are not the packages that we directly intend to install but are required for the functioning of the package that we want to install.
- The dependencies are automatically communicated and installed.

Loading a Package: Code-Based and GUI

Code-Based

```
library(package = ggplot2)
require(ggplot2)
library("ggplot2") #Both ways will work. But using quotes is better and more
apt.
```

1. *The "library"* or *"require"* function is used for loading a library. The package which is to be loaded is mentioned in the *package* argument of the function as a character string.

GUI

- To load a library using GUI, first locate the library in the "Packages" window on the right-hand side of Rstudio (Fig. 5.4).
- Select the checkbox against the respective library which is to be attached (Fig. 5.5).

  The package will be attached ultimately.

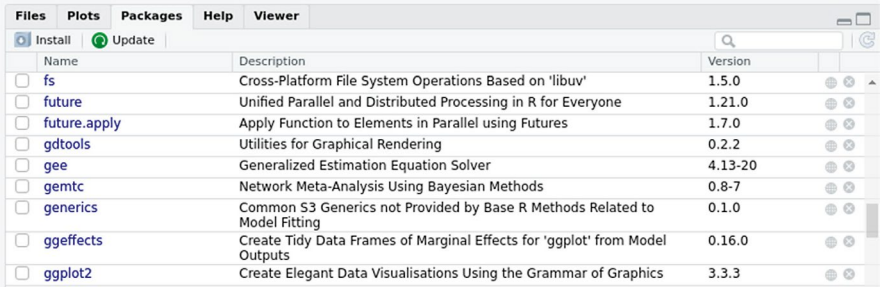

Fig. 5.4 "Packages" in Rstudio

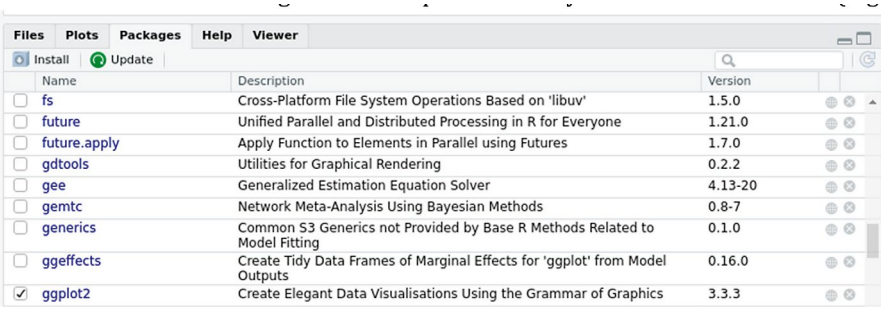

Fig. 5.5 Selection of required package. Once selected, the required package will be in the selected state (loaded currently in the environment)

Detaching a Package: Code-Based and GUI

You can detach a package after the completion of its utilization. But this is not routinely done, as closing an 'R' session will automatically detach all the loaded packages.

Code-Based

```
#Detaching a package.
detach(name = package:ggplot2) #Please note to use the term package.
```

1. The "`detach`" function is used for unloading a library. The package to be detached is mentioned in the `name` argument of the function.
2. As this function is not exclusively for detaching a package, "`package:`" should be mentioned to avoid conflict with other R objects.

GUI

- To detach a package using GUI, locate the package in the "Packages" window on the right-hand side of Rstudio. In our example case, it is "`ggplot2`" which is currently checked (Fig. 5.5).
- Uncheck the box against the respective library, which is to be detached (Fig. 5.6).
- The package will finally get detached.

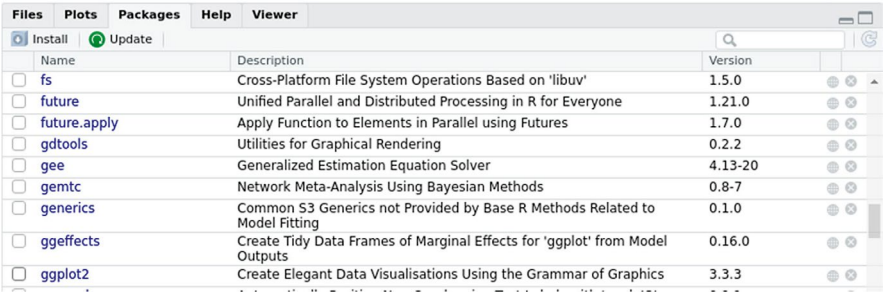

Fig. 5.6 Deselection of the required package

Removing a Package: Code-Based and GUI

Code-Based

```
remove.packages(pkgs = "ggplot2")
```

1. The "*remove.package*" function is used for removing a package. The package which is to be removed is mentioned in the "pkgs" argument of the function as a character string.

GUI

- To remove a library using GUI, locate the library in the "Packages" window on the right-hand side of Rstudio. Our example case is "ggplot2," which is currently checked (Fig. 5.5).
- Select the cross which is at the end of the respective library (Fig. 5.7).
- The package will be removed.

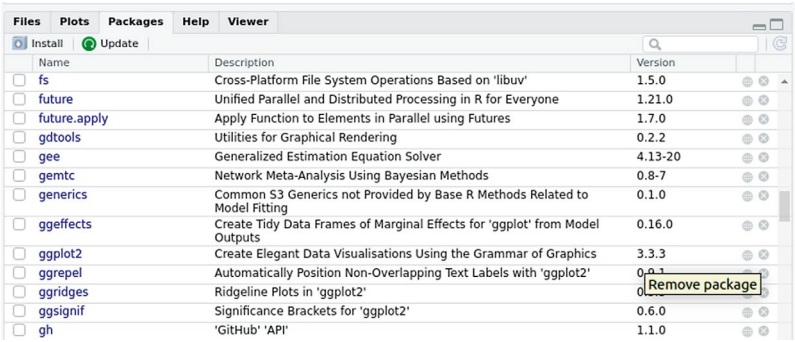

Fig. 5.7 Uninstallation of package

Citing a Package

R is an open-source tool. If we are using R for our works, we must cite R. The way to get the citation of R is.

```
citation()
##
## To cite R in publications use:
##
##   R Core Team (2021). R: A language and environment for statistical
##   computing. R Foundation for Statistical Computing, Vienna, Austria.
##   URL https://www.R-project.org/.
##
## A BibTeX entry for LaTeX users is
##
##   @Manual{,
##     title = {R: A Language and Environment for Statistical Computing},
##     author = {{R Core Team}},
##     organization = {R Foundation for Statistical Computing},
##     address = {Vienna, Austria},
##     year = {2021},
##     url = {https://www.R-project.org/},
##   }
##
## We have invested a lot of time and effort in creating R, please cite it
## when using it for data analysis. See also 'citation("pkgname")' for
## citing R packages.
```

The code gives both the citation and the BibTex entry. Like R, the packages we have read in this chapter are the individual's or team's hard work and intellect. They are giving us for free. Hence, according to the usage agreement's license, we should be citing the package if we are using it. The way to get the citation of the package in R is.

```
citation(package = "ggplot2")
##
## To cite ggplot2 in publications, please use:
##
##   H. Wickham. ggplot2: Elegant Graphics for Data Analysis.
##   Springer-Verlag New York, 2016.
##
## A BibTeX entry for LaTeX users is
##
##   @Book{,
##     author = {Hadley Wickham},
##     title = {ggplot2: Elegant Graphics for Data Analysis},
##     publisher = {Springer-Verlag New York},
##     year = {2016},
##     isbn = {978-3-319-24277-4},
##     url = {https://ggplot2.tidyverse.org},
##   }
```

We should be citing all the packages we are using, including the package we used for cleaning the data and for dataframe manipulation. By employing citing practices, we are letting others know about the package's relative importance. If a package is getting robustly cited, that reflects that the package is beneficial. Sometimes, this count might not be true for a domain-specific package, where the user base itself is narrow.

Installation of More Than One Package

The readers will often require more than one package for their projects. In that case, we can install all the required packages simultaneously. The code for installation remains the same. But we need to mention all the required packages as a character vector. Here, we are trying to install three required packages for data processing in a go.

```
install.packages(c("reshape2", "dplyr", "plyr"))
```

A similar approach can also be undertaken for the removal of packages.

Additional Knowledge

- What is FTP?
  The File Transfer Protocol (FTP) is the standard network protocol used for the transfer of computer files between a client and server on a computer network.

- A list of all the CRAN packages available can be obtained in the following link.
- https://cran.r-project.org/web/packages/available_packages_by_name.html
- What is CRAN??
- The "Comprehensive R Archive Network" (CRAN) is a collection of sites (URL and FTP) that carry identical material and up-to-date versions of codes & documentation for R.
- CRAN Mirror??

  CRAN is mirrored in many servers to tackle the workload of catering to a large number of users better. These servers are often located in universities. You can either choose the global server or select a server that is located close to you (For India, there is a server at IIT-Madras).

Getting Information About a Package

1. Let's take an example package to understand this topic. The package "`PKPDmodels: Pharmacokinetic/pharmacodynamic models`" is taken for this exercise.
2. Search the package in the CRAN link mentioned above.
3. The link for the package is https://cran.r-project.org/web/packages/PKPDmodels/index.html . Using the link, we are redirected to the web page containing information about the specific package of interest.
4. On reading the initial lines, we understood that the package provides functions to evaluate common pharmacokinetic/pharmacodynamic models and their gradients.
5. Two important pieces of information to note on this page are

   (a) Reference manual
   (b) Windows binaries (for Code-based installation)

6. The reference manual will be a PDF file containing extensive information about the package of interest.
7. Windows binary of the package will help in the manual installation of the package. The procedures for manual installation of packages include.

   (a) Select the `Packages` option in the "Files, Plot, Packages, Help, Viewer" window on the right-hand side of the Rstudio. Then select "`Install`" option within the same. A pop-up window will open.
   (b) Select the Package "Archive File(.zip,.tar.gz)" option in the "`Install`" `from` the drop-down menu option at the top.
   (c) Select the "`Browse`" option in the `Package archive` category in the middle table A file picker pop-up will appear. Select the binary windows file that you have downloaded.
   (d) The location for installing the package is to be mentioned in the "`Install to Library`" category. The recommendation is to leave the location to `default`.

(e) Check the "`Install to dependencies`" option.
(f) Press the "`Install`" button at the bottom of the window.
(g) This knowledge of manual installation will come in handy during the installation of your custom package containing the list of self-made functions.

References

1. Gohel D. ReporteRs: microsoft word and powerpoint documents generation. R package version 0.8.11.001. 2020. https://davidgohel.github.io/ReporteRs
2. Wickham H. ggplot2: elegant graphics for data analysis. New York, NY: Springer; 2016.
3. Sjoberg DD, Curry M, Hannum M, Whiting K, Zabor EC. gtsummary: presentation-ready data summary and analytic result tables. R package version 1.3.5. 2020. https://CRAN.R-project.org/package=gtsummary.
4. Morgan M. BiocManager: access the bioconductor project package repository. R package version 1.30.10. 2019. https://CRAN.R-project.org/package=BiocManager.
5. Baron KT. mrgsolve: simulate from ODE-based models. R package version 0.10.7. 2020. https://CRAN.R-project.org/package=mrgsolve.
6. Kuhn M. caret: classification and regression training. R package version 6.0–86. 2020. https://CRAN.R-project.org/package=caret
7. Ooms J. magick: advanced graphics and image-processing in R. R package version 2.6.0. 2021. https://CRAN.R-project.org/package=magick.
8. Ryan JA, Ulrich JM. quantmod: quantitative financial modelling framework. R package version 0.4.18. 2020. https://CRAN.R-project.org/package=quantmod.
9. Wickham H. Reshaping Data with the reshape package. J Stat Softw. 2007;21(12):1–20. http://www.jstatsoft.org/v21/i12/
10. Wickham H, François R, Henry L, Müller K. dplyr: a grammar of data manipulation. R package version 1.0.3. 2021. https://CRAN.R-project.org/package=dplyr.
11. Wickham H. The split-apply-combine strategy for data analysis. J Stat Softw. 2011;40(1):1–29. http://www.jstatsoft.org/v40/i01/
12. Chang W, Cheng J, Allaire JJ, Sievert C, Schloerke B, Xie Y, Allen J, McPherson J, Dipert A, Borges B. shiny: web application framework for R. R package version 1.6.0. 2021. https://CRAN.R-project.org/package=shiny.
13. Harrell FE Jr. rms: regression modeling strategies. R package version 6.0–1. 2020. https://CRAN.R-project.org/package=rms
14. Zhang E, Wu VQ, Chow SC, Zhang HG. TrialSize: R functions for chapter 3,4,6,7,9,10,11,12,14,15 of sample size calculation in clinical research. R package version 1.4. 2020. https://CRAN.R-project.org/package=TrialSize
15. Ho DE, Imai K, King G, Stuart EA. MatchIt: nonparametric preprocessing for parametric causal inference. J Stat Softw. 2011;42(8):1–28. https://www.jstatsoft.org/v42/i08/
16. Wickham H, Miller E. haven: import and export 'SPSS', 'Stata' and 'SAS' files. R package version 2.3.1. 2020. https://CRAN.R-project.org/package=haven.

Chapter 6
Visualization of Data: Basic and Advanced

Praveen Kumar-M and Anand Srinivasan

Graphical visualization is routinely utilized for the representation of data. Representing data as graphical figures is a powerful medium for communication and will instantly help understand the data. This chapter will systematically help in learning to create graphical visualization in R. We will be creating visualization using the base package of R and also using a dedicated package for a visualization, namely "ggplot2." Accordingly, the chapter has been broadly divided into two: A basic sub-chapter and an advanced sub-chapter.

The intent of providing both methodologies is to cater to the needs of a wide variety of readers and make the readers aware of different techniques. Many times, for scientific publications or reports, we would be required to create a more robust and better visualization that is fully customizable, complex to communicate more information, and powerful enough to be created in a short span of codes. The package "ggplot2" is precisely made for this usage [1]. Nevertheless, graphical visualization fundamentals would remain the same between the two sub-chapters.

In the sub-chapter of basic visualization, we will be learning to make a bar plot (frequency-based), bar plot (average and standard deviation (SD)), histogram, boxplot, scatter plot, line chart, pie chart, and density chart. We will be systematically navigating to add features and concepts in small steps to make readers understand the logic behind codes. In the advanced visualization sub-chapter, we will learn about bar plot (frequency-based), bar plot (mean and sd), line graph, and scatter

Supplementary Information The online version contains supplementary material available at https://doi.org/10.1007/978-981-97-6980-3_6.

P. Kumar-M (✉)
Clinical Sciences, Nference, Bengaluru, Karnataka, India

A. Srinivasan
Department of Pharmacology, All India Institute of Medical Sciences, Bhubaneswar, Odisha, India

A. Srinivasan et al. (eds.), *R for Basic Biostatistics in Medical Research*,
https://doi.org/10.1007/978-981-97-6980-3_6

plot. Besides, we would also be learning to add information on significance to the figure. Finally, we would look at the methodology to concatenate multiple figures to create a single figure with multiple panels. We have also added some useful techniques, such as exporting a figure directly to a PDF file and creating multiple figures side by side (using a facet grid). The packages that we will use overall for the exercise include readxl, reshape2, ggplot2, plyr, gridExtra, ggsignif, and ez [1–7]. For the basic sub-section, other than linegraph for all the other exercises, we will be using the "dm.csv" dataset.

Bar Plot

A bar plot is a graphical representation of data where rectangular bars of equal width are drawn with lengths proportional to the values they represent. Bar plots are commonly used to display and compare the values of different categories or groups.

Bar Plot for Frequency Data

Section 1: Importing of Data into the R Console

The required CSV file (*dm.csv*) is located in the working directory, and the same file has been used in the "Data handling and manipulation in R" chapter. The imported dataset is stored in the R object named "dm".

```
## Read the data
dm = read.csv("dm.csv")
## View the dataframe
# View(dm)
attach(what = dm)
```

1. The "*read.csv*" function is used to import the data that is present in the CSV format.
2. The "*attach*" function is used for attaching the dataset so that the columns(variables) can be picked up during the utilization in the graphical function without referring to the dataset each time. The dataset should be passed as a value to the "what" argument of the attach function.

Section 2: Scanning the Dataset and Fixation

```
##The "ez" package is to be installed if it has not already been installed.
#install.packages("ez")
##Loading the installed package for visualizing.
library(ez)

## Warning: package 'ez' was built under R version 4.2.3

##Employing "ezPrecis" function from the "ez" package for the determination
of the type and missing values in the dataframe.
ezPrecis(dm)

## Data frame dimensions: 40 rows, 10 columns

##                    type missing values min max
## group           numeric       0      2   1   2
## age             numeric       0     26  33  66
## sex             numeric       0      2   0   1
## weight          numeric       0     32  39 119
## ldl             numeric       0     34  75 209
## fasting_before numeric       0     34 130 276
## fasting_after  numeric       0     33  89 225
## difference      numeric       0     36  -5  96
## response        numeric       0      2   0   1
## hba1c           numeric       0     15 5.4 7.9

#Conversion of group, age, and response to numeric
group<-as.factor(group)
sex<-as.factor(sex)
response<-as.factor(response)
ezPrecis(dm)

## Data frame dimensions: 40 rows, 10 columns

##                    type missing values min max
## group           numeric       0      2   1   2
## age             numeric       0     26  33  66
## sex             numeric       0      2   0   1
## weight          numeric       0     32  39 119
## ldl             numeric       0     34  75 209
## fasting_before numeric       0     34 130 276
## fasting_after  numeric       0     33  89 225
## difference      numeric       0     36  -5  96
## response        numeric       0      2   0   1
## hba1c           numeric       0     15 5.4 7.9
```

1. *"ezPrecis"* function offers a convenient methodology for checking the data for type and missing data. The function is from the "ez" package.
2. As the group, sex, and response variables are in numeric data type, we can use *"as.factor"* function to convert them into the factor type variables.
3. We employ the *"ezPrecis"* function again to reconfirm that the conversion has taken place successfully.

Section 3: Computation of Required Values

Bar plot demonstrating the frequency of sex in the two treatment groups. The first step is to calculate the values, which need to be imputed into the graphical function *"barplot"*.

```
##Bar plot - frequency of sex in treatment groups
##Calculation of frequency so that the results could be inputted into the
graphical function
tablesexgroup<-table(sex, group)
#In the cross tables, the first factor will form the horizontal rows, and the
second factor will form the vertical column.
#In the bar plot, the vertical column headings will form the x-axis labels.
tablesexgroup

##    group
## sex  1  2
##   0  8 14
##   1 12  6
```

1. We need to plot the bar plot demonstrating the frequency (proportion) of sex in the two treatment groups. First, we need to compute the frequency using the *"table"* function. The result is stored as the r object "tablessexgroup".

Section 4: Visualization of Bar Plot

```
##Bar plot
barplot(height = tablesexgroup)
```

1. The *"barplot"* function is used to plot the bar plot (Fig. 6.1). The *"tablesexgroup"* which was calculated in the previous step, needs to be passed as the value for the argument "height" in the *"barplot"* function.
2. The arguments that are part of *"barplot"* function include "height, weight, legend.text, beside, horiz, density, angle, col, border, main, sub, xlab, ylab, xlim, ylim". Only the essential arguments are mentioned above. The utilization of the arguments will be seen in further steps

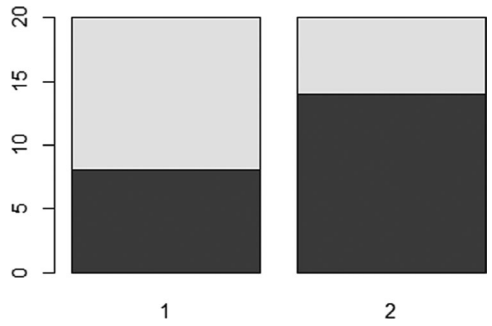

Fig. 6.1 Bar plot showing the distribution of sex across groups

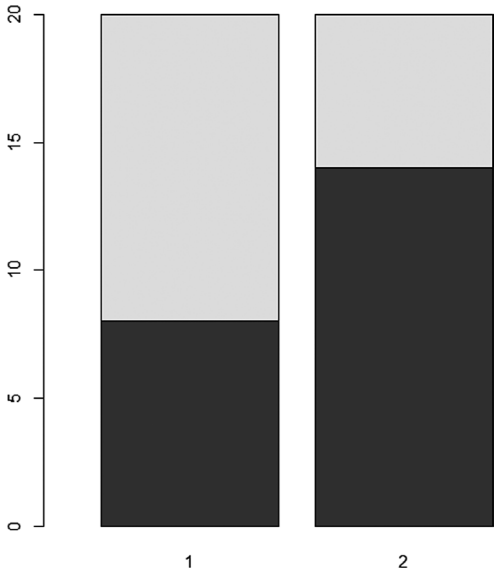

Fig. 6.2 Bar plot customization by modifying the width of the bars

Section 5: Customization of Bar Plot Using Arguments

```
##The next series of steps are for customization.
##Readjustment of the width of the bar (Fig. 6.2).
barplot(height = tablesexgroup,width = 2,xlim = c(0,6))
```

```
##Addition of color to bars (Fig. 6.3).
barplot(height = tablesexgroup,width = 2,xlim = c(0,6),col =
c("red","green"))
```

```
##Addition to labels to axes, headings, sub-headings, and legends (Fig. 6.4).
barplot(height = tablesexgroup,width = 2,xlim = c(0,6),col =
c("red","green"),names.arg = c("Drug A", "Drug B"),
        legend.text = c("Female","Male"),xlab = "Treatment Groupings",ylab =
"Number of Patients")
```

```
##Demonstration of line patterns with differing color, angle, and density.
## (Fig. 6.5).
barplot(height = tablesexgroup,width = 2,xlim = c(0,6),col =
c("red","green"),legend.text = c("Female","Male"),angle =
c(30,60),density=c(30,40),names.arg = c("Drug A", "Drug B"))
```

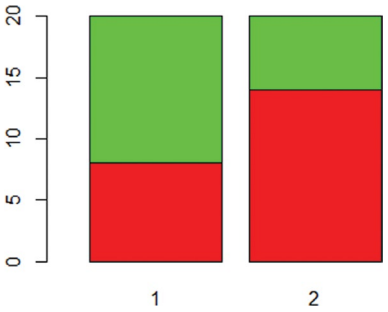

Fig. 6.3 Bar plot customization by addition of colors to the bars

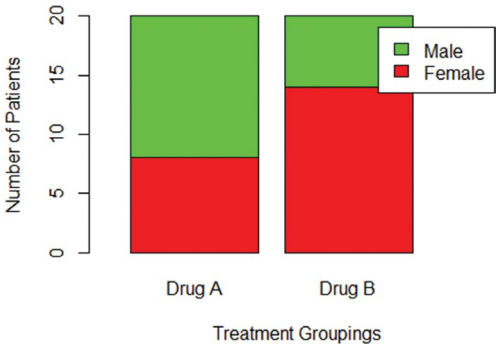

Fig. 6.4 Bar plot customization by addition of x-label, y-label, and figure legend

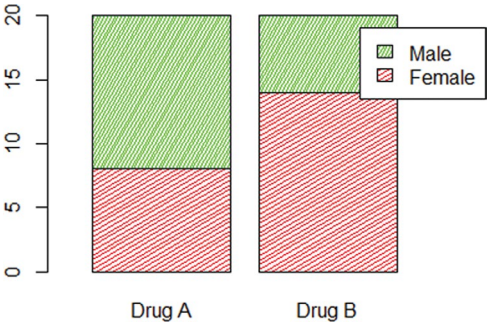

Fig. 6.5 Bar plot customization by addition and variation of angle and density of lines in bars

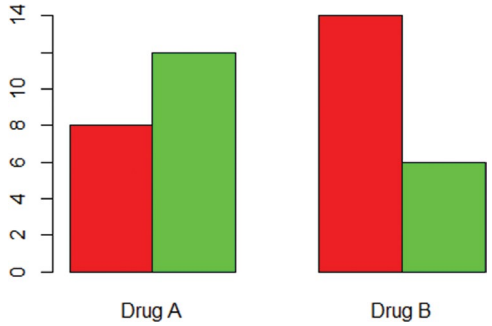

Fig. 6.6 Bar plot customization with the generation of stacked representation

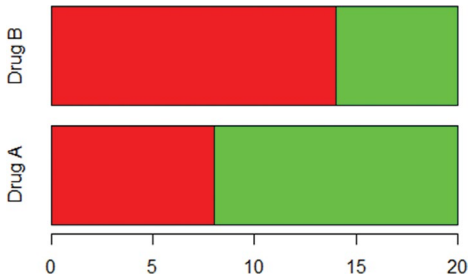

Fig. 6.7 Bar plot customization with the generation of horizontal representation

```
##Demonstration of stacked and beside representation of the bars (Fig. 6.6).
barplot(height = tablesexgroup,col = c("red","green"),beside = TRUE,names.arg
= c("Drug A", "Drug B"))
```

```
##Demonstration of the horizontal form of representation (Fig. 6.7).
barplot(height = tablesexgroup,col = c("red","green"),horiz = TRUE,names.arg
= c("Drug A", "Drug B"))
```

```
##Addition of the main heading to the bar plot (Fig. 6.8).
barplot(height = tablesexgroup,width = 2,xlim = c(0,6),col =
c("red","green"),names.arg = c("Drug A", "Drug B"),
        xlab = "Treatment Groupings",ylab = "Number of Patients",
        main = "Number of Patients of Either Sex in \n Different Treatment
Groups")
```

1. Over and above the code that was created in the previous step, we are passing the values for arguments "`width`" and "`xlim`" for customizing the width of the bar plot. It is imperative that values are passed to both "`width`" and "`xlim`" arguments simultaneously to adjust proportionally both the width and the x-axis length.(Fig. 6.2) The values are based on logic and the visual appeal that we would require.
2. The "`col`" argument is used for passing color, which we require our bar plot to be filled with. As there are two groups here, we pass two colors to the "`col`" argument (Fig. 6.3). The argument should be passed as a character vector. So, concatenation and strings need to be used appropriately.
3. There are many ways in which color values can be passed to the `col`. The simplest of them is mentioning the color directly as "green", "blue", "orange", "red", "white" and "black".
4. The "`names.arg`" takes in a character vector for naming the bar at the bottom of the bar plot. The "`legend.text`" also takes in a character vector for the creation of the legend. "`xlab`" and "`ylab`" take in a single string as an argument for labeling the x-axis and y-axis, respectively. The notation backslash followed by n(\n) could be used in between strings to insert a break in the label (Fig. 6.8).

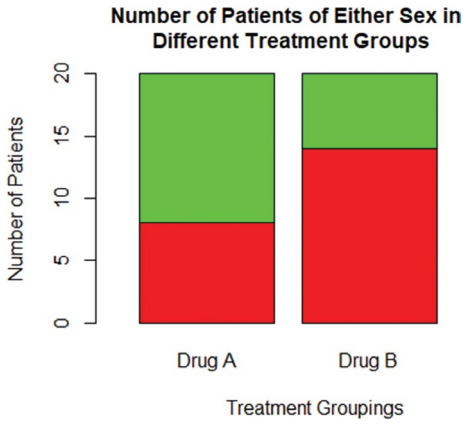

Fig. 6.8 Bar plot customization with the addition of the main heading

5. The "angle" and "density" arguments take in a numeric vector of angle and density of lines, which need to be used for differentiating between the groups in the bar plot or for representation of the bar plot (as the case may be) (Fig. 6.5).
6. The "besid" and "horiz" take the boolean operator as values. If "besid" is *FALSE,* the columns of height are stacked, and if the value is *TRUE,* the columns are juxtaposed. If "horiz" is *FALSE*, the bars are drawn vertically, and if the value is *TRUE*, the bars are drawn horizontally (Figs. 6.6 and 6.7).
7. The "main" and "sub" take in a string as the argument and are used to give the values for the main and sub-headings of the ba rplot (Fig. 6.8).
8. The "xlab" and "ylab" take in a string as an argument and are used for labeling the x-axis and y-axis, respectively (Figs. 6.4 and 6.8).
9. The "xlim" and "ylim" take in a numeric vector (2 in length) as the argument, and it is used for limiting the length of the x-axis and y-axis, respectively. The values are based on logic and the visual appeal that we would require.

Section 6: Addition of Legend as a Separate Function

```
##The legend is added as a separate function here over and above the bar plot
function above. Please note not to pass any value to legend.text argument of
'barplot' function.
barplot(height = tablesexgroup,width = 2,xlim = c(0,6),col =
c("red","green"),names.arg = c("Drug A", "Drug B"),
        xlab = "Treatment Groupings",ylab = "Number of Patients",
        main = "Number of Patients of Either Sex in \n Different Treatment
Groups")
legend(
  x = "topright",
  legend = c("Female", "Male"),
  fill = c("red","green")
)
```

```
detach(dm)
```

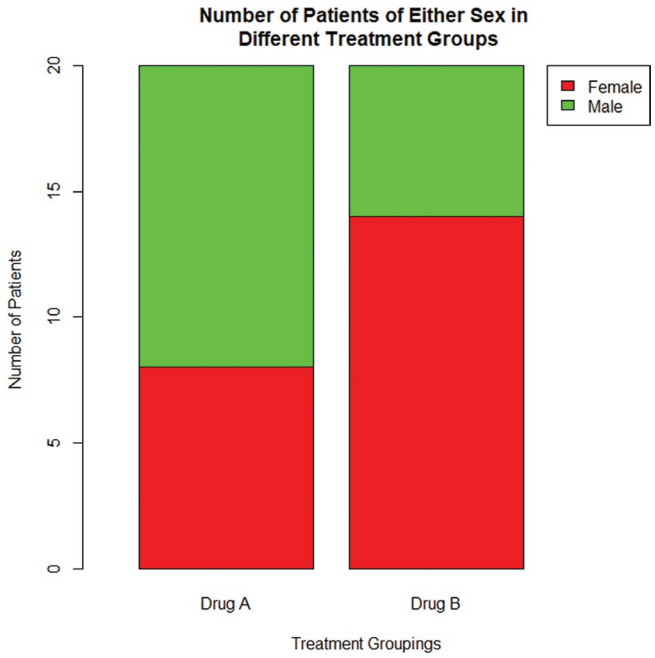

Fig. 6.9 Bar plot customization by addition of figure legend (second method)

1. The "*legend*" function is used to add legend separately over and above the bar plot graph, which would have been created in the previous step (Fig. 6.9). Please note that the function won't work as a lone function and should be preceded by a "*barplot*" function. Please note not to pass any value to the "`legend.text`" argument of the "*barplot*" function.
2. The "`X`" argument of the "*legend*" function is for mentioning the location of the legend where it needs to be placed. The possible values include "bottomright", "bottom", "bottomleft", "left", "topleft", "top", "topright", "right", "center".
3. The "`legend`" and "`fill`" argument takes in a character vector as a value for denoting the legend and color that needs to be filled in the legend. Please note that the order should be concordant with that of the representation in the graph.

Section 7: Exporting an Image

1. GUI-based export of an image (created using the graphical function) is the simplest (Fig. 6.10). The method demonstrated here is universal to export an image formed using any function.

Fig. 6.10 Quick options for saving the figure in image or PDF format

Fig. 6.11 Multiple file formats and file size options are present for saving and customization

2. First, let us populate the figure that we want to export by running the appropriate code in the **console**. The figure would have been populated in the "`Plots`" tab in the right lower part of the Rstudio window. Under the "`Plots`" option, select "`export`", which will give us two options, namely "`save as figure`" and "`save as pdf`". Click on the required option.
3. A pop-up box would have been opened. Select the image format (available image formats are `PNG`, `JPEG`, `TIFF`, `BMP`, `Metafile`, `SVG`, and `EPS`), *File name*, File directory (by selecting *directory*), *Width,* and *Height*. There are options for *Maintaining the aspect ratio* and updating the preview (for viewing the image before exporting with changes made, if any) (Fig. 6.11).
4. On clicking *Save*, the image gets exported. We can cancel the process of exporting mid-way by selecting the *Cancel* option.

Bar Plot for Average and SD Data

Bar plot demonstrating the mean Hba1c values along with positive deflection for SD in the two treatment groups.

Section 1: Calculation of Values

```
##Calculation of mean, sd
dm = read.csv("dm.csv")
attach(what = dm)

## The following objects are masked _by_ .GlobalEnv:
##
##     group, response, sex

meanhba1c<-tapply(X = hba1c,INDEX = group ,FUN = mean)
meanhba1c

##     1     2
## 6.275 6.520

sdhba1c<-tapply(X = hba1c,INDEX = group ,FUN = sd)
sdhba1c

##         1         2
## 0.3668859 0.5836365

# The methodology of calculation of se is as described below
# numhba1c<-tapply(X = hba1c,INDEX = group , FUN = length)
# numhba1c
# sehba1c <- sdhba1c/sqrt(numhba1c)
# sehba1c
```

1. We need to plot the bar plot demonstrating the Hba1c values in the two treatment groups. For this, we first compute the *mean* and *sd*. The results are stored as the R object "meanhba1c" and "sdhba1c".
2. The *"tapply"* function is used for the calculation of mean and sd.
3. The arguments which the *"tapply"* function takes are `X` (represent the vector of values for which mean is to be determined in this case *hba1c*), `INDEX` (the factorial which divides the vector of values into groups for which mean needs to be determined individually, in this case, it is the *treatment group*) and last is the `FUN` argument which takes in the function which needs to be determined. Please note that the function needs to be passed without the curly braces. The function which is passed in this case is *"mean"* and *"sd"*.
4. Though in this case, we are using the error bar as SD, if standard error (SE) is to be plotted as an error bar, calculate SE based on the formula **se = sd/ sqrt(length).**

Section 2

Visualization of bar plot based on the mean value and addition of error bars representing sd to the bar plot.

Important Note: Please select both the function (codes), namely *"barplot"* and *"arrows"*, and run them together to get a single graph.

```
##Bar plot
output = barplot(height = meanhba1c, width = 2,xlim = c(0,6),col=
rainbow(2),names.arg = c("Drug A","Drug B"))

##Addition of error bars (based on standard deviation) to the bar
output + arrows(
  x0 = barplot(meanhba1c, width = 2,xlim = c(0,6),col= rainbow(2), ylim =
c(0, 8),names.arg = c("Drug A","Drug B")),
  y0 = meanhba1c,
  x1 = barplot(meanhba1c, width = 2,xlim = c(0,6),col= rainbow(2), ylim =
c(0, 8),names.arg = c("Drug A","Drug B")),
  y1 = meanhba1c + sdhba1c,
  code = 3,
  angle = 90,
  length = 0.2
)
## numeric(0)

detach(dm)
```

1. The *"barplot"* function is used to plot the bar plot. The `meanhba1c` needs to be passed as the value to the `height` argument of the *"barplot"* function (Fig. 6.12).
2. The *"arrows"* function is used to add error bars to the bar plot.
3. The `arrows` function takes in `x0`, `x1`, `y0`, and `y1` as arguments. The *"barplot"* function per se must be passed as the value for the x0 and x1 arguments. The `y0` and `y1` take in *meanhba1c*, *meanhba1c* + *sdhba1c* as the values. The logic is that `x` represents the starting points, and `y`(`yo` and `y1`) represents the point till which the error bars should extend below and above the top of the bar plot. As the starting point is at the top of the bar, we pass the same *"barplot"* function as the argument to `x0` and `x1`. The error bar will extend from "mean" to "mean + se", so they are passed as the values to `y0` and `y1`.
4. The other arguments in the *"arrows"* function are given the following values: `code` - 3 (specific for plotting error bar), `angle`- 90 (specific for plotting error bar), and `length` - 0.5 (changed according to requirement).

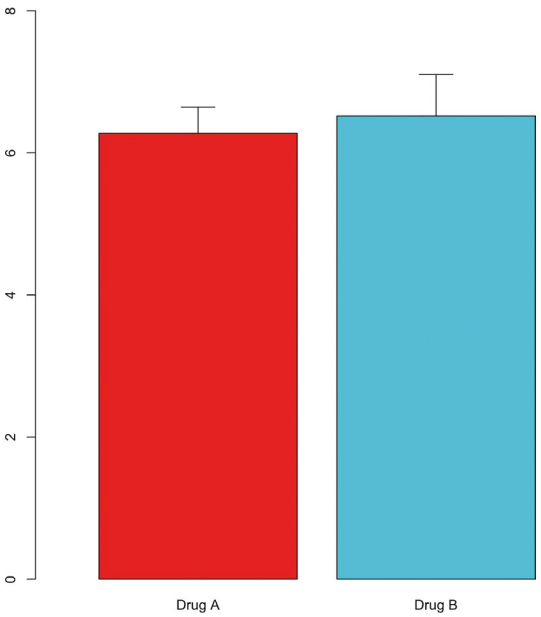

Fig. 6.12 Bar plot demonstrating mean hba1c with error bars across treatment groups

Histogram

A histogram is a graphical representation of the distribution of a dataset. It consists of a series of bars, where each bar represents a range of values (called a bin), and the height of the bar corresponds to the frequency or count of observations within that bin. Histograms are commonly used to visualize the underlying probability distribution of a continuous variable.

In this exercise, a histogram will be developed demonstrating the distribution of patients' weight in the study. There is no need to compute any values for the histogram as the plot directly takes the raw data as input.

```
##Histogram representation.
dm = read.csv("dm.csv")
attach(what = dm)

## The following objects are masked _by_ .GlobalEnv:
##
##     group, response, sex

hist(x = weight,col = "orange")

##Demonstration of breaks and other basic customization.
hist(x = weight,col = "orange",breaks = 25,ylab = "Frequency",
     xlab="Weight of the Patient in the Study",
     main = "Histogram of the Weight of Patients in the Study")
```

```
detach(dm)
```

1. The *"hist"* function is used for plotting the histogram (Fig. 6.13). The weight needs to be passed as a direct vector to the *x* argument of the *"hist"* function. The weight needs to be inputted as a vector.
2. The arguments that are part of the *"hist"* function include *break*, *density*, *angle*, *main*, *xlab*, *ylab*, *xlim*, *ylim*, *freq*. Only the essential arguments are mentioned above.
3. The *breaks* argument, if passed as a single number, represents the number of cells in the histogram. It can also be passed as a vector, in which case it represents the breakpoint in the histogram cells.
4. The *freq* is an argument that takes in logical operators. If *TRUE*, the frequencies will be taken for representation, and if *FALSE*, the density will be taken for representation instead.
5. The *main* takes in string as the argument and is used for generating the main heading of the plot.
6. The *xlab* and *ylab* take in strings as arguments and are used to name the x-axis and y-axis, respectively.
7. The xlim and ylim take in numeric vectors (2 in length) as the arguments and are used to define the limits of the x-axis and y-axis, respectively. The values are based on logic and the visual appeal that we would require.
8. The breaks in the histogram will not correspond exactly to the number of breaks mentioned with the *breaks* argument in the *"hist"* function, as R has its own calculation method for breaks.

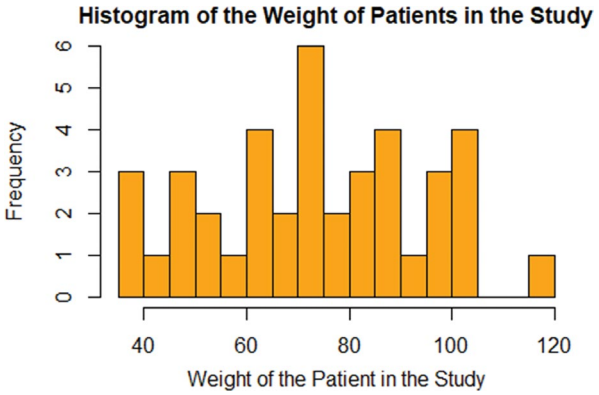

Fig. 6.13 Histogram showing the distribution of the weight of the patients

Boxplot

A boxplot, also known as a box-and-whisker plot, is a graphical representation of the distribution of a dataset. It provides a visual summary of key statistics, such as the median, quartiles, and potential outliers.

In this exercise, we will develop a boxplot for demonstrating the fasting glucose levels at the beginning and end of the study (overall). We will also make a boxplot to depict the age distribution between the intervention groups.

There is no need to compute boxplot values as the plot directly takes the raw data as input.

```
##Boxplot representation
dm = read.csv("dm.csv")
attach(what = dm)

## The following objects are masked _by_ .GlobalEnv:
##
##     group, response, sex

boxplot(fasting_before,fasting_after)

##Basic customization for completion of graph
boxplot(fasting_before,fasting_after,
        names = c("Fasting Glu Before \n Treatment","Fasting Glu After \n
Treatment"), col = c("yellow","violet"))
```

```
boxplot(formula = age~group, names = c("Drug A","Drug B"), col=
c("black","white"))
```

```
detach(dm)
```

1. The *"boxplot"* function is used to plot the boxplot (Fig. 6.14). The *fasting_before* and *fasting_after* are passed as a direct vector to the *"boxplot"* function. Please note that any number of vectors can be passed as individual arguments to the *"boxplot"* function.
2. The arguments that are part of the *"boxplot"* function include *formula*, *data*, *subset*, *x*, *notch*, *names*, *col*, and *horizontal*. Only the essential arguments are mentioned above.
3. The *formula* takes in an argument in the form of vector~groups. The argument *data* takes a dataframe, and *subset* is used for subsetting the columns from the dataframe. The argument x takes in vectors and can take any number of vectors as input.
4. The *notch* is an argument that takes in logical operator values. If *TRUE*, the notch will be added to the boxplot, and if *FALSE*, the notch will not be added to the boxplot.

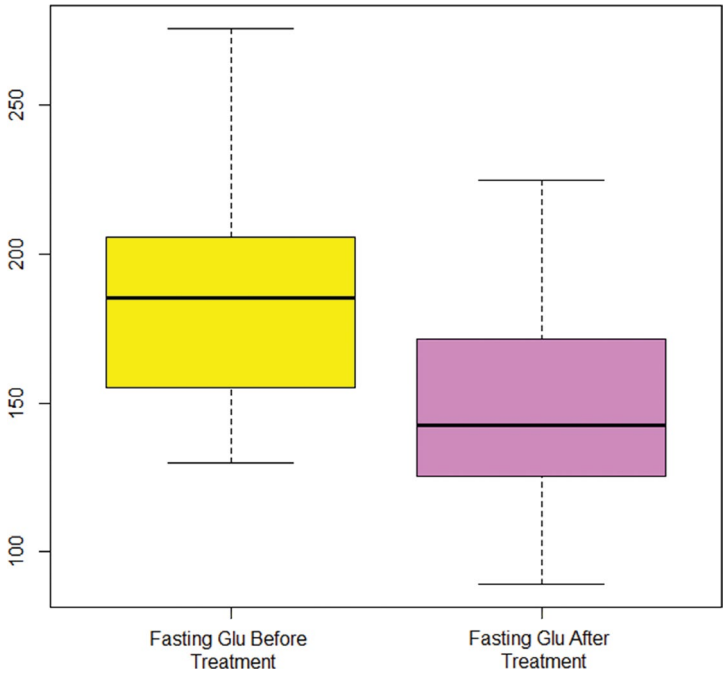

Fig. 6.14 Boxplot demonstrating the distribution of fasting glucose levels before and after treatment

5. The `col` argument has the same interpretation as that of `col` present in the `"barplot"` function.
6. The `names` argument takes in a character vector and is used for naming the boxplot at the bottom of the graph. It should be noted that the order of strings inside the character vector should match the same order in which the data was passed to the function.
7. The `horizontal` is an argument that takes in logical operator values. If *TRUE*, the horizontal boxplots will be drawn, and if *FALSE*, which is the default, vertical boxplots will be drawn.
8. The `formula` argument in the `"boxplot"` function demonstrated at the end of the code chunk (Fig. 6.15) takes input in the form of a `continuous variable~factorial variable(age ~ group` in this case). ~ is known as a tilde. The utilization of ~ can be seen in multiple functions.

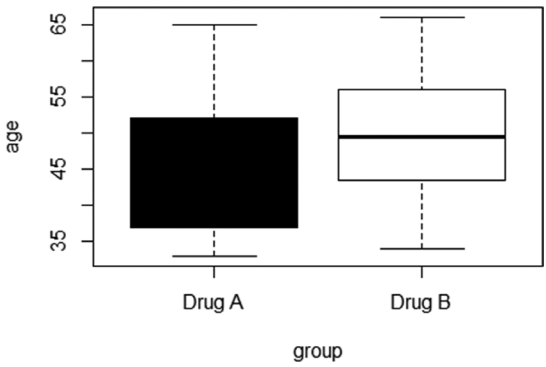

Fig. 6.15 Boxplot demonstrating the distribution of age across treatment groups

Scatter Plot

A scatter plot is a type of data visualization that displays individual data points on a two-dimensional plane. Each point represents the values of two variables, and the position of the point is determined by the values of those variables. Scatter plots are useful for visualizing relationships and patterns in data.

Section 1

Demonstration of creating a scatter plot between the fasting blood glucose before intervention and hba1c at the end of treatment (Fig. 6.16).

There is no need to compute any values for the scatter plot as the plot directly takes the raw data as input.

```
##Scatter plot representation
dm = read.csv("dm.csv")
attach(what = dm)

## The following objects are masked _by_ .GlobalEnv:
##
##     group, response, sex

plot(x = fasting_before,y = hba1c)
abline(a = lm(formula = hba1c~fasting_before), col="red")
```

1. The *"plot"* function is used for plotting the scatter plot.
2. The arguments that are part of the *"plot"* function include *x*, *y*, *main*, *sub*, *xlab*, *ylab*. Only the essential arguments are mentioned above.

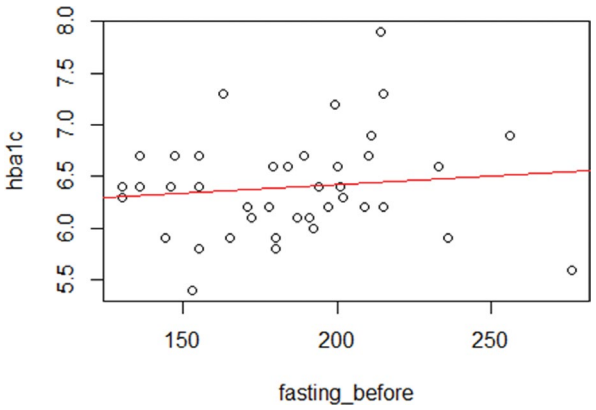

Fig. 6.16 Scatter plot between fasting plasma glucose before treatment and hba1c at the end of treatment

3. *x* and *y* arguments take a vector as input. The values to be plotted as x and y coordinates should be given in this argument. The fasting_before and hba1c need to be passed as a vector to *x* and *y* of the *"plot"* function. Since we have used *"attach"* function at the start of the document, we can pass the variable directly.
4. The function *"abline"* is used for adding the regression line to the scatter plot, which was formed using *"plot"* function. It takes in an argument known as *a*, which takes in the intercept and slope values. The *col* argument is used for defining the colors, and it takes in a character vector as input.
5. The intercept and slope are calculated through the *"lm"* function. The *"lm"* function takes an argument known as the *formula* in which the *y* and *x* need to be defined in the format of *y ~ x* (*hba1c ~ fasting_before* in this case).

Section 2

Customization of the scatter plot using arguments

```
plot(fasting_before, hba1c,xlab = "Fasting Blood Sugar mg/dL (Baseline)",
     ylab="HbA1c (End of Study)",pch =c(1,2)[response],col
=c("red","green")[response])
legend(x = "topright",legend = c("Unresponded","Responded"),col =
c("red","green"),pch = c(1,2))
```

```
detach(dm)
```

1. Over and above the code that was written in the previous step, we will provide input for the various arguments.
2. The `main` and `sub` take in a string as the input and are used to define the values for the main and sub-heading of the scatter plot (Fig. 6.17).
3. The `xlab` and `ylab` take in a string as the input and are used to label the x-axis and y-axis, respectively.
4. The `xlim` and `ylim` take in a numeric vector (2 in length) as the input, and it is used for defining the limits of the x-axis and y-axis, respectively.
5. The `col` argument is used to define the color of the data points on the scatter plot. It takes a string as input, and all the data points are uniformly colored based on the color mentioned. There are many ways in which color values can be passed to the `col` argument. The simplest of them is mentioning the color directly as "green", "blue", "orange", "red", "white", or "black".
6. In the code above, we wanted our data points to be colored according to the response, so we passed a character vector to the argument `col` and then mentioned the "response", (factorial value in this case) in square brackets next to the character vector.
7. The `"pch"` argument is used for passing the pattern of dots that we want the data points on the scatter plot to represent. If an integer is provided as input, all the dots get uniformly patterned based on the number mentioned.
8. In the code above, we wanted our dots to be patterned according to the response, so we passed a character vector as input to `pch` and mentioned the "response", (factorial value in this case) in square brackets next to the character vector.

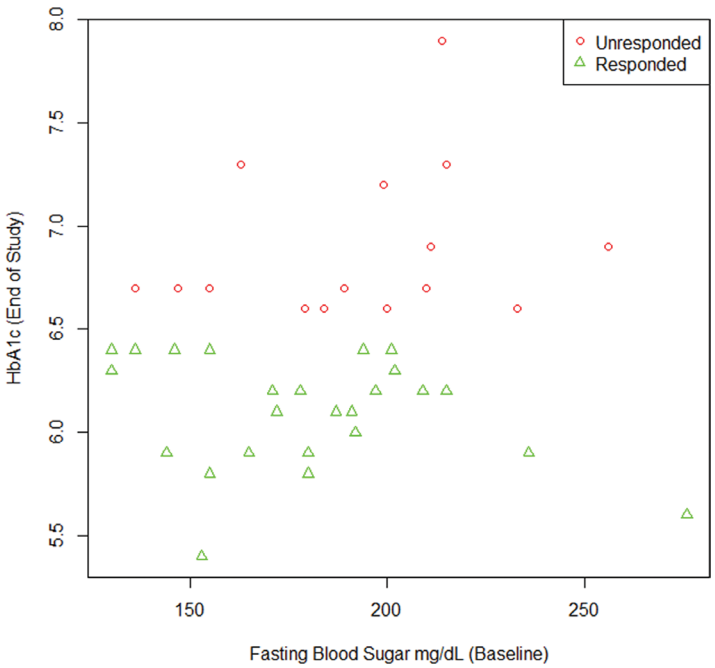

Fig. 6.17 Scatter plot customization with the addition of responder information

Line Chart

A line chart is a type of data visualization that displays data points connected by straight-line segments. It is often used to show trends or patterns over a continuous interval.

Section 1

We will proceed with creating a line chart, and the required file to be selected for this exercise is "`linechart.csv`". The dataset contains mean height as per age and Ethnicity. The aim is to plot a line chart representing heights at different ages based on Ethnicity.

```
# detach(dm) (If you have not already done)
##Reading the data
#Importing a new dataset for demonstration of line graph
line = read.csv("linechart.csv")
# View(line)
require(ez)
ezPrecis(line)

## Data frame dimensions: 35 rows, 4 columns

##              type missing values min max
## X         numeric       0     35   1  35
## Ethnicity numeric       0      5   1   5
## Age       numeric       0      7   2  20
## Height    numeric       0     22 100 220

line$Ethnicity <- as.factor(line$Ethnicity)
ezPrecis(line)

## Data frame dimensions: 35 rows, 4 columns

##              type missing values min max
## X         numeric       0     35   1  35
## Ethnicity  factor       0      5   1   5
## Age       numeric       0      7   2  20
## Height    numeric       0     22 100 220

attach(what = line)
```

1. As we are using a new dataset for demonstrating the line plots, we need to detach the *dm* dataset which we attached at the beginning of the graphical exercise. This forms a critical coding practice.
2. The "*read.csv*" function has been explained under the bar plot section. Please refer to the same.
3. The data is then visualized.

4. The *"ezPrecis"* function offers a convenient methodology for checking the data for type and missing data. The function is from *ez* package.
5. As Ethnicity is in numeric data type, we are using *"as.factor"* function to convert it into a factorial/categorical variable.
6. We employ *"ezPrecis"* function again to reconfirm that conversion has taken place successfully.

Section 2

To develop the line chart, we need to create an outline box for the plot in which the lines will be added in subsequent steps. This is followed by plotting the lines, and the title and legends are added at the end.

```
# Set up the empty plot in which the lines can be added.
# In order to set up the empty plot we need to know the range of the
parameters.
rangeofx <- range(Age)
rangeofx

## [1]  2 20

rangeofy <- range(Height)
rangeofy

## [1] 100 220

plot(x = rangeofx, y = rangeofy, type ="n",xlab="Age (in years)",
     ylab = "Height (in cm's)" )
# Adding lines to the already created empty plot.
# We will be adding lines to the plot sequentially.

lines(Age[Ethnicity==1],Height[Ethnicity==1],type = "b",
      lwd=1.5,lty=1,col="green",pch=18)
lines(Age[Ethnicity==2],Height[Ethnicity==2],type = "b",
      lwd=1.5,lty=2,col="blue",pch=19)
lines(Age[Ethnicity==3],Height[Ethnicity==3],type = "b",
      lwd=1.5,lty=3,col="red",pch=20)
lines(Age[Ethnicity==4],Height[Ethnicity==4],type = "b",
      lwd=1.5,lty=4,col="yellow",pch=21)
lines(Age[Ethnicity==5],Height[Ethnicity==5],type = "b",
      lwd=1.5,lty=5,col="orange",pch=22)

# Add a title to the plot
title(main = "Growth of Children in Different Ethnic Group")

# Now, its time to add a legend to the plot
legend(x = "topleft", legend = c(1,2,3,4,5), cex=0.9,
col=c("green","blue","red","yellow","orange"),
       pch=c(18,19,20,21,22), lty=c(1,2,3,4,5), title="Ethnicity Groups")
```

```
detach(name = line)
```

1. The "*plot*" function is used for plotting the outline of the box. As described earlier, it will take *x* and *y* as the arguments (Fig. 6.18).
2. The "*range*" function is used to determine the limits of the series and has a numeric vector as input. So "Age" and "Height" are passed as arguments to the "*range*" function separately, and the output is stored as *rangeofx* and *rangeofy*, respectively.
3. *rangeofx* and *rangeofy* are passed as values for the *x* and *y* arguments of the plot function.
4. The "*lines*" function is used for plotting the lines to the box. It takes *x*, *y*, *pch*, *lty*,*lwd*, *col* as arguments. Only the essential arguments are mentioned here.
5. The *x* and *y* arguments of the "*line*" function take in a numeric vector as input. The input for *x* and *y* is obtained by indexing the corresponding vectors for the required condition to be satisfied. For example, "Age[Ethnicity==1], Height[Ethnicity==1]" represents selecting the Age and Height vectors and subsetting them with the required Ethnicity condition (1 in this case) so that only the data (Age and Height) from participants whose Ethnicity is 1 will be selected for both the vectors.
6. The *type* argument takes in strings as values. The value b is passed as type in this case.

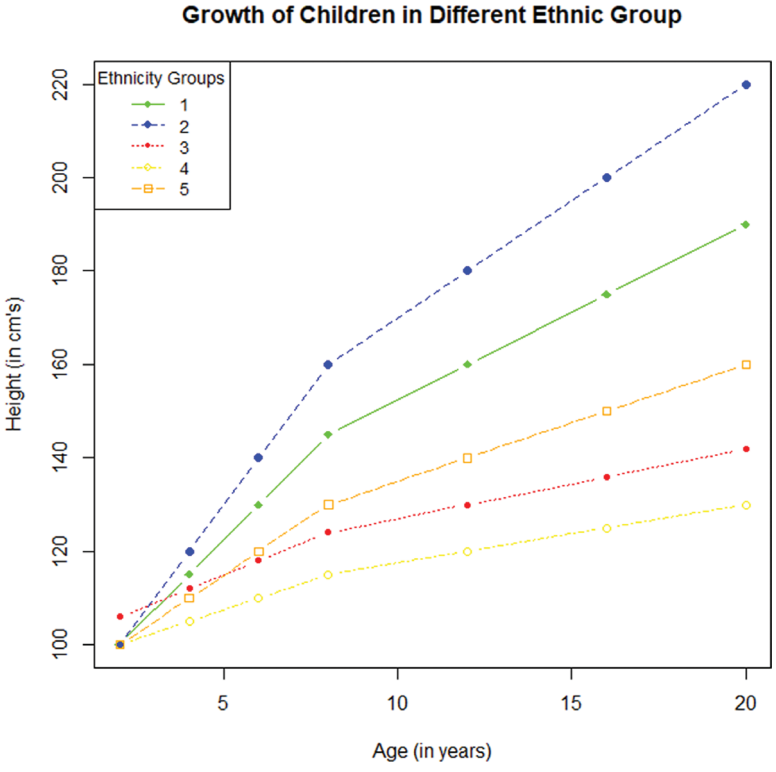

Fig. 6.18 Line chart showing the growth of children in different ethnic age groups

7. The *lty* argument stands for line type and takes numeric integers as values.
8. The *lwd* argument stands for line width and takes numeric integers as values.
9. The *pch* argument is used to select the pattern of dots representing the data points. It takes a numeric integer as the input.
10. The *col*, *lty*, and *pch* are given different values to differentiate between the Ethnicity groups in the plot.
11. The *"title"* function is used for adding features on the previously created scatter plot (line graph in this case). The *main* argument takes a string as input and is used for naming the plot.
12. The *"legend"* function is already explained in the bar plot. The arguments *pch* and *lty* take in a numeric vector as input and should match the *pch* and *lty* of the lines that are passed for representing the different Ethnicity groups in the line graph (also in the same order). The *cex* argument denotes the size of the legend.
13. The *col* argument takes in a character vector as input and is used to pass different colors that represent different ethnicity groups based on values given in the line graph (also in the same order).
14. The *"detach"* function is used for detaching the dataset line.

Pie Chart

A pie chart is a circular statistical graphic that is divided into slices to illustrate numerical proportions. Each slice represents a proportionate part of the whole data set.

Section 1

The aim is to plot two pie charts beside each other representing the responded and unresponded patients in treatment groups—1 (A) and 2 (B).

```
##Piechart representation of unresponded and responded in treatment
groups 1 and 2
dm1<-subset(dm,subset = group == 1)
dm2<-subset(dm,subset = group == 2)
tableresponse1<-table(dm1$response)
tableresponse1

##
##  0  1
##  5 15

tableresponse2<-table(dm2$response)
par(mfrow = c(1,2))

pie(x = tableresponse1,labels = c("Unresponded","Responded"),
   col = rainbow(2),main ="Treatment Group A")
pie(x = tableresponse2,labels = c("Unresponded","Responded"),
    col = rainbow(2),main = "Treatment Group B")
```

1. The *"subset"* function is used to create the subset of the *dm* dataset based on groups 1 and 2. The logical operator == is used for equating. The function *"subset"* is discussed in the chapter "Data handling and manipulation in R".
2. The *"par"* function is used to set the graphics to accept and display graphs based on requirements (2 pie charts side by side in this case). The *mfrow* argument takes in a numerical vector of length 2, with the first value representing rows and the second value representing columns. In this case, the value is set to `c(1,2)` to represent 1 row and 2 columns (Fig. 6.19).
3. The *"pie"* function takes in the arguments *x*, *labels*, *clockwise*, *density*, *angle*, *col*, and *main*. Only the essential arguments are described here.
4. The *x* argument takes input as a numeric vector. We use the *"table"* function to compute the frequency of response (in 2 groups separately) and name the output as *tableresponse1* and *tableresponse2*. The values are individually passed to *"pie"* functions.
5. The `density`, `angle`, `main`, and `col.` have the same interpretation as that described in the previous sections.
6. The *clockwise* takes in logical values as input, and it is used to denote whether the pie chart should be drawn clockwise(if TRUE) or anti-clockwise(if FALSE).

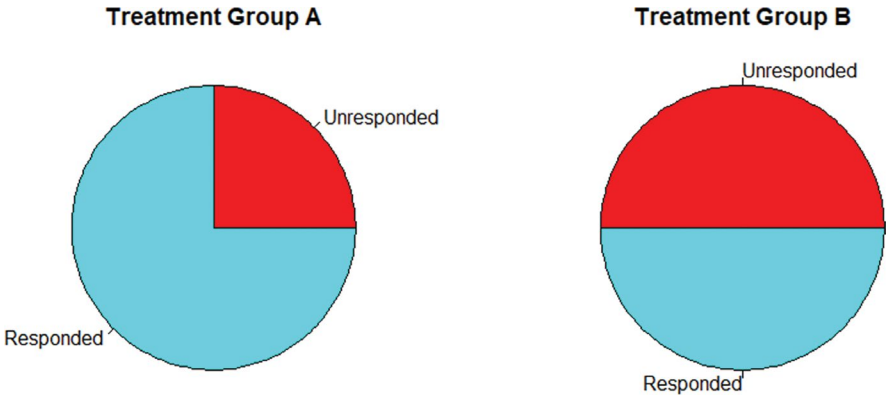

Fig. 6.19 Pie chart showing responders and nonresponders in Group A and Group B

Section 2

Labels representing percentages of responses are to be added to the pie chart.

```
#Addition of percentage to label
tableresponse1per<-(tableresponse1/sum(tableresponse1))*100
tableresponse1per

##
##  0  1
## 25 75

tableresponse2per<-(tableresponse2/sum(tableresponse2))*100
tableresponse2per

##
##  0  1
## 50 50

labeltableresponse1per<-
paste(c("Unresponded","Responded"),tableresponse1per,sep = " ")
labeltableresponse2per<-paste(c("Unresponded","Responded"),tableresponse2per,
sep = " ")
label1<-paste(labeltableresponse1per,"%",sep = "")
label2<-paste(labeltableresponse2per,"%",sep = "")
par(mfrow = c(1,2))
pie(tableresponse1,labels = label1,
    col = rainbow(2),main ="Treatment Group A")
pie(tableresponse2,labels = label2,
    col = rainbow(2),main ="Treatment Group B")
```

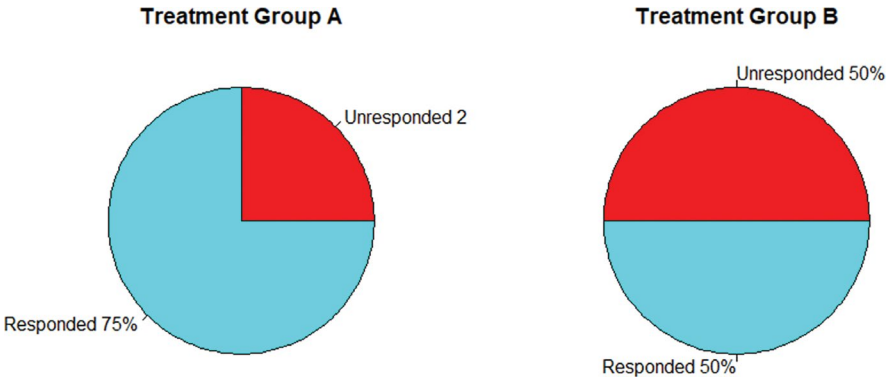

Fig. 6.20 Pie chart customization with the addition of labels for responders and nonresponders

1. The percentage of responses is calculated for each group.
2. The *"paste"* function is used to create appropriate labels. The labels are stored as variables (*label1* and *label2*).
3. The values (label variables) are passed to the *labels* argument of the *"pie"* function (Fig. 6.20).

Density Plot

A density plot, also known as a kernel density plot, is a graphical representation of the distribution of a continuous variable. It provides a smooth estimate of the probability density function of the data.

Section 1

The aim is to plot two density plots side by side representing the LDL cholesterol concentration in two intervention groups (Fig. 6.21).

```
attach(what = dm)

## The following objects are masked _by_ .GlobalEnv:
##
##     group, response, sex

##Density plot for overall LDL cholesterol in the study
par(mfrow =c (1,2))
plot(density(x = ldl[group==1]),main = "LDL in Group 1")
plot(density(x = ldl[group==2]), main = "LDL in Group 2")
```

```
detach(name = dm)
```

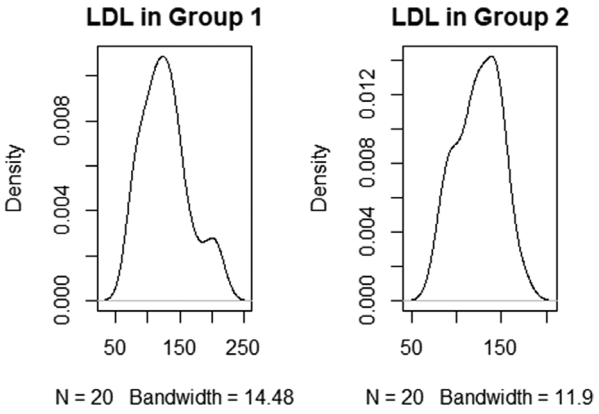

Fig. 6.21 Density plot for depicting overall LDL cholesterol in group 1 and group 2

1. The *"density"* function is used to calculate the density of LDL cholesterol values. It takes a numeric vector as the value for the argument `x`. Indexing is used to select each group's values separately.
2. The determined density values are then passed as input to the *"plot"* function.
3. Additional customization inherent to the *"plot"* function that is applicable in this context can also be performed.

Line Chart Using "for" Loop

The aim is to draw the line chart (demonstrated above) using *"for"* loop. The intention of demonstrating the *"for"* loop is to make the readers aware that the lines of codes can be condensed by utilizing *"for"* loops for identical repetitive codes that vary only in the variable selected for execution.

```
line<-read.csv("linechart.csv")
require(ez)
ezPrecis(line)

## Data frame dimensions: 35 rows, 4 columns

##              type missing values min max
## X         numeric       0     35   1  35
## Ethnicity numeric       0      5   1   5
## Age       numeric       0      7   2  20
## Height    numeric       0     22 100 220

line$Ethnicity <- as.factor(line$Ethnicity)
ezPrecis(line)

## Data frame dimensions: 35 rows, 4 columns

##              type missing values min max
## X         numeric       0     35   1  35
## Ethnicity  factor       0      5   1   5
```

```
## Age        numeric       0      7   2  20
## Height     numeric       0     22 100 220

rangeofx <- range(line$Age)
rangeofx

## [1]  2 20

rangeofy <- range(line$Height)
rangeofy

## [1] 100 220

plot(x = rangeofx, y = rangeofy, type ="n",xlab="Age (in years)",
     ylab = "Height (in cm's)" )


ltyline=1:5
colline=rainbow(5)
pchline=18:22

for (i in 1:5) {
  lines(line$Age[line$Ethnicity==i], line$Height[line$Ethnicity==i],
type="b", lwd=1.5,lty=ltyline[i], col=colline[i], pch=pchline[i])
}
```

1. The initial part of the code has already been explained under the "line chart" section.
2. Two numeric vectors are created for *lty* and *pch* as *ltyline* and *pchline,* respectively. Appropriate lty and pch values are placed in the vector according to requirement.
3. A character vector named *colline* is created, and it contains the required colors to be passed to the lines.
4. The *"for"* loop iterates value 1:5 to an iterating variable `i`. The iterating variable passes the corresponding value it gets to the statements inside the *"for"* loop (Fig. 6.22).

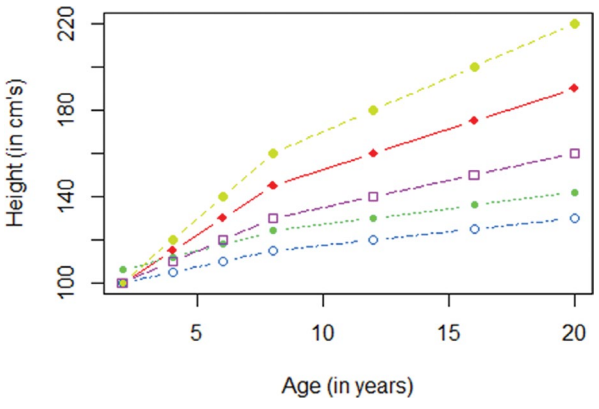

Fig. 6.22 Generation of line plot using a more elegant 'for loop' method

5. In the first line inside *"for"* loop, the "line" dataframe is read, and the required values are subsetted based on Ethnicity.
6. The *ltyline*, *pchline,* and *colline* are subsetted based on indexing, the values of which are passed by the iterating variable.
7. The "linechart.csv" file is reloaded to the *line* variable at the end of the loop so as to get ready for another round of `for` loop.
8. In this case, looping will occur from values 1 to 5.

Advanced Visualization

In the advanced visualization section, we will explore the use of the "ggplot2" package in generating different types of plots.

Bar Plot with Frequency (Advanced)

A randomized control trial was conducted in 80 patients (40 patients in each arm) to compare the effect of intravitreal Treatment_AZX vs. Treatment_HJK in patients of retinopathy. Optical coherence tomography was used to measure cystoid macular edema (CME) at baseline, week eight, and week 10. CME was graded as nil, focal, or diffuse. Draw a bar plot depicting the frequency of distribution of different CME grades by time point of measurement and treatment.

Section 1: Importing and Preparation of Data

Since the data is present in the Excel sheet, we need a dedicated package. We would be using "readxl" package here. The "readxl" package has a function known as *"read_excel"* that is used to import the data specifically from the Excel sheet.

```
# install.packages("readxl")
library(readxl)

## Warning: package 'readxl' was built under R version 4.2.3

cme <- read_xlsx("cme.xlsx")
colnames(cme)

## [1] "Treatment"    "CME_Baseline" "CME_Week8"    "CME_Week10"

View(cme)
```

1. Installation of `readxl` package using the *"install.packages"* function. This is required if the `readxl` package is not installed in your package library.
2. The *"read.excel"* function is used to import the data, which is present in Excel format, and the data is stored as a dataframe in the R object *cme*.

Section 2: Melting the Data into the Long Format

Melting and casting of data constitute essential components of data restructuring to enable data visualization. Melting stands for converting the wide format to long format data, and casting stands for the conversion of long format to wide format. For, in the above example, the CME of the patient for each month was written in different columns. This form of data entry is correct, where we dedicate one row for each patient and then mention the patient's relevant characteristics in each column. But for plotting as a graph, we would require the values to be in a single column and the factor that divides them to be correspondingly present in a separate column. Hence, restructuring of the column becomes essential. The package which is used for restructuring is "reshape2."

```
# install.packages("reshape2")
library("reshape2")

## Warning: package 'reshape2' was built under R version 4.2.3

cmemelt <- melt(data = cme, measure.vars =
c("CME_Baseline","CME_Week8","CME_Week10"), id.vars = c("Treatment"))
colnames(cmemelt)

## [1] "Treatment" "variable"  "value"

head(cmemelt)

##       Treatment     variable   value
## 1 Treatment_AZX CME_Baseline  Diffuse
## 2 Treatment_AZX CME_Baseline   Focal
## 3 Treatment_AZX CME_Baseline  Diffuse
## 4 Treatment_AZX CME_Baseline  Diffuse
## 5 Treatment_AZX CME_Baseline      No
## 6 Treatment_AZX CME_Baseline      No

dim(cmemelt)

## [1] 240   3

cmemelt$Treatment<-factor(x = cmemelt$Treatment, levels =
c("Treatment_AZX","Treatment_HJK"))
cmemelt$value<-factor(x = cmemelt$value,levels = c("No","Focal","Diffuse"))
cmemelt$variable<-factor(x = cmemelt$variable,levels =
c("CME_Baseline","CME_Week8", "CME_Week10"))

80* 3

## [1] 240
```

The steps for melting the data are provided below.

1. Install the `reshape2` library using the *"install.packages"* function.
2. Load the `reshape2` library using the *"library"* function.
3. The *"melt"* function is used to melt the data. It takes three arguments for the execution, namely *data, measure.vars, and* `id.vars`. The *data* argument is for denoting the data. The *measure.vars* is used to denote those columns representing the measures to be converted to a long format. The *id.vars* is used to denote those columns that are used for identification. The *measure.vars* and *id.vars* take input in the form of a character vector. We will assign the melted dataset to an object known as *cmemelt*.
4. The *"colnames"* function is used to visualize the column names of the dataframe. We can see that two new columns, namely *variable* and *value*, have been created. Meanwhile, the column *Treatment* was retained from the previous dataset *cme*. The *variable* column contains the time point of assessment, and the *value* column contains the patient's CME grading. The number of rows in the dataset *cmemelt* is 240, which is 80 * 3 (80 is the number of rows in the *cme* dataframe).
5. The *"factor"* function is used to assign the order of factor levels. This is essential for obtaining the required format of appearance in the plot.

Section 3: Creation of the Bar Plot

```
# install.packages("ggplot2")
library("ggplot2")

## Warning: package 'ggplot2' was built under R version 4.2.3

CME_Barplot <-
  ggplot(data = cmemelt, mapping = aes(x = variable, fill = value)) +
  geom_bar(stat = "count", position = "dodge")+ facet_grid("Treatment") +
scale_x_discrete(
    name = "Time Point of Measurement",
    breaks = c(
      "CME_Baseline",
      "CME_Week8",
      "CME_Week10"
    ),
    labels = c("Baseline", "Week 8","Week 10")
  ) + scale_fill_discrete(name = "CME")+ scale_y_continuous(name =
"Frequency", limits = c(0, 30)) + theme_bw(
  )
CME_Barplot
```

```
#To check if the figure has plotted the right values, we can use a simple
function
table(cmemelt$variable,cmemelt$value,cmemelt$Treatment)

## , ,  = Treatment_AZX
##
##
##                No Focal Diffuse
##   CME_Baseline 14     9     17
##   CME_Week8     5    13     22
##   CME_Week10    9    15     16
##
## , ,  = Treatment_HJK
##
##
##                No Focal Diffuse
##   CME_Baseline  8    19     13
##   CME_Week8     7    22     11
##   CME_Week10   15    14     11
```

1. The "`ggplot`" function is the primary standard for many ggplot2 visualizations. It takes in the arguments `data` and `mapping`. The `mapping` is an aesthetic argument, which takes `x`, `y`, `fill`, and `group` as the arguments. `x` denotes the variable in the x-axis (Fig. 6.23). `y` denotes the values on the y-axis. In this case, we are letting the y values be computed, and hence, we are not mentioning the y-axis values. The `fill` is used to mention the column to be used as the basis for coloring the bars and the legend formation for the graph. If we are not mentioning the values for different aesthetics in this function, we should mention the values in the visualization type-specific function (discussed below).
2. Here, we have three factors [Treatment (*Treatment*), Timepoint of measurement (*variable*), and CME grading (*value*)]. We have mentioned the *value* and *variable* with the "`ggplot`" function. The *treatment* is thereby used for faceting the graph. Facet can be visualized as a window by the readers.
3. The "`geom_bar`" is the function for the bar plot in ggplot2. The other possible functions for different visualization types are "`geom_point`" for point placement, "`geom_line`" for line placement, and "`geom_histogram`" for histogram. The "`geom_bar`" also has an aesthetic argument. However, if we have mentioned the required values in the "`ggplot`" function, the aesthetic arguments in the geom_bar function are not required to be mentioned again. Moreover, two other arguments exist for this function (`stat` and `position`). The `stat` can take either `identity` or `count` as values. The `identity` is used to directly use the values provided, whereas the `count` is a statistical function to calculate the frequency based on the `x` and `fill` and `facet` values provided. The `position` can take either `dodge` or `stack` as values that represent the pattern and position of the bar in the plot.

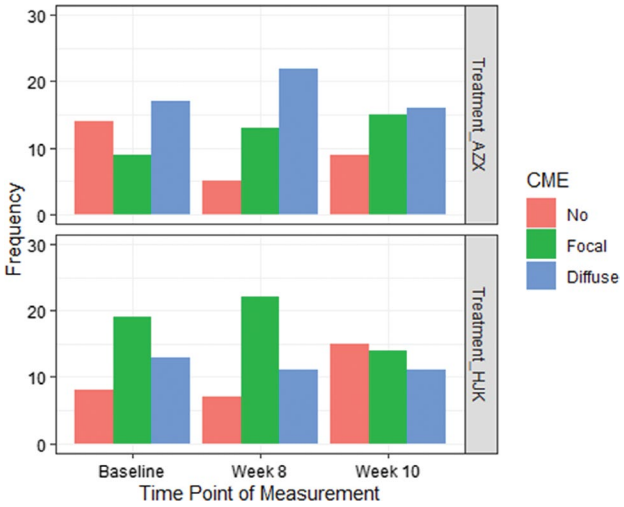

Fig. 6.23 Bar plots showing the frequency of various grades of CME at three study time points in treatment groups AZK and HJK

4. The other argument to be passed needs to be thought based on the data supplied. Here, we have both the `x` and the `fill` values as categorical (discrete) values rather than continuous values. Hence, we will be using the `"scale_x_discrete"` and `"scale_fill_discrete"` to modify the names and labels for these values. However, the y-axis is a continuous variable, and hence, we use the `"scale_y_continous"` function to modify the same. The difference between `breaks` and `labels` is that `breaks` take in a character vector to mention the factor values present in the dataset, whereas the `labels` take in the character vector corresponding to the values of the `breaks` that have to be mentioned in the figure.
5. The `"theme_bw"` function is added to make the background dark-on-light appearance. The other possible themes are `"theme_gray"`, `"theme_light"`, `"theme_void"`, and `"theme_dark"`.
6. The programming pattern of ggplot2 employs adding subsequent pieces of the function using + and ultimately producing the figures. If we have mentioned the same function more than once, the last-mentioned function will be taken for the figure.

Bar Plot for Average and SD Data (Advanced)

An experiment was conducted to assess the effect of six different interventions (Treatment A, B, C, D, E, and F) on weight of rats over 3 weeks. The intervention was administered once daily to the rat for the entire span of 3 weeks, and the weight

was measured at the end of each week. In each intervention arm, there were eight rats in total. We aim to plot a bar plot to depict each intervention arm's mean and SD over three weeks. The *ID* column represents the identification number of the rat. *Interventions* represent the treatment given to the rat. *Baseline_Weight*, *Week_1_Weight*, *Week_2_Weight*, and *Week_3_Weight* represent the rat's weight (grams) measured in the baseline, week 1, week 2, and week 3.

Section 1: Importing and Preparation of Data

```
# install.packages(readxl)
library(readxl)
wa <- read_xlsx("wgtrat.xlsx")
# View(wa)
wa$Interventions = factor(x = wa$Interventions, levels = c("Treatment_A",
"Treatment_B", "Treatment_C", "Treatment_D", "Treatment_E", "Treatment_F"))
wa$Interventions

##  [1] Treatment_C Treatment_C Treatment_C Treatment_C Treatment_C
Treatment_C
##  [7] Treatment_C Treatment_C Treatment_D Treatment_D Treatment_D
Treatment_D
## [13] Treatment_D Treatment_D Treatment_D Treatment_D Treatment_A
Treatment_A
## [19] Treatment_A Treatment_A Treatment_A Treatment_A Treatment_A
Treatment_A
## [25] Treatment_B Treatment_B Treatment_B Treatment_B Treatment_B
Treatment_B
## [31] Treatment_B Treatment_B Treatment_E Treatment_E Treatment_E
Treatment_E
## [37] Treatment_E Treatment_E Treatment_E Treatment_E Treatment_F
Treatment_F
## [43] Treatment_F Treatment_F Treatment_F Treatment_F Treatment_F
Treatment_F
## 6 Levels: Treatment_A Treatment_B Treatment_C Treatment_D ... Treatment_F
```

1. The `"View"` function enables us to visualize the data.
2. The `"factor"` function is used for assigning the *Interventions* in *wa* dataframe as a factorial variable. The factor function takes an additional argument known as `levels` for assigning the order of factors in the *Interventions* column. A character vector is provided as input for the `levels`. The `levels` argument, however, is not mandatory.
3. We visualize the *Interventions* column in the console to check the order of levels.

Section 2: Melting the Data into the Long Format

```
wamelt <- melt(data = wa, measure.vars = c("Baseline_Weight", "Week1_Weight",
"Week2_Weight", "Week3_Weight"), id.vars = c("Interventions","ID"))
colnames(wamelt)

## [1] "Interventions" "ID"          "variable"     "value"
```

The `melt` function has been explained detailedly in Section 2 of "Bar plot with frequency (Advanced)".

Section 3: Computation of mean and sd

The data set contains the raw values of rat weights in the intervention group. Each intervention group has eight rats, but the final bar plot needs to be plotted with the intervention group's mean rather than each rat. Hence, we need to compute the mean as well as the SD of the intervention group. This is the characteristic requirement of a bar plot, where a pre-computation is required before the final plot is constructed. For your understanding, such a computation is not required for boxplots, scatter plots, or histograms, which take the raw value from the dataset. Even in the previous bar plot for frequency, pre-computation was required. However, the computation was not manually made and was integrated into the function *"geom_bar"* with the argument *stat* taking the value of `count`.

```
library(plyr)
## Warning: package 'plyr' was built under R version 4.2.3

waagg<-ddply(.data = wamelt,.variables = c("Interventions","variable"),
summarise,mean = mean(value), sd = sd(value))
waagg

##    Interventions        variable    mean       sd
## 1    Treatment_A Baseline_Weight 347.875 27.39363
## 2    Treatment_A    Week1_Weight 339.750 28.97166
## 3    Treatment_A    Week2_Weight 339.875 29.21564
## 4    Treatment_A    Week3_Weight 350.000 24.83661
## 5    Treatment_B Baseline_Weight 324.000 18.60875
## 6    Treatment_B    Week1_Weight 316.250 22.75177
## 7    Treatment_B    Week2_Weight 324.375 20.94167
## 8    Treatment_B    Week3_Weight 345.000 18.63177
## 9    Treatment_C Baseline_Weight 368.000 42.59108
## 10   Treatment_C    Week1_Weight 357.750 41.61988
## 11   Treatment_C    Week2_Weight 348.375 39.27899
## 12   Treatment_C    Week3_Weight 373.500 42.79519
## 13   Treatment_D Baseline_Weight 342.500 31.88372
## 14   Treatment_D    Week1_Weight 346.875 29.56077
## 15   Treatment_D    Week2_Weight 347.125 27.72538
## 16   Treatment_D    Week3_Weight 362.375 31.36394
```

```
## 17   Treatment_E Baseline_Weight 338.375 32.24432
## 18   Treatment_E    Week1_Weight 328.000 25.85675
## 19   Treatment_E    Week2_Weight 330.000 33.95796
## 20   Treatment_E    Week3_Weight 331.750 32.39819
## 21   Treatment_F Baseline_Weight 337.875 29.37169
## 22   Treatment_F    Week1_Weight 319.625 26.45178
## 23   Treatment_F    Week2_Weight 334.375 28.25363
## 24   Treatment_F    Week3_Weight 345.625 30.88198
```

1. Installation of `plyr` library using the *"install.packages"* function.
2. Loading the `plyr` library using the *"library"* function.
3. The *"ddply"* function aggregates the data and computes the mean and sd. It takes 3 arguments for the execution, namely `.data`, `.variables`, and `.fun`.
4. `.data` takes the dataframe as the input.
5. `.variables` takes the column names on the basis of which the computation is to be executed.
6. .fun takes the function to be applied on each piece. In this case, we have used the *"summarise"* function.
7. *"mean"* and *"sd"* are specifically mentioned to execute those functions.

Section 4: Creation of the Bar Plot

```
waaggbarplot<-ggplot(data = waagg, mapping =  aes(
  x = variable,
  y = mean,
  fill = Interventions
  ))+ geom_bar(position = position_dodge(),
  colour = "black",
  stat = "identity") + geom_errorbar(aes(ymin = mean - sd, ymax = mean +
  sd),
  width = .2,
  position = position_dodge(0.9))+
scale_x_discrete(
  name = "Time Points",
  breaks = c("Baseline_Weight", "Week1_Weight", "Week2_Weight",
"Week3_Weight"),
  labels = c("Baseline", "Week 1", "Week 2", "Week 3")
  )+
  scale_fill_discrete(
  name = "Interventions",
  breaks =
c("Treatment_A","Treatment_B","Treatment_C","Treatment_D","Treatment_E","
Treatment_F"),
  labels = c("Treatment A", "Treatment B", "Treatment C", "Treatment D",
"Treatment E","Treatment F"
  )
  ) + ylab("Mean Weight (in gms)")

waaggbarplot
```

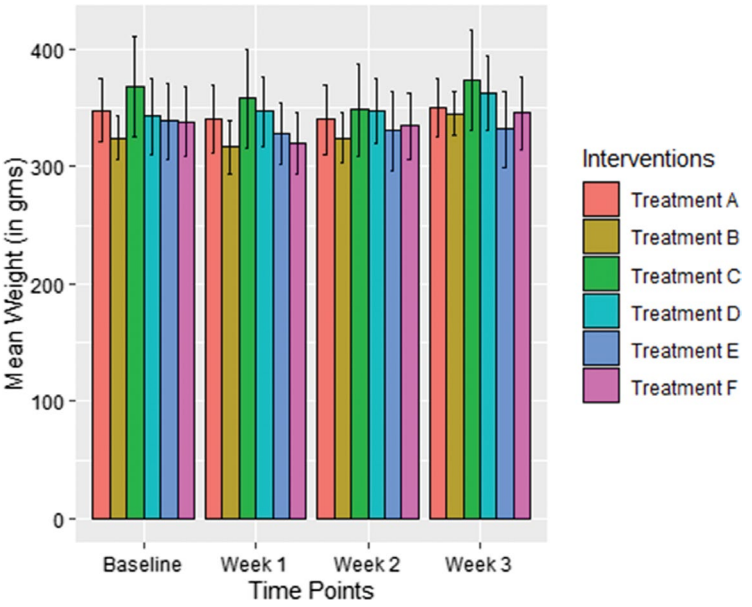

Fig. 6.24 Bar plots showing the mean weight of rats at different study points in different treatment groups

In addition to the argument that we had passed earlier, we pass a new function, "geom_errorbar", to add the SD (Fig. 6.24). As the aesthetics required for this function have not been mentioned earlier, we mention them as arguments for this function. So, we pass two arguments, `ymin`, and `ymax`, with the appropriate values.

Section 5: Additional Modification to the Bar Plot

5A: Modifying the Background Theme

```
waaggbarplot+theme(
  panel.grid.major = element_blank(),
  panel.grid.minor = element_blank(),
  panel.background = element_blank(),
  axis.line = element_line(colour = "black")
  )+theme(panel.grid.major = element_blank(),
          panel.grid.minor = element_blank(),
          legend.title = element_text(size = 12,face = 'bold'),
          legend.text = element_text(size = 10),
          axis.text.x = element_text(
          size = 12,
          ),
          axis.text.y = element_text(size = 10),
          axis.title = element_text(size = 12, face = 'bold'),
          axis.line = element_line(),
          plot.background = element_rect(size = 1, color = "red"),
          panel.background = element_blank()
          )

## Warning: The `size` argument of `element_rect()` is deprecated as of
ggplot2 3.4.0.
## i Please use the `linewidth` argument instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```

The bar plot generated can be further customized by using "*theme*" function (Fig. 6.25).

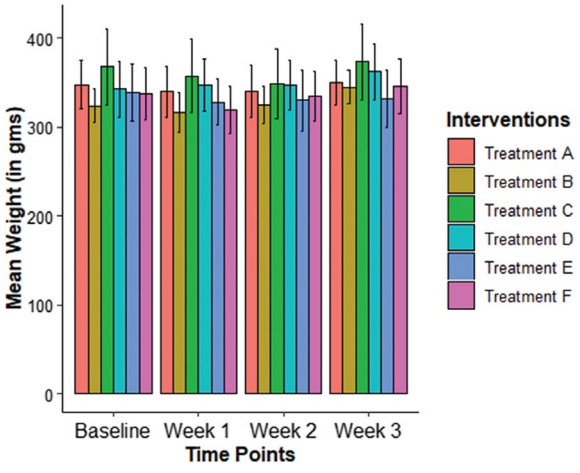

Fig. 6.25 Bar plot customization by modification of background and fonts

5B: Storing the Created Image to a PDF File

By writing an image directly into a file, we can automate the workflow seamlessly.

```
pdf(file="Barplot.pdf", width=8,height = 5)
waaggbarplot
dev.off()
```

Line Graph (Advanced)

In the same experiment mentioned above for the bar plot (weight of rat), our aim now is to plot a line graph depicting the mean alone of each intervention arm over a span of three weeks (Fig. 6.26). The line graph will help understand the weight change pattern in each group over the evaluated time points. The steps of sections 1, 2, and 3 will remain the same, and the modification will be made in Section 4.

```
waagglinegraph <- ggplot(
  data = waagg,
  mapping =  aes(
    x = variable,
    y = mean,
    group = Interventions,
    color = Interventions
  )
) + geom_point() + geom_line() +
  scale_x_discrete(
    name = "Time Points",
    breaks = c(
      "Baseline_Weight",
      "Week1_Weight",
      "Week2_Weight",
      "Week3_Weight"
    ),
    labels = c("Baseline", "Week 1", "Week 2", "Week 3")
  ) +
  scale_color_discrete(
    name = "Interventions",
    breaks = c("Treatment_A","Treatment_B","Treatment_C","Treatment_D",
"Treatment_E","Treatment_F"
    ),
    labels = c("Treatment A", "Treatment B", "Treatment C", "Treatment D",
"Treatment E", "Treatment F"
    )
  ) + ylab("Mean Weight (in gms)")
waagglinegraph
```

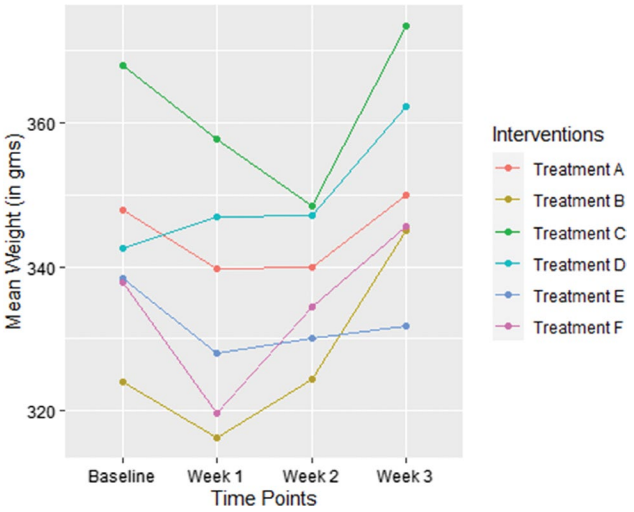

Fig. 6.26 Line graph showing the mean weight of rats at different study points in different treatment groups

We use *"geom_point"* and *"geom_line"* instead of *"geom_bar"*, and use *color* and *group* instead of *fill*. The `fill` is used to color inside the object (such as a bar). The line graph has only points & lines to be colored. Hence, we supply the *color* argument. The *group* argument is utilized to specify which set of points should be connected by lines in the graph. Theme customization can be performed for a line graph similar to that of a bar graph.

Scatter Plot (Advanced)

For the scatter plot exercise, we will try to replicate the scatter plot that we made in the basic section of the scatter plot with a slight modification. We will use the *dm* dataset and plot the scatter plot between the fasting blood glucose at baseline vs. hba1c at the end of the intervention. We should be representing the responders as well as the nonresponders in the scatter plot. In addition, we should represent the treatment groups (Fig. 6.27).

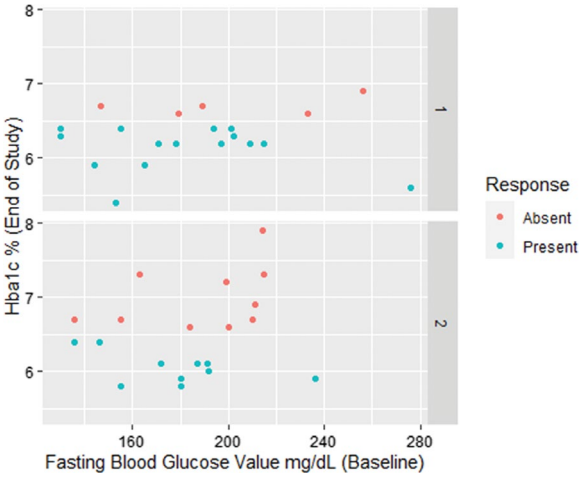

Fig. 6.27 Scatter plot showing fasting plasma glucose before treatment vs. hba1c after treatment factored based on response and faceted based on treatment

```
dm = read.csv("dm.csv")

ggplot(dm, aes(x = fasting_before, y = hba1c)) +
  geom_point(aes(color = factor(response))) +facet_grid(group)+
  scale_x_continuous(name = "Fasting Blood Glucose Value mg/dL (Baseline)") +
  scale_y_continuous(name = "Hba1c % (End of Study)") +
  scale_color_discrete(
    name = "Response",
    breaks = c(0, 1),
    labels = c("Absent", "Present")
  )
```

Joining of Multiple Figures

For publication, we would often be required to join multiple figures side-by-side. The `gridExtra` package comes in handy in this case. Our task here is to create a single figure containing two panels side-by-side, the bar graph and the line graph that we created above.

```
library(gridExtra)

## Warning: package 'gridExtra' was built under R version 4.3.1

gridExtra::grid.arrange(waaggbarplot,waagglinegraph,ncol =1)
```

The *"grid.arrange"* function takes in values of the R objects (created by earlier methods) separated by commas (Fig. 6.28). We can mention the number of columns the figure should contain using the *ncol* argument present in the *"grid.*

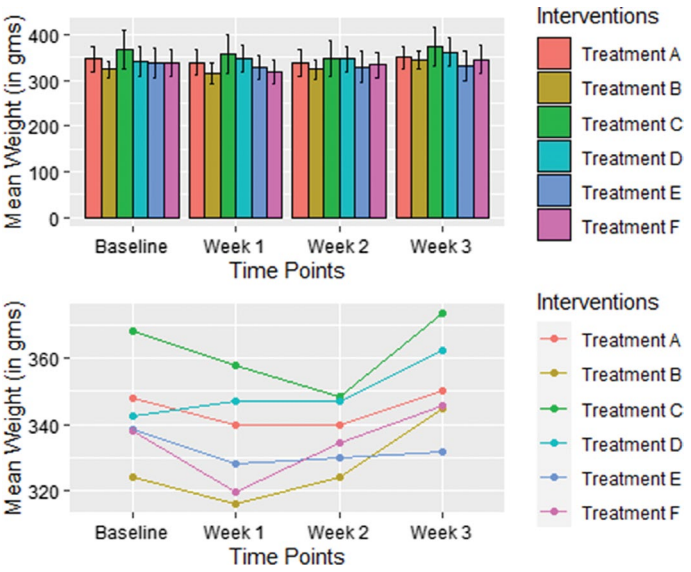

Fig. 6.28 Joining of bar plot and line graph generated in a single figure programmatically

arrange" function. Here, we pass two R objects (waaggbarplot and waagglinegraph) that we have created. And we had mentioned the `ncol` as 1, representing the number of columns as 1.

Addition of Significance Value to the Graph

Suppose we wish to add a p-value of <0.001, representing the overall difference among the interventions with time calculated by mixed-model ANOVA (calculations not shown) in the line graph created, we can use the package `ggsignif` for this task. The package also provides functions to do statistical analysis. In this exercise, we will restrict with the addition of p-value to the figure.

```
library(ggsignif)

## Warning: package 'ggsignif' was built under R version 4.2.3

waagglinegraph + geom_signif(
y_position = c(380),
xmin = c(1),
xmax = c(4),
annotation = c("p<0.001"),
tip_length = 0.02
)
```

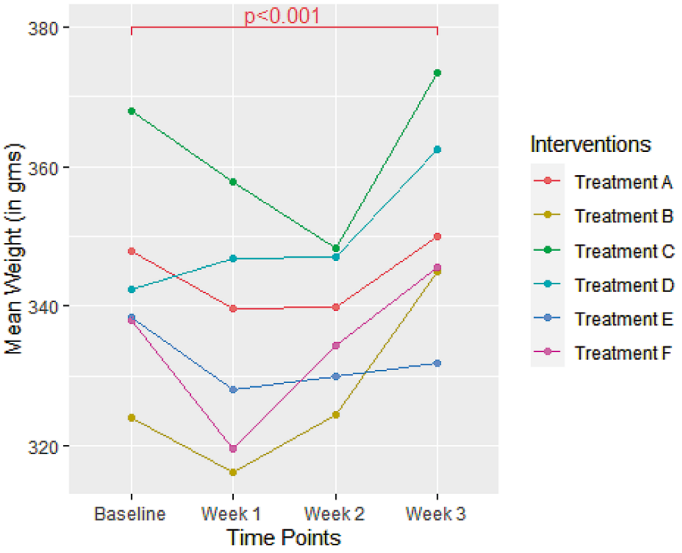

Fig. 6.29 Linegraph customization by addition of statistical significance information

The *"geom_signif"* function takes in arguments *y_position*, *xmin*, *xmax*, *annotation*, and *tip_length* (Fig. 6.29). The *y_position* stands for the y-coordinate in which the line needs to be drawn. The `xmin` and `xmax` stand for the x-axis coordinates between which the line needs to extend. The *tip_length* stands for the length of the tips at the end. The *annotation* is the string that represents the p-value.

References

1. Wickham H. ggplot2: elegant graphics for data analysis. New York, NY: Springer; 2016.
2. Wickham H. Reshaping Data with the reshape package. J Statist Softw. 2007;21(12):1–20. http://www.jstatsoft.org/v21/i12/
3. Wickham H, Bryan J. readxl: read excel files. R package version 1.3.1. 2019. https://CRAN.R--project.org/package=readxl.
4. Wickham H. The split-apply-combine strategy for data analysis. J Statist Softw. 2011;40(1):1–29. http://www.jstatsoft.org/v40/i01/
5. Auguie B. gridExtra: miscellaneous functions for "grid" graphics. R package version 2.3. 2017. https://CRAN.R-project.org/package=gridExtra
6. Ahlmann-Eltze C. ggsignif: significance brackets for 'ggplot2'. R package version 0.6.0. 2019. https://CRAN.R-project.org/package=ggsignif
7. Lawrence MA. ez: easy analysis and visualization of factorial experiments. R package version 4.4–0. 2016. https://CRAN.R-project.org/package=ez

Chapter 7
Inferential Statistics for Hypothesis Testing of Parametrically Distributed Data

Inderjeet Singh, Anand Srinivasan, and Archana Mishra

The readers would have learned about inferential statistics and hypothesis testing in "An Overview of Statistical Analysis Plan for Clinical Studies (Chap. 2)." Before applying inferential statistics, the distribution of data needs to be checked. Parametric tests can only be used when the data follows a normal distribution. Thus, the normality of data should be checked before selecting appropriate tests. A specific test can be chosen based on the research question and study design. In this chapter, we will explain various parametric tests and their implementation in example data sets.

What is Normal Distribution

It is a form of data distribution in which the data is symmetrically distributed around a mean. For a standard normal distribution, the mean is zero, and the standard deviation(SD) is one. Let us create a vector of random numbers that have a mean of zero and an SD of one and then plot it in a histogram (Fig. 7.1).

```
hist(rnorm(10,000,0,1))
```

Supplementary Information The online version contains supplementary material available at https://doi.org/10.1007/978-981-97-6980-3_7.

I. Singh
Experimental Drug Development Centre, Singapore, Singapore

A. Srinivasan · A. Mishra (✉)
Department of Pharmacology, All India Institute of Medical Sciences, Bhubaneswar, Odisha, India

A. Srinivasan et al. (eds.), *R for Basic Biostatistics in Medical Research*,
https://doi.org/10.1007/978-981-97-6980-3_7

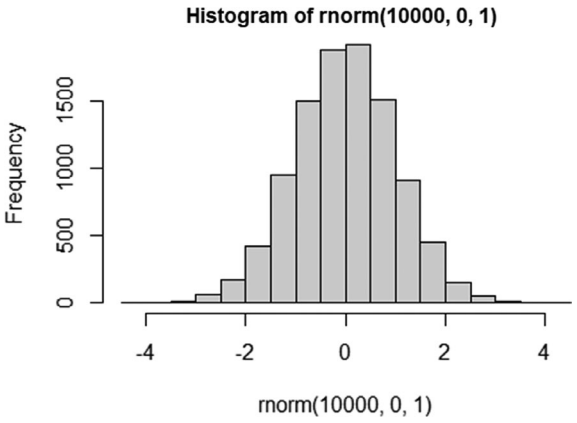

Fig. 7.1 Histogram of 10,000 random numbers with a mean of 0 and SD of 1

1. The *"hist"* function is used to create a histogram.
2. *"rnorm"* is used to generate a vector of normally distributed random numbers. In this case, 10,000 random numbers are generated with a mean of 0 and an SD of 1.

Apart from the assumption of normality, the assumption of homogeneity of variances should also be satisfied. Let us learn the parametric tests with a few worked-up examples in the case scenarios given below.

Student/Independent/Unpaired t-Test

Student t-test is applied to check whether there is any significant difference between the mean values in two unrelated or independent groups.

Assumptions Made

1. The data should be normally distributed in the comparison groups
2. There should be homogeneity of variances in the comparison groups
3. The data points should be independent of each other, i.e., one data point should not influence other data point (s)[This is a study design-based assumption]

Hypothesis

Null hypothesis: The mean difference between the groups is equal to zero.

Alternate hypothesis: The mean difference between the groups is not equal to zero.

Case Scenario 1

A clinical trial was conducted in patients with diabetes mellitus where the efficacy of two antidiabetic drugs, namely, "Drug A" in group 1 and "Drug B" in group 2, were compared for a decrease in fasting plasma glucose values and HbA1c levels at follow-up visits. The data was stored in the form of a *.csv* file and has been given as *"dm.csv"* (Table 7.1).

Here, we will first check if there was a difference in hba1c between the groups and then see if there was an improvement within the group at three months of therapy when compared to the baseline.

Step 1: Importing Data into the R Console

We will import the data file *dm.csv* into an R object *mydat*.

```
mydat <- read.csv("dm.csv").
```

Step 2: Checking for Assumptions

Checking for Normality of Data

Methods to Check the Normality of the Data

Visual Method-By plotting the data.

Plot the data in the form of a frequency distribution i.e., a histogram.

```
hist(mydat$hba1c).
```

Table 7.1 Description of variables for the "*dm.csv*" dataset

SNo	Variable	Explanation
1	group	1-Drug A, 2-Drug B
2	age	Numerical value indicating the age of the participant
3	sex	1-Male, 0-Female
4	weight	Numerical value indicating weight of the participant
5	ldl	LDL values in baseline
6	fasting_before	Numerical value indicating baseline fasting glucose
7	fasting_after	Numerical value indicating fasting glucose at 3 months
8	difference	Difference between baseline and 3 months fasting glucose
9	response	Hba1c < 6.5 at 3 months considered as responded
10	hba1c	HbA1c after 3 months of treatment

1. The "*hist*" function generates a histogram with the given data of hba1c (Fig. 7.2).

If we want to visualize the symmetry more clearly, we may calculate the kernel density estimates and plot the same.

```
d <- density(mydat$hba1c)
plot(d)
```

1. The "*density*" function calculates the kernel density.
2. The "*plot*" function is a generic function for plotting R objects (Fig. 7.3).

The second way to visually assess normality is by plotting a Q-Q plot.

Q-Q Plot

In this plot, we compare the quantiles of the true normal distribution with the quantiles of the data set under analysis (Fig. 7.4).

```
qqnorm(mydat$hba1c)
qqline(mydat$hba1c)
```

1. The "*qqnorm*" function is used to plot the given data in a sorted order versus quantiles from a standard normal distribution. The x-axis plots the theoretical quantiles, i.e., quantiles from a standard normal distribution, whereas the y-axis plots quantiles from the given data.
2. The *"qqline"* adds a line to the theoretical normal QQ plot passing through the first and third quartiles.

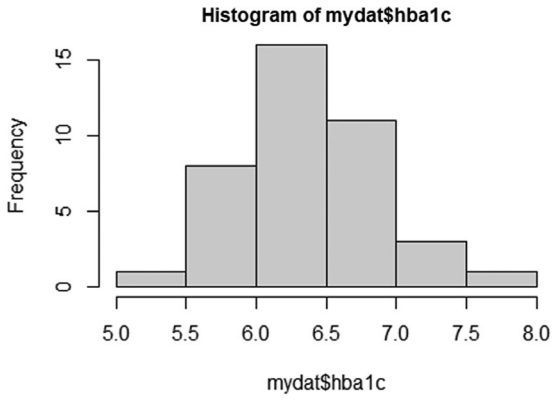

Fig. 7.2 Histogram for plotting number of patients with given hba1c values to confirm normal distribution of data

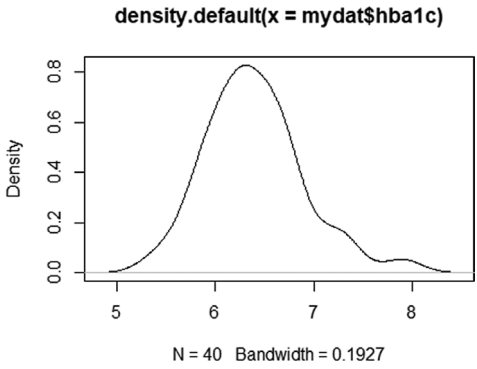

Fig. 7.3 Density distribution of HbA1c value to confirm normal distribution of data

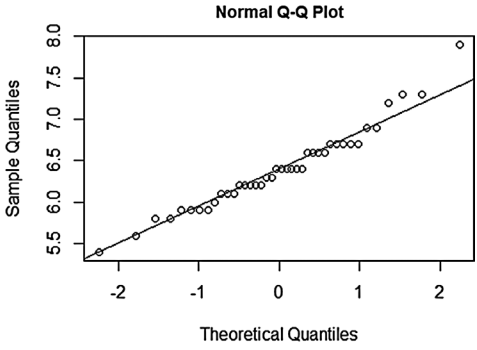

Fig. 7.4 Q-Q plot to confirm normality

3. We can appreciate the data points intertwined closely around the straight line (data points not deviating considerably from the straight line). Thus, we can assume that the data for *hba1c* is distributed normally.

Inferential Method, i.e., Statistically Testing the Normality

There are many tests for statistically testing the normality of the data; the most popularly used are the Shapiro–Wilk test and Kolmogorov–Smirnov test. We commonly use the Shapiro-Wilk test, and thus, we are discussing the same here. Shapiro-Wilk test is a hypothesis test that analyzes the data to verify if its distribution deviates significantly from the normal distribution. If $p < 0.05$,

```
shapiro.test(mydat$hba1c)

##
##  Shapiro-Wilk normality test
##
## data:  mydat$hba1c
## W = 0.96242, p-value = 0.2024

tapply(mydat$hba1c,mydat$group,shapiro.test)

## $`1`
##
##  Shapiro-Wilk normality test
##
## data:  X[[i]]
## W = 0.94121, p-value = 0.2528
##
##
## $`2`
##
##  Shapiro-Wilk normality test
##
## data:  X[[i]]
## W = 0.92917, p-value = 0.1488
```

1. The *"shapiro.test"* function applies the Shapiro-Wilk test on the data column for hba1c. However, here, we want to compare hba1c between the groups, and thus, the Shapiro-Wilk test will have to be performed on both groups separately.
2. The *"tapply"* function allows us to compute statistics for each indexing variable of a function. The arguments of the functions are *x, Index, and function*. The *function* being specified in the argument (*shapiro.test*) is applied to the parameter *x* (here hba1c) while data is split as specified in the grouping variable supplied to the *Index*.
3. We can see that the p-values for the two groups are 0.2528 and 0.1488; both the values are greater than 0.05, and thus we assume that data follows a normal distribution.

Checking for the Assumption of Homogeneity of Variance [1]

This assumption checks for data variability and how similar the variability is across the groups. Each of the comparison groups is expected to show similar variability. If the variances are not equal, we have to adjust for the unequal variance, or alternative analytical approaches may be used.

```
library(car)

## Warning: package 'car' was built under R version 4.2.3

## Loading required package: carData

## Warning: package 'carData' was built under R version 4.2.3

leveneTest(mydat$hba1c,mydat$group)

## Warning in leveneTest.default(mydat$hba1c, mydat$group): mydat$group
coerced to
## factor.

## Levene's Test for Homogeneity of Variance (center = median)
##       Df F value  Pr(>F)
## group  1  4.9479 0.03214 *
##       38
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

1. The *"leveneTest"* function is used to test for homogeneity of variances. It is a function from a package "*car*". Thus, as discussed earlier, we should install the package and load the library in the workspace prior to performing operations with this function. The arguments are the response variable(hba1c) and the factor-defining variable (group).
2. The output shows that $p < 0.05$, and thus homogeneity is not assumed. We should specify this while further testing our hypothesis.

Things to look at:

1. Did you encounter a warning??? How to interpret it. Warning is different from error. It alerts us to be aware of the information conveyed through warning at the time of interpretation

##Checking for the independence of samples: This is ensured during the planning of the study that data points are mutually exclusive.

Step 3: Accompanying Calculation

The data is normally distributed; thus, we calculate the mean and SD of the data for reporting purposes.

```
by(mydat$hba1c,mydat$group,mean)

## mydat$group: 1
## [1] 6.275
## ------------------------------------------------------------
## mydat$group: 2
## [1] 6.52

by(mydat$hba1c,mydat$group,sd)

## mydat$group: 1
## [1] 0.3668859
## ------------------------------------------------------------
## mydat$group: 2
## [1] 0.5836365
```

1. We have calculated the mean and SD of the hba1c, grouping them by the variable group using "*by"* function.
2. The mean(SD) for groups 1 and 2 are 6.275(0.367) and 6.52(0.584) respectively. Thus, we can easily calculate the mean difference of 0.245 between the groups

Step 4: Analysis of Data

The next step after testing assumptions is to estimate the probability of observing the difference that we observed if it is repeated multiple times. Before this, we should identify the test we should apply for testing differences between two independent samples. You must remember from previous chapters that since we have one variable (hba1c) and two groups and that the data follows a normal distribution, we should use the *Independent/Student's t-test*. Our null hypothesis states that the mean of the groups for HbA1c are equal.

If the probability is low, we reject the assumption that the means of the two groups are *equal* and accept the alternate assumption that the means of the two groups are *not equal.*

Independent/Student's t-Test

```
t.test(mydat[mydat$group==1,]$hba1c,mydat[mydat$group==2,]$hba1c,
       alternate="two.sided",paired=F, var.equal=F,conf.level = 0.95)
##
##  Welch Two Sample t-test
##
## data:  mydat[mydat$group == 1, ]$hba1c and mydat[mydat$group == 2, ]$hba1c
## t = -1.5894, df = 31.988, p-value = 0.1218
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -0.55899535  0.06899535
## sample estimates:
## mean of x mean of y
##     6.275     6.520
```

1. The *"t.test"* function has been used to compare the mean of hba1c between two groups.
2. Arguments for *"t.test"* are as follows.

   - *x*(hba1c) and *y*(group) are the vectors of data values that need to be compared.
   - *alternate*—The alternate assumption in terms of the sidedness of the statistical test used in hypothesis testing. It can be either "two.sided", "greater" or" less". Greater and lesser are used for one-sided hypothesis testing in superiority and non-inferiority trials.
   - *paired*—TRUE if we are performing a paired t-test. As we are performing an independent t-test here, paired should be kept FALSE.
   - *var.equal*—TRUE if the variances of the two groups are equal. As variances were not equal, we have used FALSE in the above example.
   - *conf.level*—usually 0.95 is used.

```
t.test(hba1c ~ group, data= mydat,alternate="two.sided",
paired=F,var.equal=F,conf.level = 0.95)
##
##  Welch Two Sample t-test
##
## data:  hba1c by group
## t = -1.5894, df = 31.988, p-value = 0.1218
## alternative hypothesis: true difference in means between group 1 and
group 2 is not equal to 0
## 95 percent confidence interval:
##  -0.55899535  0.06899535
## sample estimates:
## mean in group 1 mean in group 2
##           6.275           6.520
```

This is an alternate method of performing an independent t-test where we provide a formula in the form of hba1c as a function of the grouping variable. The rest of the arguments are kept the same as earlier.

Step 5:Conclusion

We had already calculated that the mean difference between the groups was 0.245. Now, applying the t-test, we get a 95% confidence interval of –0.559 to 0.0690 with a p-value of 0.1218. Thus, we can conclude that the difference was not statistically significant. We can report it as "The mean difference between the groups was 0.245 (95%CI:-0.559 to 0.0690; p = 0.1218), which was found not to be statistically significant".

Paired t-Test

The paired t-test is applied when each subject within a group has a paired measurement, i.e., before and after intervention values. The test calculates whether the mean change within the group is significant or not.

Assumptions Made

1. The difference between the paired samples should be normally distributed.
2. There should be homogeneity of variances.

Hypothesis

Null hypothesis: The mean difference between the paired samples is equal to zero.

Alternate hypothesis: The mean difference between the paired samples is not equal to zero.

Comparison of Change in Fasting Plasma Glucose Over 3 Months within the Two Groups in Case Scenario 1 Dataset

For this comparison, we need to compare pre-intervention and post-intervention results group-wise. Thus, We first separate the before and after values (here glucose values) within the groups.

Step 1: Loading the Data and Checking into the Data Structure

```
mydat  <- read.csv("dm.csv")
str(mydat)

## 'data.frame':    40 obs. of  10 variables:
##  $ group         : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ age           : int  46 36 53 63 62 65 33 46 61 38 ...
##  $ sex           : int  1 0 0 0 1 1 1 1 1 1 ...
##  $ weight        : int  101 97 119 48 88 58 40 71 41 62 ...
##  $ ldl           : int  104 100 81 86 75 115 146 132 138 141 ...
##  $ fasting_before: int  155 153 130 147 130 171 197 201 194 209 ...
##  $ fasting_after : int  111 133 100 126 101 146 147 125 192 113 ...
##  $ difference    : int  44 20 30 21 29 25 50 76 2 96 ...
##  $ response      : int  1 1 1 0 1 1 1 1 1 1 ...
##  $ hba1c         : num  6.4 5.4 6.3 6.7 6.4 6.2 6.2 6.4 6.4 6.2 ...
```

Step 2: Converting the Grouping Variable into Type "Factor"

```
####converting to factors###
mydat$group <- as.factor(mydat$group)
```

The *"as.factor"* function converts the grouping variable into a factorial variable.

Step 3: Separation of Pre and Post Intervention Data Group Wise

```
grp1fpgafter<- mydat$fasting_after[which(mydat$group==1)]
grp2fpgafter<- mydat$fasting_after[which(mydat$group==2)]
grp1fpgbefore<- mydat$fasting_before[which(mydat$group==1)]
grp2fpgbefore<- mydat$fasting_before[which(mydat$group==2)]
```

1. In the above codes, group 1 and group 2 values for before and after fasting glucose levels are separated.
2. "[]" allows to subset the fasting values, and "*which"* function enables indexing the value satisfying the condition that follows after "*which"*.

Step 4: Checking for Assumption

Normality and homogeneity of variances needs to be checked for the difference between pre-intervention and post-intervention data groupwise. We will avoid repeating our earlier discussions on normality and visualization of data.

```
by(mydat$difference,mydat$group,shapiro.test)

## mydat$group: 1
##
##  Shapiro-Wilk normality test
##
## data:  dd[x, ]
## W = 0.95938, p-value = 0.5315
##
## ------------------------------------------------------------
## mydat$group: 2
##
##  Shapiro-Wilk normality test
##
## data:  dd[x, ]
## W = 0.95648, p-value = 0.4762

leveneTest(mydat$difference, mydat$group)

## Levene's Test for Homogeneity of Variance (center = median)
##       Df F value  Pr(>F)
## group  1  3.6293 0.06436 .
##       38
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Step 5: Analysis of Data

```
t.test(grp1fpgbefore,grp1fpgafter,paired=TRUE, alternate="two.sided",
       var.equal=T,conf.level = 0.95)
##
##  Paired t-test
##
## data:  grp1fpgbefore and grp1fpgafter
## t = 6.0383, df = 19, p-value = 8.279e-06
## alternative hypothesis: true mean difference is not equal to 0
## 95 percent confidence interval:
##  26.46162 54.53838
## sample estimates:
## mean difference
##           40.5
```

```
t.test(grp2fpgbefore,grp2fpgafter,paired=TRUE, alternate="two.sided",
       var.equal=T,conf.level = 0.95)
##
##  Paired t-test
##
## data:  grp2fpgbefore and grp2fpgafter
## t = 7.4512, df = 19, p-value = 4.736e-07
## alternative hypothesis: true mean difference is not equal to 0
## 95 percent confidence interval:
##  24.44951 43.55049
## sample estimates:
## mean difference
##              34
```

1. The function and the arguments for independent (unpaired) and paired sample t-tests are the same, i.e., "*t.test"*, except that we have to mark paired = TRUE for the paired t-test.
2. Bothered by 8.279e-06? It can be simply put as 8.279 * 10^-6, which is 0.000008279.

Step 6: Conclusion

The mean change from baseline to three months follow-up was 40.5 (95%CI: 26.462 to 54.538; $p < 0.001$) in Group 1(Drug A), whereas the mean change was 34 (95%CI: 24.450 to 43.550; $p < 0.001$) for Group 2(Drug B). The change in both groups was statistically significant.

Analysis of Variance (ANOVA)

One-Way ANOVA

This is used to test whether there is a significant difference between the mean values of more than two groups.

Assumptions Made

1. The data should be normally distributed in the comparison groups.
2. There should be homogeneity of variances in the comparison groups.
3. The data points should be independent of each other i.e., one data point should not influence other data point(s).

Case Scenario 2

A clinical trial was conducted to test the efficacy of three doses (0.6 g, 1.2 g, 1.8 g) of "Drug L" and placebo on patients with diabetes mellitus. Patients were screened, recruited, and randomized into four groups, and baseline data was collected. They were called for follow-up at 12 weeks and 24 weeks for fasting glucose and HbA1c. The data for the trial has been saved as *DMdat.csv,* and the variables were as follows (Table 7.2):

In order to determine which of the following treatments causes an improvement in HbA1c scores, we need to test the change in HbA1c and fasting glucose across the groups. Let us start with the change in Hba1c at 24 weeks i.e., hcfb.

We can recall that comparing means of more than two groups [i.e., Variable = 1 timepoint and Groups = 4]; thus, one-way analysis of variance (one-way ANOVA) should be used.

Table 7.2 Description of variables for the "DMdat.csv" dataset

SNo	Variable	Explanation
1	Age	Numerical value indicating the age of the participant
2	Gender	1-Male, 0-Female
3	BMI	Numerical value indicating body mass index
4	Wt	Numerical value indicating the weight of the participant
5	Hb0	Haemoglobin values in the baseline
6	Treat	Four treatments groups: L0.6,L1.2,L1.8,P
7	HbA1c0	Numerical value indicating HbA1c at baseline
8	FPG0	Numerical value indicating baseline fasting glucose
9	HbA1c12	Numerical value indicating HbA1c at 12 weeks
10	HbA1c24	Numerical value indicating HbA1c at 24 weeks
11	FPG12	Numerical value indicating fasting glucose at 12 weeks
12	priorRx	Prior treatment, whether Insulin or Non-insulin
13	hcfb	Change in HbA1c at follow-up
14	fcfb	Change in Fasting glucose at follow-up

Hypothesis

Null

There is no difference in change in mean HbA1c amongst the groups.

Alternate

There is a difference in change in mean HbA1c amongst the groups.

Step 1: Loading and Visualizing the Data

```
DMdat<- read.csv("DMdat.csv")
str (DMdat)

## 'data.frame':    400 obs. of  15 variables:
##  $ SID    : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ Age    : int  51 60 64 53 36 40 34 36 55 62 ...
##  $ Gender : int  0 0 1 1 0 1 1 1 1 1 ...
##  $ BMI    : int  24 24 24 25 34 25 33 27 28 31 ...
##  $ Wt     : int  81 54 67 54 87 63 91 67 78 67 ...
##  $ Hb0    : num  18 14.3 15.3 7.1 17.5 13.1 13.4 13.4 14.8 14.6 ...
##  $ Treat  : chr  "L1.2" "L1.8" "L0.6" "L1.2" ...
##  $ HbA1c0 : num  9 10.4 10 9 9.5 9.2 9.2 10.1 8.3 9.4 ...
##  $ FPG0   : int  211 256 243 211 227 217 217 246 189 224 ...
##  $ HbA1c12: num  7.1 7.5 9.2 7.7 9.2 9.8 7.8 9.8 7.1 9.2 ...
##  $ HbA1c24: num  6.71 9.3 8.8 7.31 8.8 ...
##  $ FPG12  : int  160 205 233 161 215 211 180 230 137 188 ...
##  $ priorRx: chr  "Insulin" "Insulin" "Insulin" "Insulin" ...
##  $ hcfb   : num  1.9 2.9 0.8 1.3 0.3 -0.6 1.4 0.3 1.2 0.2 ...
##  $ fcfb   : int  51 51 10 50 12 6 37 16 52 36 ...
```

Step 2: Converting Grouping Variable to Factor

```
DMdat$Treat <- as.factor(DMdat$Treat)
```

Step 3: Checking Assumptions for Employing Parametric Tests

Checking Normality and Homogeneity of Variances

```
by(DMdat$hcfb,DMdat$Treat,shapiro.test)

## DMdat$Treat: L0.6
##
##  Shapiro-Wilk normality test
##
## data:  dd[x, ]
## W = 0.98598, p-value = 0.3455
##
## ------------------------------------------------------------
## DMdat$Treat: L1.2
##
##  Shapiro-Wilk normality test
##
## data:  dd[x, ]
## W = 0.98557, p-value = 0.2513
##
## ------------------------------------------------------------
## DMdat$Treat: L1.8
##
##  Shapiro-Wilk normality test
##
## data:  dd[x, ]
## W = 0.97619, p-value = 0.08376
##
## ------------------------------------------------------------
## DMdat$Treat: P
##
##  Shapiro-Wilk normality test
##
## data:  dd[x, ]
## W = 0.97519, p-value = 0.09647

leveneTest(DMdat$hcfb, DMdat$Treat)

## Levene's Test for Homogeneity of Variance (center = median)
##        Df F value Pr(>F)
## group   3  1.6902 0.1685
##       396
```

We are sure our readers are well equipped to understand the output for normality and homogeneity of variances. We can appreciate that HbA1c data at baseline is normally distributed ($p > 0.05$ for Shapiro-Wilk), and the variances are equal($p > 0.05$ for the Levene test). Now, we will move ahead with the analysis.

The function we use for ANOVA is "*anova_test*". This is a function from package *rstatix*. Prior to starting the analysis, installation of *rstatix*, *tidyverse,* and *ggpubr* packages is required [2–4]. We should initiate installing packages and calling the libraries.

```
install.packages(c("rstatix", "tidyverse", "ggpubr"))
library(rstatix)

## Warning: package 'rstatix' was built under R version 4.2.3

##
## Attaching package: 'rstatix'

## The following object is masked from 'package:stats':
##
##     filter

library(tidyverse)
## Warning: package 'tidyverse' was built under R version 4.2.3
## Warning: package 'ggplot2' was built under R version 4.2.3
## Warning: package 'tibble' was built under R version 4.2.3
## Warning: package 'tidyr' was built under R version 4.2.3
## Warning: package 'readr' was built under R version 4.2.3
## Warning: package 'purrr' was built under R version 4.2.3
## Warning: package 'dplyr' was built under R version 4.2.3
## Warning: package 'stringr' was built under R version 4.3.1
## Warning: package 'forcats' was built under R version 4.3.1
## Warning: package 'lubridate' was built under R version 4.2.3
## ── Attaching core tidyverse packages ──────────────────────
tidyverse 2.0.0 ──
## ✓ dplyr     1.1.3     ✓ readr     2.1.4
## ✓ forcats   1.0.0     ✓ stringr   1.5.0
## ✓ ggplot2   3.4.3     ✓ tibble    3.2.1
## ✓ lubridate 1.9.2     ✓ tidyr     1.3.0
## ✓ purrr     1.0.1

## ── Conflicts ──────────────────────────────────────── tidyverse_
conflicts() ──
## ✓ dplyr::filter()  masks rstatix::filter(), stats::filter()
## ✓ dplyr::lag()     masks stats::lag()
## ✓ dplyr::recode() masks car::recode()
## ✓ purrr::some()    masks car::some()
## i Use the conflicted package (http://conflicted.r-lib.org/) to force all
conflicts to become errors

library (ggpubr)

## Warning: package 'ggpubr' was built under R version 4.2.3
```

Let us quickly view the group-wise HbA1c values at baseline.

```
plot(DMdat$hcfb ~ DMdat$Treat)
```

The "*plot*" function will plot change in HbA1c at 24 weeks from baseline for all the treatment groups (Fig. 7.5).

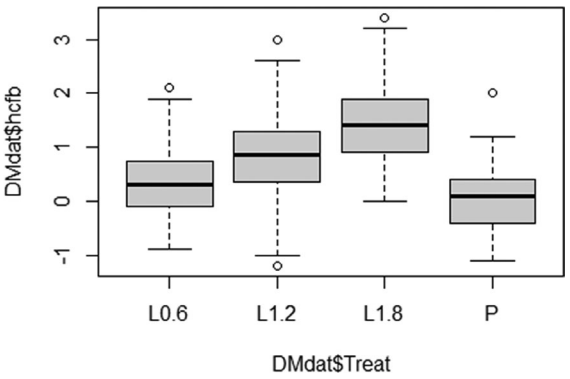

Fig. 7.5 Boxplot to demonstrate the change in HbA1c in the four groups

Step 4: Calculation of Summary Statistics

```
DMdat %>%
  group_by(Treat) %>%
  get_summary_stats(hcfb, type = "mean_sd")

## # A tibble: 4 × 5
##   Treat variable     n  mean    sd
##   <fct> <fct>    <dbl> <dbl> <dbl>
## 1 L0.6  hcfb       104 0.335 0.608
## 2 L1.2  hcfb       116 0.801 0.729
## 3 L1.8  hcfb        94 1.44  0.735
## 4 P     hcfb        86 0.031 0.617
```

1. *Introducing pipe operator*: The pipe operator is a special operational function available under the dplyr package, which allows us to pass the result of one function/argument to the other one in sequence. It is generally denoted by the symbol %>%.
2. The "*group_by"* function groups the HbA1c0 values according to the treatment groups.
3. The "*get_summary_stats"* will create summary statistics for the grouped variable, and the summary statistics needed here are *mean* and *sd*.

Step 5: Analysis of Data

```
DMdat %>% anova_test(hcfb~Treat)

## ANOVA Table (type II tests)
##
##   Effect DFn DFd      F        p p<.05   ges
## 1  Treat   3 396 75.929 8.15e-39     * 0.365
```

1. The "*anova_test*" function is used to perform ANOVA. The arguments of the ANOVA include a formula to analyze *hcfb* as a function of treatment groups. The formula is represented using ~ to create a model between the outcome variable and grouping variable.
2. Visualizing the outcome, $p < 0.001$, which implies that there is a difference between any of the comparisons. The next step is to know which mean is different from the other.

Step 6: Post-hoc Analysis

To know and understand the comparisons between the means, we perform post hoc tests. There are a number of post-hoc tests like Bonferroni, Tukey HSD, Scheffe's, Dunnett's, etc., which differ in their application and p-adjustment methods.

Let us use "*TukeyHSD*" here for the post hoc comparisons.

```
pwc<- DMdat%>%tukey_hsd(hcfb~Treat)
pwc

## # A tibble: 6 × 9
##   term  group1 group2 null.value estimate conf.low conf.high    p.adj
## * <chr> <chr>  <chr>       <dbl>    <dbl>    <dbl>     <dbl>    <dbl>
## 1 Treat L0.6   L1.2            0    0.466    0.230    0.702  3.20e- 6
## 2 Treat L0.6   L1.8            0    1.11     0.857    1.35   0
## 3 Treat L0.6   P               0   -0.303   -0.558   -0.0485 1.22e- 2
## 4 Treat L1.2   L1.8            0    0.640    0.397    0.882  2.09e-10
## 5 Treat L1.2   P               0   -0.769   -1.02    -0.521  0
## 6 Treat L1.8   P               0   -1.41    -1.67    -1.15   0
## # i 1 more variable: p.adj.signif <chr>
```

1. The "*tukey_hsd*" function runs a test on studentized distribution to assess the significance of the difference between the groups.
2. We can appreciate from the output that all the comparisons show significant differences.
3. It is noteworthy that if the p-value for *anova_test* > 0.05, we could conclude that there is no significant difference across the groups, and post-hoc analysis would not be needed.

Step 7: Conclusion

The mean and SD for L0.6, L1.2, L1.8, and Placebo groups were 0.335(0.608), 0.801 (0.729), 1.44 (0.735), and 0.031 (0.617), respectively. $F(3,396) = 75.93$; $p < 0.001$, which implies that there lies some difference somewhere, which needs to be identified. A post-hoc analysis shows that there were differences across all the comparisons between treatment groups.

Two-way ANOVA

A two-way ANOVA is used when we want to check whether the combination of two independent variables has a significant effect on the mean of a dependent variable across more than two groups.

Assumptions Made

1. The value of the dependent variable should be normally distributed in all the groups.
2. Homogeneity of variances around the mean should be present.
3. The independence of samples is ensured by the study's design.

In the case scenario 2 data, we may hypothesize that there might be some interaction between prior treatment with insulin and the current treatment the patient has received, which might affect the HbA1c changes. As two factors are involved, we will need to perform a *two-way ANOVA* to explore this.

Here we are considering prior treatment with insulin (yes or no) as the second factor and test whether HbA1c change (hcfb) is impacted by any prior history of insulin use. As the data is already loaded and the normality of data has been ensured, we can proceed to further steps.

Hypothesis

Null

1. There is no difference in means of change in HbA1c at any level of the treatment group.
2. There is no difference in means of change in HbA1c at any level of the prior treatment with insulin.
3. The effect of the treatment group does not depend on the effect of the prior treatment with insulin (a.k.a. no interaction effect).

Alternate

1. There is a difference in change in HbA1c by treatment group.
2. There is a difference in change in HbA1c by prior treatment with insulin.
3. There is an interaction effect between the treatment group and prior treatment with insulin.

Step 1: Explore the Data

```
class(DMdat$priorRx)

## [1] "character"

DMdat$priorRx <- as.factor(DMdat$priorRx)
```

1. We are checking the class of the variable *priorRx* and converting it to factorial variable with 2 levels.

```
boxplot(DMdat$hcfb ~ DMdat$Treat:DMdat$priorRx)
```

We can visualize the interaction graphically by plotting boxplots (Fig. 7.6).

Step 2: Analyse the Data (Two-Way ANOVA)

```
DMdat %>% anova_test(hcfb ~ Treat * priorRx)

## ANOVA Table (type II tests)
##
##           Effect DFn DFd      F        p p<.05   ges
## 1         Treat   3 392 77.014 3.50e-39     * 0.371
## 2       priorRx   1 392  1.986 1.60e-01       0.005
## 3 Treat:priorRx   3 392  1.724 1.62e-01       0.013
```

1. All the types of ANOVA tests use the function *"anova_test"* and the arguments may vary depending on the type of ANOVA.
2. In this case, we have two grouping variables, namely *Treat* and *priorRx,* and we wish to show the interaction effect. Thus, they have been combined using a multiplication operator.

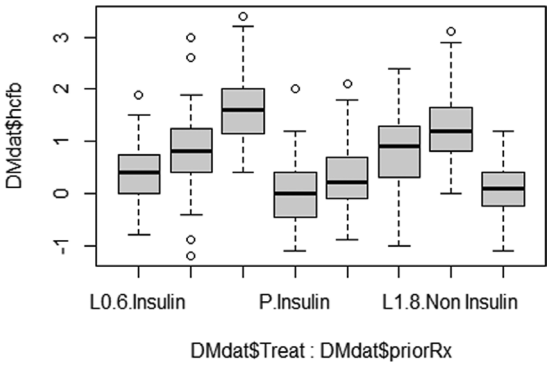

Fig. 7.6 Boxplot for interaction effect between treatment group and prior treatment with insulin

3. We can appreciate that there is a significant effect of treatment groups but no difference with prior treatment with insulin (p = 0.16) as well as the interaction between treatment group and prior treatment with insulin (p = 0.162).

Repeated Measures ANOVA

We already know that when we have the same measure (paired sample) from a participant on two occasions, we perform a paired t-test; however, if we have measures within a group at more than two-time points, repeated measures ANOVA is used.

Assumptions

1. The continuous dependent variable should be normally distributed.
2. The categorical independent variable should have three or more levels.
3. No outliers should be present in any of the repeated measurements.
4. Assumption of sphericity (constant variance across the time points).

Let us assume the scenario where we want to observe if there is any significant change in HbA1c values over three or more time points in only one group, i.e., "L1.8" (In the study, HbA1c was estimated at baseline, 12 weeks, and 24 weeks for all the groups). In this case, we have to perform *repeated measure ANOVA* to test the significant changes of the related values(HbA1c) at more than two-time points.

Hypothesis

Null

There is no difference in mean HbA1c levels at the three-time points (for the "L1.8" group).

Alternate

There is a difference in mean HbA1c levels at the three-time points (for the "L1.8" group).

Step 1: Loading the Dataset

```
data1 <- read.csv("DMdat.csv")
data <- data1[which(data1$Treat=="L1.8"),]
str(data)

## 'data.frame':    94 obs. of  15 variables:
##  $ SID    : int  2 9 12 23 24 26 29 42 46 52 ...
##  $ Age    : int  60 55 50 47 45 63 69 62 64 56 ...
##  $ Gender : int  0 1 1 1 1 0 0 1 0 1 ...
##  $ BMI    : int  24 28 26 21 28 30 33 30 28 23 ...
##  $ Wt     : int  54 78 56 75 78 74 85 76 65 62 ...
##  $ Hb0    : num  14.3 14.8 11.5 15.5 14.4 16.8 11.5 12.5 14.5 17.8 ...
##  $ Treat  : chr  "L1.8" "L1.8" "L1.8" "L1.8" ...
##  $ HbA1c0 : num  10.4 8.3 9.8 9.8 9.4 8.6 10.2 10.2 8.2 9.8 ...
##  $ FPG0   : int  256 189 173 237 224 198 249 249 185 173 ...
##  $ HbA1c12: num  7.5 7.1 7.2 8.6 7.8 6.6 8.2 8.5 7.2 6.4 ...
##  $ HbA1c24: num  9.3 6.72 6.8 8.2 8.21 ...
##  $ FPG12  : int  205 137 128 191 178 120 209 200 127 128 ...
##  $ priorRx: chr  "Insulin" "Insulin" "Insulin" "Insulin" ...
##  $ hcfb   : num  2.9 1.2 2.6 1.2 1.6 2 2 1.7 1 3.4 ...
##  $ fcfb   : int  51 52 45 46 46 78 40 49 58 45 ...
```

1. We have imported the data into our working space using *"read.csv"*.
2. We have subsetted the data for the "L1.8" group as we want to evaluate the change in HbA1c in this group over time.
3. We can visualize that data has four columns viz., ID, and three values of HbA1c at baseline, 12 weeks, and 24 weeks. This is known as the *wide format* for dataframe.
4. Next, we need to check the structure of the data.

Step 2: Converting Data from Wide to Long Format

```
data <- data %>% gather(key = "time", value = "score", HbA1c0, HbA1c12,
HbA1c24)                          %>% convert_as_factor(SID,time)
```

1. A *"gather()"* function is used for collecting (gather) multiple columns and converting them into a key-value pair, which is the time point and score of HbA1c.
2. We can visualize this change in format by typing the name of the R object assigned, i.e., *data* in this case, in the console and running it.
3. We can also confirm the format change by randomly sampling five observations per time point. We will subset the data to show only four variables, i.e., SID, Age, time, and score, to avoid unnecessary confusion.

```
data[,c(1,2,13,14)] %>% sample_n_by(time, size = 5)

## # A tibble: 15 × 4
##    SID     Age time    score
##    <fct> <int> <fct>   <dbl>
##  1 199      63 HbA1c0   8.7
##  2 390      38 HbA1c0   9.4
##  3 26       63 HbA1c0   8.6
##  4 149      48 HbA1c0   8.3
##  5 388      55 HbA1c0   9.9
##  6 192      66 HbA1c12  8.1
##  7 297      52 HbA1c12  6.4
##  8 117      47 HbA1c12  8.2
##  9 262      66 HbA1c12  6.4
## 10 140      43 HbA1c12  8
## 11 70       60 HbA1c24  6
## 12 64       71 HbA1c24  9.8
## 13 362      55 HbA1c24  7.12
## 14 192      66 HbA1c24  7.71
## 15 194      63 HbA1c24  7.01
```

4. We can see that five observations per timepoint, i.e., a total of fifteen observations, are shown in the output. The key to HbA1c values has been represented as HbA1c0, HbA1c12, and HbA1c24 and clubbed together in one column, “time”, and the corresponding HbA1c values are mentioned in column “score”

Step 3: Calculating Summary Statistics

```
data %>%
  group_by(time) %>%
  get_summary_stats(score,type="mean_sd")

## # A tibble: 3 × 5
##   time    variable     n  mean    sd
##   <fct>   <fct>    <dbl> <dbl> <dbl>
## 1 HbA1c0  score       94  9.08 0.662
## 2 HbA1c12 score       94  7.64 0.909
## 3 HbA1c24 score       94  7.52 1.10
```

1. This function calculates summary stats while the data is grouped by time points.

Step 4: Visualising the Data in Plots

```
bxp <- ggboxplot(data, x = "time", y = "score", add = "point")
bxp
```

1. We have plotted the boxplot for HbA1c data at each time point; individual data points have been plotted by adding the argument *add* to the *ggboxplot* function (Fig. 7.7).

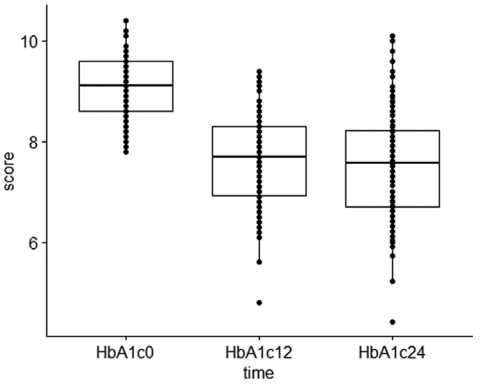

Fig. 7.7 Boxplot for representing HbA1c values at different time-points

Step 5: Checking Assumptions

```
data %>%
  group_by(time) %>%
  identify_outliers(score)

## # A tibble: 2 × 16
##   time    SID     Age Gender   BMI    Wt   Hb0 Treat  FPG0 FPG12
priorRx  hcfb
##   <fct>   <fct> <int>  <int> <int> <int> <dbl> <chr> <int> <int>
<chr>   <dbl>
## 1 HbA1c12 158      56      1    33    79   8.8 L1.8    179   143
Insulin   3.2
## 2 HbA1c24 158      56      1    33    79   8.8 L1.8    179   143
Insulin   3.2
## # i 4 more variables: fcfb <int>, score <dbl>, is.outlier <lgl>,
## #   is.extreme <lgl>
```

1. We can appreciate that there are no extreme outliers (which we need to remove) for the given data. Thus, we can move ahead with our analysis.

```
#Checking normality by Shapiro test
data %>%
  group_by(time) %>%
  shapiro_test(score)

## # A tibble: 3 × 4
##   time    variable statistic      p
##   <fct>   <chr>        <dbl>  <dbl>
## 1 HbA1c0  score        0.975 0.0677
## 2 HbA1c12 score        0.984 0.319
## 3 HbA1c24 score        0.991 0.806

#Checking normality by Q-Q plot
ggqqplot(data, "score", facet.by = "time")
```

1. We performed the Shapiro-Wilk test and generated a Q-Q plot to check the normality of the data (Fig. 7.8). As the p-values are >0.05, the data was assumed to be normally distributed.
2. It should be noted that the function *"anova_test"*, which will be discussed in the next section, will correct itself for sphericity if required, and there is no need to look for sphericity explicitly.

Step 6: Analysis of the Data: Repeated Measures ANOVA

```
rp.anova <- anova_test(data=data, dv=score, wid=SID, within = time)
get_anova_table(rp.anova)

## ANOVA Table (type III tests)
##
##    Effect DFn DFd       F        p p<.05   ges
## 1    time   2 186 233.894 1.69e-51     * 0.381
```

1. The *"anova_test"* function is used to generate an ANOVA table comparing the data across more than two-time points.
2. Arguments of the *"anova_test"* for repeated measures have been defined as *data*, dependent continuous variable *dv*, column specifying subject identifiers *wid*, within-subject factors *within*.
3. We can see that p is 1.69 *10^-51, i.e., <0.001. Thus, we can conclude that there is a significant difference in HbA1c values between any of the two-time points. Once it is understood there is a significant difference in HbA1c levels across the time points, post-hoc analysis is done to locate which pairs of means are different from each other.

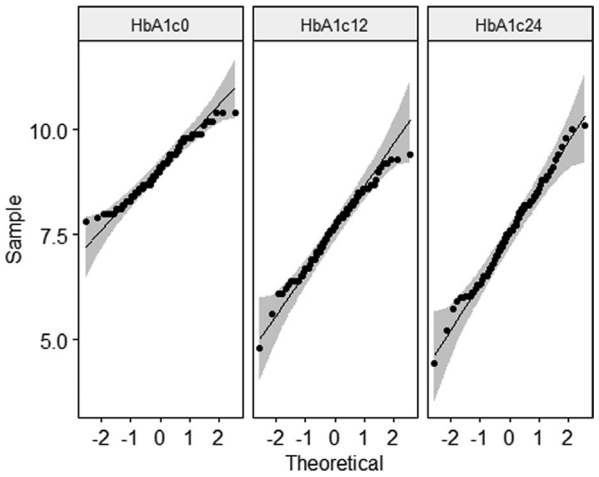

Fig. 7.8 Q-Q plot to confirm the normal distribution of data at each time-point

Step 7: Post-hoc Analysis

```
pwc <- data %>% pairwise_t_test (score~time, paired = T,
                                 p.adjust.method = "bonferroni")
pwc

## # A tibble: 3 × 10
##   .y.   group1 group2    n1    n2 statistic    df        p    p.adj
p.adj.signif
## * <chr> <chr>  <chr>  <int> <int>     <dbl> <dbl>    <dbl>    <dbl> <chr>
## 1 score HbA1c0 HbA1c…    94    94      19.0     93 8.73e-34 2.62e-33 ****
## 2 score HbA1c0 HbA1c…    94    94      17.9     93 5.93e-32 1.78e-31 ****
## 3 score HbA1c… HbA1c…    94    94       1.52    93 1.32e- 1 3.96e- 1 ns
```

1. The function *"pairwise_t_test"* carries out as many t-tests as the number of comparisons, and for adjustment of p-value, the argument *p.adjust.method* is used. Here, we have used the "bonferroni" method for post-hoc analysis.
2. The output shows that there is a statistically significant difference between HbA1c values in all the time points.

Step 8: Conclusion

A repeated measures ANOVA shows that mean HbA1c values are significantly different across the three time points: $F(2,186) = 233.89$, $p < 0.001$. The mean (SD) at baseline, week 12, and week 24 were 9.08(0.662), 7.64(0.909) and 7.52 (1.10), respectively. On post-hoc analysis, the mean HbA1c values at 12 weeks and 24 weeks were significantly different from baseline. The change from week 12 to week 24 was not significant.

Step 9: Presenting Results in Plot Form

```
bxp<- ggboxplot(data, x = "time", y="score", add="point")
```

```
pwc <- pwc %>% add_xy_position(x="time")
bxp +
  stat_pvalue_manual(pwc)+
  labs (subtitle = get_test_label(rp.anova, detailed=T),
        caption = get_pwc_label(pwc)
        )
```

The boxplot created can visually appreciate the change in HbA1c levels at different time points, with the significance marked between the time points (Fig. 7.9).

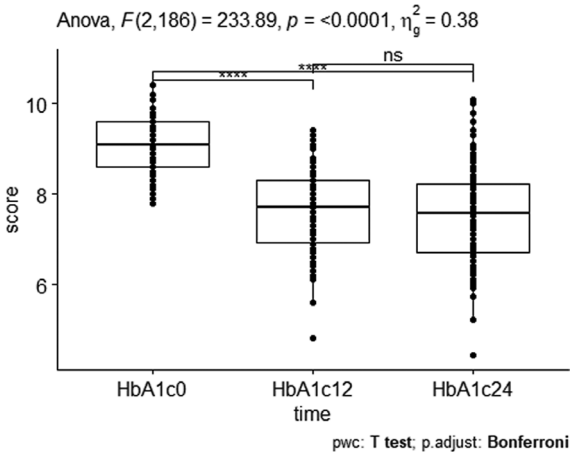

Fig. 7.9 Representation of the results in plot format with detailed caption for statistical inference

Mixed-Model ANOVA (Two-Way Repeated Measures)

Assumptions

1. The dependent variable should be measured on a continuous scale and normally distributed.
2. The within-participant variables should consist of at least three categorical values, i.e., "related groups" or "matched pairs".
3. Between participant factors should consist of at least two categorical values, "independent groups".
4. There should not be any significant outliers.
5. There should be homogeneity of variances across the differences between the groups.
6. Assumption of sphericity (constant variance across the time points).

Coming back to visualizing the entire dataset of *DMdat.csv*, we can appreciate that HbA1c and Fasting plasma glucose values have been collected for participants in all four groups over three time points. Thus, in these cases, to understand the change in HbA1c and fasting glucose levels across groups over two or more time points, we need to carry out two-way repeated measures ANOVA.

Since it might be a bit complicated to understand the null and alternative hypotheses for mixed-model ANOVA, it is easier to understand that we assume there is some interaction effect between the group and time variables, which needs to be explored and calculated through analysis.

Let us visualize the data to look at the changes in HbA1c over a period of time and between the groups.

Step 1. Loading and Visualizing Data

```
set.seed(123)
data2 <- read.csv("DMdat.csv")
str(data2)

## 'data.frame':    400 obs. of  15 variables:
##  $ SID    : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ Age    : int  51 60 64 53 36 40 34 36 55 62 ...
##  $ Gender : int  0 0 1 1 0 1 1 1 1 1 ...
##  $ BMI    : int  24 24 24 25 34 25 33 27 28 31 ...
##  $ Wt     : int  81 54 67 54 87 63 91 67 78 67 ...
##  $ Hb0    : num  18 14.3 15.3 7.1 17.5 13.1 13.4 13.4 14.8 14.6 ...
##  $ Treat  : chr  "L1.2" "L1.8" "L0.6" "L1.2" ...
##  $ HbA1c0 : num  9 10.4 10 9 9.5 9.2 9.2 10.1 8.3 9.4 ...
##  $ FPG0   : int  211 256 243 211 227 217 217 246 189 224 ...
##  $ HbA1c12: num  7.1 7.5 9.2 7.7 9.2 9.8 7.8 9.8 7.1 9.2 ...
##  $ HbA1c24: num  6.71 9.3 8.8 7.31 8.8 ...
##  $ FPG12  : int  160 205 233 161 215 211 180 230 137 188 ...
##  $ priorRx: chr  "Insulin" "Insulin" "Insulin" "Insulin" ...
##  $ hcfb   : num  1.9 2.9 0.8 1.3 0.3 -0.6 1.4 0.3 1.2 0.2 ...
##  $ fcfb   : int  51 51 10 50 12 6 37 16 52 36 ...

data2$Treat <- as.factor(data2$Treat)
```

1. The *"set.seed()"* function in R is used to create reproducible results when writing code that involves creating variables that take on random values.
2. We checked the structure of our data frame and converted the *Treat* vector to a factorial variable

Step 2: Preparing Data into Long Format

```
data2 <- data2 %>% gather(key = "Time", value = "HbA1c", HbA1c0, HbA1c12,
HbA1c24) %>%
  convert_as_factor(SID,Time)
```

1. As the repeated measure ANOVA function works with a long data format rather than a wide data format, we gathered the data in a long format and converted patient identifier and time point variables into factorial variables

Confirming the Long Format Conversion

```
data2 %>% sample_n_by(Treat, Time, size = 1)

## # A tibble: 12 × 14
##    SID     Age Gender   BMI    Wt   Hb0 Treat  FPG0 FPG12 priorRx
hcfb  fcfb
##    <fct> <int>  <int> <int> <int> <dbl> <fct> <int> <int> <chr>
<dbl> <int>
##  1 116      48      1    29    65  11.4 L0.6    176   165 Insulin
0.7    11
##  2 288      51      1    26    79  16.7 L0.6    176   163 Non Insul…
-0.1   13
##  3 169      48      1    24    62  14.3 L0.6    230   220 Insulin
0.3    10
##  4 51       43      0    28    78  12.4 L1.2    176   149 Insulin
1.3    27
##  5 241      61      1    29    77  16.5 L1.2    227   199 Non Insul…
0.6    28
##  6 162      38      1    26    78  14.6 L1.2    214   171 Insulin
0.9    43
##  7 213      46      1    20    98  16.6 L1.8    182   149 Non Insul…
1.2    33
##  8 192      66      1    27    78  13.2 L1.8    198   158 Insulin
0.5    40
##  9 61       68      1    20    65  15.2 L1.8    176   111 Insulin
1.8    65
## 10 113      31      1    32    93  10.5 P       227   237 Insulin
-1.1  -10
## 11 332      59      1    22    78  12.2 P       227   217 Non Insul…
0.9    10
## 12 266      52      1    25    81  17.4 P       240   230 Non Insul…
-0.9   10
## # i 2 more variables: Time <fct>, HbA1c <dbl>
```

We get to see one random row for each Treatment group and time point.

Step 3: Calculating Summary Statistics

```
data2 %>%
  group_by(Time, Treat) %>%
  get_summary_stats(HbA1c, type = "mean_sd")

## # A tibble: 12 × 6
##    Treat Time    variable     n  mean    sd
##    <fct> <fct>   <fct>    <dbl> <dbl> <dbl>
##  1 L0.6  HbA1c0  HbA1c      104  9.13 0.732
##  2 L1.2  HbA1c0  HbA1c      116  9.11 0.704
##  3 L1.8  HbA1c0  HbA1c       94  9.08 0.662
##  4 P     HbA1c0  HbA1c       86  8.91 0.764
##  5 L0.6  HbA1c12 HbA1c      104  8.78 0.977
##  6 L1.2  HbA1c12 HbA1c      116  8.29 1.03
##  7 L1.8  HbA1c12 HbA1c       94  7.64 0.909
##  8 P     HbA1c12 HbA1c       85  9.03 1.01
##  9 L0.6  HbA1c24 HbA1c      104  8.39 0.971
## 10 L1.2  HbA1c24 HbA1c      116  7.9  1.03
## 11 L1.8  HbA1c24 HbA1c       94  7.52 1.10
## 12 P     HbA1c24 HbA1c       86  8.64 0.992
```

Step 4: Visualizing the Data

```
bxp <- ggboxplot(
  data2, x = "Treat", y = "HbA1c",
  color = "Time", palette = "jco")
bxp

## Warning: Removed 1 row containing non-finite values (`stat_boxplot()`).
```

We have created a boxplot to visualize the trends in this data with the x-axis as the treatment group and the y-axis as HbA1c values. Different colors have been imparted for different time points. The color palette can be chosen (Fig. 7.10).

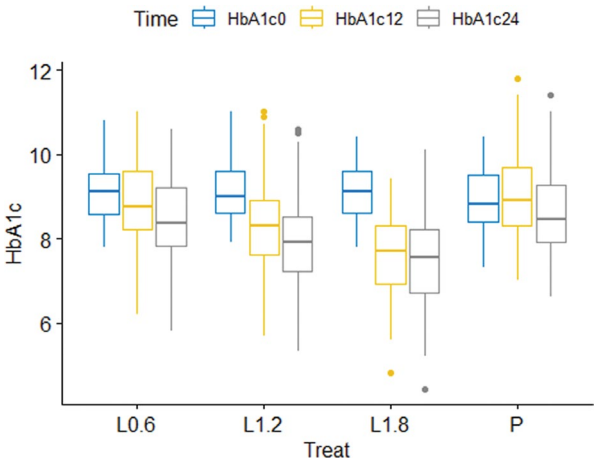

Fig. 7.10 Boxplot to visualize the trends of change in HbA1c in each treatment group and comparison of treatment groups

Step 5: Checking Assumptions

```
data2 %>%
  group_by(Time, Treat) %>%
  identify_outliers(HbA1c)

## # A tibble: 8 × 16
##   Treat Time    SID     Age Gender   BMI    Wt   Hb0  FPG0 FPG12
priorRx    hcfb
##   <fct> <fct>   <fct> <int>  <int> <int> <int> <dbl> <int> <int>
<chr>     <dbl>
## 1 L1.2  HbA1c12 76       64      1    29    87  14.8   243   234
Insulin    -0.9
## 2 L1.2  HbA1c12 343      69      0    20    72  14     272   216 Non
Insu…  -0.1
## 3 L1.8  HbA1c12 158      56      1    33    79   8.8   179   143
Insulin     3.2
## 4 P     HbA1c12 172      42      0    34    76  17.2   275   281
Insulin    -0.8
## 5 L1.2  HbA1c24 76       64      1    29    87  14.8   243   234
Insulin    -0.9
## 6 L1.2  HbA1c24 343      69      0    20    72  14     272   216 Non
Insu…  -0.1
## 7 L1.8  HbA1c24 158      56      1    33    79   8.8   179   143
Insulin     3.2
## 8 P     HbA1c24 172      42      0    34    76  17.2   275   281
Insulin    -0.8
## # i 4 more variables: fcfb <int>, HbA1c <dbl>, is.outlier <lgl>,
## #   is.extreme <lgl>

data2 %>%
  group_by(Time, Treat) %>%
  shapiro_test(HbA1c)

## # A tibble: 12 × 5
##    Treat time variable statistic      p
##    <fct> <fct>   <chr>        <dbl>  <dbl>
##  1 L0.6  HbA1c0  HbA1c        0.977 0.0665
##  2 L1.2  HbA1c0  HbA1c        0.979 0.0660
##  3 L1.8  HbA1c0  HbA1c        0.975 0.0677
##  4 P     HbA1c0  HbA1c        0.976 0.107
##  5 L0.6  HbA1c12 HbA1c        0.992 0.800
##  6 L1.2  HbA1c12 HbA1c        0.989 0.477
##  7 L1.8  HbA1c12 HbA1c        0.984 0.319
##  8 P     HbA1c12 HbA1c        0.976 0.116
##  9 L0.6  HbA1c24 HbA1c        0.992 0.813
## 10 L1.2  HbA1c24 HbA1c        0.989 0.474
## 11 L1.8  HbA1c24 HbA1c        0.991 0.806
## 12 P     HbA1c24 HbA1c        0.977 0.133

data2 %>%
  group_by(Time) %>%
  levene_test(HbA1c ~ Treat)

## # A tibble: 3 × 5
##   Time      df1   df2 statistic     p
##   <fct>   <int> <int>     <dbl> <dbl>
## 1 HbA1c0      3   396     0.471 0.703
```

```
## 2 HbA1c12     3   395     0.322 0.809
## 3 HbA1c24     3   396     0.535 0.659

box_m(data2[, "HbA1c", drop = FALSE], data2$Treat)

## # A tibble: 1 × 4
##   statistic p.value parameter method
##       <dbl>   <dbl>     <dbl> <chr>
## 1        NA      NA         3 Box's M-test for Homogeneity of
Covariance Matric…
```

1. Assumptions for normality, homogeneity of variance, homogeneity of covariances and outliers in data were checked.
2. There were no extreme outliers in the data, the data was normally distributed, and homogeneity of variances and covariances was ensured.
3. It should be noted that the function *"anova_test"*, which will be discussed in the next section, will correct itself for sphericity if required, and there is no need to look for sphericity explicitly.

Step 6: Analysing the Data

```
res.aov <- anova_test(
  data = data2, dv = HbA1c, wid = SID,
  between = Treat, within = Time)

## Warning: NA detected in rows: 425.
## Removing this rows before the analysis.

get_anova_table(res.aov)

## ANOVA Table (type III tests)
##
##        Effect  DFn    DFd       F        p p<.05   ges
## 1      Treat 3.00 395.00  16.755 2.89e-10     * 0.095
## 2       Time 1.34 529.27 403.353 1.98e-82     * 0.153
## 3 Treat:Time 4.02 529.27  54.077 2.31e-38     * 0.068
```

1. The *"anova_test"* function is used to check if there is a difference between various groups at different time points.
2. The arguments for a mixed model will include *between* for grouping variables and *within* for variables for time points. The *dv,* as discussed earlier, is the argument for specifying the dependent continuous variable.
3. We can appreciate the effect of the treatment group and time points and that there is a statistically significant two-way interaction between treatment and time. Thus, a post-hoc analysis should be done.

Step 7: Post-hoc Analysis

A significant two-way interaction, which we found in the above step, means that the effect of one factor, i.e., treatment group, on the outcome variable, i.e., HbA1c, depends on the levels of the other factor, i.e., time point and vice-versa. In these cases, the next step would be to test the effect of one factor on every level of the second factor. If this is significant, multiple pairwise comparisons are further done to locate which groups are different.

Evaluating the Effect of Treatment on Each Time Point

Simple Main Effect

```
#Effect of treatment at each time point
one.way_group <- data2 %>%
  group_by(Time) %>%
  anova_test(dv = HbA1c, wid = SID, between = Treat) %>%
  get_anova_table() %>%
  adjust_pvalue(method = "bonferroni")

## Warning: There was 1 warning in `mutate()`.
## i In argument: `data = map(.data$data, .f, ...)`.
## Caused by warning:
## ! NA detected in rows: 25.
## Removing this rows before the analysis.

one.way_group

## # A tibble: 3 × 9
##   Time    Effect   DFn   DFd     F        p `p<.05`   ges    p.adj
##   <fct>   <chr>  <dbl> <dbl> <dbl>    <dbl> <chr>   <dbl>    <dbl>
## 1 HbA1c0  Treat      3   396  1.82 1.43e- 1 ""      0.014 4.29e- 1
## 2 HbA1c12 Treat      3   395 35.7  1.9 e-20 "*"     0.213 5.7 e-20
## 3 HbA1c24 Treat      3   396 22.2  2.64e-13 "*"     0.144 7.92e-13
```

1. Here, we have used “bonferroni” post-hoc to check the effect of groups on each time point.
2. The output shows that the treatment groups were not different at baseline, but a significant interaction can be seen at 12 and 24 weeks.

Multiple Pairwise Comparisons

Next, we need to perform a posthoc pairwise comparison by “bonferroni” method to understand the difference between the groups at each timepoint in HbA1c levels.

```
pwc_group <- data2 %>%
  group_by(Time) %>%
  pairwise_t_test(HbA1c ~ Treat, p.adjust.method = "bonferroni")

pwc_group

## # A tibble: 18 × 10
##    Time  .y.   group1 group2    n1    n2        p p.signif    p.adj
p.adj.signif
##  * <fct> <chr> <chr>  <chr>  <int> <int>    <dbl> <chr>       <dbl> <chr>
##  1 HbA1… HbA1c L0.6   L1.2     104   116 8.33e- 1 ns       1   e+ 0 ns
##  2 HbA1… HbA1c L0.6   L1.8     104    94 6.09e- 1 ns       1   e+ 0 ns
##  3 HbA1… HbA1c L1.2   L1.8     116    94 7.5 e- 1 ns       1   e+ 0 ns
##  4 HbA1… HbA1c L0.6   P        104    86 3.42e- 2 *        2.05e- 1 ns
##  5 HbA1… HbA1c L1.2   P        116    86 4.89e- 2 *        2.93e- 1 ns
##  6 HbA1… HbA1c L1.8   P         94    86 1.13e- 1 ns       6.79e- 1 ns
##  7 HbA1… HbA1c L0.6   L1.2     104   116 2.77e- 4 ***      1.66e- 3 **
##  8 HbA1… HbA1c L0.6   L1.8     104    94 5.74e-15 ****     3.44e-14 ****
##  9 HbA1… HbA1c L1.2   L1.8     116    94 2.68e- 6 ****     1.61e- 5 ****
## 10 HbA1… HbA1c L0.6   P        104    86 8.45e- 2 ns       5.07e- 1 ns
## 11 HbA1… HbA1c L1.2   P        116    86 2.6 e- 7 ****     1.56e- 6 ****
## 12 HbA1… HbA1c L1.8   P         94    86 3.95e-19 ****     2.37e-18 ****
## 13 HbA1… HbA1c L0.6   L1.2     104   116 4.72e- 4 ***      2.83e- 3 **
## 14 HbA1… HbA1c L0.6   L1.8     104    94 5.76e- 9 ****     3.46e- 8 ****
## 15 HbA1… HbA1c L1.2   L1.8     116    94 7.76e- 3 **       4.66e- 2 *
## 16 HbA1… HbA1c L0.6   P        104    86 9.32e- 2 ns       5.59e- 1 ns
## 17 HbA1… HbA1c L1.2   P        116    86 6.15e- 7 ****     3.69e- 6 ****
## 18 HbA1… HbA1c L1.8   P         94    86 1.37e-12 ****     8.23e-12 ****
```

1. The *"pairwise_t_test"* using "bonferroni" correction was done. Furthermore, we should evaluate the effect of time on the HbA1c levels in groups.
2. Pairwise comparisons show that there are significant differences in HbA1c at 12 weeks between treatment groups L0.6 and L1.2, L0.6 and L1.8, L1.2 and L1.8, L1.2 and P & L1.8 and P. There were differences at 24 weeks between L0.6 and L1.2, L0.6 and L1.8, L1.2 and L1.8, L1.2 and P & L1.8 and P.

Evaluating the Effect of Timepoint on Each Treatment

Simple Main Effect

```
one.way_time <- data2 %>%
  group_by(Treat) %>%
  anova_test(dv = HbA1c, wid = SID, within = Time) %>%
  get_anova_table() %>%
  adjust_pvalue(method = "bonferroni")

## Warning: There was 1 warning in `mutate()`.
## i In argument: `data = map(.data$data, .f, ...)`.
## Caused by warning:
## ! NA detected in rows: 91.
## Removing this rows before the analysis.

one.way_time

## # A tibble: 4 × 9
##   Treat Effect   DFn   DFd       F        p `p<.05`   ges    p.adj
##   <fct> <chr>  <dbl> <dbl>  <dbl>    <dbl> <chr>   <dbl>    <dbl>
## 1 L0.6  Time       1 103.  121.   4.69e-19 *       0.103 1.88e-18
## 2 L1.2  Time       1 115.  264.   1.43e-31 *       0.227 5.72e-31
## 3 L1.8  Time       2 186   234.   1.69e-51 *       0.381 6.76e-51
## 4 P     Time       1  84.2   9.58 3   e- 3 *       0.029 1.2 e- 2
```

1. The output shows that the effect of time is significant on the treatment effects.
2. A post-hoc analysis is needed for pairwise comparison between the time points for the treatment groups.

Multiple Pairwise Comparison

```
pwc_time <- data2 %>%
  group_by(Treat) %>%
  pairwise_t_test(
    HbA1c ~ Time, paired = TRUE,
    p.adjust.method = "bonferroni")
pwc_time

## # A tibble: 12 × 11
##    Treat .y.   group1  group2     n1    n2 statistic    df
p     p.adj
##  * <fct> <chr> <chr>   <chr>   <int> <int>     <dbl> <dbl>     <dbl>
    <dbl>
##  1 L0.6  HbA1c HbA1c0  HbA1c12   104   104      6.02   103 2.74e-  8
8.22e-  8
##  2 L0.6  HbA1c HbA1c0  HbA1c24   104   104     12.7    103 9.18e- 23
2.75e- 22
##  3 L0.6  HbA1c HbA1c12 HbA1c24   104   104    345.     103 1.41e-159
4.23e-159
##  4 L1.2  HbA1c HbA1c0  HbA1c12   116   116     12.4    115 4.85e- 23
1.46e- 22
##  5 L1.2  HbA1c HbA1c0  HbA1c24   116   116     18.4    115 4.95e- 36
1.49e- 35
##  6 L1.2  HbA1c HbA1c12 HbA1c24   116   116    220.     115 9.41e-153
2.82e-152
##  7 L1.8  HbA1c HbA1c0  HbA1c12    94    94     19.0     93 8.73e- 34
2.62e- 33
##  8 L1.8  HbA1c HbA1c0  HbA1c24    94    94     17.9     93 5.93e- 32
1.78e- 31
##  9 L1.8  HbA1c HbA1c12 HbA1c24    94    94      1.52    93 1.32e-  1
3.96e-  1
## 10 P     HbA1c HbA1c0  HbA1c12    86    85     -1.22    84 2.24e-  1
6.72e-  1
## 11 P     HbA1c HbA1c0  HbA1c24    86    86      2.47    85 1.5 e-  2
4.6 e-  2
## 12 P     HbA1c HbA1c12 HbA1c24    85    86     80.5     84 2.72e- 81
8.16e- 81
## # i 1 more variable: p.adj.signif <chr>
```

1. The pairwise comparisons show that all comparisons between time points were significant for L0.6 and L1.2, while, in L1.8, the comparison between 12 and 24 weeks was not significant, and in the placebo group, the comparison between baseline and 12 weeks was not significant.

Step 8: Conclusion

A mixed model (two-way repeated measure) ANOVA was performed. There was a significant interaction between treatment and time on HbA1c values. $F(4,530)$ <0.001. Therefore, the treatment variable was analyzed at each time point, and pairwise comparisons show that there are significant differences in HbA1c at 12 weeks

between treatment groups L0.6 and L1.2, L0.6 and L1.8, L1.2 and L1.8, L1.2 and P and L1.8 and P. There were differences at 24 weeks between L0.6 and L1.2, L0.6 and L1.8, L1.2 and L1.8, L1.2 and P & L1.8 and P.

References

1. Fox J, Weisberg S. An R Companion to Applied Regression. 3rd ed. Thousand Oaks CA: Sage; 2019. https://socialsciences.mcmaster.ca/jfox/Books/Companion/
2. Kassambara A. rstatix: Pipe-Friendly Framework for Basic Statistical Tests. R package version 0.7.2. 2023. https://rpkgs.datanovia.com/rstatix/.
3. Wickham H, Averick M, Bryan J, Chang W, McGowan L, François R, Grolemund G, Hayes A, Henry L, Hester J, Kuhn M, Pedersen T, Miller E, Bache S, Müller K, Ooms J, Robinson D, Seidel D, Spinu V, et al. Welcome to the tidyverse. J Open Source Softw. 2019;4(43):1686.
4. Kassambara A. ggpubr: 'ggplot2' Based Publication Ready Plots. R package version 0.6.0. 2023. https://rpkgs.datanovia.com/ggpubr/.

Chapter 8
Inferential Statistics for the Hypothesis Testing of Non-parametric Data

Praveen Kumar-M and Archana Mishra

The tests for parametric data, which the readers learned in the earlier chapter, can only be applied if the data follows a normal distribution. We opt for non-parametric tests if the data does not follow a normal distribution and the sample size is small. Specific non-parametric tests can be used based on the context of the question at hand and the type of the available data. Non-parametric tests compute test statistics by comparing the ranks of observations or differences in observations between two or more groups. This chapter will explain the different available non-parametric tests and their implementation in R.

To decide whether the proper non-parametric test should be applied, first look at whether the data in hand is continuous or categorical. In the case of a continuous variable, for comparison between two-time points within a group (paired data), we apply the Wilcoxon signed-rank test, and to compare the data from two independent groups (unpaired data), the Mann-Whitney U test, also known as the Wilcoxon rank-sum test, is used. We apply the Friedman test to compare measures between >2 time points in a group for a continuous variable. For comparison between >2 groups, we use the Kruskal Wallis test. In the categorical variable, we need to look out for the cell values in the expected contingency table to compare unpaired groups. An expected contingency table is a table that is formed during the calculation of Chi-square statistics describing the frequency distribution of the two categorical variables. If all the expected tables' values are >5, we apply the Chi-square test; if >20% of the expected table values are <5, we go for the Fisher exact test. The Fisher exact test can only be applied if the data dimension

Supplementary Information The online version contains supplementary material available at https://doi.org/10.1007/978-981-97-6980-3_8.

P. Kumar-M (✉)
Clinical Sciences, Nference, Bengaluru, Karnataka, India

A. Mishra
Department of Pharmacology, All India Institute of Medical Sciences, Bhubaneswar, Odisha, India

A. Srinivasan et al. (eds.), *R for Basic Biostatistics in Medical Research*,
https://doi.org/10.1007/978-981-97-6980-3_8

is 2 by 2; for 2 by R, where R can be >2, the Fisher test with the Monte Carlo simulation is applied. The Mcnemar test is to be used for comparison between a paired set of categorical variables. In addition to the choice of tests mentioned here, some assumptions must be checked before using a particular test. We have mentioned these assumptions under each test. The next paragraph gives the chapter's general outline for each test.

We start each test with an introduction to the study design for which the test is applicable. This will be followed by stating the assumptions, null and alternative hypotheses, and providing a specific case scenario and associated experimental data to work on. We will first import the data into the workspace. Thereafter, we will test for fulfillment of the assumptions on the experimental data. If the assumptions are fulfilled, we will conduct the test. If more than one method is available for a particular step, we have also tried to mention alternative methods. Some additional calculations have been explained, such as descriptive statistics and graphical representation that must be undertaken to aid the test's interpretation. Finally, we will present the results and give a conclusion statement to the case scenario.

Wilcoxon Signed-Rank Test

Wilcoxon sign rank test is the non-parametric counterpart of the paired t-test. By paired, we mean that we have measured the observation in the same group of individuals two times.

Assumptions Made

1. Data fails the assumption of paired t-test.
2. One dependent variable that is measured at a continuous level (even ordinal can be considered).

Hypothesis

Null hypothesis: The median difference between the paired groups equals zero.

Alternate hypothesis: The median difference between the paired groups is not equal to zero.

Case Scenario

A single-arm study was conducted on twenty participants to understand the benefits of the non-vegetarian diet on hemoglobin. The non-vegetarian diet was defined as an intake of at least 150 gm of boneless chicken daily. The hemoglobin estimation

was done for all the participants at baseline and after 6 months of the above-mentioned non-vegetarian diet. You are asked to analyze the data to find if there was a significant change in hemoglobin levels with the intervention. The data has been named "Wilcoxon_rank.csv". The data contains three columns: Patient Id, Hemoglobin levels at baseline (Hb_baseline), and Hemoglobin levels at 6 months (Hb_6Months).

Step 1: Importing the Data into the R Console

We will import the data file and name the imported object as *datawx.*

```
datawx <- read.csv(file  =  "Wilcoxon_rank.csv")
View(datawx)
```

1. The "*read.csv*" function imports the data in CSV format. The arguments of the function are *file*, *header*, *sep*, *row.names*, *col.names*, *na.strings*. We have used only the essential argument in this case. The *file* argument takes in the name of the file from the working directory or the *path of the file*.
2. The data is then visualized in the viewer window using the "*View*" function.

Step 2: Checking for Assumptions

We can check for the distribution of data (whether normal or not) using the graphical method (plotting histogram) or the Shapiro-Wilk test (Fig. 8.1). The data distribution is non-parametric if the histogram is skewed to any of the sides. Similarly, if the p-value is <0.05 for the Shapiro-Wilk test, the data is assumed to follow a non-parametric distribution. It is reiterated here that the normality assumption should be checked on the change in the estimate of hemoglobin between the two-time points and not on the raw data.

```
datawx$Diff_6Months_Baseline <-
  datawx$Hb_6Months - datawx$Hb_Baseline
hist(datawx$Diff_6Months_Baseline)
```

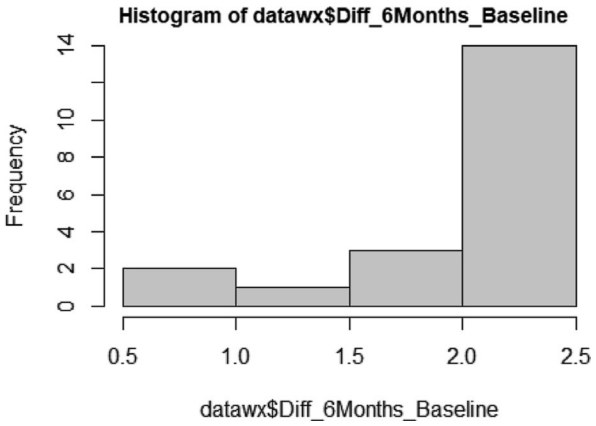

Fig. 8.1 Histogram of the Hb difference between 6 months and baseline

```
shapiro.test(datawx$Hb_6Months - datawx$Hb_Baseline)

##
##  Shapiro-Wilk normality test
##
## data:  datawx$Hb_6Months - datawx$Hb_Baseline
## W = 0.70555, p-value = 4.559e-05

#hist(datawx$Diff_6Months_Baseline, breaks = 5)
#datawx datawx can be visualised for confirmation
attach(what = datawx)
```

1. The difference is calculated as a simple subtraction of the values obtained at both the time points and assigning a new name(*Diff_6Months_Baseline*) to the computed variable.
2. The "*hist*" function is used for plotting histograms. The function has already been explained under the "Visualization of data—basic and advanced" chapter.
3. The "*shapiro.test*" function is used to check for the normality of the distribution.
4. The "*attach*" function is used to attach the dataframe to the working environment. The dataframe needs to be passed as the value to `what` argument of the function. This helps avoid the repetitive entry of the `R-object-name-$-variable-name` inside the functions while subsetting the variables (columns). The `variable name` can be directly passed as the value inside functions.
5. In this case, the histogram appears skewed, and the p-value of the Shapiro-Wilk test is significant, thereby showing that the distribution is non-parametric (Fig. 8.1).

Step 3: Accompanying Calculation

We require median values to write the final result and interpret the output from the Wilcoxon signed-rank test. So, the next step is the computation of the median of the paired data from both time points.

```
median(x = Hb_Baseline)

## [1] 11.55

median(Hb_6Months)

## [1] 13.45

median(Hb_6Months-Hb_Baseline)

## [1] 2.1

quantile(Hb_6Months-Hb_Baseline)

##    0%   25%   50%   75%  100%
## 0.500 1.975 2.100 2.200 2.500
```

1. The "*median*" function is used to calculate the median, and the input is a numeric vector for argument *x*. Another argument in this function is *na.rm*, which takes a logical argument as its value. When *na.rm* is *TRUE*, the NAs in the dataset are removed, which becomes essential in case of missing values in the dataset. *na.rm* argument set to *FALSE* is the default option.

Step 4: Wilcoxon Signed-Rank Test

```
wilcox.test(x = Hb_6Months, y = Hb_Baseline, paired = TRUE)

## Warning in wilcox.test.default(x = Hb_6Months, y = Hb_Baseline, paired =
TRUE):
## cannot compute exact p-value with ties

##
##  Wilcoxon signed rank test with continuity correction
##
## data:  Hb_6Months and Hb_Baseline
## V = 210, p-value = 9.251e-05
## alternative hypothesis: true location shift is not equal to 0
```

1. The "`wilcox.test`" function is used for performing the Wilcoxon signed-rank test, and the arguments used are `x` (takes in numeric vector as value), `y` (takes in numeric vector), `paired` (logical operator as value: *TRUE* for paired data, *FALSE* for unpaired data), `alternative` (takes in one among the following as value: *two.sided*, *greater* or *less*), `mu` (a number which constitutes the null hypothesis), `conf.level`(mention the confidence interval as value for this argument), `data`, `formula,` and `subset`
2. The `data` argument of the "`wilcox.test`" function takes in the dataframe as the value (which is not required here because of the use of the "`attach`" function), and the `subset` is used for subsetting the columns from the dataframe if required. The `formula` argument takes input in the form of vector~groups.

Things to look at:

Did you encounter a warning??? This has already been communicated that a warning is different from an error. It alerts us to beware of the information conveyed through warning during interpretation.

In this case, a *warning* arose due to ties when comparing the groups.

Conclusion

The Wilcoxon signed-rank test was performed, and a median increase in Hb of 2.1 gm/dL (IQR: 1.975–2.200) at 6 months following a regular non-vegetarian diet compared to the baseline was observed (please note that the median is accompanied by IQR in the results), which was found to be statistically significant ($p < 0.001$).

Mann-Whitney U Test

The Mann-Whitney U test, also known as the Wilcoxon rank-sum test, is a non-parametric statistical test used to determine if there is a significant difference between two independent and non-normally distributed groups. The test compares the distribution of ranks between two independent groups to determine if there is a significant difference in their medians. It is suitable for ordinal or continuous data that does not meet the assumptions of parametric tests, i.e., the unpaired t-test.

Assumptions Made

1. Data fails the assumption of independent t-test
2. One dependent variable is measured at a continuous or ordinal level.
3. One independent variable consists of two independent groups.
4. Independence of observation both within and between groups (study design-based).

Hypothesis

Null hypothesis: The distribution of values in one group is equal to that in another group. In terms of medians, it can be stated that there is no difference in the medians of the two groups.

Alternative hypothesis: There is a significant difference between the distributions of the two groups. In terms of medians, it can be stated that there is a difference in the medians of the two groups.

Case Scenario

A clinical trial was conducted to evaluate and compare the efficacy of two drugs, "Treatment_A" and "Treatment_B," on patients with type 2 diabetes mellitus after 6 months of therapy. The glycated hemoglobin (Hba1c) was measured at baseline and at the end of 6 months of treatment. Baseline values of HbA1c were comparable between the groups. You are asked to compare the groups' Hba1c values at 6 months. The data obtained for Hba1c at 6 months has been stored in the name of the *MWU.csv*. The data contains three columns: Patient Id, Treatment group (Treatment), and Hba1c values at 6 months (Hba1c_6_Month).

Step 1: Importing of Data into the R Console

```
datamw <- read.csv("MWU.csv")
View(datamw)
str(object = datamw)

## 'data.frame':    40 obs. of  3 variables:
##  $ X            : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ Treatment    : chr  "Treatment_A" "Treatment_A" "Treatment_A"
"Treatment_A" ...
##  $ Hba1c_6_Month: num  5.7 5.5 5.7 4.9 5.4 5.7 5.3 4.9 5.5 5.7 ...

###If the grouping variable is not a factor, then you need to convert the
variable to factor manually.
datamw$Treatment <- as.factor(datamw$Treatment)
str(datamw)

## 'data.frame':    40 obs. of  3 variables:
##  $ X            : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ Treatment    : Factor w/ 2 levels "Treatment_A",..: 1 1 1 1 1 1 1
1 1 1 ...
##  $ Hba1c_6_Month: num  5.7 5.5 5.7 4.9 5.4 5.7 5.3 4.9 5.5 5.7 ...
```

1. We have imported the data using "*read.csv*" function and assigned the same to an R object *datamw*.
2. After that, the "*str*" function is used to understand each variable's data type. The "*str*" function has an *object* as an argument to which the *R object* for the dataframe is the input. It is to be ensured that the grouping variable is a factorial datatype.

Step 2: Checking for Assumptions

```
###Is the data non-parametric???
tapply(datamw$Hba1c_6_Month, datamw$Treatment, shapiro.test)

## $Treatment_A
##
##  Shapiro-Wilk normality test
##
## data:  X[[i]]
## W = 0.86375, p-value = 0.009142
##
##
## $Treatment_B
##
##  Shapiro-Wilk normality test
##
## data:  X[[i]]
## W = 0.8531, p-value = 0.006009

###Determination of the distribution of values in the two groups.
datamw1 <-
  subset(datamw, Treatment == "Treatment_A")
datamw2 <-
  subset(datamw, Treatment == "Treatment_B")
par(mfrow = c(1, 2))
hist(
  datamw1$Hba1c_6_Month,
  breaks = 4,
  xlim = c(4, 8)
)
hist(
  datamw2$Hba1c_6_Month,
  breaks = 4,
  xlim = c(4, 8)
)
```

```
par(mfrow = c(1, 1))
```

1. The "*tapply*" function is a helpful tool for applying a function to subsets of a vector or other data structure based on a factor or list of factors. It is convenient to use when you want to split your data into groups and perform some operation on each group separately. In this context, the "`tapply`" function was used to perform the Shapiro-Wilk test.
2. The arguments which the "`tapply`" function takes are `X` (The vector or array that needs to be split into groups and apply a function to; in this case, *Hba1c at 6th month*), `INDEX` (A list or factor that specifies the grouping; in this case, it is the *treatment*) and last is the `FUN` argument that specifies the function to be applied to each group; in this case is "*shapiro.test*".
3. The "`subset`" function is used to create the subset of the *datamw* dataset based on groups "Treatment_A" and "Treatment_B". The logical operator == is used for equating. The function "`subset`" is explained in the chapter "Data handling and manipulation in R".
4. The given data distribution is not normal because the p-value computed using the Shapiro-Wilk test is less than 0.05 in both groups. We should apply non-parametric tests for comparison.
5. Having the same x-axis limits in both graphs is important. This makes the comparison uniform. The distribution looks similar, and hence, we would be computing an alternate hypothesis: "The median of two groups is not equal" (Fig. 8.2).

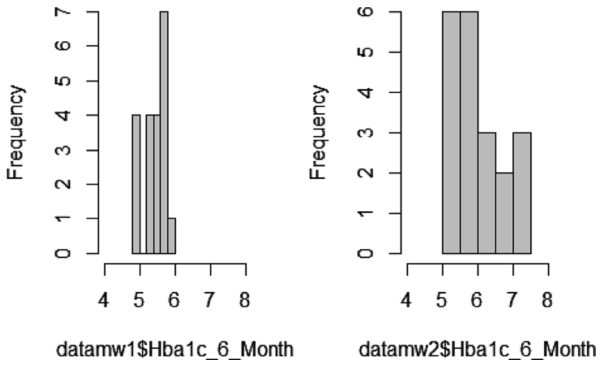

Fig. 8.2 Histograms of Hba1c level at 6 months for Treatment A (left) and Treatment B (right)

Step 3: Accompanying Calculation

The data is not normally distributed; thus, we calculate the median and interquartile range for reporting purposes.

Option 3a: Method 1

```
###Computation of the median of the two groups by using one of the 2 methods
mentioned below.

###First is the tapply method, where we can compute the median directly on
the original data
tapply(X = datamw$Hba1c_6_Month,
       datamw$Treatment,
       FUN = median)

## Treatment_A Treatment_B
##         5.5         5.8

tapply(X = datamw$Hba1c_6_Month,
       datamw$Treatment,
       FUN = IQR)

## Treatment_A Treatment_B
##       0.400       1.075

tapply(X = datamw$Hba1c_6_Month,
       datamw$Treatment,
       FUN = quantile)

## $Treatment_A
##   0%  25%  50%  75% 100%
##  4.9  5.3  5.5  5.7  5.9
##
## $Treatment_B
##    0%   25%   50%   75%  100%
## 5.300 5.500 5.800 6.575 7.200
```

Option 3b: Method 2

```
#Second is the median function method, which can be applied to the subsets

median(x = datamw1$Hba1c_6_Month)

## [1] 5.5

median(datamw2$Hba1c_6_Month)

quantile(datamw1$Hba1c_6_Month)

##   0%  25%  50%  75% 100%
##  4.9  5.3  5.5  5.7  5.9

quantile(datamw2$Hba1c_6_Month)

##    0%   25%   50%   75%  100%
## 5.300 5.500 5.800 6.575 7.200

## [1] 5.8
```

1. The "*tapply*" function has been used to calculate the median in `Option 3a`. The input for the *FUN* argument of the "*tapply*" function is the `median`.
2. The "*median*" function is directly employed to compute the median.

Step 4: Analysis of Data: Mann Whitney U Test

Option 4a: Method 1

```
### Option 4a:

#First method
wilcox.test(x = datamw1$Hba1c_6_Month,
            y = datamw2$Hba1c_6_Month,paired = FALSE)

## Warning in wilcox.test.default(x = datamw1$Hba1c_6_Month, y =
## datamw2$Hba1c_6_Month, : cannot compute exact p-value with ties

##
##  Wilcoxon rank sum test with continuity correction
##
## data:  datamw1$Hba1c_6_Month and datamw2$Hba1c_6_Month
## W = 106.5, p-value = 0.01137
## alternative hypothesis: true location shift is not equal to 0
```

1. The "*wilcox.test*" function was used to perform the Mann-Whitney U test to compare the median of the two groups. The function is explained in the Wilcoxon test section. The input for *paired* argument for "*wilcox.test*" function should be kept as FALSE for the Mann-Whitney U test.

Option 4b: Method 2

```
#Second method
wilcox.test(formula = Hba1c_6_Month ~ Treatment, data = datamw, paired =
FALSE)

## Warning in wilcox.test.default(x = DATA[[1L]], y = DATA[[2L]], ...):
cannot
## compute exact p-value with ties

##
##  Wilcoxon rank sum test with continuity correction
##
## data:  Hba1c_6_Month by Treatment
## W = 106.5, p-value = 0.01137
## alternative hypothesis: true location shift is not equal to 0
```

1. This is an alternate method of performing a Mann-Whitney U test where we provide a formula in the form of `Hba1c_6_Month` as a function of the grouping variable `Treatment`. The rest of the arguments are kept the same as earlier.

Final Conclusion

Median HbA1c score at 6 months was significantly (statistically) different between the "Treatment_A" (median (IQR)—5.5 (5.3–5.7)) and "Treatment_B" (median (IQR)—5.8 (5.50–6.58)) with p = 0.01.

Step 5: Creation of Population Pyramid

Visualizing HbA1c (glycated hemoglobin) distribution group-wise can be done by creating a population pyramid (Fig. 8.3). We need to categorize the continuous data into groups or bins and then make the pyramid based on the means of these groups. We will use *pyramid* package for this [1].

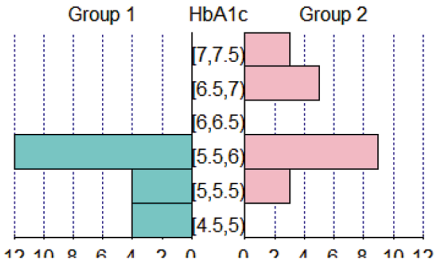

Fig. 8.3 Pyramidal figure for the distribution of Hba1c levels in Treatment A and B groups

```
duration1 <- datamw1$Hba1c_6_Month
range(duration1)

## [1] 4.9 5.9

breaks1 <- seq(4.5, 7.5, by = 0.5)
duration1.cut <- cut(duration1, breaks1, right = FALSE)
duration1.cut

##  [1] [5.5,6) [5.5,6) [5.5,6) [4.5,5) [5,5.5) [5.5,6) [5,5.5) [4.5,5)
[5.5,6)
## [10] [5.5,6) [4.5,5) [5.5,6) [5.5,6) [4.5,5) [5,5.5) [5.5,6) [5,5.5)
[5.5,6)
## [19] [5.5,6) [5.5,6)
## Levels: [4.5,5) [5,5.5) [5.5,6) [6,6.5) [6.5,7) [7,7.5)

duration1.freq <- table(duration1.cut)
duration1.freq

## duration1.cut
## [4.5,5) [5,5.5) [5.5,6) [6,6.5) [6.5,7) [7,7.5)
##       4       4      12       0       0       0

duration2 <- datamw2$Hba1c_6_Month
range(duration2)

## [1] 5.3 7.2

breaks2 <- seq(4.5, 7.5, by = 0.5)
duration2.cut <- cut(duration2, breaks2, right = FALSE)
duration2.cut

##  [1] [5.5,6) [7,7.5) [5.5,6) [5,5.5) [5.5,6) [6.5,7) [6.5,7) [5.5,6)
[7,7.5)
## [10] [5.5,6) [5,5.5) [5.5,6) [6.5,7) [6.5,7) [5.5,6) [7,7.5) [5.5,6)
[5,5.5)
## [19] [5.5,6) [6.5,7)
## Levels: [4.5,5) [5,5.5) [5.5,6) [6,6.5) [6.5,7) [7,7.5)

duration2.freq <- table(duration2.cut)
duration2.freq

## duration2.cut
## [4.5,5) [5,5.5) [5.5,6) [6,6.5) [6.5,7) [7,7.5)
##       0       3       9       0       5       3
```

```
duration <- data.frame(duration1.freq, duration2.freq)
duration

##   duration1.cut Freq duration2.cut Freq.1
## 1       [4.5,5)    4       [4.5,5)      0
## 2       [5,5.5)    4       [5,5.5)      3
## 3       [5.5,6)   12       [5.5,6)      9
## 4       [6,6.5)    0       [6,6.5)      0
## 5       [6.5,7)    0       [6.5,7)      5
## 6       [7,7.5)    0       [7,7.5)      3

durationpyramid <-
  data.frame(duration$Freq, duration$Freq.1, duration$duration1.cut)
durationpyramid

##   duration.Freq duration.Freq.1 duration.duration1.cut
## 1             4               0                [4.5,5)
## 2             4               3                [5,5.5)
## 3            12               9                [5.5,6)
## 4             0               0                [6,6.5)
## 5             0               5                [6.5,7)
## 6             0               3                [7,7.5)

require("pyramid")

## Loading required package: pyramid

pyramid(durationpyramid,
        Llab = "Group 1",
        Rlab = "Group 2",
        Clab = "HbA1c")
```

1. The "*cut*" function is used to categorize HbA1c values into groups. The "*pyramid*" function is used to plot the population pyramid.

Kruskal-Wallis Test

The Kruskal-Wallis test is a non-parametric statistical test used to determine whether there are statistically significant differences between the medians of three or more independent groups. It is an extension of the Mann-Whitney U test for comparing two groups. It is to be kept in mind that the Kruskal-Wallis test is a non-parametric alternative to the one-way analysis of variance (ANOVA) for comparing more than two groups when the assumptions of ANOVA are not met. If the Kruskal-Wallis test indicates significant differences, you may need to perform post-hoc tests to identify which specific groups differ from each other.

Assumptions Made

1. Data is not normally distributed.
2. One dependent variable is measured at a continuous or ordinal level.

3. One independent variable consisting of more than two categorical independent groups
4. Independence of observation both within and between groups (Study design-based).

Hypothesis

Null hypothesis: The distribution of scores amongst the groups is equal.

Alternate hypothesis: The median or the distribution of the score for the groups is not equal.

Case Scenario

A clinical trial was conducted to evaluate and compare the effect of four antidepressant drugs in 31 patients with depression. Patients were grouped into Group 1, Group 2, Group 3, and Group 4 for four drugs. The primary outcome measure was the severity of depression as measured using the Hamilton depression rating scale (HDRS) after 6 months of therapy. HDRS is a 17-item scale scale that ranges from 0 to 52, with a lower score meaning a better prognosis. An evaluation was made at the end of the follow-up. There was no significant difference in HDRS scores at baseline. The required data was saved in a file named "*KW.csv*". The data contains three variables: ID, Treatment, and HDRS_Score.

Step 1: Importing Data into the R Console

```
datakw <- read.csv("KW.csv")
str(datakw)

## 'data.frame':    31 obs. of  3 variables:
##  $ X         : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ Treatment : chr  "Group_1" "Group_1" "Group_1" "Group_2" ...
##  $ HDRS_Score: int  5 7 5 17 6 10 10 6 19 17 ...

#If the grouping variable is not a factor, then you need to convert the
variable to factor manually.
datakw$Treatment <- as.factor(x = datakw$Treatment)
str(datakw)

## 'data.frame':    31 obs. of  3 variables:
##  $ X         : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ Treatment : Factor w/ 4 levels "Group_1","Group_2",..: 1 1 1 2 1 2
2 1 2 3 ...
##  $ HDRS_Score: int  5 7 5 17 6 10 10 6 19 17 ...
```

1. "*read.csv*" function is used to load the data into working environment and assigned to an R object *datakw*.
2. The grouping variable shows the data type as integer, we need to convert the variable type to factor using the "*as.factor*" function.

Step 2: Checking for Assumptions

Checking for normality of data

```
tapply(datakw$HDRS_Score, datakw$Treatment, shapiro.test)

## $Group_1
##
##  Shapiro-Wilk normality test
##
## data:  X[[i]]
## W = 0.79133, p-value = 0.03359
##
##
## $Group_2
##
##  Shapiro-Wilk normality test
##
## data:  X[[i]]
## W = 0.82519, p-value = 0.03943
##
##
## $Group_3
##
##  Shapiro-Wilk normality test
##
## data:  X[[i]]
## W = 0.80781, p-value = 0.03468
##
##
## $Group_4
##
##  Shapiro-Wilk normality test
##
## data:  X[[i]]
## W = 0.73003, p-value = 0.007813
```

We have run the Shapiro-Wilk test for HDRS-Score group-wise and can observe that the p-value for the same is <0.05 for all the groups; thus, the assumption of normality is not met.

Checking for distribution of data

Option 2a

```
boxplot(formula = HDRS_Score~Treatment,data = datakw, col = c("violet",
"yellow","red","white"))
```

Option 2b

```
datakw1 <- subset(datakw, Treatment == "Group_1")
#datakw1
datakw2 <- subset(datakw, Treatment == "Group_2")
datakw3 <- subset(datakw, Treatment == "Group_3")
datakw4 <- subset(datakw, Treatment == "Group_4")
datassl <-
  list(datakw1$HDRS_Score,
    datakw2$HDRS_Score,
    datakw3$HDRS_Score,
    datakw4$HDRS_Score
  )
boxplot(datassl, col = rainbow(4))
```

1. The "*boxplot*" function is used to create a boxplot to check the distribution of the data across groups. The arguments used in the "*boxplot*" function have been explained in the "Visualization of data—basic and advanced" chapter (Figs. 8.4 and 8.5).
2. The distribution does not appear similar for all the groups, and hence, we will use the Kruskal Wallis to test the alternate hypothesis that the median and the distribution across the groups are not equal.

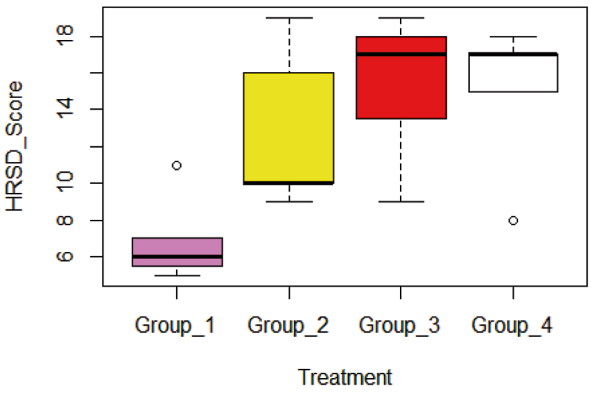

Fig. 8.4 Boxplot for HDRS score across the treatment groups (Method 1)

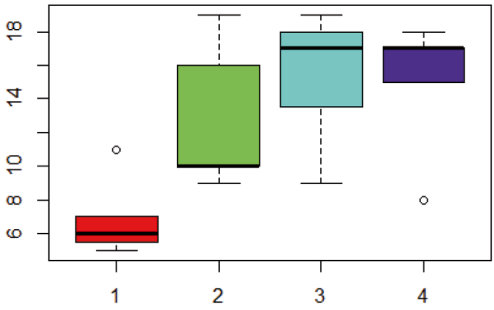

Fig. 8.5 Boxplot for HDRS score across the treatment groups (Method 2)

Step 3: Accompanying Calculation

We need to calculate the median and IQR to report the results.

Option 3a: Method 1

```
#First is the tapply method, where we can compute directly on the original
data
tapply(X = datakw$HDRS_Score,
       datakw$Treatment,
       FUN = median)

## Group_1 Group_2 Group_3 Group_4
##       6      10      17      17

tapply(X = datakw$HDRS_Score,
       datakw$Treatment,
       FUN = IQR)

## Group_1 Group_2 Group_3 Group_4
##    1.50    6.00    3.25    2.00

tapply(X = datakw$HRSD_Score,
       datakw$Treatment,
       FUN = quantile)

## $Group_1
##   0%  25%  50%  75% 100%
##  5.0  5.5  6.0  7.0 11.0
##
## $Group_2
##   0%  25%  50%  75% 100%
##    9   10   10   16   19
##
## $Group_3
##    0%   25%   50%   75%  100%
##  9.00 14.75 17.00 18.00 19.00
##
## $Group_4
##   0%  25%  50%  75% 100%
##    8   15   17   17   18
```

Option 3b: Method 2

```
#Second is the median function method, which can be applied on the subsets
created
median(datakw1$HDRS_Score)

## [1] 6

median(datakw2$HDRS_Score)

## [1] 10

median(datakw3$HDRS_Score)

## [1] 17

median(datakw4$HDRS_Score)

## [1] 17

IQR(datakw1$HDRS_Score)

## [1] 1.5

IQR(datakw2$HDRS_Score)

## [1] 6

IQR(datakw3$HDRS_Score)

## [1] 3.25

IQR(datakw4$HDRS_Score)

## [1] 2

# Length will be required for writing the report
# tapply(X = datakw$HDRS_Score,
#        datakw$Treatment,
#        FUN = length)

quantile(datakw1$HRSD_Score)

##   0%  25%  50%  75% 100%
##  5.0  5.5  6.0  7.0 11.0

quantile(datakw2$HRSD_Score)

##   0%  25%  50%  75% 100%
##    9   10   10   16   19

quantile(datakw3$HRSD_Score)

##    0%   25%   50%   75%  100%
##  9.00 14.75 17.00 18.00 19.00

quantile(datakw4$HRSD_Score)

##   0%  25%  50%  75% 100%
##    8   15   17   17   18
```

1. In `Option 3a`, "*tapply*" function with the *median* as input for the *FUN* argument is used; in `Option 3b`, the "*median*" function is used for calculating the median.

Step 4: Kruskal Wallis Test

Option 4a: Method 1

```
#First method
kruskal.test(formula = HDRS_Score ~ Treatment, data = datakw)

##
##  Kruskal-Wallis rank sum test
##
## data:  HDRS_Score by Treatment
## Kruskal-Wallis chi-squared = 15.088, df = 3, p-value = 0.001743
```

Option 4b: Method 2

```
#Second method
kruskal.test(x = datassl)

##
##  Kruskal-Wallis rank sum test
##
## data:  datassl
## Kruskal-Wallis chi-squared = 15.088, df = 3, p-value = 0.001743
```

1. The "*kruskal.test*" function performs the Kruskal Wallis test.
2. In `Option 4a`, the *data* argument of the "*kruskal.test*" function takes dataframe (*datakw*) as input. The input for the argument *formula* is HDRS_Score as a function of the treatment group (HDRS_Score ~ Treatment).
3. In `Option 4b`, the `x` argument of "*kruksal.test*" function is given a value of *list* (in this case, `datassl`, which was created to plot boxplot, is passed)
4. We observe that the output shows $p < 0.05$, and thus, we conclude that there is a difference between any of the two treatment groups. After that, we need to perform a post hoc analysis to identify the groups that are different from each other.

Step 5: Additional Information

1. Dunn's test evaluates pairwise comparison and reports significance based on the differences in the mean rank of the groups being compared. We will use *dunn.test* package for this purpose [2].

2. Performing multiple comparisons in a dataset increases the chances of type 1 error. Bonferroni correction is applied over the Dunn test for multiplicity adjustments.

Option 5a

```
#install.packages(dunn.test)
require(dunn.test)

## Loading required package: dunn.test

dunn.test(x = datakw$HDRS_Score,g = datakw$Treatment,kw = TRUE,method =
"Bonferroni")

##   Kruskal-Wallis rank sum test
##
## data: x and group
## Kruskal-Wallis chi-squared = 15.0879, df = 3, p-value = 0
##
##
##                             Comparison of x by group
##                                   (Bonferroni)
## Col Mean-|
## Row Mean |    Group_1    Group_2    Group_3
## ---------+---------------------------------
##  Group_2 |  -2.424066
##          |     0.0460
##          |
##  Group_3 |  -3.619480  -1.341073
##          |    0.0009*     0.5397
##          |
##  Group_4 |  -3.075720  -0.838228   0.442890
##          |    0.0063*     1.0000     1.0000
##
## alpha = 0.05
## Reject Ho if p <= alpha/2

### Option 5b:
dunn.test(x = datassl, method = "Bonferroni",kw = FALSE)

##
##                          Comparison of datassl by group
##                                   (Bonferroni)
## Col Mean-|
## Row Mean |          1          2          3
## ---------+---------------------------------
##        2 |  -2.424066
##          |     0.0460
##          |
##        3 |  -3.619480  -1.341073
##          |    0.0009*     0.5397
##          |
##        4 |  -3.075720  -0.838228   0.442890
##          |    0.0063*     1.0000     1.0000
##
## alpha = 0.05
## Reject Ho if p <= alpha/2
```

1. The "*dunn.test*" function is used for performing post-hoc tests. The *"dunn.test"* function's arguments include *x*, *g*, *kw*, and *method*.
2. In Option 5a, the input for *x* argument of "*dunn.test*" should be a numeric vector to be analyzed (datakw**$**HDRS_Score) and *g* argument is the vector for groups (datakw$Treatment).
3. In Option 5b, the *x* argument of dunn.test takes in a list as the value (datassl is passed in this case); *g* argument is not needed for this option.
4. The *method* argument is given *Bonferroni* as the input. The *kw* argument takes in logical operators as value. If *TRUE*, the Kruskal Wallis test results are shown, and if *FALSE*, the test results are not shown.

Conclusion

The HDRS score at the end of the study for four groups was group 1—median(IQR) 6 (5.5–7.0); group 2–10 (10–16); group 3–17 (14.75–18.00); group 4–17 (15–17). The median HDRS score was statistically different among the four groups assessed by the Kruskal Wallis test p = 0.04. The posthoc Dunn test with Bonferroni correction showed that scores in group 1 significantly differed from those in groups 2, 3, and 4.

Friedman Test

The Friedman test is instrumental when dealing with repeated measures or related samples and does not assume a specific distribution for the data. Friedman test is the non-parametric counterpart of repeated measures ANOVA.

Assumptions Made

1. Data is not normally distributed
2. One dependent variable is measured at a continuous or ordinal level.
3. The observations within each group should be independent of each other. This assumption ensures that dependencies between observations do not affect the ranking process.

Hypothesis

Null hypothesis: The distribution of scores across the time points in a group is the same.**Alternate hypothesis:** The distribution of scores differs between at least two time points.

Case Scenario

IL6 is an essential biomarker for predicting the severity of the disease in patients affected with COVID-19. The investigator recruited 20 patients with COVID-19 whose IL-6 levels were known prior to infection with COVID-19, and IL-6 levels were estimated at admission and 4 days after admission. The investigator wishes to compare the IL-6 levels across the three time points. The data has been saved in a file named "*FM.csv*". The data frame has four variables: patient ID, IL6_preCOVID (IL6 levels prior to COVID infection), IL6_AdmCOVID (IL6 levels at the time of admission for COVID), and IL6_postCOVID IL6 levels 4 days post admission).

Step 1: Importing of Data into the R Console

After importing, we assigned the dataset to an R object *datafm.*

```
datafm <- read.csv("FM.csv")
str(datafm)

## 'data.frame':    20 obs. of  4 variables:
##  $ ID           : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ IL6_preCOVID : num  5.9 4.8 6.5 2.3 5.6 8.9 2.3 7.8 11.3 14.5 ...
##  $ IL6_AdmnCOVID: num  32.5 35.6 35.4 34.5 36.5 35.9 28.9 48.5 18.9 19.5
...
##  $ IL6_postCOVID: num  59.1 66.4 64.3 66.7 67.4 62.9 55.5 89.2 26.5 24.5
...

View(datafm)
```

Step 2: Checking for Assumptions

```
shapiro.test(x = datafm$IL6_preCOVID-datafm$IL6_postCOVID)

##
##  Shapiro-Wilk normality test
##
## data:  datafm$IL6_preCOVID - datafm$IL6_postCOVID
## W = 0.88965, p-value = 0.0265

shapiro.test(x = datafm$IL6_postCOVID-datafm$IL6_AdmnCOVID)

##
##  Shapiro-Wilk normality test
##
## data:  datafm$IL6_postCOVID - datafm$IL6_AdmnCOVID
## W = 0.89647, p-value = 0.03542

shapiro.test(x = datafm$IL6_AdmnCOVID-datafm$IL6_preCOVID)
##
##  Shapiro-Wilk normality test
##
## data:  datafm$IL6_AdmnCOVID - datafm$IL6_preCOVID
## W = 0.87441, p-value = 0.01406
```

1. Given that overall, we have a small sample size and the distribution of IL6 values is non-parametric, we should perform the Friedman test to compare the IL6 values at three different time points.

Step 3: Accompanying Calculation

We require median values at the three time points for writing the final result and interpretation of the Friedman test. So, the next step is the computation of the median of the three groups.

Option 3a: Method 1

```
#First method
median(datafm$IL6_preCOVID)

## [1] 10.4

median(datafm$IL6_AdmnCOVID)

## [1] 32.65

median(datafm$IL6_postCOVID)

## [1] 53.75

IQR(datafm$IL6_preCOVID)

## [1] 7.625

IQR(datafm$IL6_AdmnCOVID)

## [1] 16.2

IQR(datafm$IL6_postCOVID)

## [1] 38.15

quantile(datafm$IL6_preCOVID)

##     0%    25%    50%    75%   100%
##  2.300  6.350 10.400 13.975 15.900

quantile(datafm$IL6_AdmnCOVID)

##     0%    25%    50%    75%   100%
## 14.500 19.475 32.650 35.675 48.500

quantile(datafm$IL6_postCOVID)

##    0%   25%   50%   75%  100%
## 15.50 25.55 53.75 63.70 89.20
```

Option 3b: Method 2

```
#summary method
summary(object = datafm)

##        ID         IL6_preCOVID   IL6_AdmnCOVID   IL6_postCOVID
##  Min.   : 1.00   Min.   : 2.30   Min.   :14.50   Min.   :15.50
##  1st Qu.: 5.75   1st Qu.: 6.35   1st Qu.:19.48   1st Qu.:25.55
##  Median :10.50   Median :10.40   Median :32.65   Median :53.75
##  Mean   :10.50   Mean   :10.04   Mean   :28.86   Mean   :47.12
##  3rd Qu.:15.25   3rd Qu.:13.97   3rd Qu.:35.67   3rd Qu.:63.70
##  Max.   :20.00   Max.   :15.90   Max.   :48.50   Max.   :89.20
```

Option 3c: Method 3

```
#apply method
apply(X = datafm[,c(2:4)],MARGIN = 2,FUN = median)

##  IL6_preCOVID IL6_AdmnCOVID IL6_postCOVID
##         10.40         32.65         53.75

apply(X = datafm[,c(2:4)],MARGIN = 2,FUN = IQR)

##  IL6_preCOVID IL6_AdmnCOVID IL6_postCOVID
##         7.625        16.200        38.150

apply(X = datafm[,c(2:4)],MARGIN = 2,FUN = quantile)

##      IL6_preCOVID IL6_AdmnCOVID IL6_postCOVID
## 0%          2.300        14.500         15.50
## 25%         6.350        19.475         25.55
## 50%        10.400        32.650         53.75
## 75%        13.975        35.675         63.70
## 100%       15.900        48.500         89.20
```

1. In `Option 3a`, the "*median*" function is used, in `Option 3b`, the "*summary*" function is used for calculating the median, and in `Option 3c`, the "*apply*" function is used.
2. `The` "*summary*" function has an argument *object*, the input for which should be the R object assigned for the dataframe (*datafm*). It returns `Min`, `first Quartile`, `Median`, `Mean`, `third Quartile`, and `Max` as the output.
3. The "*apply*" function has the arguments *X*, *MARGIN*, and *FUN*. The *X* takes in an array, data frame, or matrix (in this case, *datafm*), and *MARGIN* takes in either 1 (row-wise) or 2 (column-wise) as value (in this case, 2). The *FUN* argument takes the function as input (in this case, *median*).
4. The median values at each time point are:

   Pre COVID-19 - Median (IQR) - 10.4 (6.350–13.975)
   Admission COVID-19 - Median (IQR) - 32.65 (19.475–35.675)
   Post COVID-19 - Median (IQR) - 53.75 (25.55–63.70)

Step 4: Analysis of Data: Friedman Test

The input for the Friedman test should be a matrix, where each column corresponds to a different time point, and each row corresponds to a paired observation across the groups. The Friedman test is designed for repeated measures or matched-pairs data.

```
#We do not require the first column X. So, we are removing the first column.
datafmsub <- subset(datafm, select = c(2, 3, 4))
#datafmsub

#Another way of doing the same thing is
datafmsub <- datafm[, -1]
#datafmsub

#The input for the Friedman test needs to be given as a matrix. I am so
converting the dataframe to a matrix.
datafmmat <- as.matrix(x = datafmsub)
#datafmmat

friedman.test(y = datafmmat)

##
##  Friedman rank sum test
##
## data:  datafmmat
## Friedman chi-squared = 40, df = 2, p-value = 2.061e-09
```

1. The **"*subset*" function is used to create a subset of a data frame based on specified conditions (selecting specified columns). Alternatively, indexing using square brackets can be performed to subset the data frame.**
2. The "*as.matrix*" function converts the dataframe into a matrix.
3. The "*friedman.test*" function is used for performing the Friedman test; the argument *y* takes the matrix as the input value (*datamat*).
4. The output for the Friedman test shows $p < 0.05$.

Step 5: Additional Information

1. Having found that the Friedman test was statistically significant, the next step is to conduct the post hoc test to perform pairwise comparisons.
2. The post hoc test for the Friedman test is the multiple pairwise Wilcoxon test. When we perform multiple comparisons, we increase the chance of committing the type 1 error. So, we apply Bonferroni correction for multiplicity adjustments.
3. We need to melt the variable present in separate columns into a single column for the computation of multiple pairwise Wilcoxon tests. We will use the *reshape2* package for this purpose [3].

```
require(reshape2)

## Loading required package: reshape2

## Warning: package 'reshape2' was built under R version 4.2.3

datafmmelt <- melt(datafmsub)

## No id variables; using all as measure variables

str(datafmmelt)

## 'data.frame':    60 obs. of  2 variables:
##  $ variable: Factor w/ 3 levels "IL6_preCOVID",..: 1 1 1 1 1 1 1
1 1 1 ...
##  $ value   : num  5.9 4.8 6.5 2.3 5.6 8.9 2.3 7.8 11.3 14.5 ...

attach(datafmmelt)
pairwise.wilcox.test(x = datafmmelt$value,
                     g = datafmmelt$variable,
                     p.adjust.method = "b",
                     paired = TRUE
                     )

## Warning in wilcox.test.default(xi, xj, paired = paired, ...): cannot
compute
## exact p-value with ties

## Warning in wilcox.test.default(xi, xj, paired = paired, ...): cannot
compute
## exact p-value with ties

## Warning in wilcox.test.default(xi, xj, paired = paired, ...): cannot
compute
## exact p-value with ties

##
##  Pairwise comparisons using Wilcoxon signed rank test with continuity
correction
##
## data:  datafmmelt$value and datafmmelt$variable
##
##                IL6_preCOVID IL6_AdmnCOVID
## IL6_AdmnCOVID  0.00029      -
## IL6_postCOVID 0.00029       0.00029
##
## P value adjustment method: bonferroni
```

1. The "*melt*" function is used to convert variables represented in separate columns (for different time points) into a single column and create a variable-value pair. The subsetted data frame in the previous step (*datafmsub*) is given as an input, and a new data frame with two columns, namely `value` and `variable`, is obtained as an output. We can observe that the first line of output for the melt function states, "`No id variables; using all as measure variables.`" This shows the essence of removing the first column from the original data set.

2. The "*pairwise.wilcox.test*" is applied to compute the pairwise Wilcoxon test. The *x* argument of "*pairwise.wilcox.test*" is given an input of numeric vector (`datafmmelt$value`), and *g* argument of "*pairwise.wilcox.test*" function needs grouping variable as input (`datafmmelt$variable`).
3. The *p.adjust.method* argument of "*pairwise.wilcox.test*" needs an input of the adjustment method to address multiple comparisons (**b** for Bonferroni). The *paired* argument takes in logical operators as values. If the comparisons are made between paired samples, then the input should be *TRUE*.

Final Conclusion

The IL6 between the three different periods was statistically significantly different, with a p-value of <0.001. The median score of the IL6 pre-COVID-19, admission COVID-19, and post-COVID-19 were 10.4 (6.350–13.975), 32.65 (19.475–35.675), and 53.75 (25.55–63.70), respectively.

Chi-Square Test

The chi-square test is a statistical test used to determine whether the distribution of categorical variables is the same or different across different groups. It is specifically applied when comparing the distribution of a categorical variable in two or more independent groups.

Assumptions Made

1. The samples are independent.
2. The categorical variable is measured on a nominal scale.
3. The expected frequency for each cell is greater than 5.

Hypothesis

Null hypothesis: The distribution of the categorical variable is the same across all groups.

Alternate hypothesis: The distribution of the categorical variable is different across at least two groups.

Case Scenario

A randomized control trial was conducted to assess the efficacy of Drug A, Drug B, and placebo in resolving lesions in psoriasis. The principal investigator recruited a total of 120 patients (40 patients in each group), and the study drug was applied weekly for 2 months. The total observation period was 3 months, including the period of intervention. At the end of the observation period, the lesions were assessed for disappearance or persistence in each of the participants enrolled in the three groups. The data has been provided as "*Chisq.csv*". It consists of two variables: Intervention (Placebo, A and B) and Response (Present or Absent).

Step 1: Importing of Data into the R Console

```
datacs <- read.csv("Chisq.csv")
str(datacs)

## 'data.frame':    120 obs. of  2 variables:
##  $ intervention: chr  "Placebo" "A" "Placebo" "Placebo" ...
##  $ Response    : chr  "Present" "Present" "Present" "Absent" ...

table(datacs$Intervention)

##
##       A       B Placebo
##      40      40      40
```

Step 2: Calculation of Frequency of Events

The frequency table containing the counts of the dichotomous outcome for each intervention is generated as follows.

```
datacstable <- table(datacs)
datacstable

##             Response
## Intervention Absent Present
##      A           15      25
##      B           28      12
##      Placebo     17      23
```

1. The "`table`" function is used to create a contingency table, which is a tabular representation of the counts of different categories or combinations of variables. The table function takes one or more factors as input and returns a table showing the counts of observations for each combination of factor levels. In this case, it shows the frequency of the presence and absence of lesions in each of the three intervention groups in the *datacs* dataset.

Step 3: Computation of Chi-Square Statistic

```
chiresult <- chisq.test(datacstable)
chiresult

##
##  Pearson's Chi-squared test
##
## data:  datacstable
## X-squared = 9.8, df = 2, p-value = 0.007447
```

1. The "*chisq.test*" function is used to perform the chi-square test. It takes the contingency table as the argument (datacstable), and the output is assigned to an R object *chiresult*.
2. The *chiresult* is the R object that stores the result of the chi-square test. The *chiresult* can be used to access various components of the test result, such as the expected and observed frequency, chi-square statistic, degrees of freedom, and the p-value.

Step 4: Checking for Assumptions

Unlike other tests, where assumptions are checked a priori, the assumption of adequate counts in each cell is tested once the test is already run to ensure the validity of the results obtained.

```
chiresult$expected

##             Response
## Intervention Absent Present
##      A           20      20
##      B           20      20
##      Placebo     20      20

#Round the values for better conceptualization
round(x = chiresult$expected,digits = 0)

##             Response
## Intervention Absent Present
##      A           20      20
##      B           20      20
##      Placebo     20      20

#As all the values in the expected table are greater than 5, we ensure
that the chi-square test result is valid.

chiresult$observed
```

```
##             Response
## Intervention Absent Present
##      A           15      25
##      B           28      12
##      Placebo     17      23
```

1. The *expected*(using *chiresult$expected*) and *observed*(using *chiresult$observed*) values are extracted from the *chiresult* list.
2. The "*round*" function is used to round off the result. It takes in two arguments, namely *x* and *digits*). The *x* takes in a numeric vector as input, and *digits* take in the number of digits to which the numeric values need to be rounded off.
3. The results are valid, and as p = 0.007, there is a significant difference between at least any of the between-group comparisons, which needs to be identified using post-hoc tests.

Step 5: Posthoc Test

```
#Posthoc test
library(chisq.posthoc.test)

## Warning: package 'chisq.posthoc.test' was built under R version 4.2.3

chisq.posthoc.test(datacstable, method = "bonferroni")

##   Dimension     Value    Absent   Present
## 1         A Residuals -1.936492  1.936492
## 2         A  p values  0.316845  0.316845
## 3         B Residuals  3.098387 -3.098387
## 4         B  p values  0.011675  0.011675
## 5   Placebo Residuals -1.161895  1.161895
## 6   Placebo  p values  1.000000  1.000000
```

1. When we obtain a $p < 0.05$ in the chi-square test for 2 x c table, where ($c > 2$), we should further perform post-hoc analysis to identify where the difference lies.
2. We have a dedicated package *chisq.posthoc.test* for post hoc analysis of the chi-square test [4].
3. The function "*chisq.posthoc.test"* is used for post hoc analysis. The arguments are the R object assigned to the table created for the chi-square test and the method for adjusting the p-value. Here, we have used the Bonferroni method.
4. The p-value (0.016) in the Drug B group is significant ($p < 0.05$).

Step 6: Accompanying Calculation

To report the association between two categorical variables, we need to calculate the odds ratio for reporting purposes. The chi-square test or Fisher exact test is for hypothesis testing, while the odds ratio is calculated as a measure of effect size. We will calculate the odds ratio using the *epitools* package [5].

```
# Computation of odds ratio with 95% C.I.

# epitools package is used to compute the odds ratio.

if (require("epitools") == TRUE) {
  library(epitools)
} else {
  install.packages("epitools")
}

## Loading required package: epitools

#Method 1
epitools::epitab(x = datacstable,rev = "both",method = "oddsratio",oddsratio
= "wald")

## $tab
##             Response
## Intervention Present        p0 Absent        p1 oddsratio
lower    upper
##      Placebo      23 0.3833333     17 0.2833333 1.0000000
NA       NA
##      B            12 0.2000000     28 0.4666667 3.1568627 1.2554061
7.938294
##      A            25 0.4166667     15 0.2500000 0.8117647 0.3313751
1.988568
##             Response
## Intervention    p.value
##      Placebo         NA
##      B       0.02357552
##      A       0.81970137
##
## $measure
## [1] "wald"
##
## $conf.level
## [1] 0.95
##
## $pvalue
## [1] "fisher.exact"

# 4 available methods of determination of 95% C.I. for oddsratio are `wald`,
`fisher`, `midp`, and `small`. The default method is `wald`.
```

```
#Method 2
epitools::oddsratio(datacstable,correction = TRUE,rev = "both")

## $data
##             Response
## Intervention Present Absent Total
##      Placebo      23     17    40
##      B            12     28    40
##      A            25     15    40
##      Total        60     60   120
##
## $measure
##             odds ratio with 95% C.I.
## Intervention  estimate     lower    upper
##      Placebo 1.0000000        NA       NA
##      B       3.0936397 1.2412128 8.056188
##      A       0.8146731 0.3270204 2.012907
##
## $p.value
##             two-sided
## Intervention midp.exact fisher.exact chi.square
##      Placebo         NA           NA         NA
##      B       0.01501369   0.02357552 0.02421223
##      A       0.65679626   0.81970137 0.81947698
##
## $correction
## [1] TRUE
##
## attr(,"method")
## [1] "median-unbiased estimate & mid-p exact CI"
```

Final Conclusion

At the end of 3 months of therapy, 15 psoriasis patients(36%) in the Drug A, 28 patients (70%) in the Drug B group, and 17 patients(42%) in the placebo group responded to the therapy. The difference in proportions was statistically significant, p = 0.007. The odds ratio disappearance of lesions in Drug A versus placebo was 0.81(95% CI: 0.33 to 1.98; p = 0.819), and that for Drug B versus placebo was 3.15 (95% CI: 1.25 to 7.93; p = 0.023).

Step 6: Additional Information

If the expected frequency for each cell is less than 5, the chi-square test does not remain valid, and we need to perform the Fisher exact test.

Subdiv 1—Fisher Exact Test

```
#Fisher exact
# what to do if the expected count in any one of the cell is less than 5.
# Go for Fisher exact test. How will you do that?
# Let's learn with an example.
datafr <- matrix(c(5, 4, 4, 5),
                    nrow = 2,
                    ncol = 2,
                    byrow = TRUE)
datafr <- as.table(datafr)
chisq.test(datafr)

## Warning in chisq.test(datafr): Chi-squared approximation may be incorrect

##
##  Pearson's Chi-squared test with Yates' continuity correction
##
## data:  datafr
## X-squared = 0, df = 1, p-value = 1
```

1. We have created a 2 X 2 matrix using the function "*matrix",* and we have further transformed it into a table using the function "*as.table"*.
2. Next, we run a chi-square test on the same.
3. Please note that you are getting a warning for the chi-square test. Why is this warning occurring??
4. Let us check assumptions by calculating the expected frequencies for the contingency table.

```
chisq.test(datafr)$expected

## Warning in chisq.test(datafr): Chi-squared approximation
may be incorrect

##     A   B
## A 4.5 4.5
## B 4.5 4.5

round(chisq.test(datafr)$expected)

## Warning in chisq.test(datafr): Chi-squared approximation
may be incorrect

##   A B
## A 4 4
## B 4 4
```

5. We can see that the expected frequency in all the cells is less than 5. Thus, we should perform Fisher's exact test on the given data.

```
#Let's do the fisher exact test and please note that there
are no warning now.
fisher.test(datafr)

##
##  Fisher's Exact Test for Count Data
##
## data:  datafr
## p-value = 1
## alternative hypothesis: true odds ratio is not equal to 1
## 95 percent confidence interval:
##   0.1736323 14.4729449
## sample estimates:
## odds ratio
##   1.523879
```

6. The *"fisher.test"* function is used to perform the test, and the results show that there is no significant difference between the two groups.

Subdiv 4—Proportion Computation

```
#Computation proportion:
# We need to find the proportion of the dichotomous variable in each
independent variable to understand the chi-square test result to a deeper
extent

#Convert the table to dataframe
datacsframe <- as.data.frame(datacstable)
datacsframe

##   Intervention Response Freq
## 1            A   Absent   15
## 2            B   Absent   28
## 3      Placebo   Absent   17
## 4            A  Present   25
## 5            B  Present   12
## 6      Placebo  Present   23

#Create a new column known as proportion
datacsframe$Proportion <- datacsframe$Freq
datacsframe
```

```
##   Intervention Response Freq Proportion
## 1            A   Absent   15         15
## 2            B   Absent   28         28
## 3      Placebo   Absent   17         17
## 4            A  Present   25         25
## 5            B  Present   12         12
## 6      Placebo  Present   23         23

# As the total number of patients in each individual group is 40, we can
divide the newly formed column "proportion" by the common number 40.
datacsframe$Proportion <- datacsframe$Proportion / 40
datacsframe

##   Intervention Response Freq Proportion
## 1            A   Absent   15      0.375
## 2            B   Absent   28      0.700
## 3      Placebo   Absent   17      0.425
## 4            A  Present   25      0.625
## 5            B  Present   12      0.300
## 6      Placebo  Present   23      0.575

# If the total number in each group was not equal, we could have divided by
the corresponding total number in each group by equating the proportion
column to that of the intervention using a conditional operator.
# datacsframe$Proportion[datacsframe$Intervention == "Drug A_Cream"] <-
#   datacsframe$Proportion / 40
# datacsframe$Proportion[datacsframe$Intervention == "Drug B_Cream"] <-
#   datacsframe$Proportion / 40
# datacsframe$Proportion[datacsframe$Intervention == "Placebo"] <-
#   datacsframe$Proportion / 40
#datacsframe
```

With the above codes, we can calculate frequencies and proportions for the presence and absence of response in each drug group.

Mcnemar Test

McNemar's test is a statistical test used to determine whether there is a significant difference between two paired proportions. It is specifically designed for analyzing paired nominal or categorical data in a 2 × 2 contingency table. McNemar's test is often applied in situations where the same subjects are measured or observed at two different points in time or under two different conditions.

Assumptions Made

1. The data must be paired or matched.
2. The dependent variable is dichotomous or binary.
3. The pairs are independent of each other.

Hypothesis

Null Hypothesis: There is no difference in the proportions between the two related groups.

Alternate Hypothesis: There is a significant difference in the proportions between the two related groups.

Case Scenario

The hospital selected a group of 50 residents for training in performing CPR. A pre-training and post-training test was conducted by administration of a 100-point questionnaire that assessed residents' knowledge. The cut-off score for designating a resident knowledgeable in CPR was kept at 60. Those individuals who scored more than 60 were classified as knowledgeable, and those less than 60 were classified as not knowledgeable. The aim was to compare the knowledgeability before and after the CPR training.

```
tablemacne<-matrix(data = c(10,40,20,30),nrow = 2,ncol = 2,byrow =
FALSE,dimnames = list(c("knowledgeable","Not knowledgeable"),c("Before CPR
training","After CPR training")))
tablemacne

##                   Before CPR training After CPR training
## knowledgeable                      10                 20
## Not knowledgeable                  40                 30

mcnemar.test(x = tablemacne)

##
##  McNemar's Chi-squared test with continuity correction
##
## data:  tablemacne
## McNemar's chi-squared = 6.0167, df = 1, p-value = 0.01417
```

1. We have created a matrix using the "*matrix*" function, and this matrix is given as an input to the "*mcnemar.test*" function.
2. We can appreciate that there was a significant increase in knowledgeable residents as measured by the CPR questionnaire (p = 0.014).

Additional Information

The Cochrane Q test is an extension of the Mcnemar test if more than two-time points of data are present. Cochrane Q is an alias for the Friedman test. Friedman's test is used for the continuous variable, whereas the Cochrane Q is for the

categorical variable. In the above example, suppose the trainer assessed in the middle of the training. The trainer then wishes to compare the knowledge among participants across the three-time CPR training points; the Cochrane Q test would qualify for assessment. The "`cochran.qtest`" function within the `RVAideMomoire` package can be used for the calculation. It is suggested that the readers self-explore the function further.

Cochran Mantel Haenszel Test

The Cochran-Mantel-Haenszel (CMH) test is a statistical test used to assess the association between two categorical variables while controlling for the effects of one or more confounding variables. It is particularly useful when analyzing data in stratified contingency tables. The CMH test is an extension of the Mantel-Haenszel test, allowing for the analysis of data in multiple strata.

Assumptions Made

1. The data are collected in strata (groups) based on a third variable.
2. The relationship between the two categorical variables remains constant across strata.
3. The dependent variable is dichotomous or binary.

Hypothesis

Null Hypothesis: There is no association between the two categorical variables after adjusting for the effects of the confounding variable(s) (no stratum-specific association).

Alternate Hypothesis: There is a significant association between the two categorical variables even after adjusting for the effects of the confounding variable(s).

Case Scenario

In a clinical pharmacology experiment on healthy human volunteers, an investigator evaluated the efficacy of Dexamethasone compared to that of placebo in protecting the participant from the appearance of wheal following injection of an allergic agent. First, the investigator injected the study drug (Dexamethasone or Placebo), and 10 min later, the allergic agent was injected into the forearm. The study was planned as a double-blind, randomized multicenter trial involving five government institutes in India. The outcome evaluated was the number of patients in whom

wheal did not appear. The study was conducted to check if the odds of non-appearance of allergic reaction with allergic agents are similar between the intervention groups, Dexamethasone, and placebo, across study sites.

```
# The dataset which Cochran Mantel Haenszel fuction takes is in the form of
array object.
# Hence the data is entered in the form of array
# The vector supplied to the data argument of array could be imported
separately from a excel file, if you wish to avoid errors during entry.

MHData <-
array(data = c(5, 40, 45, 10,
       6, 45, 44, 5,
        8, 49, 42, 1,
        3, 48, 47, 2,
        4, 44, 46, 6),
      dim = c(2, 2, 5),
      dimnames = list(
          Drug = c("Dexa", "Placebo"),
          Response = c("Appeared", "None"),
          Institution = c("Institute 1", "Institute 2", "Institute 3",
"Institute 4", "Institute 5")))
```

1. The "*array*" function is used to create an array. As the data contains three dimensions: Study groups, Institutes, and Response; an array needs to be created.

```
## Classical Mantel-Haenszel test
mantelhaen.test(x = MHData,alternative = "two.sided",conf.level = 0.95)

##
##  Mantel-Haenszel chi-squared test with continuity correction
##
## data:  MHData
## Mantel-Haenszel X-squared = 315.66, df = 1, p-value < 2.2e-16
## alternative hypothesis: true common odds ratio is not equal to 1
## 95 percent confidence interval:
##  0.006250748 0.021759188
## sample estimates:
## common odds ratio
##        0.01166238

# ## Exact conditional test
# mantelhaen.test(MHData, exact = TRUE)
```

1. The "*mantelhaen.test*" function is used to perform the test.
2. The *x* argument in "*mantelhaen.test*" function takes the argument as an array. In the array, the first and second dimensions should correspond to data and the third dimension should correspond to *strata*.

3. Other (optional) arguments which can be passed to "`mantelhaen.test`" function are `alternative`, `correct`, `exact`, *and* `conf.level`. The `correct` argument takes logical operator as input: *TRUE* if continuity correction needs to be applied, *FALSE* if continuity correction need not be applied, `alternative` (takes in one among the following as value: *two.sided*, *greater* or *less*), `exact` (logical operator as value: if *TRUE* an exact test is performed) and `conf.level` takes in confidence interval value as input.

Final Conclusion

The common odds ratio as assessed by the Cochran Mantel Haenszel test is 0.01 with a p-value of <0.001. This means that the odds ratio of the appearance of allergic reaction following administration of an allergic agent with Dexamethasone pre-intervention is 0.01 times as compared to that of placebo across all the study sites.

References

1. Nakazawa M. pyramid: draw population pyramid. R package version 1.5. 2019. https://CRAN.R-project.org/package=pyramid.
2. Dinno A. dunn.test: Dunn's Test of Multiple Comparisons Using Rank Sums. R package version 1.3.5. 2017. https://CRAN.R-project.org/package=dunn.test
3. Wickham H. Reshaping data with the reshape package. J Stat Softw. 2007;21(12):1–20. http://www.jstatsoft.org/v21/i12/
4. Beasley M, Schumacker RE. Perform posthoc analysis based on residuals of Pearson's Chi squared Test for Count Data based on T. 1995. https://doi.org/10.1080/00220973.1995.9943797.
5. Aragon TJ. Epitools: epidemiology tools. R package version. 0.5–10.1. 2020. https://CRAN.R--project.org/package=epitools.

Chapter 9
Computation of Sample Size for Clinical Studies

Anand Srinivasan, Rituparna Maiti, and Archana Mishra

Introduction to Sample Size in Clinical Research

As we already discussed, research is done to test a hypothesis by using appropriate statistical tests. Ideally, observations from the entire population are required to arrive at a conclusion regarding the hypothesis. However, this is an impossible task except in the census, where data from every individual is recorded. Thus, each study is conducted on a sample drawn from the concerned population. A sample is a set of participants (or data) selected from the population, less in number (size), but adequately represents the population from which it is drawn so that true inferences about the population can be made from the results obtained. The important thing to understand here is how big this sample should be. If the sample drawn is too small, the study may fail to answer its research question, even if it is executed rigorously. The study may fail to detect important effects or may estimate too imprecisely. On the other hand, a sample size that is too large may lead to compromises in data quality and make the study difficult and wasteful of resources.

What Is the Sample Size? [1]

The sample size is the appropriate number of observations required to obtain statistical significance. Sample size determination, which can answer the research question, is one of the first steps in planning and executing clinical research. There are four important elements for the determination of sample size.

A. Srinivasan (✉) · R. Maiti · A. Mishra
Department of Pharmacology, All India Institute of Medical Sciences, Bhubaneswar, Odisha, India
e-mail: anandsrinivasan@aiimsbhubaneswar.edu.in

A. Srinivasan et al. (eds.), *R for Basic Biostatistics in Medical Research*,
https://doi.org/10.1007/978-981-97-6980-3_9

Table 9.1 Effect of change of various determinants of sample size on sample size

Name of the parameter	Direction of change in parameter	Sample size
Effect size	⇩	⇧
Variance/standard deviation	⇩	⇩
Alpha error	⇩	⇩
Beta error	⇩	⇩

Factors Influencing Sample Size (Table 9.1)

1. Effect size (e.g., Mean difference): The smallest difference (between the participating groups) that is clinically meaningful is taken as effect size.
2. Variance of effect size: This is the variability associated with the effect size.
3. Alpha error: This is the probability of rejecting a null hypothesis when it is true, i.e., it is the probability of finding a difference when there is none. This represents a false positive error.
4. Beta error (determines the power of the study): This is the probability of failing to reject a null hypothesis when it is false. This represents a false negative error.

A trade-off between the four determinants is required to obtain an optimal sample size for the conduct of a clinical study. Sample size can be calculated using the precision approach and power approach. In the power approach, we focus on the smallest difference, which can be clinically relevant to detect. In contrast, in the precision approach, we want to fix the width of the confidence interval of a parameter (e.g., incidence or prevalence) for which we want to determine the sample size.

Though the basic concept of sample size calculation remains the same, the formula for sample size calculation differs based on design considerations and effect estimates of the objectives outlined in the study. Discussing individual formulas for sample size calculation is beyond the scope of this book. We will focus on explaining codes for calculating sample size for each type of study.

Sample Size Calculation for Mean

Sample Size Determination for Detecting the Difference in Continuous Outcome Between Two Groups (Unpaired Data)

Consider the following scenario: A study has been designed to determine the difference in the outcome measured as a continuous variable between two groups at the end of the follow-up period. For example, the study is designed to determine if the difference in the level of a biomarker between two groups at the end of the follow-up period is significant or not. As per the principles of sample size determination, we need to identify effect size, variance, alpha, and beta error. For this example, we will try to calculate the sample size using a formula and dedicated R packages.

Let us calculate the sample size required per group to detect a mean difference of 4 for an outcome with a pooled standard deviation of 6. A two-sided test is performed, allowing a false positive rate of 5% (alpha—0.05). The sample should be powerful enough to detect the difference 80% of the time (beta—0.2).

```
####Formula approach
#n = number needed per group (continuous variable)
m1 = 10
m2 = 6
sd = 6
ef = (m1-m2)^2
n = 2*((qnorm(0.975,0,1)+(qnorm(0.8,0,1)))^2)*((sd^2)/ef)
cat("The sample size required per group is",n)
## The sample size required per group is 35.31996
```

1. The values m1 and m2 indicate means in the two groups, and sd is the standard deviation. The function *"qnorm"* returns the z value for the numbers included in the function. We have used *"qnorm"* to determine the z values of alpha and beta.
2. In the above example, we have coded the formula directly.
3. The *"cat"* function converts arguments to characters or strings and prints them, which we can understand from the output.

We can perform this task easily using packages dedicated to the calculation of sample size. The two most relevant packages for the calculation of sample size are:

1. `TrialSize` [2]
2. `Pwr` [3]

Functions from other packages like *epiR4* and *pROC5* are also used.

Let us load all the essential packages for further operations

```
library(TrialSize)
## Warning: package 'TrialSize' was built under R version 4.3.0
library(pwr)
## Warning: package 'pwr' was built under R version 4.3.1
library(epiR)
## Warning: package 'epiR' was built under R version 4.2.3
## Loading required package: survival
## Package epiR 2.0.66 is loaded
## Type help(epi.about) for summary information
## Type browseVignettes(package = 'epiR') to learn how to use epiR for
applied epidemiological analyses
##
library(pROC)
## Warning: package 'pROC' was built under R version 4.3.1
## Type 'citation("pROC")' for a citation.
##
## Attaching package: 'pROC'
## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
```

```
Package approach
n = TwoSampleMean.Equality(alpha = 0.05,beta = 0.2,sigma = 6,
k=1,margin = 4)
cat("The sample size required per group is",n)
## The sample size required per group is 35.31996
```

The arguments for the "*TwoSampleMean.Equality*" function require inputting the alpha, beta, pooled standard deviation (sigma), allocation ratio (k), and mean difference (margin), which would provide the sample size upon running the function.

From now on, we will follow the package approach to calculate sample sizes in this chapter.

Sample Size to Detect a Paired Mean Difference

Let us calculate the sample size for a single-arm study to detect a mean change of 20 from the baseline in a biomarker and pooled standard deviation of 40. A two-sided test is performed, allowing a false positive rate of 5% (alpha—0.05). The sample should be powerful enough to detect the difference 80% of the time (beta—0.2).

```
n = pwr.t.test(n = NULL,d = (20/40),sig.level = 0.05,power = 0.8,type =
"paired",alternative = "two.sided")
cat("The sample size required is",n[[1]])
## The sample size required is 33.36713
```

In the above function "*pwr.t.test*", the argument *d* is the ratio of the mean difference to pooled variance. The *type* can be chosen to be paired or unpaired based on the t-test to be applied for analysis later.

The Sample Size for a Single Mean

Let us calculate the sample size for a study where we want to compare the mean weight of babies born to mothers with Iron deficiency anemia with the population mean. We want to detect a difference of 2 and a standard deviation of 4. A two-sided test is performed, allowing a false positive rate of 5% (alpha—0.05). The sample should be powerful enough to detect the difference 80% of the time (beta—0.2).

```
n = OneSampleMean.Equality(alpha = 0.05,beta = 0.2,sigma = 4, margin = 2)
cat("The sample size required is",n)
## The sample size required is 31.39552
```

The function *"OneSampleMean.Equality"* requires arguments for alpha error, beta error, the mean difference between sample and population means (margin), and standard deviation (sigma).

Sample Size to Detect Mean Difference for More Than Two Groups

Let us calculate the sample size to detect a minimum difference of 4 between three groups, a standard deviation of 10 at a two-sided significance level of 5%, and the power of the study to be 80%.

```
n = OneWayANOVA.pairwise(alpha = 0.05,beta = 0.2,tau = 3,sigma = 10,
margin = 4)
cat("The sample size required per group is",n)
## The sample size required per group is 130.8639
```

1. As we should perform an ANOVA on the data to draw conclusions, we use the *"OneWayANOVA.pairwise"* function. This can be used for three or more groups. The rest of the arguments are similar to previous codes except for an additional argument *tau*.
2. In the above example, *tau* indicates the number of comparisons possible. The number of possible comparisons can be determined by using the $n(n-1)/2$ formula, where n = number of groups. Thus, in the above scenario, we need to calculate the sample size for comparison between three groups, and thus 3*2/2 = 3 comparisons. The *tau* value obtained is 3.

Sample Size Using Confidence Interval Approach

Let us assume a scenario where we need to measure a parameter like systolic blood pressure (continuous outcome) in a population. The sample size required to measure the blood pressure in the population depends on the confidence interval width desired. For example, if we need to measure the blood pressure and the accepted width of 95% confidence interval is 10, provided the standard deviation of the population is 15, then it can be determined as follows.

```
ncim = function(CI, width, sd){
  z = 1-((1-CI)/2)
  n = (4*((qnorm(z,0,1))^2)*sd^2)/(width^2)
  return(n)
}
n = ncim(CI = 0.95,width = 10,sd = 15)
cat("The sample size required is",n)
## The sample size required is 34.57313
```

1. We learned how to create our own function in earlier chapters. Here, we are creating a function "*ncim*".
2. The components of the function are (a) CI: Confidence interval (CI), (b) width: The desired width of the confidence interval, and (c) sd: The standard deviation of the population.
3. This function uses the formula for the determination of sample size in estimating mean from a normal distribution, i.e., n = (4 * z^2 * (SD)^2)/(width)^2, where z is the z score corresponding to half of the confidence interval.

We will explore the relation of change in confidence interval (for fixed standard deviation) with sample size.

Consider a sample mean (sm) of 20 for a parameter measured in plasma with a standard deviation (sd) of 40. The population mean for the same parameter is 40. The following code demonstrates the effect of confidence intervals on the sample size required to demonstrate a difference between the sample and population mean with statistical significance.

```
n = 6
sd = 40
se_x = sd/sqrt(n)
se_y = sd/sqrt(n)
sm = 20
poolse = sqrt(((se_y^2)+(se_x^2))/2)
a = seq(-40,60,length=601)
m = dnorm(a,sm,poolse)
plot(a,m,xlim = range(-40,60),ylim = range(0,0.06),type = "l",
     xlab = "Parameter value",ylab = "Probability")

no = c(7,11,18,28,40,50)
for (i in 1:length(no)){
  mx = 160
  my = 180
  n = no[i]
  se_x = sd/sqrt(n)
  se_y = sd/sqrt(n)
  sm = 20
  poolse = sqrt(((se_y^2)+(se_x^2))/2)
  a = seq(-40,60,length=601)
  m = dnorm(a,sm,poolse)
  lines(a,m,type = "l")}
abline(v = 40, col = "red")
```

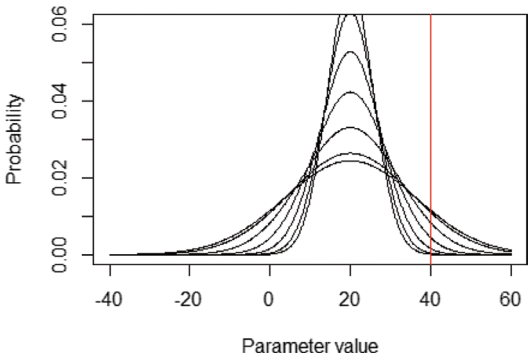

Fig. 9.1 Plot demonstrating the relationship between the confidence interval (due to varying sample size) and the sample size required to demonstrate a significant difference between the sample mean and population mean

It can be observed that as the sample size (in each group) increases through 6, 7, 11, 18, 28, 40, and 50, the width of the bell curve decreases (Fig. 9.1). This indicates the confidence interval decreases as the sample size increases. With the above-mentioned parameters, to detect a difference of 20, it is evident from the above graph that a sample size of 28 or above would be sufficient to demonstrate the significant difference between the sample and population mean.

We shall now explore the impact of the change in standard deviation (with a fixed sample size) on the confidence interval for the same parameter. Let us consider a scenario similar to the one mentioned above. In the previous scenario, we kept the standard deviation constant with varying sample sizes, whereas, for this scenario, we will keep the sample size constant with varying standard deviation.

```
n = 40
sd = c(20,30,40,50,60,70)
se_x = sd[1]/sqrt(n)
se_y = sd[1]/sqrt(n)
poolse = sqrt(((se_y^2)+(se_x^2))/2)
a = seq(10,50,length=601)
m = dnorm(a,20,poolse)
plot(a,m,xlim = range(10,50),ylim = range(0,0.15),type = "l",
     xlab = "Parameter value",ylab = "Probability")

for (i in 2:length(sd)){
  sdv = sd[i]
  se_x = sdv/sqrt(n)
  se_y = sdv/sqrt(n)
  poolse = sqrt(((se_y^2)+(se_x^2))/2)
  a = seq(10,50,length=601)
  m = dnorm(a,20,poolse)
  lines(a,m,type = "l")
}
abline(v = 40, col = "red")
```

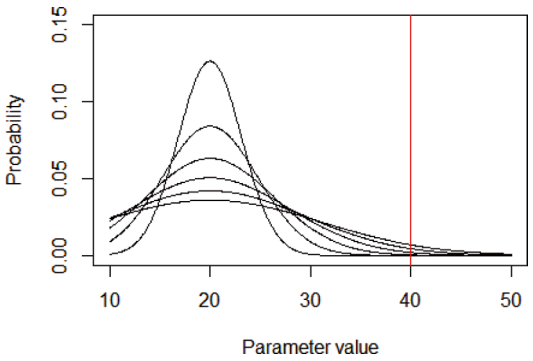

Fig. 9.2 Plot demonstrating the relationship between the confidence interval (due to varying standard deviation) and the sample size required to demonstrate a significant difference between the sample mean and population mean

In Fig. 9.2, we can observe that when the sd increases through 20, 30, 40, 50, 60, and 70 for the fixed sample size of 40, the width of the bell curve increases. To detect a significant difference of 20, the standard deviation of the sample parameter should be 20 or less.

Sample Size Calculation for Proportion

Calculation of Sample Size to Detect the Difference Between Two Proportions

Calculate the sample size to detect a difference between two proportions for a categorical outcome where the proportion for one group is 0.4 and the other is 0.6. A two-sided test is performed, allowing a false positive rate of 5% (alpha—0.05). The sample size should be powerful enough to detect the difference 80% of the time (beta—0.2).

```
n = TwoSampleProportion.Equality(alpha = 0.05,beta = 0.2,p1 = 0.4,p2 = 0.6,k
= 1)
cat("The sample size required per group is",n)
## The sample size required per group is 94.18656
```

The information required for the calculation is an alpha error, beta error, proportion in group 1, proportion in group 2, groups, and the allocation ratio.

Calculation of Sample Size for Odds Ratio

Let us calculate the sample size for a study to detect the association between smoking and pregnancy outcomes. We intend to detect an odds ratio of 2, where exposure in the control group was 40%. A two-sided test is performed, allowing a false positive rate of 5% (alpha—0.05). The sample should be powerful enough to detect the difference 80% of the time (beta—0.2).

```
#library(epiR)
n = epi.sscc(OR = 2,p0 = 0.4,n = NA,power = 0.8,r = 1, sided.test =
2,conf.level = 0.95,method = "unmatched")
cat("The sample size required per group is",n[[2]])
## The sample size required per group is 133
```

1. This function requires *epiR* package.
2. The *"epi.ccsize"* function requires arguments for the prevalence of exposure amongst the controls (p0), the expected odds ratio (OR), (allocation ratio = number in controls/number in cases (r)), the method can be chosen as matched or unmatched based on the design of the study.

Calculation of Sample Size for a Single Proportion

Let us calculate the sample size to determine if the percentage of stillbirths in underweight mothers (20%) is equal to or different from that of stillbirths in the population in an underdeveloped country (30%).

```
n = OneSampleProportion.Equality(alpha = 0.05,beta = 0.2,p = 0.3,
differ = -0.1)
cat("The sample size required is",n)
## The sample size required is 164.8265
```

1. The function *"OneSampleProportion.Equality"* uses arguments to specify the response rate in the sample and the difference between sample and population response rates.

Calculation of Sample Size for Proportion Using Confidence Interval Approach

```
ncip = function(CI, width, p){
  z = 1-((1-CI)/2)
  n = (4*(qnorm(z,0,1)^2)*p*(1-p))/(width^2)
  return(n)
}
n = ncip(CI = 0.95,width = 0.1,p = 0.8)
cat("The sample size required is",n)
## The sample size required is 245.8534
```

Similar to the function created to calculate sample size for the continuous outcome in confidence interval approach, we have created a function for proportions (categorical outcome) as well.

Sample Size Calculation for Correlation Studies

Let us calculate the sample size to study a minimum correlation of 0.5 between the gestational age of the mother and the birth weight of newborn babies. A two-sided test is performed, allowing a false positive rate of 5% (alpha—0.05). The sample should be powerful enough to detect the difference 80% of the time (beta—0.2). This study has been designed to show a correlation between two variables. If the expected correlation is known, the sample size can be calculated as follows.

```
n = pwr.r.test(n = NULL,r = 0.5,sig.level = 0.05,power = 0.8)
cat("The sample size required is",n[[1]])
## The sample size required is 28.24841
```

1. The function *"pwr.r.test"* uses arguments for expected correlation (r).

Sample Size Calculation for Diagnostic Studies

Sensitivity and specificity of a new test determine the diagnostic accuracy of a test. Plotting sensitivity versus specificity gives us a receiver operating characteristics (ROC) curve, and the area under the curve (AUC) indicates the accuracy of differentiating cases and controls. This area is used to calculate the sample size required in diagnostic studies.

Let us calculate the sample size required to establish the diagnostic accuracy of a new test with an AUC of 0.65. A two-sided test is performed, allowing a false positive rate of 5% (alpha—0.05). The sample should be powerful enough to detect the difference 80% of the time (beta—0.2) between the two groups.

```
n = power.roc.test(auc = 0.65,ncontrols = NULL,nases = NULL,
                   sig.level = 0.05,power = 0.8,k=1)
cat("The sample size required per group is",n[[1]])
## The sample size required per group is 54.55418
```

In the function *"power.roc.test"* appropriate values are assigned to the arguments except for *ncases* and *ncontrols*. The last two arguments mentioned are specified "NULL" to calculate the sample size. The argument *k* is the allocation ratio between the groups.

References

1. Nayak BK. Understanding the relevance of sample size calculation. Indian J Ophthalmol. 2010;58(6):469–70.
2. Zhang E, Wu VQ, Chow S-C, Zhang HG. TrialSize: R functions for chapter 3,4,6,7,9,10,11,12,14,15 of sample size calculation in clinical research. R package version 1.4. 2020. https://CRAN.R-project.org/package=TrialSize
3. Champely S. pwr: Basic functions for power analysis. R package version 1.3-0. 2020. https://CRAN.R-project.org/package=pwr
4. Stevenson M, Sergeant E, Firestone S. epiR: tools for the analysis of epidemiological data. R package version 2.0.66. 2023. https://CRAN.R-project.org/package=epiR
5. Robin X, Turck N, Hainard A, et al. pROC: an open-source package for R and S+ to analyze and compare ROC curves. BMC Bioinformatics. 2011;12:77.

Chapter 10
Correlation and Linear Regression Analysis for Continuous Outcome

Inderjeet Singh and Archana Mishra

Introduction to Correlation Analysis [1]

In daily life, the word correlation is used to denote some form of connection, such as the quality of the pages of a book and its price. However, in statistical terms, correlation analysis is a bivariate analysis and is used to assess the strength of association between two quantitative variables, which is denoted by the correlation coefficient (r). The value of the correlation coefficient ranges between −1 to +1. A value closer to −1 denotes a strong negative correlation, i.e., the value of one variable decreases with an increase in the other. A value closer to +1 denotes a strong positive correlation, i.e., the value of one variable increases with an increase in the other, while a value of zero (0) denotes no correlation. However, it should be kept in mind that correlation does not imply causation. Additionally, the absence of a linear relationship does not rule out a nonlinear association. The three most common types of correlation tests are:

1. Pearson's product-moment correlation test
2. Kendall's correlation test
3. Spearman's correlation test

Supplementary Information The online version contains supplementary material available at https://doi.org/10.1007/978-981-97-6980-3_10.

I. Singh
Experimental Drug Development Centre, Singapore, Singapore

A. Mishra (✉)
Department of Pharmacology, All India Institute of Medical Sciences, Bhubaneswar, Odisha, India

A. Srinivasan et al. (eds.), *R for Basic Biostatistics in Medical Research*,
https://doi.org/10.1007/978-981-97-6980-3_10

The choice of correlation depends on the distribution of data and other assumptions. The assumptions to be fulfilled for applying Pearson's product-moment correlation test are:

1. Both variables should be continuous on an interval or ratio scale.
2. Both variables should follow a normal distribution.
3. The relationship between the variables should be linear.
4. There should be no outliers, and if present, the outliers should be handled before the analysis.

The Spearman's and Kendall's correlation tests are applied when the data is not distributed normally, and they measure monotonic relationships. These analyses are performed on the ranked data, and the measurement focuses on the direction of association and not on the constancy of rate. The data could be represented on a continuous or ordinal scale. Kendall's method is more robust and is usually the preferred method between these two tests.

With this background knowledge, let us move ahead and perform correlation on our dataset. If we want to check whether weight has any association with HbA1c values, we can determine the correlation coefficient to conclude regarding the association. Prior to this, we will determine the normality of the data.

```
mydat <- read.csv("dm.csv")
attach(mydat)
shapiro.test(weight)
##
##  Shapiro-Wilk normality test
##
## data:  weight
## W = 0.97325, p-value = 0.4533
shapiro.test(hba1c)
##
##  Shapiro-Wilk normality test
##
## data:  hba1c
## W = 0.96242, p-value = 0.2024
plot(weight, hba1c)
cor(weight,hba1c,use="everything", method="pearson")
## [1] -0.3853913
```

1. We are now well acquainted with reading the data file into the working environment and assigning it to an R object.
2. We know that the data for weight and hba1c are continuous variables.
3. The test for the normality in data distribution demonstrates that both variables are normally distributed.
4. Next, we need to draw a scatter plot to check if the association is linear. (Fig. 10.1) Thus, we have confirmed that the method to be used for this analysis should be Pearson's.
5. The "`cor`" function assesses the correlation between two variables. The function uses arguments for variables (*x* and *y* are the two variables of numeric class, *use* is

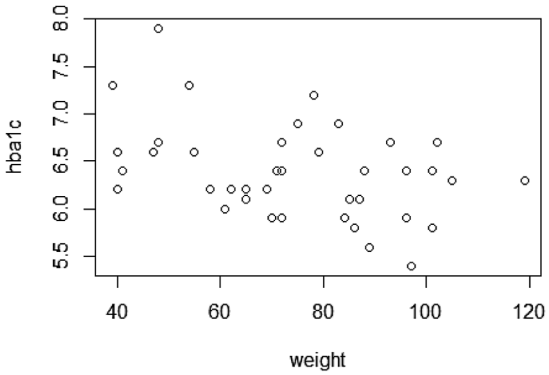

Fig. 10.1 Scatter plot for checking linearity of association between HbA1c and weight

an optional argument providing an option for handling the analysis in the presence of missing values; either of these could be used based on missingness in data—*everything* (default), *all.obs*, *complete.obs*, *na.or.complete*, or *pairwise.complete.obs* and *method* to specify which correlation coefficient is to be computed, whether *pearson* (default), *kendall*, or *spearman*. We already decided earlier that we want to compute Pearson's coefficient as the data is distributed normally.

6. Please note that we have used *everything* for the *use* argument in this case, as the data was complete. We can check what happens when we use all data in the presence of missing values.

```
mydat1<- read.csv("dm_missing.csv",na.strings=c(",","-",";"))
summary(mydat1)
##      group          age              sex             weight
##  Min.   :1.0   Min.   :34.00   Min.   :0.0000   Min.   : 39.00
##  1st Qu.:1.0   1st Qu.:41.00   1st Qu.:0.0000   1st Qu.: 61.00
##  Median :1.5   Median :47.00   Median :0.0000   Median : 72.00
##  Mean   :1.5   Mean   :47.95   Mean   :0.4474   Mean   : 74.62
##  3rd Qu.:2.0   3rd Qu.:56.00   3rd Qu.:1.0000   3rd Qu.: 89.00
##  Max.   :2.0   Max.   :66.00   Max.   :1.0000   Max.   :119.00
##                NA's   :3       NA's   :2        NA's   :3
##       ldl         fasting_before  fasting_after     difference
##  Min.   : 75.0   Min.   :130.0   Min.   : 89.0   Min.   : -9.00
##  1st Qu.:101.0   1st Qu.:155.0   1st Qu.:118.2   1st Qu.: 21.25
##  Median :121.5   Median :182.0   Median :132.5   Median : 30.00
##  Mean   :125.4   Mean   :183.2   Mean   :139.3   Mean   : 44.82
##  3rd Qu.:145.5   3rd Qu.:200.8   3rd Qu.:161.5   3rd Qu.: 79.00
##  Max.   :209.0   Max.   :276.0   Max.   :192.0   Max.   :126.00
##  NA's   :2       NA's   :2       NA's   :2       NA's   :2
##     response         hba1c
##  Min.   :0.000   Min.   :5.400
##  1st Qu.:0.000   1st Qu.:6.100
##  Median :1.000   Median :6.400
##  Mean   :0.625   Mean   :6.397
##  3rd Qu.:1.000   3rd Qu.:6.700
##  Max.   :1.000   Max.   :7.900
##
cor(mydat1$weight,mydat1$hba1c,use="everything", method="pearson")
## [1] NA
```

1. In this case, we are reading the dataset named *dm_missing.csv,* and with the argument *na.strings*, we are specifying that if any of the characters comma, dash, or semi-colon are encountered in the dataset, they should be treated as NAs.
2. When we check the summary of this dataset, we can note there are three missing values (NA) in the weight column.
3. Using the *use* argument, we set how to handle the NAs in the data set. *use* = "everything" in the presence of missing values gives NA in the output instead of the correlation coefficient.
   Similarly, *use* = "all.obs" in the presence of NA gives an error.

```
cor(mydat1$weight,mydat1$hba1c,use="all.obs", method="pearson")
## [1] NA
```

When use = "na.or.complete", it typically means that the function should include cases with missing values if they are complete for the specific operation being performed.

```
cor(mydat1$weight,mydat1$hba1c,use="na.or.complete", method="pearson")
## [1] -0.3776013
```

When use = "complete.obs", this specifies that only complete pairs of observations should be used for the correlation calculation.

```
cor(mydat1$weight,mydat1$hba1c,use="complete.obs", method="pearson")
## [1] -0.3776013
```

Thus, it is important to specify the string for the *use* argument based on our choice of handling the missing data to get the desired result.

Hypothesis Test of Correlation

A correlation test can be used to test whether there is a linear relationship between the variables in the population. The null hypothesis is that the population correlation coefficient is zero (0). The alternate hypothesis is that there is a linear relationship, and the population correlation coefficient is not zero.

Let us move ahead with the same example and check whether there is any relationship between weight and hba1c in the population.

```
cor.test(weight,hba1c, alternative = "two.sided", method="pearson", conf.
level=0.95 )
##
##  Pearson's product-moment correlation
##
## data:  weight and hba1c
## t = -2.5746, df = 38, p-value = 0.01406
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  -0.62220330 -0.08396184
## sample estimates:
##        cor
## -0.3853913
```

1. The *"cor.test"* function tests the hypothesis of whether an observed relationship is statistically relevant i.e., it estimates the p-value.
2. The arguments of the function require specification of the two numeric variables x (weight) and y (hba1c), alternative hypothesis, confidence interval (the default is 95%), and the method (*pearson, kendall or spearman*)
3. The output shows a correlation coefficient of −0.385 and a p-value of 0.01406.

Conclusion

There is a weak correlation between weight and hba1c with a Pearson's coefficient (r) of −0.385 (95% CI: −0.622 to −0.084) and p-value = 0.014, which is statistically significant.

Regression Analysis

While correlation shows the strength of association, regression analysis is a statistical technique used to examine the relationship between one dependent variable and one or more independent variables. The goal is to understand and quantify the nature of the relationship and to make predictions or estimations based on that understanding. In simpler terms, regression analysis helps us to understand how changes in one variable are associated with changes in another. Thus, the independent variables that show a good correlation with the dependent variable can be used to predict the dependent variable in a regression model.

The basic idea is to fit a mathematical model to observed data, where the model represents the relationship between the variables. The most common form of regression analysis is linear regression, where a straight line is used to model the relationship between the variables. Based on the type of dependent variable, regression can be classified into linear (continuous dependent variable) and logistic (categorical dependent variable), irrespective of the type of independent variable. Based on the number of independent variables, regression can be classified into single (one independent variable) and multiple (more than one independent variable) regression. The dependent variable is the main variable of interest or the outcome that you are trying to predict or explain. Independent variables are the variables that are used to predict or explain the variation in the dependent variable. In this chapter, we will learn how to perform linear regression. Unlike the analysis in hypothesis testing, where assumptions are satisfied before choosing parametric or nonparametric tests, in regression analysis, we first create a model and then test the assumptions for the model to hold valid for reporting purposes.

Simple Linear Regression
The most common form of regression analysis is simple linear regression. For example, $y = a + b(x)$, where x is an independent variable and y is a dependent variable.

Multiple Linear Regression
Two or more independent variables are used in multiple regression. The equation appears like this: y = a + b1(x1) + b2(x2) + b3(x3), where x is the vector of independent variables and y is the dependent variable.

Performing Linear Regression

We should check for linear relationships between independent and dependent variables. We can draw scatter plots to visualize the same.

Simple Linear Regression with Continuous Independent Variable

Let us check how the change in fasting plasma glucose levels (fcfb) affects the change in HbA1c (hcfb) in the given dataset.

Step 1: Loading the Dataset

```
dm <- read.csv("DMdat.csv")
head(dm,2)
##   SID Age Gender BMI Wt  Hb0 Treat HbA1c0 FPG0 HbA1c12 HbA1c24
FPG12 priorRx
## 1   1  51      0  24 81 18.0  L1.2    9.0  211     7.1    6.71
160 Insulin
## 2   2  60      0  24 54 14.3  L1.8   10.4  256     7.5    9.30
205 Insulin
##   hcfb fcfb
## 1  1.9   51
## 2  2.9   51
attach(dm)
View(dm)
str(dm)
## 'data.frame':    400 obs. of  15 variables:
##  $ SID    : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ Age    : int  51 60 64 53 36 40 34 36 55 62 ...
##  $ Gender : int  0 0 1 1 0 1 1 1 1 1 ...
##  $ BMI    : int  24 24 24 25 34 25 33 27 28 31 ...
##  $ Wt     : int  81 54 67 54 87 63 91 67 78 67 ...
##  $ Hb0    : num  18 14.3 15.3 7.1 17.5 13.1 13.4 13.4 14.8 14.6 ...
##  $ Treat  : chr  "L1.2" "L1.8" "L0.6" "L1.2" ...
##  $ HbA1c0 : num  9 10.4 10 9 9.5 9.2 9.2 10.1 8.3 9.4 ...
##  $ FPG0   : int  211 256 243 211 227 217 217 246 189 224 ...
##  $ HbA1c12: num  7.1 7.5 9.2 7.7 9.2 9.8 7.8 9.8 7.1 9.2 ...
##  $ HbA1c24: num  6.71 9.3 8.8 7.31 8.8 ...
##  $ FPG12  : int  160 205 233 161 215 211 180 230 137 188 ...
##  $ priorRx: chr  "Insulin" "Insulin" "Insulin" "Insulin" ...
##  $ hcfb   : num  1.9 2.9 0.8 1.3 0.3 -0.6 1.4 0.3 1.2 0.2 ...
##  $ fcfb   : int  51 51 10 50 12 6 37 16 52 36 ...
```

1. We are reading our earlier *.csv* file into the working environment and viewing the head of the dataset.

Step 2: Checking for Linearity

```
plot (fcfb, hcfb)
```

1. We can visualize that there is a linear relationship between a change in fasting blood glucose (fcfb) and a change in HbA1c (hcfb). Thus, we can move ahead and perform regression analysis (Fig. 10.2).

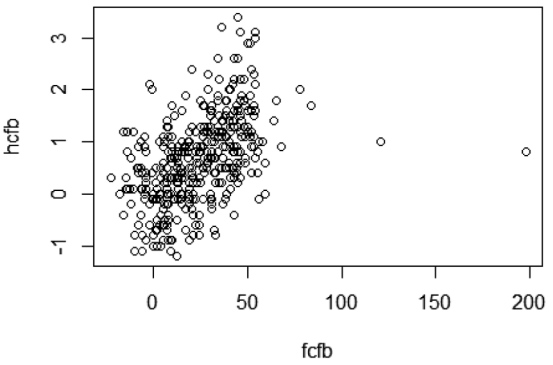

Fig. 10.2 Scatter plot for checking linearity of association between change in fasting blood glucose and change in HbA1c

Step 3: Creating a Simple Linear Regression Model

```
mod1<- lm(hcfb~ fcfb, data = dm)
mod1
##
## Call:
## lm(formula = hcfb ~ fcfb, data = dm)
##
## Coefficients:
## (Intercept)         fcfb
##     0.21893      0.01882
```

1. The "*lm*" function is used for both simple and multiple regression analysis. The arguments for the *lm* function are the *formula* to be used for the regression and the *data*. For a simple linear regression, the *formula* can be read as 'y' as a function of 'x' (y~x), where y is a dependent variable and x is an independent variable. The *data* argument is used to specify the dataset from where the variables x and y will be taken.

Step 4: Summary of the Model

```
summary(mod1)
##
## Call:
## lm(formula = hcfb ~ fcfb, data = dm)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -3.14611 -0.45337 -0.04248  0.43077  2.33398
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 0.218933   0.054393   4.025 6.82e-05 ***
## fcfb        0.018824   0.001685  11.175  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.7399 on 398 degrees of freedom
## Multiple R-squared:  0.2388, Adjusted R-squared:  0.2369
## F-statistic: 124.9 on 1 and 398 DF,  p-value: < 2.2e-16
```

1. The "`summary`" function provides us with statistical summaries such as coefficients, standard errors, t-values, p-values, and other relevant information like R-squared and F—statistics about the fitted model.

Step 5: Conclusion

We performed a simple linear regression analysis to estimate the change in HbA1c based on the change in fasting plasma glucose values. The model was statistically significant with adjusted R-squared = 0.237; $F(1,398) = 124.9$; $p < 0.001$. The beta coefficient for change in fasting blood glucose was 0.0188 (se = 0.00168); $p < 0.001$.

Next, we need to check for the assumptions, which have been discussed at the end of the chapter.

Simple Linear Regression with Categorical Independent Variable

Let us perform a regression analysis to estimate the effect of the treatment group (Treat) on change in HbA1c (hcfb). Please note that the independent variable in this case (Treat) is categorical.

Step 1: Loading the Dataset

We are using the same dataset *DMdat.csv.*

Step 2: Converting the "Treat" Variable into a Factorial Variable

```
dm$Treat<-as.factor(dm$Treat)
```

1. First of all, we convert the variable Treat to factor after we have read the dataframe into our working space.

Step 3: Setting the Baseline Category to Compare With

```
library(dplyr)
## Warning: package 'dplyr' was built under R version 4.2.3
##
## Attaching package: 'dplyr'
## The following objects are masked from 'package:stats':
##
##     filter, lag
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
dm <- dm %>%
  mutate(Treat = relevel(Treat, ref = "P"))
```

In this example, we set the baseline category to the treatment group "Placebo".

Step 4: Creating a Regression Model

```
mod2<- lm(hcfb ~ Treat, data = dm)
summary(mod2)
##
## Call:
## lm(formula = hcfb ~ Treat, data = dm)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.00086 -0.43462 -0.00086  0.45957  2.19914
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.03140    0.07305   0.430  0.66759
## TreatL0.6    0.30322    0.09874   3.071  0.00228 **
## TreatL1.2    0.76947    0.09640   7.982 1.57e-14 ***
## TreatL1.8    1.40903    0.10109  13.939  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.6774 on 396 degrees of freedom
## Multiple R-squared:  0.3652, Adjusted R-squared:  0.3604
## F-statistic: 75.93 on 3 and 396 DF,  p-value: < 2.2e-16
```

1. Next, we run the regression analysis using *"lm"* function as used in the earlier example.
2. Finally, we call the *"summary"* function to obtain the results for reporting.
3. The output for the categorical variable should be read as the difference between the value of the given level of the variable and the reference level.

Conclusion

We performed a simple linear regression analysis to estimate the change in HbA1c based on the treatment group. The model was statistically significant with adjusted R-squared = 0.360; $F(3,396) = 75.93$; $p < 0.001$. The beta coefficient for the treatment group L0.6 is 0.303 (se = 0.099), i.e., the change in HbA1c in this group was 0.303 more than the placebo group, which was statistically significant ($p = 0.002$). The change in HbA1c in group L1.2 versus placebo was 0.769 (se = 0.096), which was statistically significant ($p < 0.001$). The change in HbA1c in group L1.8 versus placebo is 1.409 (se = 0.101) with $p > 0.001$.

Multiple Linear Regression

As we already discussed, two or more independent variables are used in multiple regression, irrespective of whether they are continuous or categorical.

Let us create a multiple linear regression to estimate the change in HbA1c levels based on treatment group, weight, change in fasting blood glucose, BMI, age, fasting blood glucose, and HbA1c at baseline.

Step 1: Loading the Dataset (the Dataset Is Already Loaded)

Step 2: Converting Categorical Variable(s) into Type Factor

```
Treat <- as.numeric(Treat)
## Warning: NAs introduced by coercion
```

Step 3: Checking for Linearity Between Independent and Dependent Variables

```
plot(Wt, hcfb)

plot(BMI, hcfb)

plot(Age, hcfb)

plot(FPG0, hcfb)

plot(HbA1c0, hcfb)
```

We can appreciate the linearity in the relationships. (Figs. 10.3, 10.4, 10.5, 10.6, and 10.7).

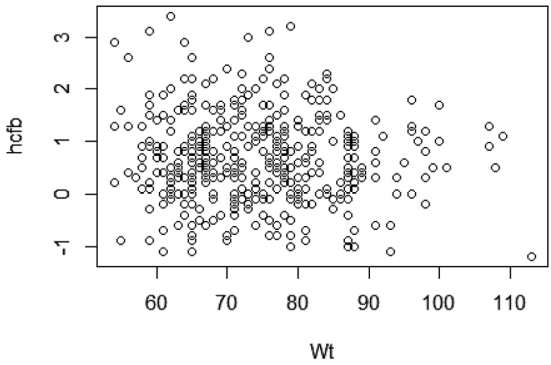

Fig. 10.3 Scatter plot for checking linearity of association between change in HbA1c and weight

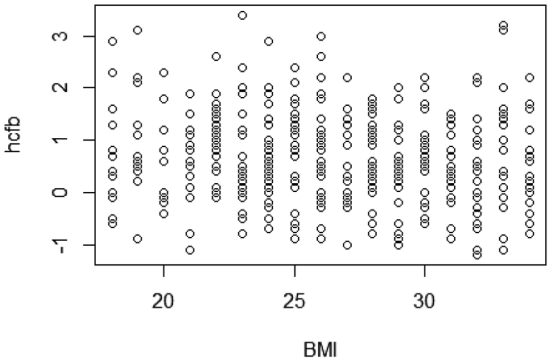

Fig. 10.4 Scatter plot for checking linearity of association between change in HbA1c and BMI

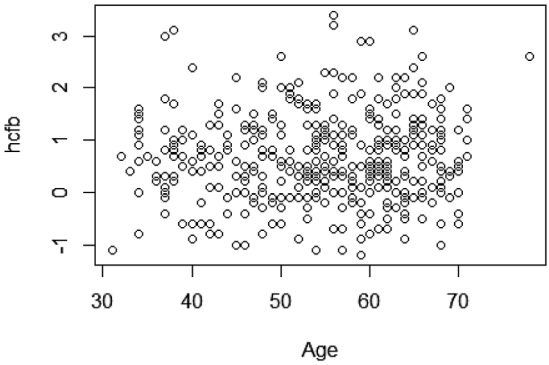

Fig. 10.5 Scatter plot for checking linearity of association between change in HbA1c and age

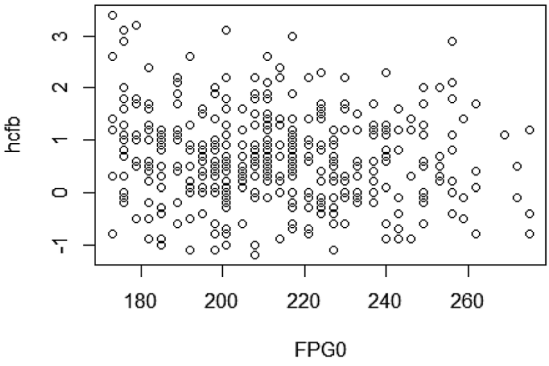

Fig. 10.6 Scatter plot for checking linearity of association between change in HbA1c and fasting glucose at baseline

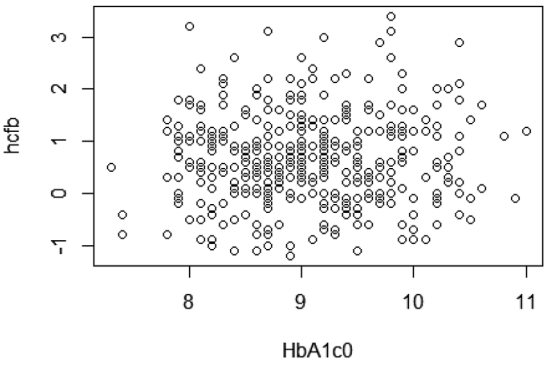

Fig. 10.7 Scatter plot for checking linearity of association between change in HbA1c and HbA1c at baseline

Step 4: Creating the Model

```
mod4 <- lm(hcfb ~ Treat + Wt +fcfb + BMI + Age + FPG0 + HbA1c0, data = dm)
summary(mod4)
##
## Call:
## lm(formula = hcfb ~ Treat + Wt + fcfb + BMI + Age + FPG0 + HbA1c0,
##     data = dm)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.84026 -0.45182 -0.00098  0.40941  2.17458
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -0.377759   0.590865  -0.639  0.52298
## TreatL0.6    0.190051   0.106249   1.789  0.07443 .
## TreatL1.2    0.620500   0.121637   5.101 5.28e-07 ***
## TreatL1.8    1.167382   0.147804   7.898 2.91e-14 ***
## Wt          -0.001720   0.003158  -0.545  0.58638
## fcfb         0.003215   0.002328   1.381  0.16810
## BMI         -0.009924   0.007684  -1.292  0.19726
## Age          0.003796   0.003396   1.118  0.26429
## FPG0        -0.008298   0.002610  -3.179  0.00159 **
## HbA1c0       0.266621   0.085680   3.112  0.00200 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.6672 on 390 degrees of freedom
## Multiple R-squared:  0.3936, Adjusted R-squared:  0.3796
## F-statistic: 28.12 on 9 and 390 DF,  p-value: < 2.2e-16
```

1. The function "*lm*" is used to create a model in simple and multiple linear regression.
2. All the independent variables are added with a plus sign.
3. The beta coefficients obtained as an output are partial beta coefficients.

Step 5: Selecting the Best Model Out of All Independent Variables To Be Studied

```
stepwise <- step(mod4)
## Start:  AIC=-313.86
## hcfb ~ Treat + Wt + fcfb + BMI + Age + FPG0 + HbA1c0
##
##          Df Sum of Sq    RSS     AIC
## - Wt      1     0.132 173.74 -315.56
## - Age     1     0.556 174.17 -314.58
## - BMI     1     0.743 174.35 -314.15
## - fcfb    1     0.849 174.46 -313.91
## <none>                173.61 -313.86
## - HbA1c0  1     4.311 177.92 -306.05
## - FPG0    1     4.499 178.11 -305.63
## - Treat   3    34.804 208.41 -246.78
##
## Step:  AIC=-315.56
## hcfb ~ Treat + fcfb + BMI + Age + FPG0 + HbA1c0
##
##          Df Sum of Sq    RSS     AIC
## - Age     1     0.585 174.33 -316.21
## - BMI     1     0.742 174.48 -315.85
## <none>                173.74 -315.56
## - fcfb    1     0.948 174.69 -315.38
## - HbA1c0  1     4.436 178.18 -307.47
## - FPG0    1     4.624 178.37 -307.05
## - Treat   3    34.684 208.43 -248.75
##
## Step:  AIC=-316.21
## hcfb ~ Treat + fcfb + BMI + FPG0 + HbA1c0
##
##          Df Sum of Sq    RSS     AIC
## - BMI     1     0.720 175.05 -316.56
## <none>                174.33 -316.21
## - fcfb    1     1.042 175.37 -315.83
## - HbA1c0  1     4.329 178.66 -308.40
## - FPG0    1     4.634 178.96 -307.72
## - Treat   3    34.638 208.97 -249.72
##
## Step:  AIC=-316.56
## hcfb ~ Treat + fcfb + FPG0 + HbA1c0
##
##          Df Sum of Sq    RSS     AIC
## <none>                175.05 -316.56
## - fcfb    1     1.022 176.07 -316.23
## - HbA1c0  1     4.777 179.82 -307.79
## - FPG0    1     5.103 180.15 -307.07
## - Treat   3    34.849 209.90 -249.94
summary(stepwise)
##
## Call:
## lm(formula = hcfb ~ Treat + fcfb + FPG0 + HbA1c0, data = dm)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.93287 -0.44568  0.00884  0.41368  2.11577
##
```

```
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -0.574014   0.431350  -1.331 0.184047
## TreatL0.6    0.197889   0.105964   1.868 0.062575 .
## TreatL1.2    0.612251   0.120804   5.068 6.20e-07 ***
## TreatL1.8    1.169732   0.147196   7.947 2.04e-14 ***
## fcfb         0.003495   0.002307   1.515 0.130560
## FPG0        -0.008765   0.002589  -3.385 0.000784 ***
## HbA1c0       0.278172   0.084944   3.275 0.001151 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.6674 on 393 degrees of freedom
## Multiple R-squared:  0.3885, Adjusted R-squared:  0.3792
## F-statistic: 41.62 on 6 and 393 DF,  p-value: < 2.2e-16
```

1. In this example, "`lm`" was used in the earlier step to fit the initial full model with all the variables we wanted to include in the model.
2. The "`step`" function is used to perform the stepwise regression. By default, the "`step`" function selects the model based on AIC values.
3. We can appreciate that variables Treat, fcfb, FPG0 and HbA1c0 are included in the best-fit model, and the variables Wt, BMI, and Age are dropped from the model.

Conclusion

A multiple linear regression was performed to estimate the effect of the Treatment group, change in fasting blood glucose, fasting blood glucose, and HbA1c at baseline on the dependent variable change in HbA1c. The model is statistically significant with an adjusted R-squared = 0.379; $F(6,393) = 41.62$; $p < 0.001$. We can mention the estimates, standard errors, and p-values of the independent variables. We can report the result in tabular form as obtained in the output.

Once the model is built, to keep the linear regression model valid, we should check a few assumptions before we report and use the model.

Testing the Assumptions [2]

1. *Linearity*: This assumption should be tested prior to choosing the independent variables. This is required because if the relation is nonlinear, then the predictions made by the linear regression model will not be true.
2. *Checking for outliers and influencers*: Outliers are data points that deviate significantly from the overall pattern of the data. In the context of regression analy-

sis, outliers can disproportionately influence the estimation of regression coefficients and may distort the fit of the model. Outliers can be identified by examining residual plots.

Influencers are data points that have a substantial impact on the estimates of regression parameters when included or excluded from the analysis. These can significantly affect the slope, intercept, and overall fit of the regression model. Influencers could be identified by observing Cook's distance, which is a measure of the impact of deleting a given observation on the regression coefficients. Large Cook's distances (>1) indicate influential points.

3. *Homoscedasticity*: It refers to the assumption that the variability, or "spread," of the residuals (the differences between observed and predicted values) is constant across all levels of the independent variable(s). Visually, homoscedasticity can be assessed by examining a plot of the residuals against the predicted values. In a scatterplot of residuals versus predicted values, we would ideally see a random spread of points with no discernible pattern. If there is a noticeable pattern or the spread of residuals widens or narrows systematically across the range of predicted values, it suggests heteroscedasticity, violating the homoscedasticity assumption.
4. *Normality of residuals*: The residuals should have a constant variance, should be normally distributed, and should be independent of each other.
5. *Independence or no collinearity*: Collinearity occurs when two or more independent variables in a regression model are highly correlated with each other. It can create challenges in estimating the individual contributions of each predictor variable to the dependent variable. A common metric for detecting collinearity is the Variance inflation factor (VIF). VIF measures how much the variance of an estimated regression coefficient increases if the predictors are correlated. We can check the independence or collinearity of independent variables with VIF results: <1: no collinearity, 1–5: moderate collinearity >5: strong collinearity.

   If VIF > 5 for any of the two variables, one of the variables should be dropped.

Checking for Assumptions

Assumption of Linearity

The scatter plot is the best and simplest way. We have seen this before performing linear regression.

Checking Outliers

We will identify outliers in the datasets by quantifying standardized residuals. A value of >3 is considered an outlier.

```
rstandard(stepwise)
##           1           2           3           4           5           6
##  1.552763325  2.236548340  0.738302823  0.652742804 -0.653661486 -1.359116772
##           7           8           9          10          11          12
##  0.865781692  0.249466182 -0.346764691 -0.303766364  0.522779387  0.985531559
##          13          14          15          16          17          18
## -0.524340885  0.065058154  2.696085508  0.175331676 -0.904161380 -0.879470386
##          19          20          21          22          23          24
##  1.501076889  0.519086038 -0.551046407 -0.450227522 -0.309894343  0.289313564
##          25          26          27          28          29          30
##  2.922719691  0.720315263 -0.364084332 -0.975777153  0.921730445  0.075864216
##          31          32          33          34          35          36
## -0.073899356  0.625709394 -0.096327368  0.025130786 -0.457108393  0.048458663
##          37          38          39          40          41          42
##  1.768353655 -0.293863074 -0.152811020 -0.020458699  0.332265350  0.420451443
##          43          44          45          46          47          48
## -0.027343415  1.457990150  0.848697212 -0.691944862 -0.076389156 -0.367345461
##          49          50          51          52          53          54
##  0.496990987  0.092160591  0.774183856  2.222871150  1.436134526  0.569822306
```

Assumption of Homoscedasticity

This can be confirmed by residual values against predicted values.

```
plot(stepwise$residuals ~ stepwise$fitted.values)
```

If the assumption of homoscedasticity is not met or there is a heteroscedasticity, we can visualize a pattern in the plot. (Fig. 10.8).

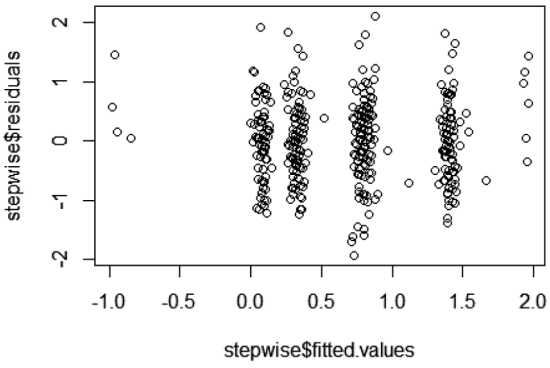

Fig. 10.8 Scatter plot for checking association between residuals and fitted values of the model

Assumption of Normality of Residuals

```
hist(mod1$residuals)
```

```
hist(stepwise$residuals)
```

We can appreciate a normal pattern for the two models in the histogram. (Figs. 10.9 and 10.10).

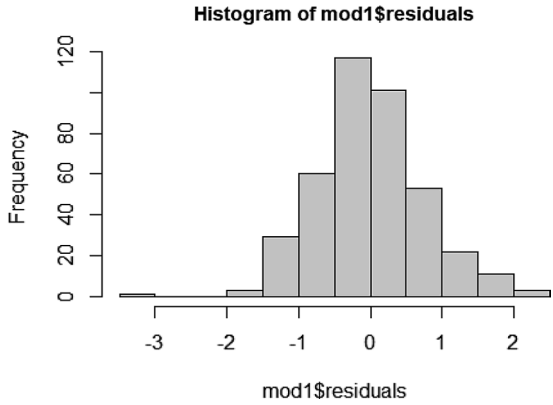

Fig. 10.9 Histogram for distribution of residuals in the model (mod1)

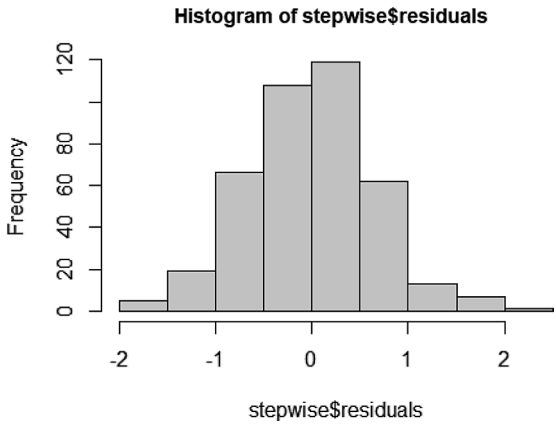

Fig. 10.10 Histogram for distribution of residuals in the model (stepwise)

Assumption of No Collinearity

1. The "*vif*" function is used to estimate the collinearity between independent variables.
2. We can observe that VIF is <5 for all the variables, and thus, the assumption of no collinearity is made.

```
library(car)
## Warning: package 'car' was built under R version 4.2.3
## Loading required package: carData
## Warning: package 'carData' was built under R version 4.2.3
##
## Attaching package: 'car'
## The following object is masked from 'package:dplyr':
##
##     recode
vif(stepwise)
##            GVIF Df GVIF^(1/(2*Df))
## Treat  2.351137  3        1.153134
## fcfb   2.305408  1        1.518357
## FPG0   3.304947  1        1.817951
## HbA1c0 3.325568  1        1.823614
```

Checking for Influencers in Data

```
influence.measures(stepwise)
## Influence measures of
##   lm(formula = hcfb ~ Treat + fcfb + FPG0 + HbA1c0, data = dm) :
##
##        dfb.1_  dfb.TL0. dfb.TL1.2 dfb.TL1.8  dfb.fcfb  dfb.FPG0  dfb.HA10
## 1    1.44e-02 -3.93e-02  9.58e-03 -7.79e-02  1.08e-01  2.57e-03 -9.76e-03
## 2   -1.87e-01 -2.85e-02 -3.56e-02  7.82e-02  4.49e-02  9.02e-02  4.09e-02
## 3   -4.23e-02  4.77e-02  7.37e-03  1.09e-02 -1.62e-02  1.08e-02  1.62e-02
## 4    5.99e-03 -1.56e-02  5.39e-03 -3.10e-02  4.32e-02  9.63e-04 -3.98e-03
## 5    1.79e-02 -1.47e-02 -5.65e-02 -3.08e-02  4.32e-02 -2.55e-03 -8.18e-03
## 6    7.97e-03 -1.00e-01 -2.90e-02 -3.47e-02  4.87e-02  2.90e-03 -6.50e-03
## 7   -4.74e-03 -7.19e-03  3.01e-02 -1.32e-02  1.80e-02  1.10e-03  1.96e-03
## 8   -1.54e-02 -2.59e-02 -2.66e-02 -2.49e-02  1.45e-02  9.11e-04  1.08e-02
## 9   -1.88e-02  1.13e-03  3.45e-03 -1.20e-02 -8.23e-03 -9.59e-04  1.14e-02
## 10   4.67e-03 -1.04e-02  1.29e-02  1.51e-02 -2.08e-02 -2.76e-03 -7.17e-04
## 11  -2.30e-02 -3.84e-02 -3.07e-02 -2.28e-02 -9.56e-03 -1.73e-03  1.97e-02
## 12  -9.08e-02 -2.25e-02 -1.62e-02  2.46e-02 -1.36e-02 -2.24e-01  2.15e-01
## 13   3.67e-02  1.18e-02 -8.57e-03  1.79e-02 -2.36e-02 -1.08e-02 -1.31e-02
## 14   2.63e-03 -4.27e-03 -3.37e-03 -2.59e-03 -1.41e-03 -1.86e-03  5.25e-04
## 15   1.27e-01 -6.14e-04  1.20e-01 -1.80e-02  2.89e-02 -3.11e-02 -4.97e-02
## 16   1.53e-03 -1.37e-02 -1.24e-02 -1.06e-02  1.07e-03 -3.18e-03  3.26e-03
## 17  -2.41e-02 -7.01e-02 -2.29e-02 -2.59e-02  3.55e-02  1.03e-02  6.33e-03
## 18  -2.27e-02 -8.50e-03 -5.45e-02 -1.35e-02  1.83e-02  8.70e-03  6.67e-03
## 19   9.96e-02  1.18e-01  3.52e-02  3.67e-02 -4.84e-02 -3.19e-02 -3.37e-02
## 20   1.17e-02  2.80e-02 -6.65e-03 -8.45e-03  1.22e-02 -1.71e-03 -5.37e-03
## 21  -1.41e-02 -4.99e-02 -2.56e-02 -2.96e-02  4.08e-02  7.25e-03  2.90e-03
```

```
## 22   3.77e-02 -3.00e-02 -6.70e-03 -9.88e-03  1.46e-02 -1.00e-02 -1.41e-02
## 23   1.37e-02  9.51e-04  9.11e-04 -1.49e-02 -8.16e-04 -8.97e-03 -1.24e-03
## 24  -5.24e-03 -3.23e-04 -4.29e-04  1.40e-02  7.69e-04  6.04e-03 -1.42e-03
## 25  -2.25e-01 -2.44e-01 -2.11e-01 -1.68e-01 -2.85e-03  1.97e-02  1.45e-01
## 26   2.68e-02 -2.79e-02 -4.78e-02 -2.26e-02  8.26e-02  7.55e-03 -2.05e-02
## 27   2.17e-03  2.29e-02  1.60e-02  1.03e-02  1.43e-02  6.69e-03 -9.83e-03
## 28   8.08e-02  2.40e-02 -1.33e-02  3.57e-02 -4.76e-02 -2.63e-02 -2.70e-02
## 29  -6.60e-02  2.22e-03  7.41e-03  5.75e-02 -1.71e-02  2.96e-02  1.59e-02
## 30   1.50e-02 -1.91e-03 -1.39e-03  8.66e-04 -9.61e-04  2.85e-02 -2.85e-02
## 31   5.82e-03 -8.07e-04 -5.25e-03 -2.32e-03  3.35e-03 -1.56e-03 -2.17e-03
## 32   4.63e-02 -4.60e-02 -4.19e-02 -3.72e-02  3.94e-03 -2.28e-02 -3.31e-03
## 33  -4.41e-03 -8.76e-03 -4.39e-03 -4.99e-03  6.79e-03  1.55e-03  1.40e-03
## 34  -1.25e-03 -2.12e-03 -1.91e-03 -1.61e-03  2.39e-04 -2.21e-05  9.89e-04
## 35  -2.53e-02  3.77e-02  3.68e-02  3.41e-02 -1.24e-02  1.29e-02  2.08e-04
## 36   6.26e-04  6.16e-04  3.27e-03  1.11e-03 -1.51e-03 -2.15e-04 -2.05e-04
## 37   3.64e-02 -1.03e-01 -6.76e-02 -3.98e-02 -8.15e-02 -4.06e-02  2.69e-02
## 38  -8.28e-03  2.34e-02  2.16e-02  1.92e-02 -3.83e-03  7.29e-03 -3.64e-03
## 39   3.82e-03 -1.08e-02 -2.77e-03 -3.48e-03  4.93e-03 -7.71e-04 -1.60e-03
## 40   1.56e-03 -1.24e-03 -9.67e-05 -1.96e-04  3.07e-04 -4.51e-04 -5.62e-04
## 41  -2.29e-02  1.30e-02 -1.03e-02 -1.11e-02  1.46e-02  6.67e-03  8.21e-03
## 42  -2.97e-02 -3.88e-03 -4.62e-03  1.67e-02  5.41e-03  1.42e-02  6.59e-03
## 43   1.97e-03  1.65e-03  9.64e-04  3.93e-04  1.65e-03  3.49e-05 -1.43e-03
## 44   7.58e-02  1.47e-01  8.91e-02  1.02e-01 -1.39e-01 -2.69e-02 -2.40e-02
## 45  -1.49e-02 -1.32e-02  2.01e-02 -2.36e-02  3.25e-02  6.88e-03  3.58e-03
## 46  -4.22e-02  7.45e-03  1.55e-02 -1.34e-02 -3.08e-02  7.79e-04  2.34e-02
## 47  -6.16e-03  9.11e-04 -1.76e-03  2.58e-03 -3.75e-03  1.42e-03  2.46e-03
## 48   3.53e-02 -1.84e-02  4.17e-03  3.16e-03 -3.38e-03 -9.71e-03 -1.30e-02
## 49   1.36e-02 -3.51e-02 -2.93e-02 -2.37e-02 -5.61e-03 -1.29e-02  6.77e-03
## 50  -4.66e-04 -1.24e-03  2.43e-03 -2.33e-03  3.19e-03  1.81e-04  1.44e-04
## 51   6.10e-02  8.78e-03  4.73e-02  8.90e-03 -1.06e-02 -1.69e-02 -2.25e-02
## 52  -2.06e-01 -5.10e-02 -3.67e-02  5.58e-02 -3.07e-02 -5.07e-01  4.87e-01
## 53  -2.87e-02  3.00e-02  4.95e-02  1.31e-01 -8.13e-02  2.58e-02 -2.77e-03
## 54   1.55e-02 -2.07e-04 -1.83e-03  2.50e-02  5.47e-03  3.53e-03 -1.14e-02
## 55   2.17e-02 -2.82e-03 -5.69e-03 -2.86e-02  1.06e-02 -1.20e-02 -3.56e-03
## 56  -4.68e-03  2.72e-02  2.69e-03  3.45e-03 -4.79e-03  1.70e-03  1.41e-03
## 57  -5.88e-03 -3.88e-02  3.51e-03  4.46e-03 -6.34e-03  1.29e-03  2.39e-03
## 58  -6.34e-02  6.38e-02 -1.91e-02 -1.94e-02  2.54e-02  2.01e-02  2.15e-02
## 59   1.38e-02 -1.89e-03  8.20e-03 -5.49e-03  8.07e-03 -2.92e-03 -5.72e-03
## 60  -2.30e-02  7.39e-02  1.65e-01  1.97e-01 -2.73e-01 -9.23e-03  1.90e-02
## 61   4.00e-02 -8.86e-03 -1.76e-02  1.17e-03  3.38e-02 -1.47e-03 -2.16e-02
## 62  -7.63e-03 -5.90e-04  1.40e-02 -1.35e-04 -7.81e-06  2.18e-03  2.76e-03
## 63  -4.75e-02 -1.61e-02  9.67e-03 -2.48e-02  3.30e-02  1.42e-02  1.69e-02
## 64   5.48e-02  1.15e-02  6.94e-03 -1.65e-02  1.16e-02  1.25e-01 -1.22e-01
## 65   7.01e-02 -2.04e-02 -3.99e-02  7.25e-02  7.67e-02  1.82e-02 -5.30e-02
## 66   5.26e-02  4.21e-03 -6.20e-02  1.14e-03  1.87e-04 -1.26e-02 -2.08e-02
##         dffit cov.r   cook.d     hat inf
## 1    0.181777 0.988 4.70e-03 0.01347
## 2    0.330461 0.951 1.54e-02 0.02115
## 3    0.088146 1.023 1.11e-03 0.01407
## 4    0.074872 1.024 8.02e-04 0.01300
## 5   -0.077094 1.024 8.50e-04 0.01374
## 6   -0.142883 0.996 2.91e-03 0.01091
## 7    0.082888 1.014 9.82e-04 0.00909
## 8    0.037338 1.040 2.00e-04 0.02196
## 9   -0.041561 1.030 2.47e-04 0.01419
## 10  -0.036998 1.031 1.96e-04 0.01465
## 11   0.066349 1.029 6.30e-04 0.01588
```

```
## 12   0.252249 1.066 9.09e-03 0.06149   *
## 13  -0.068516 1.030 6.72e-04 0.01682
## 14   0.007649 1.032 8.38e-06 0.01367
## 15   0.291420 0.904 1.19e-02 0.01137   *
## 16   0.019335 1.030 5.35e-05 0.01204
## 17  -0.100210 1.016 1.44e-03 0.01214
## 18  -0.088181 1.014 1.11e-03 0.00996
## 19   0.193298 0.994 5.32e-03 0.01626
## 20   0.053989 1.024 4.17e-04 0.01072
## 21  -0.070274 1.029 7.07e-04 0.01603
## 22  -0.062881 1.034 5.66e-04 0.01917
## 23  -0.036933 1.031 1.95e-04 0.01404
## 24   0.031282 1.028 1.40e-04 0.01158
## 25   0.434588 0.892 2.65e-02 0.02122   *
## 26   0.114397 1.034 1.87e-03 0.02463
## 27  -0.043067 1.030 2.66e-04 0.01383
## 28  -0.138499 1.021 2.74e-03 0.01975
## 29   0.125681 1.021 2.26e-03 0.01826
## 30   0.031154 1.190 1.39e-04 0.14462   *
## 31  -0.010042 1.037 1.44e-05 0.01817
## 32   0.082418 1.028 9.72e-04 0.01708
## 33  -0.012765 1.036 2.33e-05 0.01730
## 34   0.003258 1.035 1.52e-06 0.01657
## 35  -0.056469 1.030 4.56e-04 0.01506
## 36   0.004817 1.028 3.32e-06 0.00981
## 37   0.212732 0.976 6.43e-03 0.01419
## 38  -0.033232 1.029 1.58e-04 0.01266
## 39  -0.016364 1.029 3.83e-05 0.01137
## 40  -0.002695 1.036 1.04e-06 0.01710
## 41   0.044278 1.034 2.81e-04 0.01749
## 42   0.057003 1.033 4.65e-04 0.01809
## 43  -0.004231 1.042 2.56e-06 0.02344
## 44   0.219280 1.002 6.85e-03 0.02206
## 45   0.087483 1.016 1.09e-03 0.01052
## 46  -0.089334 1.026 1.14e-03 0.01642
## 47  -0.010521 1.037 1.59e-05 0.01866
## 48  -0.053771 1.037 4.14e-04 0.02102
## 49   0.056119 1.026 4.51e-04 0.01261
## 50   0.009181 1.028 1.21e-05 0.00985
## 51   0.100156 1.024 1.43e-03 0.01648
## 52   0.571852 0.993 4.62e-02 0.06149   *
## 53   0.175804 0.996 4.40e-03 0.01472
## 54   0.061328 1.024 5.38e-04 0.01147
## 55  -0.052764 1.031 3.99e-04 0.01548
## 56   0.042230 1.025 2.55e-04 0.00991
## 57  -0.066519 1.020 6.33e-04 0.00979
## 58   0.147855 1.002 3.12e-03 0.01314
## 59   0.029345 1.030 1.23e-04 0.01301
## 60  -0.277226 1.710 1.10e-02 0.40574   *
## 61   0.073546 1.037 7.74e-04 0.02224
## 62   0.029383 1.026 1.24e-04 0.00942
## 63   0.089657 1.027 1.15e-03 0.01714
## 64  -0.142048 1.079 2.89e-03 0.06203   *
## 65   0.280654 0.921 1.11e-02 0.01230   *
```

Influence diagnostics use the function **influence.measures**

1. Cook's distance can be observed from the data when this command is run.
2. None of the Cook's distance is >1. Thus, we assume data has no influencers.

Thus, we can assume that the model which was created in the name of stepwise was valid and can be reported in the results.

References

1. Bewick V, Cheek L, Ball J. Statistics review 7: correlation and regression. Crit Care. 2003;7(6):451–9.
2. Schmidt AF, Finan C. Linear regression and the normality assumption. J Clin Epidemiol. 2018;98:146–51.

Chapter 11
Logistic Regression Analysis for Categorical Outcome

Guruprasad Padmanaban, Archana Mishra, and Anand Srinivasan

Introduction to Logistic Regression [1]

A logistic regression defines a relationship between one or more explanatory variables and a single discrete dependent variable that is categorical (binary variable in most cases). We can use it to create a prediction model or test hypotheses regarding predictive factors by disentangling the impact of many predictor variables on a categorical variable. Any combination of continuous, binary, or categorical data can be used as predictor variables. A logarithmic transformation is used to model a linear relationship. The result is a set of regression coefficients showing the relationship between each predictor variable and the binary outcome after all other variables in the model. The model formed is $\log_e[p/(1-p)] = b0 + b1x1 + b2x2 + b3x3 + b4x4 + \ldots + bnxn$, where $\log_e[p/(1-p)]$ is the logit transformation, p is the proportion with the outcome, b0 is the intercept, and b1,b2,b3,…., bn are the log-transformed partial regression coefficients. These need to be back-transformed to obtain the odds ratio.

The assumption for performing logistic regression:

The central assumption is that, for continuous predictor variables, the relationship between the predictor variable and the outcome is linear on the logit scale.

Applications of logistic regression

Supplementary Information The online version contains supplementary material available at https://doi.org/10.1007/978-981-97-6980-3_11.

G. Padmanaban
Metadata Management for Clinical Operations, Novartis, Hyderabad, India

A. Mishra · A. Srinivasan (✉)
Department of Pharmacology, All India Institute of Medical Sciences, Bhubaneswar, Odisha, India
e-mail: anandsrinivasan@aiimsbhubaneswar.edu.in

A. Srinivasan et al. (eds.), *R for Basic Biostatistics in Medical Research*,
https://doi.org/10.1007/978-981-97-6980-3_11

1. Logistic regression would be used in order to examine the association between a causal variable and a binary output variable while accounting for confounding variables that may be continuous or categorical.
2. The development of prognostic indicators may be done using logistic regression analysis.
3. Identifying the factors that can differentiate between the two levels of an outcome variable in a discriminant analysis.

Let us start performing a logistic regression with the case mentioned below.

Case Scenario

The Cleveland Heart Study [2] is a long-term, population-based, prospective epidemiological investigation focused on cardiovascular diseases. It was initiated to study the causes of heart diseases and to identify risk factors associated with their development. The primary goal of the study is to understand the risk factors contributing to cardiovascular diseases, including heart attacks and strokes. The outcome variable is the angiographic disease status (Value 0: <50% diameter narrowing and 1–4: >50% diameter narrowing) denoted by *num*. The explanation of the variables is given in the Table 11.1.

Table 11.1 The list of variables used in Cleveland Heart Study

SNo	Variable	Explanation
1	age	A numerical value indicating the age of the participant
2	sex	1-Male, 0-Female
3	cp	Chest pain type
		– Value 1: typical angina
		– Value 2: atypical angina
		– Value 3: non-anginal pain
		– Value 4: asymptomatic
4	trestbps	Resting blood pressure (in mmHg on admission)
5	chol	Serum cholesterol in mg/dL
6	fbs	Fasting blood sugar >120 mg/dL (1 = true; 0 = false)
7	restecg	Resting electrocardiographic results
		– Value 0: Normal
		– Value 1: ST-T wave abnormality (T wave inversions
		– Value 2: Showing probable or definite left
8	thalch	Maximum heart rate achieved
9	exang	Exercise-induced angina (1 = yes; 0 = no)
10	oldpeak	ST depression induced by exercise relative to rest
11	slope	Slope: The slope of the peak exercise ST segment
		– Value 1: Upsloping
		– Value 2: Flat
		– Value 3: Downsloping
12	ca	Number of major vessels (0–3) colored by fluoroscopy
13	thal	3 = normal; 6 = fixed defect; 7 = reversible defect
14	num	Angiographic disease status

This dataset has been shared as "cleveland_heart.csv". It contains data from 297 participants. Our goal is to find out the relationship between the given variables and the narrowing of coronary arteries. The variables age, gender, diabetes, hypertension, dyslipidemia, etc., are associated with coronary atherosclerosis and is already known through literature search.

In this case, this analysis may have two objectives: (1) To predict whether the patient has significant coronary artery narrowing by just looking at the explanatory variables alone (without a coronary angiography) so that invasive procedures can be avoided if not needed. (2) To quantify the relationship between the explanatory (age, gender, diabetes, hypertension, etc.) variables and coronary artery narrowing.

Step 1: Loading the Dataset

```
cleveland_heart <- read.csv("cleveland_heart.csv")

#let us explore the different variables
dim(cleveland_heart)
## [1] 297  15
names(cleveland_heart)
##  [1] "Id"       "age"      "sex"      "cp"       "trestbps" "chol"
##  [7] "fbs"      "restecg"  "thalch"   "exang"    "oldpeak"  "slope"
## [13] "ca"       "thal"     "num"
```

1. The *"read.csv"* function is used to load the dataset into our working environment.
2. The *"dim"* function enables us to know the number of rows and columns. Each row represents one participant, and each column denotes a vector of values for one variable.
3. The *"names"* function gives the names of the columns as the output.

Step 2: Exploring the Variables

```
str(cleveland_heart)
## 'data.frame':    297 obs. of  15 variables:
##  $ Id      : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ age     : int  63 67 67 37 41 56 62 57 63 53 ...
##  $ sex     : int  1 1 1 1 0 1 0 0 1 1 ...
##  $ cp      : int  1 4 4 3 2 2 4 4 4 4 ...
##  $ trestbps: int  145 160 120 130 130 120 140 120 130 140 ...
##  $ chol    : int  233 286 229 250 204 236 268 354 254 203 ...
##  $ fbs     : int  1 0 0 0 0 0 0 0 0 1 ...
##  $ restecg : int  2 2 2 0 2 0 2 0 2 2 ...
##  $ thalch  : int  150 108 129 187 172 178 160 163 147 155 ...
##  $ exang   : int  0 1 1 0 0 0 0 1 0 1 ...
##  $ oldpeak : num  2.3 1.5 2.6 3.5 1.4 0.8 3.6 0.6 1.4 3.1 ...
##  $ slope   : int  3 2 2 3 1 1 3 1 2 3 ...
##  $ ca      : int  0 3 2 0 0 0 2 0 1 0 ...
##  $ thal    : int  6 3 7 3 3 3 3 3 7 7 ...
##  $ num     : int  0 2 1 0 0 0 3 0 2 1 ...
```

1. The "*str*" function enables us to understand the data type of each variable.

```
summary(cleveland_heart$age)
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   29.00   48.00   56.00   54.54   61.00   77.00
```

2. The "*summary*" function allows the values in the column to be summarized in terms of minimum value, maximum value, mean, median, first and third quartile.
3. The summary can be viewed for all the continuous variables.

Step 3: Identifying Categorical Variables and Convert Them into Factor Data Type

```
cleveland_heart$sex <- as.factor(cleveland_heart$sex)
cleveland_heart$cp <- as.factor(cleveland_heart$cp)
cleveland_heart$fbs <- as.factor(cleveland_heart$fbs)
cleveland_heart$restecg <- as.factor(cleveland_heart$restecg)
cleveland_heart$exang <- as.factor(cleveland_heart$exang)
cleveland_heart$slope <- as.factor(cleveland_heart$slope)
cleveland_heart$ca <- as.factor(cleveland_heart$ca)
cleveland_heart$thal <- as.factor(cleveland_heart$thal)
cleveland_heart$num <- as.factor(cleveland_heart$num)

# let us confirm the changes in data type
str(cleveland_heart)
## 'data.frame':    297 obs. of  15 variables:
##  $ Id      : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ age     : int  63 67 67 37 41 56 62 57 63 53 ...
##  $ sex     : Factor w/ 2 levels "0","1": 2 2 2 2 1 2 1 1 2 2 ...
##  $ cp      : Factor w/ 4 levels "1","2","3","4": 1 4 4 3 2 2 4
4 4 4 ...
##  $ trestbps: int  145 160 120 130 130 120 140 120 130 140 ...
##  $ chol    : int  233 286 229 250 204 236 268 354 254 203 ...
##  $ fbs     : Factor w/ 2 levels "0","1": 2 1 1 1 1 1 1 1 1 2 ...
##  $ restecg : Factor w/ 3 levels "0","1","2": 3 3 3 1 3 1 3 1 3 3 ...
##  $ thalch  : int  150 108 129 187 172 178 160 163 147 155 ...
##  $ exang   : Factor w/ 2 levels "0","1": 1 2 2 1 1 1 2 1 2 ...
##  $ oldpeak : num  2.3 1.5 2.6 3.5 1.4 0.8 3.6 0.6 1.4 3.1 ...
##  $ slope   : Factor w/ 3 levels "1","2","3": 3 2 2 3 1 1 3 1 2 3 ...
##  $ ca      : Factor w/ 4 levels "0","1","2","3": 1 4 3 1 1 1 3
1 2 1 ...
##  $ thal    : Factor w/ 3 levels "3","6","7": 2 1 3 1 1 1 1 1 3 3 ...
##  $ num     : Factor w/ 5 levels "0","1","2","3",..: 1 3 2 1 1 1 4
1 3 2 ...
```

1. The *"as.factor"* function converts the data type of the column vector into a factorial variable. We can confirm the data types of the variables using the function "*str*".

Step 4: Identifying and Converting the Outcome Data into a Binary Variable if Applicable

```
table(cleveland_heart$num)
##
##   0   1   2   3   4
## 160  54  35  35  13
```

1. The *"table"* function creates a frequency table of the levels of the vector containing factorial variables. The "0" indicates <50% narrowing, and "1–4" indicates increasing narrowing. To simplify the analysis, let's now consider binary levels (0 and 1) for our outcome for this analysis. We will include all narrowing >50% into one level. We will create a new variable *num2* with two levels: 0 for narrowing <50% and 1 for narrowing >50%.

```
cleveland_heart$num2 <- ifelse(cleveland_heart$num == 0, 0, 1)
table(cleveland_heart$num, cleveland_heart$num2)
##
##       0   1
##   0 160   0
##   1   0  54
##   2   0  35
##   3   0  35
##   4   0  13
```

2. The function *"ifelse"* is a conditional function that allows conditional operations to be performed on the vectors or data frames element-wise. The arguments of the function are *text_expression, x, y*; where *text_expression* is a logical that needs to be evaluated, *x* is the value to be returned if the condition expressed in *text_expression* is true, and *y* is the value to be returned if the condition expressed in *text_expression* is false.
3. The *"table"* function allows the tabulation of categorical variables and gives its frequency as output. In this case, we can see that the levels 1–4 from column *num* get converted to level 1 in column *num2*.

Step 5: Performing Univariate Logistic Regression

Before we run a predictive model, it is advisable to run univariate logistic models for each possible predictor variable. Univariate logistic regression helps identify individual predictors that may have a significant association with the outcome variable. This step is useful for preliminary variable screening, allowing you to focus on the most promising variables for inclusion in the multiple logistic regression model.

```
glm_age <- glm(num2~age, data = cleveland_heart, family = binomial)
summary(glm_age)
##
## Call:
## glm(formula = num2 ~ age, family = binomial, data = cleveland_heart)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.6070  -1.0744  -0.8323   1.1701   1.7105
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -3.05122    0.76862  -3.970  7.2e-05 ***
## age          0.05291    0.01382   3.829 0.000128 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 409.95  on 296  degrees of freedom
## Residual deviance: 394.25  on 295  degrees of freedom
## AIC: 398.25
##
## Number of Fisher Scoring iterations: 4
summary(glm(num2~sex, data = cleveland_heart, family = binomial))
##
## Call:
## glm(formula = num2 ~ sex, family = binomial, data = cleveland_heart)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.2765  -1.2765  -0.7768   1.0815   1.6404
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -1.0438     0.2326  -4.488 7.18e-06 ***
## sex1          1.2737     0.2725   4.674 2.95e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 409.95  on 296  degrees of freedom
## Residual deviance: 386.12  on 295  degrees of freedom
## AIC: 390.12
##
## Number of Fisher Scoring iterations: 4
summary(glm(num2~cp, data = cleveland_heart, family = binomial))
##
## Call:
## glm(formula = num2 ~ cp, family = binomial, data = cleveland_heart)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.6077  -0.6992  -0.6371   0.8014   1.8410
##
```

```
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -0.8267     0.4532  -1.824 0.068116 .
## cp2          -0.6650     0.5844  -1.138 0.255133
## cp3          -0.4573     0.5256  -0.870 0.384269
## cp4           1.7978     0.4906   3.664 0.000248 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 409.95  on 296  degrees of freedom
## Residual deviance: 328.75  on 293  degrees of freedom
## AIC: 336.75
##
## Number of Fisher Scoring iterations: 4
summary(glm(num2~trestbps, data = cleveland_heart, family = binomial))
##
## Call:
## glm(formula = num2 ~ trestbps, family = binomial, data =
cleveland_heart)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.4854  -1.0990  -0.9304   1.2272   1.4942
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) -2.494132   0.905112  -2.756  0.00586 **
## trestbps     0.017745   0.006807   2.607  0.00914 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 409.95  on 296  degrees of freedom
## Residual deviance: 402.88  on 295  degrees of freedom
## AIC: 406.88
##
## Number of Fisher Scoring iterations: 4
summary(glm(num2~chol, data = cleveland_heart, family = binomial))
##
## Call:
## glm(formula = num2 ~ chol, family = binomial, data = cleveland_heart)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -1.546  -1.102  -1.029   1.233   1.405
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) -0.929987   0.576336  -1.614    0.107
## chol         0.003130   0.002279   1.373    0.170
##
## (Dispersion parameter for binomial family taken to be 1)
##
```

```
##     Null deviance: 409.95  on 296  degrees of freedom
## Residual deviance: 408.03  on 295  degrees of freedom
## AIC: 412.03
##
## Number of Fisher Scoring iterations: 4
summary(glm(num2~fbs, data = cleveland_heart, family = binomial))
##
## Call:
## glm(formula = num2 ~ fbs, family = binomial, data = cleveland_heart)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -1.119  -1.111  -1.111   1.245   1.245
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -0.15781    0.12588  -1.254    0.210
## fbs1         0.01805    0.33064   0.055    0.956
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 409.95  on 296  degrees of freedom
## Residual deviance: 409.94  on 295  degrees of freedom
## AIC: 413.94
##
## Number of Fisher Scoring iterations: 3
summary(glm(num2~restecg, data = cleveland_heart, family = binomial))
##
## Call:
## glm(formula = num2 ~ restecg, family = binomial, data = cleveland_heart)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.6651  -0.9681  -0.9681   1.1083   1.4022
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -0.5145     0.1704  -3.018  0.00254 **
## restecg1      1.6131     1.1672   1.382  0.16698
## restecg2      0.6792     0.2380   2.854  0.00432 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 409.95  on 296  degrees of freedom
## Residual deviance: 400.28  on 294  degrees of freedom
## AIC: 406.28
##
## Number of Fisher Scoring iterations: 4
summary(glm(num2~thalch, data = cleveland_heart, family = binomial))
##
## Call:
## glm(formula = num2 ~ thalch, family = binomial, data = cleveland_heart)
##
```

```
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.1547  -0.9287  -0.6085   1.0657   2.1340
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)  6.472142   1.002574   6.456 1.08e-10 ***
## thalch      -0.044312   0.006627  -6.687 2.28e-11 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 409.95  on 296  degrees of freedom
## Residual deviance: 351.97  on 295  degrees of freedom
## AIC: 355.97
##
## Number of Fisher Scoring iterations: 4
summary(glm(num2~exang, data = cleveland_heart, family = binomial))
##
## Call:
## glm(formula = num2 ~ exang, family = binomial, data = cleveland_heart)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.6966  -0.8699  -0.8699   0.7357   1.5200
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -0.7768     0.1522  -5.103 3.34e-07 ***
## exang1        1.9454     0.2831   6.871 6.37e-12 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 409.95  on 296  degrees of freedom
## Residual deviance: 355.48  on 295  degrees of freedom
## AIC: 359.48
##
## Number of Fisher Scoring iterations: 4
summary(glm(num2~oldpeak, data = cleveland_heart, family = binomial))
##
## Call:
## glm(formula = num2 ~ oldpeak, family = binomial, data =
cleveland_heart)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.3774  -0.8920  -0.7638   1.0196   1.6580
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -1.0828     0.1823  -5.940 2.86e-09 ***
## oldpeak       0.9161     0.1381   6.635 3.25e-11 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
```

```
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 409.95  on 296  degrees of freedom
## Residual deviance: 350.48  on 295  degrees of freedom
## AIC: 354.48
##
## Number of Fisher Scoring iterations: 4
summary(glm(num2~slope, data = cleveland_heart, family = binomial))
##
## Call:
## glm(formula = num2 ~ slope, family = binomial, data = cleveland_heart)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.4483  -0.7743  -0.7743   0.9288   1.6438
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -1.0512     0.1936  -5.429 5.65e-08 ***
## slope2        1.6686     0.2637   6.327 2.50e-10 ***
## slope3        1.3389     0.4816   2.780  0.00543 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 409.95  on 296  degrees of freedom
## Residual deviance: 365.16  on 294  degrees of freedom
## AIC: 371.16
##
## Number of Fisher Scoring iterations: 4
summary(glm(num2~ca, data = cleveland_heart, family = binomial))
##
## Call:
## glm(formula = num2 ~ ca, family = binomial, data = cleveland_heart)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.9479  -0.7736  -0.7736   0.8834   1.6446
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -1.0531     0.1731  -6.083 1.18e-09 ***
## ca1           1.7928     0.3167   5.660 1.51e-08 ***
## ca2           2.5412     0.4529   5.611 2.01e-08 ***
## ca3           2.7878     0.6497   4.291 1.78e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 409.95  on 296  degrees of freedom
## Residual deviance: 333.93  on 293  degrees of freedom
## AIC: 341.93
##
## Number of Fisher Scoring iterations: 4
summary(glm(num2~thal, data = cleveland_heart, family = binomial))
##
```

```
## Call:
## glm(formula = num2 ~ thal, family = binomial, data = cleveland_heart)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.7024  -0.7151  -0.7151   0.7316   1.7257
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -1.2333     0.1868  -6.601 4.07e-11 ***
## thal6         1.9264     0.5338   3.609 0.000307 ***
## thal7         2.4148     0.2886   8.367  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 409.95  on 296  degrees of freedom
## Residual deviance: 323.39  on 294  degrees of freedom
## AIC: 329.39
##
## Number of Fisher Scoring iterations: 4
```

1. We have to run the *"glm"* function for 'num2' as a function of age and assign it to an R object glm_age. Next, we call for a summary of the object *glm_age* to identify the significance of the association. The p-value 0.000128 shows a significant association.
2. Similarly, we have to run the *summary* of *"glm"* function for each of the predictor variables. We can observe that all the variables except 'chol' and 'fbs' have a significant association with the dependent variable 'num2'.
3. While univariate logistic regression provides valuable insights, it's important to note that the results may change when multiple predictors are considered simultaneously. The ultimate goal is to build a robust and accurate model that captures the combined effects of multiple variables on the outcome. We are now ready to perform multiple logistic regression.

Step 6: Running the Multiple Logistic Regression Model

```
names(cleveland_heart)
##  [1] "Id"       "age"      "sex"      "cp"       "trestbps" "chol"
##  [7] "fbs"      "restecg"  "thalch"   "exang"    "oldpeak"  "slope"
## [13] "ca"       "thal"     "num"      "num2"
glm_fit <- glm(num2~age+sex+cp+trestbps+restecg+thalch+exang+oldpeak+slope
+ca+thal, data = cleveland_heart, family = binomial)
```

1. The function to use is "*glm*". The term before "~" is the outcome variable in the formula argument. The terms after the "~" are the explanatory variables separated by the "+". The "family = binomial" instructs R to run the logistic regression model. The model has been assigned to an R object *glm_fit*.

```
summary(glm_fit)
##
## Call:
## glm(formula = num2 ~ age + sex + cp + trestbps + restecg + thalch +
##     exang + oldpeak + slope + ca + thal, family = binomial, data =
cleveland_heart)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.9584  -0.4830  -0.1262   0.3038   2.9714
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -5.42258    2.87663  -1.885 0.059424 .
## age         -0.01999    0.02462  -0.812 0.416739
## sex1         1.50311    0.53136   2.829 0.004673 **
## cp2          1.53614    0.80320   1.913 0.055810 .
## cp3          0.39251    0.69210   0.567 0.570625
## cp4          2.47885    0.70545   3.514 0.000442 ***
## trestbps     0.02627    0.01155   2.275 0.022931 *
## restecg1     1.06852    2.35609   0.454 0.650179
## restecg2     0.55652    0.38913   1.430 0.152669
## thalch      -0.01824    0.01149  -1.588 0.112343
## exang1       0.63238    0.44294   1.428 0.153379
## oldpeak      0.43366    0.23803   1.822 0.068474 .
## slope2       1.28588    0.48412   2.656 0.007905 **
## slope3       0.48319    0.93714   0.516 0.606129
## ca1          2.16830    0.50403   4.302 1.69e-05 ***
## ca2          3.16379    0.77174   4.100 4.14e-05 ***
## ca3          2.03773    0.88461   2.304 0.021249 *
## thal6       -0.28551    0.79588  -0.359 0.719798
## thal7        1.46327    0.43958   3.329 0.000872 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 409.95  on 296  degrees of freedom
## Residual deviance: 185.10  on 278  degrees of freedom
## AIC: 223.1
##
## Number of Fisher Scoring iterations: 6
```

2. The "*summary*" function allows us to review the model properties.
3. Next, we will interpret the output of the "*summary*" function. There are five rows in the output:
   (a) Call—the original call to the "*glm*" function.

(b) The 'Deviance residuals' part of the output provides the summary of the model performance. Deviance residuals are scaled versions of the components of the binomial log-likelihood.
(c) Coefficients provide the weights associated with each explanatory variable.
(d) Null and residual deviance—The null deviance shows how well the model predicts the response variable with only the intercept. The residual deviance indicates how well a model can predict the response variable. The lower the value, the better the model can predict the value of the response variable.
(e) The number of Fisher scoring iterations—Fisher's scoring algorithm is a derivative of Newton's method for solving maximum likelihood problems. (This can be ignored. Explaining this is outside the scope of this book)

We need to look at the coefficients section in detail to quantify the associations. There are five columns for this section:

(a) Variable name—Contains the names of the explanatory variables.
(b) Estimate—Coefficients for the "log" of the variable values. The estimate is the "log of the Odds Ratio" of the outcome variable when there is one unit change in the corresponding independent variable.
(c) Std. Error—it is the "standard deviation/square root(n)" of the estimate.
(d) z value—Z values are calculated by dividing the coefficient estimate by the standard error. They show the degree of uncertainty that surrounds the coefficient's point estimate. It follows the z distribution with a mean of 0 and a standard deviation of 1. Hence, a z value of >2 indicates that the estimate for the variable is statistically significant from 0, and hence, the variable is associated with the outcome.
(e) p-value—P-value establishes the likelihood of extreme outcomes from a statistical hypothesis test by assuming that the null hypothesis is true. "$p < 0.05$" means a statistically significant association.

Step 7: Interpretation of the Model Output

1. In order to quantify the association, we need to extract the regression coefficients for the variables. For example, the coefficient for age is −0.0199. This means that the logarithm of the odds ratio for coronary artery narrowing is −0.0199 with every 1-year increase in age.
2. But, it may be difficult to interpret the "log odds ratio". Thus, we should calculate the Odds Ratio, which becomes more interpretable.
3. Perform the inverse function of *"log"* which is 'exponent' which can be done in R by the *"exp"* function.

```
or_age <- exp(-0.0199)
or_age
## [1] 0.9802967
```

4. The exp(estimate) for any continuous predictor variable can be interpreted as age: For every unit increase in age, the odds of having a coronary artery narrowing is 0.98 (not significant in terms of p-value). All the other continuous variables can be interpreted similarly.
5. The exp(estimate) for any categorical predictor variable can be interpreted like sex as follows: For the change in sex from females to males, the odds of having coronary artery narrowing increases by log odds ratio of 1.50311, i.e., 4.49 times.
6. Now that we know how to interpret the output of the estimate column, we need to find out which variables are significantly related to coronary artery narrowing. To achieve this objective, we need to observe their p-value (fifth column). The variables having a p-value of <0.05 are significantly associated with the outcome variable. The following variables are significantly related:

   (a) sex1—Males compared to females.
   (b) cp4—as compared to the baseline of cp1
   (c) trestbps
   (d) slope2—the slope of 2 as compared to 1.
   (e) ca—1.0, 2.0, and 3.0 as compared to baseline of 0.0.
   (f) thal—7.0 as compared to 3.0.

7. For all the categorical variables, a reference category is decided, and the output for all other categories is interpreted depending on the reference category as we did for the variable 'sex'.

Step 8: Evaluation of the Model

Once the model is built, we can check how good a model we have built. For this, we need to first compare the model predictions with the actual status of the patients.

To predict the output of coronary artery narrowing on the same data

```
pred_lr <- predict(glm_fit, type = "response")
summary(pred_lr)
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## 0.001351 0.062285 0.341485 0.461280 0.914219 0.998988
```

To convert probabilities of ≤0.5 and >0.5 to outputs of 0 and 1, respectively

```
cleveland_heart$pred_num2 <- ifelse(pred_lr > 0.5, 1, 0)
```

To look at the predicted output vs. actual output (aka confusion matrix)

```
table(cleveland_heart$num2, cleveland_heart$pred_num2)
##
##       0   1
##   0 147  13
##   1  23 114
```

1. We have created a table of the true status of narrowing (in rows) versus the predicted status of narrowing (in columns). This is called a *confusion matrix*. Now, the diagonals indicate the correct predictions by the logistic regression model, which indicates the accuracy of the model. Accuracy is the ratio of correct predictions to the overall number of predictions.
   Similarly, four other parameters can be calculated:
   (a) Sensitivity or Recall—Proportion of true positives predicted correctly.
   (b) Specificity—Proportion of true negatives predicted correctly.
   (c) Positive Predictive Value (PPV) or Precision—Proportion of true positives among positive predictions.
   (d) Negative Predictive Value (NPV)—Proportion of true negatives among negative predictions.
   Lets calculate all of these from the confusion matrix.

```
N = 147+13+23+114
accuracy = (147+114)/N ; accuracy
## [1] 0.8787879
sensitivity = 114/(114+23) ; sensitivity
## [1] 0.8321168
specificity = 147/(147+13) ; specificity
## [1] 0.91875
ppv = 114/(114+13) ; ppv
## [1] 0.8976378
npv = 147/(147+23) ; npv
## [1] 0.8647059
```

2. We can obtain our model's sensitivity, specificity, PPV, and NPV from the confusion matrix, as shown in the codes above.
3. Finally, we may calculate the Area Under the Curve (AUC) of the Receiver Operator Characteristic (ROC) curve. AUC can take a maximum value of 1. Hence, a value close to 1 indicates that the model is good. The ROC is a plot with sensitivity on the y-axis and (1—specificity) on the x-axis. Let us calculate the AUC. (ROC analysis using package `pROC` needs to be done) [3]

```
library(pROC)
## Warning: package 'pROC' was built under R version 4.3.1
## Type 'citation("pROC")' for a citation.
##
## Attaching package: 'pROC'
## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
roc_lr <- roc(cleveland_heart$num2, pred_lr)
## Setting levels: control = 0, case = 1
## Setting direction: controls < cases
auc(roc_lr)
## Area under the curve: 0.9394
```

The AUC of the ROC curve is 0.939, indicating that our logistic regression model can differentiate between coronary artery narrowing <50% versus >50% with an accuracy of 93.9%, which means our model performs very well in predicting the outcome (The explanation of codes for ROC has been provided in the next chapter).

References

1. Sperandei S. Understanding logistic regression analysis. Biochem Med (Zagreb). 2014;24(1):12–8.
2. Janosi A, Steinbrunn W, Pfisterer M, Detrano R. Heart disease data set. 1988. Available from: https://archive.ics.uci.edu/ml/datasets/Heart+Disease
3. Robin X, Turck N, Hainard A, et al. pROC: an open-source package for R and S+ to analyze and compare ROC curves. BMC Bioinformatics. 2011;12:77.

Chapter 12
Receiver Operating Characteristic (ROC) Curve Analysis for Diagnostic Studies

Anand Srinivasan and Archana Mishra

Inroduction [1]

In previous chapters, we dealt with hypothesis testing used to make inferences about population parameters or test the validity of assumptions. Hypothesis testing involves comparing observed data to a null hypothesis and determining whether the evidence supports rejecting the null hypothesis in favor of an alternative. On the other hand, receiver operating characteristic analysis is primarily used to evaluate the performance of classification models, particularly in binary classification scenarios such as diagnosis of the presence or absence of a disease. In modern medicine, the assessment of diagnostic tests is crucial not only for confirming the presence of a disease but also for excluding the disease in healthy individuals. When dealing with diagnostic tests that yield binary outcomes (positive/negative), the conventional evaluation involves using sensitivity and specificity as indicators of the test's accuracy compared to a gold standard. However, in situations where test results are expressed on an ordinal scale (e.g., a 5-point scale: "definitely normal," "probably normal," "uncertain," "probably abnormal," "definitely abnormal") or as continuous values, sensitivity, and specificity can be computed across various threshold values. Consequently, sensitivity and specificity fluctuate across different thresholds, with an inverse relationship between the two. The best threshold is the one where the sum of specificity and sensitivity is the maximum.

To represent this relationship graphically, a plot of sensitivity against 1-specificity, known as the Receiver Operating Characteristic (ROC) curve, is constructed. The

Supplementary Information The online version contains supplementary material available at https://doi.org/10.1007/978-981-97-6980-3_12.

A. Srinivasan (✉) · A. Mishra
Department of Pharmacology, All India Institute of Medical Sciences, Bhubaneswar, Odisha, India
e-mail: anandsrinivasan@aiimsbhubaneswar.edu.in

A. Srinivasan et al. (eds.), *R for Basic Biostatistics in Medical Research*,
https://doi.org/10.1007/978-981-97-6980-3_12

Area Under the Curve (AUC) derived from the ROC curve is a meaningful measure of accuracy. This curve assumes a central role in evaluating tests' diagnostic capability to discern subjects' true condition, identifying optimal cut-off values, and comparing two alternative diagnostic tasks conducted on the same subjects. The ROC curve and AUC provide valuable insights into the effectiveness and discriminatory power of diagnostic tests across a spectrum of possible results, offering a comprehensive approach to diagnostic test evaluation beyond binary outcomes. A curve with more discriminant capacity is found progressively towards the upper left corner of the ROC space. With this introduction, we will move ahead and perform an ROC analysis.

Case Scenario

A study was done to evaluate the diagnostic efficacy of the Ki-67 labeling index in diagnosing patients with glioblastoma. The data contains the diagnosis of glioblastoma present or absent via a 'gold standard' test (0 = glioblastoma absent and 1 = glioblastoma present). Ki-67 labeling index has been provided as the values from a 'new' test. You are assigned a task to evaluate the new test for determining accuracy and a threshold value that can be used to differentiate between diseased and non-diseased.

Step 1: Loading the Dataset

```
r_dat <- read.csv("roc.csv")
```

Step 2: Checking the Data Type of the Variables and Performing Required Conversions

```
str(r_dat)
## 'data.frame':    100 obs. of  2 variables:
##  $ gold_std: int  0 0 0 0 0 0 0 0 0 0 ...
##  $ new     : int  34 30 34 29 31 29 26 35 31 26 ...
r_dat$gold_std <- as.factor (r_dat$gold_std)
```

1. The variable gold_std is binary categorical variable and thus needs to be converted into factor data type.

Step 3: Performing ROC Analysis

In order to perform an ROC analysis, we first need to install and load a package *pROC* [2]

```
library(pROC)
## Warning: package 'pROC' was built under R version 4.3.1
## Type 'citation("pROC")' for a citation.
##
## Attaching package: 'pROC'
## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
par_roc <- roc(r_dat$gold_std,r_dat$new)
## Setting levels: control = 0, case = 1
## Setting direction: controls > cases
par_roc
##
## Call:
## roc.default(response = r_dat$gold_std, predictor = r_dat$new)
##
## Data: r_dat$new in 50 controls (r_dat$gold_std 0) > 50 cases
(r_dat$gold_std 1).
## Area under the curve: 0.8996
```

1. The single function *"roc"* from the package does the entire ROC analysis required. In the required arguments, the gold standard in its binomial form should come first, followed by the continuous data from the new test.
2. *par_roc* is the R object that contains all the output parameters.

Step 4: Interpreting the Output

```
auc(par_roc)
## Area under the curve: 0.8996
```

1. The *"auc"* function gives the area under the curve for the ROC. The maximum AUC is 1. The higher the AUC, the better the accuracy of the tests will be. In this example, we have obtained an AUC of 0.8996, which implies that the new test can discriminate between glioblastoma present and absent participants with 89.96% accuracy.
2. The next step is to determine the best possible cut-off value for the new parameter. We can visualize the possible specificities and sensitivities for various cut-off values and decide the best possible value manually, as provided below.

```
values = data.frame(par_roc$thresholds, par_roc$sensitivities,
                    par_roc$specificities)
values
##    par_roc.thresholds par_roc.sensitivities par_roc.specificities
## 1                 Inf                  1.00                  0.00
## 2                36.0                  1.00                  0.02
## 3                34.5                  1.00                  0.04
## 4                33.5                  1.00                  0.12
## 5                32.5                  1.00                  0.18
## 6                31.5                  1.00                  0.32
## 7                30.5                  1.00                  0.48
## 8                29.5                  0.98                  0.56
## 9                28.5                  0.92                  0.74
## 10               27.5                  0.86                  0.84
## 11               26.5                  0.62                  0.86
## 12               25.5                  0.46                  0.96
## 13               24.5                  0.32                  0.96
## 14               23.5                  0.26                  1.00
## 15               22.5                  0.24                  1.00
## 16               21.5                  0.18                  1.00
## 17               20.5                  0.06                  1.00
## 18               18.5                  0.04                  1.00
## 19               -Inf                  0.00                  1.00
```

1. We can observe the sensitivity and specificity at each threshold point. The best threshold can be determined by a trade-off between sensitivity and specificity. This process is automated in the package by the *"coords"* function.

```
coords(par_roc,"best",ret=c("t", "sp", "se","tp","tn","fp","fn",
                            "ppv","npv"))
##           threshold specificity sensitivity tp tn fp fn       ppv       npv
## threshold      27.5        0.84        0.86 43 42  8  7 0.8431373
0.8571429
```

1. The function *"coords"* provides the best value, which can be explored further along with its sensitivity, specificity, and other values. The arguments of the function being the R object for the ROC analysis, followed by "best" for the best possible threshold and an argument for the formation of a vector with desired parameters like the threshold value, specificity, sensitivity, true positive, true negative, false negative, positive predictive value and negative predictive value for the threshold chosen.
2. We can appreciate that at a threshold of 27.5, the new test has a specificity of 84% and a sensitivity of 86%.
3. The confidence intervals for the specificity at a given sensitivity and sensitivity at a given specificity can be determined by the following code.

```
ci.se(par_roc,specificities = 0.84)
## 95% CI (2000 stratified bootstrap replicates):
##    sp se.low se.median se.high
##  0.84 0.4733      0.84    0.96
ci.sp(par_roc,sensitivities = 0.86)
## 95% CI (2000 stratified bootstrap replicates):
##    se sp.low sp.median sp.high
##  0.86 0.6575    0.8224  0.9238
```

Step 5: Generation of ROC Plot

```
plot(par_roc,xlab = "Specificity",ylab = "Sensitivity", col="Red")
```

1. The *"plot"* function plots the output (Fig. 12.1) of R object *par_roc* with the x-axis label as "Specificity" and the y-axis label as "sensitivity".
2. The diagonal line in the ROC space connects the point (0, 0) to (1, 1). Along this line, the sensitivity and specificity are equal, and the area under the curve (AUC) is 0.5. Essentially, if a classifier's performance is equivalent to random chance, its ROC curve would coincide with the diagonal line.

Thus, we hope that the readers can perform ROC analysis at ease with just a few lines of code.

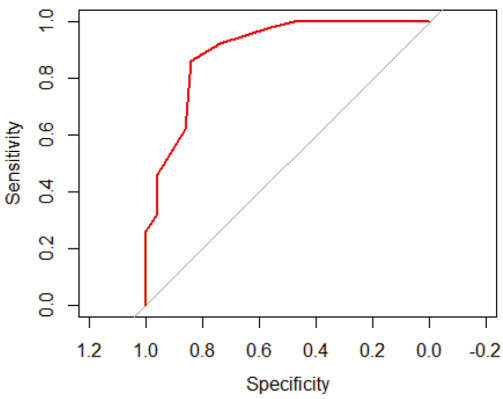

Fig. 12.1 Receiver Operating Characteristic (ROC) curve showing the graph generated by plotting the specificity and sensitivity combinations coordinates for various cut-off points

References

1. Hajian-Tilaki K. Receiver operating characteristic (ROC) curve analysis for medical diagnostic test evaluation. Caspian J Intern Med. 2013 Spring;4(2):627–35.
2. Robin X, Turck N, Hainard A, et al. pROC: an open-source package for R and S+ to analyze and compare ROC curves. BMC Bioinformatics. 2011;12:77.

Chapter 13
Survival Analysis for Time to Event-Based Outcome

Guruprasad Padmanaban, Archana Mishra, and Anand Srinivasan

Introduction [1]

Survival analysis is a collection of statistical procedures for data analysis for which the outcome variable of interest is time until an event occurs. Time may be expressed as years, months, or days from the beginning of follow-up till an event occurs. Events may be death, hospitalization, relapse of a disease, or any other event of interest. This type of data cannot be analyzed using usual parametric or non-parametric tests of hypothesis testing because only a few individuals experience events. Events are distributed in a skewed manner, i.e., there may be many early events, with a few occurring later in the study or vice-versa, and lastly, the actual time to event is unknown, and the data needs censoring.

Goals of Survival Analysis

1. To estimate and interpret survivor and hazard functions from survival data.
2. To compare survivor and hazard functions between two groups.
3. To assess the relationship of explanatory variables to survival time.

Supplementary Information The online version contains supplementary material available at https://doi.org/10.1007/978-981-97-6980-3_13.

G. Padmanaban
Metadata Management for Clinical Operations, Novartis, Hyderabad, India

A. Mishra · A. Srinivasan (✉)
Department of Pharmacology, All India Institute of Medical Sciences, Bhubaneswar, Odisha, India
e-mail: anandsrinivasan@aiimsbhubaneswar.edu.in

A. Srinivasan et al. (eds.), *R for Basic Biostatistics in Medical Research*,
https://doi.org/10.1007/978-981-97-6980-3_13

Survival Function
This is the probability that a person survives longer than the specified time 't'. At the start of the study, since no one has gotten the event yet, the probability of surviving past time 0 is 1. t = 0, S(t) = S(0) = 1; While, if the study period is increased to infinity, eventually nobody would survive, survivor function would reach zero.

$t = \infty$, $S(t) = S(\infty) = 0$.

Hazard Function
This is the instantaneous potential per unit of time for the event, given that the individual has survived up to time t. It is denoted by h(t). The survival model is usually written in terms of the hazard function.

There are a few commonly used models used to analyze the survival data:

1. Kaplan-Meier survival analysis: Kaplan-Meier survival analysis is a statistical method used to estimate the probability of an event, typically survival, over time. It accounts for censored data and provides a survival curve illustrating the cumulative probability of an event occurring. Widely used in medical and biological research, it helps assess the survival distribution and compare different groups. The analysis is crucial for studying time-to-event outcomes, such as patient survival in clinical trials or disease progression.
2. Comparing survival curves: The tests deployed for comparing survival curves are the log-rank test, Breslow test, and Tarone-Ware test. The log-rank test is a statistical method resembling a chi-square test for large samples, which assesses Kaplan-Meier curves by employing a criterion yielding a comprehensive comparison. This test evaluates observed versus expected cell counts across outcome categories, with the categories determined by the ordered failure times for the entire dataset under examination. In essence, it scrutinizes how well the observed event rates align with the expected rates at different time points, providing insights into the significance of differences between survival curves.
3. Cox regression: The Cox Proportional Hazards Model is a statistical method used to assess the relationship between the survival time of individuals and several predictor variables without making assumptions about the shape of the baseline hazard function. The model estimates hazard ratios, indicating the relative risk of an event occurring at any given time. It's commonly employed in medical and epidemiological research to examine factors influencing the time until an event, such as death or disease recurrence.

With this understanding of survival analysis, let us learn how to perform survival analysis on a given data.

Case Scenario

A clinical trial was conducted to evaluate the efficacy of a new anti-pancreatic cancer drug, 'pancrean' in a randomized add-on placebo-controlled trial. Forty-two patients with pancreatic cancer were enrolled and randomized to two treatment arms: 'pancrean' plus standard treatment versus placebo plus standard therapy in an allocation ratio of 1:1. The primary outcome measure is time to death. You are asked to analyze if the drug 'pancrean' has an advantage over a placebo.

Step 1: Loading the Dataset

```
panc <- read.csv("survival.csv")
```

1. We are loading the dataset in our working environment for further analysis.
2. Our data appears like this:
   The outcome in the test arm is as follows:
   – Nine failures (death).
   – 12 patients were censored. Censoring occurs when the event of interest does not occur until the participants are followed up in a clinical trial.
   – The failure/censor times are 6,6,6,7,10,13,16,22,23,6+,9+,10+,11+,17+,19+,20+,25+,32+,32+,34+,35+ ("+" indicates censoring).

The outcome in the control arm is as follows:

– 21 failures
– no censors
– The failure times are 1,1,2,2,3,4,4,5,5,8,8,8,8,11,11,12,12,15,17,22,23.

Step 2: Loading the Required Packages

We have to install packages (*survival, splines2, dplyr, ggfortify, survminer*) using function *"install.packages"* or the packages tab in the lower right panel before we start the analysis [2–6].

```
library(survival)
## Warning: package 'survival' was built under R version 4.2.3
library(splines2)
## Warning: package 'splines2' was built under R version 4.2.3
##
## Attaching package: 'splines2'
## The following object is masked from 'package:survival':
##
##     nsk
library(dplyr)
## Warning: package 'dplyr' was built under R version 4.2.3
##
## Attaching package: 'dplyr'
## The following objects are masked from 'package:stats':
##
##     filter, lag
## The following objects are masked from 'package:base':
##
```

```
##     intersect, setdiff, setequal, union
library(ggfortify)
## Warning: package 'ggfortify' was built under R version 4.2.3
## Loading required package: ggplot2
## Warning: package 'ggplot2' was built under R version 4.2.3
library(survminer)
## Warning: package 'survminer' was built under R version 4.2.3
## Loading required package: ggpubr
## Warning: package 'ggpubr' was built under R version 4.2.3
##
## Attaching package: 'survminer'
## The following object is masked from 'package:survival':
##
##     myeloma
```

Step 3: Plotting Survival Curve

Survival Curve for Test Arm

We will create a survival object and then plot the survival curve for the test arm.

```
surv_test <- Surv(panc$time[which(panc$group==1)],
panc$delta[which(panc$group==1)])
survfit_test <- survfit(surv_test ~ 1)
```

1. The "`Surv`" function is used to create a survival object. The function's arguments are the time of occurrence of the event and the event experienced or censoring done. This survival object has been named *surv_test*. We need not remind that `[which(panc$group==1)]` argument has been used to specify that we need to create a survival curve of the test group, labeled as '`1`' in the given data file.
2. The object thus created is passed on to the function "`survfit`" to generate a survival curve. The `~1` denotes that an intercept-only model (no covariates) is fitted here.

The next step is to plot the survival curve.

```
ggsurvival <- autoplot(survfit_test, conf.int.fill = "blue", censor.size = 5,
censor.colour = "red", conf.int = FALSE)  +
  labs(y="Survival", x="Time in Weeks") +
  theme_bw()
ggsurvival
```

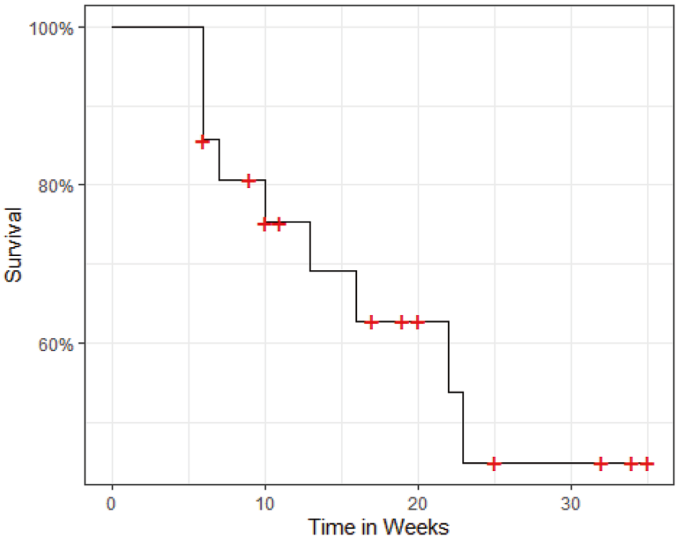

Fig. 13.1 Kaplan-Meier curve for the test ('pancrean' plus standard therapy) group

3. The *"autoplot"* function is used to plot the survival plot created using the *"survfit"* function.
4. We can appreciate that the survival is 100% at time 0. At every point after, the survival rate will either decrease or remain the same. It will never increase. The red crosses indicate censors. Censors do not affect the survival (Fig. 13.1).

Survival Curve for Control Arm

```
survfit_control <- survfit(Surv(panc$time[which(panc$group==2)],
                                panc$delta[which(panc$group==2)]) ~ 1)
ggsurvival2 <- autoplot(survfit_control, conf.int.fill = "blue", censor.size
= 5, censor.colour = "red", conf.int = FALSE)  +
  labs(y="Survival", x="Time in Weeks") +
  theme_bw()
ggsurvival2
```

5. We repeat the earlier steps from the test arm to the control arm to obtain a survival curve for the control arm. We can observe no red crosses as there were no censors for the control arm (Fig. 13.2).

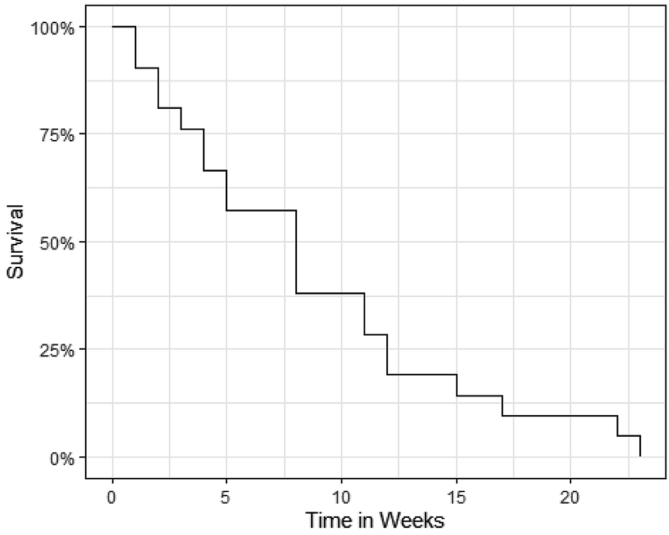

Fig. 13.2 Kaplan-Meier curve for the test (placebo plus standard therapy) group

Survival Curves of Both Arms Together

The survival curves for individual groups will be required when we have a single cohort in our study. In this case, we had two treatment groups. So, let us plot both survival curves in the same plot and observe the median survival time for both groups. The median survival time for 'pancrean' group is 23 weeks, whereas the median survival time for the placebo group is 8 weeks.

```
survfit_both <- survfit(Surv(time, delta) ~ group, data = panc)
survfit_both
## Call: survfit(formula = Surv(time, delta) ~ group, data = panc)
##
##          n events median 0.95LCL 0.95UCL
## group=1 21      9     23      16      NA
## group=2 21     21      8       4      12
ggsurvival_both <- autoplot(survfit_both, censor.size = 5, conf.int = F) +
  labs(y="Survival", x="Time in Weeks") +
  theme_bw()
ggsurvival_both
```

6. The two survival curves appear to differ from each other (Fig. 13.3). The test arm seems to have better survival than the control arm. However, the study's objective is to determine whether this difference is statistically significant. For that, we will perform a log-rank test.

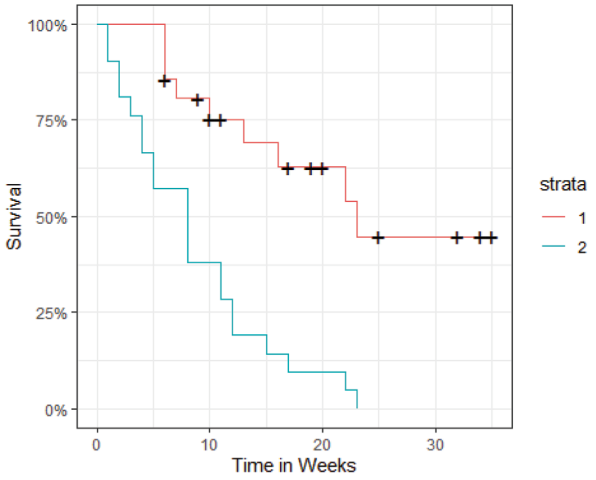

Fig. 13.3 Kaplan-Meier curve for the test and control group

Step 4: Comparing Survival Curves: Log Rank Test

```
survdiff(Surv(time, delta) ~ group, data = panc,rho = 0)
## Call:
## survdiff(formula = Surv(time, delta) ~ group, data = panc, rho = 0)
##
##           N Observed Expected (O-E)^2/E (O-E)^2/V
## group=1 21        9     19.3      5.46      16.8
## group=2 21       21     10.7      9.77      16.8
##
##  Chisq= 16.8  on 1 degrees of freedom, p= 4e-05
```

1. The "*survdiff*" function can be used to compare two survival curves. The arguments of the function are the survival curve object and the specification of the type of test used in *rho*. *rho = 0* is used for specifying log-rank or Mantel-Haenszel test, and *rho = 1* is Peto & Peto modification of the Gehan-Wilcoxon test (outside the scope of this book).
2. The test arm is significantly superior to the control concerning survival ($p < 0.001$). We conclude that the 'pancrean' is significantly better than a placebo as an add-on therapy for the treatment of pancreatic cancer.

Step 5: Modifying the Kaplan-Meier Curve

We will add some more information to the plot to make it self-explanatory.

```
ggsurvplot(survfit_both,
           pval = TRUE, conf.int = TRUE,
           risk.table = TRUE, # Add risk table
           risk.table.col = "strata", # Change risk table color by groups
           linetype = "strata", # Change line type by groups
           surv.median.line = "hv", # Specify median survival
           ggtheme = theme_bw(),
           palette = c("#999999","#333333"),
           legend.labs=c("Pancrean","Control"))
```

1. Using the same function, *"ggsurvplot"*, we added the number at risk, changed the line type to differentiate between the groups, and added the p-value to the plot (Fig. 13.4).

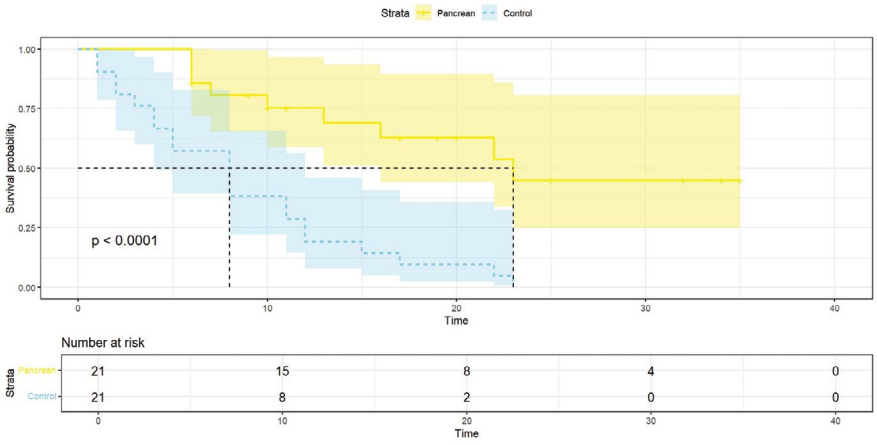

Fig. 13.4 Kaplan-Meier curve comparing test and control groups

Step 6: Calculation of Hazard Ratio

```
coxmodel1 <- coxph(Surv(time, delta) ~ group, data = panc)
summary(coxmodel1)
## Call:
## coxph(formula = Surv(time, delta) ~ group, data = panc)
##
##   n= 42, number of events= 30
##
##         coef exp(coef) se(coef)     z Pr(>|z|)
## group 1.5721    4.8169   0.4124 3.812 0.000138 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
##       exp(coef) exp(-coef) lower .95 upper .95
## group     4.817     0.2076     2.147     10.81
##
## Concordance= 0.69  (se = 0.041 )
## Likelihood ratio test= 16.35  on 1 df,   p=5e-05
## Wald test            = 14.53  on 1 df,   p=1e-04
## Score (logrank) test = 17.25  on 1 df,   p=3e-05
```

1. We have run a cox-proportional hazard model using *"coxph"* function. The arguments are a formula for the survival object as a function of the covariate (here, we have used 'group') and data specification. If more than one covariate is expected to affect the hazard ratio, they may be added using a '+' sign, as in other regression models.
2. The Cox regression results can be interpreted as follows:
   (a) *Statistical significance*. The column marked "z" gives the Wald statistic value. It corresponds to the ratio of each regression coefficient to its standard error (z = coef/se(coef)). The Wald statistic evaluates whether a given variable's beta (β) coefficient is statistically significantly different from 0. From the output above, we can conclude that the variable group has highly statistically significant coefficients.
   (b) *The regression coefficients*. The second feature in the Cox model results is the sign of the regression coefficients (coef). A positive sign means that the hazard (risk of death) is higher, and thus the prognosis worse, for subjects with higher values of that variable. The variable group is encoded as 1: 'pancrean', 2: placebo. The R summary for the Cox model gives the hazard ratio (HR) for the second group relative to the first group, that is, placebo versus 'pancrean'. The beta coefficient for group = exp(1.572) indicates that the placebo group has a higher risk of death (lower survival rates) than the 'pancrean' group in these data.

(c) *Hazard ratios*. The exponentiated coefficients (exp(coef) = exp(1.572) = 4.817), also known as hazard ratios, give the effect size of covariates. For example, the administration of a placebo increases the probability of death by 4.817 times.
(d) *Confidence intervals of the hazard ratios*. The summary output also gives upper and lower 95% confidence intervals for the hazard ratio (exp(coef)), lower 95% bound = 2.147, upper 95% bound = 10.81.
(e) *Statistical significance of the model*. Finally, the output gives p-values for three alternative tests for the overall significance of the model: The likelihood-ratio test, Wald test, and score log-rank statistics. These three methods are asymptotically equivalent. For large enough N, they will give similar results. For small N, they may differ somewhat. The Likelihood ratio test has better behavior for small sample sizes, so it is generally preferred.

We hope you will be able to analyze your time-to-event (survival) data amicably.

References

1. Rich JT, Neely JG, Paniello RC, Voelker CC, Nussenbaum B, Wang EW. A practical guide to understanding Kaplan-Meier curves. Otolaryngol Head Neck Surg. 2010;143(3):331–6.
2. Therneau TM. A package for survival analysis in R [Internet]. 2020. Available from: https://CRAN.R-project.org/package=survival
3. Wang W, Yan J. Shape-restricted regression splines with R package splines2. J Data Sci. 2021;19(3):498–517.
4. Wickham H, François R, Henry L, Müller K, Vaughan D. Dplyr: a grammar of data manipulation. R package version 1.1.3. 2023. Available from: https://CRAN.R-project.org/package=dplyr
5. Tang Y, Horikoshi M, Li W. Ggfortify: unified interface to visualize statistical result of popular R packages. The R Journal. 2016;8(2):478–89.
6. Kassambara A, Kosinski M, Biecek P. Survminer: drawing survival curves using 'ggplot2'. R package version 0.4.9. 2021. Available from: https://CRAN.R-project.org/package=survminer

Chapter 14
Conducting Randomization in Clinical Trials

Praveen Kumar-M and Archana Mishra

Randomization in clinical trials is crucial to ensure unbiased and fair allocation of participants into different treatment groups. Through a random process, individuals are assigned to receive either the experimental treatment or the control/placebo, minimizing selection bias and enhancing the validity of study results. This helps researchers achieve comparable baseline characteristics among groups, reducing the impact of confounding variables. Randomization promotes statistical validity, enabling researchers to make reliable inferences about the treatment's effectiveness. This essential component of clinical trial design enhances the reliability of findings, ultimately contributing to evidence-based medicine and improved healthcare decision-making.

The common randomization methodologies practiced in clinical trials are:

(a) Simple randomization: Participants are randomly assigned to different treatment groups without any specific pattern or restriction.
(b) Stratified randomization: Prior to randomization, participants are grouped into strata based on certain characteristics (e.g., age and gender). Randomization is then performed within each stratum, ensuring a balance of key variables across treatment groups.
(c) Block randomization: Participants are grouped into blocks, and within each block, randomization occurs. This helps ensure an equal distribution of participants across treatment groups at various points in the trial.
(d) Adaptive randomization: The randomization process is adjusted during the trial based on interim results. This method allows for modifications in treatment assignments to allocate more participants to the more effective treatment arm.

P. Kumar-M (✉)
Clinical Sciences, Nference, Bengaluru, Karnataka, India

A. Mishra
Department of Pharmacology, All India Institute of Medical Sciences, Bhubaneswar, Odisha, India

A. Srinivasan et al. (eds.), *R for Basic Biostatistics in Medical Research*,
https://doi.org/10.1007/978-981-97-6980-3_14

(e) Cluster randomization: Instead of randomizing individual participants, entire groups or clusters (e.g., hospitals, communities) are assigned to different treatments. This method is often used in public health or community-based interventions.
(f) Minimization: This method takes into account multiple participant characteristics to minimize differences between treatment groups. It aims to ensure balance in key variables, such as age or severity of the disease, during the randomization process.

The choice of randomization method depends on various factors, including the trial design, objectives, and specific considerations related to the study population and intervention. Each technique aims to enhance the reliability and validity of study results by minimizing bias and ensuring a balanced distribution of crucial variables among treatment groups.

Various online and offline tools can generate the list for randomization. This chapter will teach us to perform randomization using "randomizeR" and "blockrand" packages in R [1, 2]. The advantage of generating randomization in R is that we can document the randomization steps and exactly replicate the randomization list using the same version of the package & seed as used previously for the generation of the list. This chapter will teach us the randomization process through practical examples of different case scenarios. With each scenario, the readers will be aware of different types of randomization, their practical usage, and R codes and associated explanations. The randomization that we will be covering includes (1) simple randomization. (2) Permuted block randomization with equal block sizes. (3) Permuted block randomization with random block sizes. (4) Stratified block randomization. (5) Adaptive/Biased randomization techniques. The biased randomization will include Wei's Urn Design Randomization, Efron's Biased Coin Randomization Design, and Accelerated Biased Coin Design.

Scenario 1

A researcher wants to evaluate and compare the efficacy of two drugs, "Q" and "W" in a clinical trial. The researcher wishes to perform simple randomization with an equal allocation ratio. We are tasked to generate a randomization list for the same.

Step 1: Planning Based on the Information in Hand

1. It is a case of simple randomization. No blocks are employed.
2. Total number of treatments/groups in the study: Two: Q and W.
3. Equal allocation is planned between groups Q and W. This means that the allocation ratio is 1:1.

4. The total number of patients in the study is 100.

Step 2: Generate the Sequence

First, we need to install the package "*randomizeR*" [1].

```
library(randomizeR)
## Loading required package: ggplot2
## Loading required package: plotrix
## Loading required package: survival
r1<-rarPar(N = 100,K = 2,ratio = c(1,1),groups = c("Q","W"))
g1<-genSeq(r1)
table(g1@M)
##
##  0  1
## 50 50
```

1. The "*install.packages*" function is used to install the *randomizeR* package. The package installation has already been explained in the chapter on the *Introduction to Packages in R*.
2. The installed library is loaded using the function "*library*".
3. The "*rarPar*" function has been used for simple randomization design. The arguments of "*rarPar*" include *N*, *K*, *ratio*, and *groups*.
   - *N*—integer as input—Total sample size of study
   - *K*—integer as input—Total number of treatment/groups in the study.
   - *ratio*—numeric vector as input-allocation ratio in the study.
   - *groups*—character vector as input—The name of the treatment/groups in the study.
4. The output of the "*rarPar*" function is stored in an R object (`r1`).
5. The "*genSeq*" function is used to generate the randomization sequence. The function takes in the randomization object created using a randomization function as input (in this case, the output of the "*rarPar*" function was stored under the R object named `r1`). The "*genSeq*" function output is stored as an R object *g1*.
6. The "`table(g1@M)`" function creates a contingency table of counts. It takes one or more categorical variables (`g1@M`) as input and returns a table of the counts of observations for each of the categorical variables. The counts of the groups "0" and "1" are displayed in this example.
7. The `@M` symbol retrieves the information stored in the R object *g1* created using the "*genSeq*" function.

Step 3: Assessment of the Output Sequence Generated

```
g1@M
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12]
[,13] [,14]
## [1,]    0    1    1    0    1    0    0    0    0     1     1
0     1     0
##      [,15] [,16] [,17] [,18] [,19] [,20] [,21] [,22] [,23] [,24]
[,25] [,26]
## [1,]     1     0     1     1     1     1     1     1     1
0     0     1
##      [,27] [,28] [,29] [,30] [,31] [,32] [,33] [,34] [,35] [,36]
[,37] [,38]
## [1,]     0     1     1     0     0     0     1     1     1
0     1     1
##      [,39] [,40] [,41] [,42] [,43] [,44] [,45] [,46] [,47] [,48]
[,49] [,50]
## [1,]     1     1     0     1     0     0     1     0     0
1     0     0
##      [,51] [,52] [,53] [,54] [,55] [,56] [,57] [,58] [,59] [,60]
[,61] [,62]
## [1,]     0     0     1     0     1     0     1     1     1
1     1     1
##      [,63] [,64] [,65] [,66] [,67] [,68] [,69] [,70] [,71] [,72]
[,73] [,74]
## [1,]     0     1     1     1     1     0     1     0     1
1     0     1
##      [,75] [,76] [,77] [,78] [,79] [,80] [,81] [,82] [,83] [,84]
[,85] [,86]
## [1,]     1     1     0     0     0     1     0     0     0
0     0     0
##      [,87] [,88] [,89] [,90] [,91] [,92] [,93] [,94] [,95] [,96]
[,97] [,98]
## [1,]     1     1     0     0     1     1     0     0     0
0     0     0
##      [,99] [,100]
## [1,]     0      0
table(g1@M[1:10])
##
## 0 1
## 6 4
table(g1@M[21:30])
##
## 0 1
## 4 6
table(g1@M[90:100])
##
## 0 1
## 9 2
```

1. The "*table*" function is then used to create a contingency table, and square brackets are used to extract a subset of the sequence thus generated.

Step 4: Saving the Sequence into a File

```
saveRand(obj = g1,file = "scenario1.csv")
```

1. The "*saveRand*" function is used to save the randomization sequence to a file, and it has an *obj* argument that takes in the randomization object as value and a *file* argument that takes in character (file name to be stored) as value(`scenario1.csv`).

Practical Information

The output generated contains the given information: the time & date of document generation, the version of the randomizeR package used for the generation, the random number used for the generation, and other pertinent details that we have decided for randomization. This information will help replicate the exact steps of randomization at a later date if required. The investigator can take this sequence and generate opaque envelopes containing the serial number on the outside of the envelope (1, 2, 3,…,100) and a paper with treatment information (Q or W) inside. We are using sealed opaque envelopes to ensure allocation concealment.

```
g1_withseed<-genSeq(r1,seed = 999)
```

Here, with the "*genSeq*" function, we are using a seed argument (taking a numeric input) to ensure reproducibility in random processes. When we set a seed, we initialize the random number generator with a specific value, and if we use the same seed again, we should get the same sequence of random numbers.

We may appreciate the differences in the group allocation in various blocks (1–10, 21–30, 90–100) of the sequence, though overall counts in both groups are the same. Block randomization could address this, which ensures a balanced distribution of participants across treatment groups within each block. This helps reduce the likelihood of unequal group sizes at all time points and ensures that each treatment group has a similar number of participants.

Scenario 2

A pharmaceutical company is conducting a double-blind, randomized, controlled clinical trial to assess the efficacy of a new drug (Drug A) compared to Drug B for treating a specific medical condition in 120 patients. The research team decided to use block randomization to ensure that both treatment groups were balanced at regular intervals during enrollment. They choose block sizes of 8 to provide a reasonable balance between the two groups. We are tasked to prepare a randomization list assuming an equal allocation ratio.

Step 1: Planning Based on the Information in Hand

1. The first step is to decide the size of the block. The investigator plans a block size of 8.
2. Total number of treatments/groups in the study: 2 (A and B).
3. Equal allocation is planned in groups A and B (allocation ratio is 1:1).
4. The total number of participants in the study is 120.

Step 2: Computation of Randomization Object and Generating Sequence

```
r2<-rpbrPar(N = 120,rb = 8,K=2, ratio=c(1,1),groups = c("Treatment Group A",
"Treatment Group B"))
r2
##
## Object of class "rpbrPar"
##
## design = RPBR(8)
## rb = 8
## filledBlock = FALSE
## N = 120
## groups = Treatment Group A Treatment Group B
g2<-genSeq(r2)
```

1. The "*rpbrPar*" function is used to create a randomization object in a permuted block design. The arguments of rpbrPar include *N*, *rb*, *K*, *ratio*, and *groups*.
   - *N*—integer as input—Total sample size of study
   - *K*—integer as input—Total number of treatment/groups in the study.
   - *ratio*—numeric vector as input-allocation ratio in the study.
   - *rb*—numeric vector as input—vector containing the size of the block.
   - *groups*—character vector as input—The name given to treatment/groups in the study.
2. The output of "*rpbrPar*" function is a randomization R object represented by a variable (*r2*).
3. The "*genSeq*" function is then used to generate the randomization sequence. The function takes in a randomization object created using a randomization function as input(r2). The output is stored as a variable g2.

Step 3: Assessment of the Output Sequence Generated

```
g2@M
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12]
[,13] [,14]
## [1,]    1    0    0    1    1    1    0    0    1     0     0
1     0     1
##      [,15] [,16] [,17] [,18] [,19] [,20] [,21] [,22] [,23] [,24]
[,25] [,26]
## [1,]     1     0     1     1     0     1     0     1     0
0     1     0
##      [,27] [,28] [,29] [,30] [,31] [,32] [,33] [,34] [,35] [,36]
[,37] [,38]
## [1,]     0     0     1     0     1     1     0     1     0
0     1     1
##      [,39] [,40] [,41] [,42] [,43] [,44] [,45] [,46] [,47] [,48]
[,49] [,50]
## [1,]     1     0     1     1     0     1     1     0     0
0     1     0
##      [,51] [,52] [,53] [,54] [,55] [,56] [,57] [,58] [,59] [,60]
[,61] [,62]
## [1,]     1     0     1     1     0     0     0     1     1
0     1     0
##      [,63] [,64] [,65] [,66] [,67] [,68] [,69] [,70] [,71] [,72]
[,73] [,74]
## [1,]     0     1     1     1     0     0     0     1     0
1     1     1
##      [,75] [,76] [,77] [,78] [,79] [,80] [,81] [,82] [,83] [,84]
[,85] [,86]
## [1,]     1     0     0     0     1     0     1     1     0
0     0     1
##      [,87] [,88] [,89] [,90] [,91] [,92] [,93] [,94] [,95] [,96]
[,97] [,98]
## [1,]     1     0     0     0     1     0     1     1     1
0     1     0
##      [,99] [,100] [,101] [,102] [,103] [,104] [,105] [,106]
[,107] [,108]
## [1,]     1      1      0      1      0      0      1      0
0      1
##      [,109] [,110] [,111] [,112] [,113] [,114] [,115] [,116]
[,117] [,118]
## [1,]      1      1      0      0      0      1      1      0
1      1
##      [,119] [,120]
## [1,]      0      0
table(g2@M[1:12])
##
## 0 1
## 6 6
table(g2@M[25:36])
##
## 0 1
## 7 5
table(g2@M[109:120])
##
## 0 1
## 6 6
```

Assessment of the generated sequence can be performed by using the "`table`" function. As discussed earlier, the square bracket is used to subset. We can appreciate equal allocation in various blocks in block randomization.

Step 4: Saving the Sequence into a File

```
saveRand(obj = g2,file = "scenario2.csv")
```

Scenario 3

A pharmaceutical company is conducting a clinical trial to evaluate the effectiveness of a new treatment in 240 participants for a specific medical condition. The trial aims to compare three different treatment regimens: a standard treatment (Group K), a placebo (Group L), and an experimental drug (Group M). To ensure a balanced distribution of participants across the three groups, the researchers use random block randomization with a 1:1:2 allocation ratio.

The advantage of using unequal-sized (random) permuted block randomization is that the intervention to which the last subject will be allocated in a block cannot be predicted as the blocks are unequal in size.

Step 1: Planning Based on the Information in Hand

1. The first step is to decide the size of the block. Say the length of a block will be 4 or 8.
2. Total number of treatments/groups in the study: 3 (K, L, and M).
3. Unequal allocation is planned. The allocation ratio is 1:1:2.
4. The total number of patients in the study is 240.

Step 2: Computation of Randomization R Object and Generating Sequence

```
library(randomizeR)
r3<-rpbrPar(N = 240,rb = c(4,8), K=3, ratio=c(1,1,2), groups = c("Treatment
group K", "Treatment group L", "Treatment group M"))
r3
##
## Object of class "rpbrPar"
##
## design = RPBR(4,8)
## rb = 4 8
## filledBlock = FALSE
## N = 240
## K = 3
## ratio = 1 1 2
## groups = Treatment group K  Treatment group L  Treatment group M
g3<-genSeq(r3)
```

1. The "*rpbrPar*" function is used to create a randomization object in a permuted block design. The function has been explained under `Scenario 2`. The only difference with scenario 2 is an input of two block sizes, as we intend to perform a random block randomization.
2. The output of the "*rpbrPar*" function is a randomization R object, and it is stored in a variable (`r3`).
3. The "*genSeq*" function is then used to generate the randomization sequence. The output is stored as a variable `g3`.

Step 3: Assessment of Output Sequence Generated

```
table(g3@M)
##
##   0   1   2
##  60  60 120
barplot(table(g3@M),col = rainbow(1),xlab = "Groupings of Study",ylab =
"Frequency",names.arg = c("Treatment \n Group K"," Treatment \n Group L","
Treatment \n Group M"))

table(g3@M[133:140])
##
## 0 1 2
## 2 2 4
#plotSeq(sequences = g3)
```

The "*table*" and "*barplot*" functions can be used to assess the number of randomized participants in each group (Fig. 14.1).

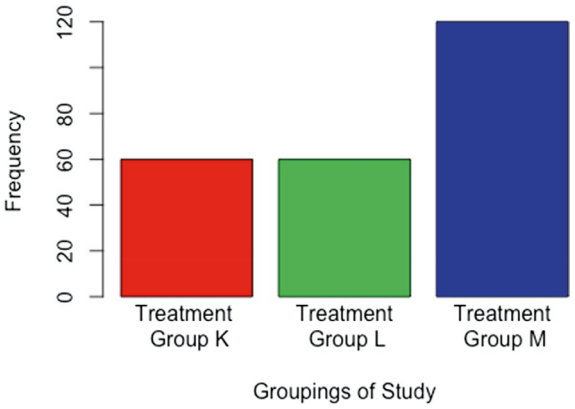

Fig. 14.1 Number of participants randomized in different treatment groups

Step 4: Saving the Sequence into a File

```
saveRand(obj = g3,file = "scenario3.csv")
```

Scenario 4

A pharmaceutical company is conducting a clinical trial to evaluate the efficacy and safety of a new medication in 700 patients with a specific medical condition in a two-arm trial (groups X and Y). Given that the patient's age (<60 years and ≥60 years) and the presence of diabetes are essential covariates influencing the response to treatment, the researchers decided to implement stratified randomization to ensure a balanced distribution of participants across treatment groups. It is observed that 100 patients are <60 years old and have diabetes, 200 patients are <60 years old and do not have diabetes, 300 patients are >60 years old and have diabetes, and 100 patients are >60 years old and do not have diabetes. The block sizes that the investigator decided to use were 2 or 4. The allocation ratio is set at 1:1. We are tasked to generate a randomization sequence for the same.

Step 1: Planning Based on the Information in Hand

1. The first step is to decide the size of the block. The block sizes are 2 and 4.
2. Total number of treatments/groups in the study: 2: X and Y.
3. Equal allocation is planned in groups X and Y. This means a 1:1 allocation ratio.
4. The total number of patients in the study is 700.

5. However, the vital information to consider is that age and diabetes are stratifying factors. The stratifying factors have to be equally distributed in the randomization sequence. So we will perform the permuted block randomization separately for 100 patients who are <60 years old and who have diabetes, 200 patients who are <60 years old and do not have diabetes, 300 patients who are ≥60 years old and have diabetes, and 100 patients who are ≥60 years and not having diabetes. After generating the sequence separately for each stratum, we will combine to generate the final sequence. So overall, we get 350 patients in each group with ages and diabetes equally distributed.

Step 2: Computation of Randomization R Object and Generating Sequence

```
Method 1
##100 patients who are <60 years old and who have diabetes
r4a<-rpbrPar(N = 100,rb = c(2,4),K=2, ratio=c(1,1),groups = c("Treatment
Group X","Treatment Group Y"))
r4a
##
## Object of class "rpbrPar"
##
## design = RPBR(2,4)
## rb = 2 4
## filledBlock = FALSE
## N = 100
## groups = Treatment Group X Treatment Group Y
g4a<-genSeq(r4a)
##200 patients who are <60 years old and do not have diabetes
r4b<-rpbrPar(N = 200,rb = c(2,4),K=2, ratio=c(1,1),groups = c("Treatment
Group X","Treatment Group Y"))
r4b
##
## Object of class "rpbrPar"
##
## design = RPBR(2,4)
## rb = 2 4
## filledBlock = FALSE
## N = 200
## groups = Treatment Group X Treatment Group Y
g4b<-genSeq(r4b)
##300 patients who are >=60 years old and have diabetes
r4c<-rpbrPar(N = 300,rb = c(2,4),K=2, ratio=c(1,1),groups = c("Treatment
Group X","Treatment Group Y"))
r4c
##
## Object of class "rpbrPar"
##
## design = RPBR(2,4)
## rb = 2 4
## filledBlock = FALSE
## N = 300
```

```
## groups = Treatment Group X Treatment Group Y
g4c<-genSeq(r4c)
##100 patients who are >=60 years and not having diabetes
r4d<-rpbrPar(N = 100,rb = c(2,4),K=2, ratio=c(1,1),groups = c("Treatment
Group X","Treatment Group Y"))
r4d
##
## Object of class "rpbrPar"
##
## design = RPBR(2,4)
## rb = 2 4
## filledBlock = FALSE
## N = 100
## groups = Treatment Group X Treatment Group Y
g4d<-genSeq(r4d)
```

The problem with using the *randomizeR* package for the sequence generation of stratified permuted block randomization is that the IDs representing the stratum are not mentioned along with the generated list. The information regarding the stratum would readily help identify the patient.

The sequence created using the *blockrand* package contains information about the stratum within the generated list. Here, we will discuss the codes with the *blockrand* package. We recommend using the *blockrand* package for stratified permuted block randomization.

```
Method 2
#install.packages("blockrand")
library(blockrand)
AgeL60.DiaYes <- blockrand(n = 100,block.sizes = c(2,4), id.prefix = 'AgeL60.
DiaYes',
                           block.prefix = 'AgeL60.DiaYes', stratum = 'AgeL60.
DiaYes')

AgeL60.DiaNo <- blockrand(n = 200,block.sizes = c(2,4), id.prefix = ' AgeL60.
DiaNo',
                          block.prefix = ' AgeL60.DiaNo', stratum = 'AgeL60.
DiaNo')

AgeGE60.DiaYes <- blockrand(n = 300,block.sizes = c(2,4), id.prefix =
'AgeGE60.DiaYes',
                            block.prefix = 'AgeGE60.DiaYes', stratum =
'AgeGE60.DiaYes ')

AgeGE60.DiaNo <- blockrand(n = 100,block.sizes = c(2,4), id.prefix = 'AgeGE60.
DiaNo',
                           block.prefix = 'AgeGE60.DiaNo', stratum = 'AgeGE60.
DiaNo')
FinalList<-rbind(AgeL60.DiaYes,AgeL60.DiaNo,AgeGE60.DiaYes,AgeGE60.DiaNo)
table(FinalList$treatment)
##
##   A   B
## 350 350
```

1. The "*blockrand*" function is used for randomized permuted block design. The arguments for "*blockrand*" include *n*, *block.sizes*, *id.prefix*, *block.prefix*, *stratum*.
   - *n*—integer as input for the total sample size of the study.
   - *K* is the integer that is inputted for the total number of groups in the study.
   - *block.sizes*—numeric vector as input for a vector containing block size.
   - *id.prefix*, *block.prefix and stratum*—character vector as input. These are the identification details, which will be prefixed with the ID of the patient, block, and stratum.
2. The "*blockrand*" function's output is a dataframe; in this case, it is stored in an R object with the name corresponding to the stratum to which the randomization belongs.
3. We then use the "*rbind*" function to combine all the sequences generated for different strata.
4. We use the "*table*" function to confirm that the number of patients is equal among the two groups. We required 350 patients per treatment group, with a total of 700 patients. However, the generated list may, at times, contain more patients. This is because the block sizes are randomly selected. There is an overshooting of the list sequence beyond requirement owing to the last set box. This would not be a problem as we can safely omit the additional patients (as we have the serial number appended to the ID sequence)

Step 3: Assessment of the Output Sequence Generated

```
#Assessment of generated sequence of `randomizeR`
barplot(table(g4a@M),col = rainbow(2),xlab = "Groupings of Study",ylab =
"Frequency",names.arg = c("Treatment \n Group X"," Treatment \n Group
Y"),ylim = c(0,200))
#Assessment of the codes of `blockrand`
barplot(table(AgeL60.DiaYes$treatment),col = c("pink","green"),xlab =
"Groupings of Study",ylab = "Frequency",names.arg = c("Treatment \n Group
X"," Treatment \n Group Y"),ylim = c(0,200),main = "Age <60 years and
Diabetes Present")
barplot(table(AgeL60.DiaNo$treatment),col = c("pink","green"),xlab =
"Groupings of Study",ylab = "Frequency",names.arg = c("Treatment \n Group
X"," Treatment \n Group Y"),ylim = c(0,200),main = "Age <60 years and
Diabetes Absent")
barplot(table(AgeGE60.DiaYes$treatment),col = c("pink","green"),xlab =
"Groupings of Study",ylab = "Frequency",names.arg = c("Treatment \n Group
X"," Treatment \n Group Y"),ylim = c(0,200),main = "Age >=60 years and
Diabetes Present")
barplot(table(AgeGE60.DiaNo$treatment),col = c("pink","green"),xlab =
"Groupings of Study",ylab = "Frequency",names.arg = c("Treatment \n Group
X"," Treatment \n Group Y"),ylim = c(0,200),main = "Age >=60 years and
Diabetes Absent")
barplot(table(FinalList$treatment),col = c("pink","green"),xlab =
"Groupings of Study",ylab = "Frequency",names.arg = c("Treatment \n Group
X"," Treatment \n Group Y"),ylim = c(0,500), main = "Overall")
```

The *"barplot"* function was used to assess the number of participants in different sub-groups. We have demonstrated only one sequence as an example in the case of `randomizeR` (Fig. 14.2). Readers are encouraged to try other sequences. In the case of `blockrand`, we have shown all the sequences.

Note that the figures were combined using PowerPoint® software after generating individual plots using the above code (Fig. 14.3). The users can also combine figures programmatically using R package such as *ggpubr*.

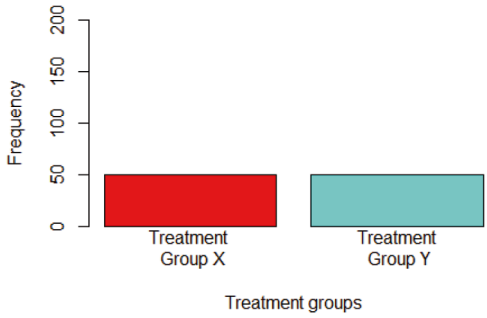

Fig. 14.2 Number of patients randomized (randomizeR package) in age < 60 years and diabetes present strata

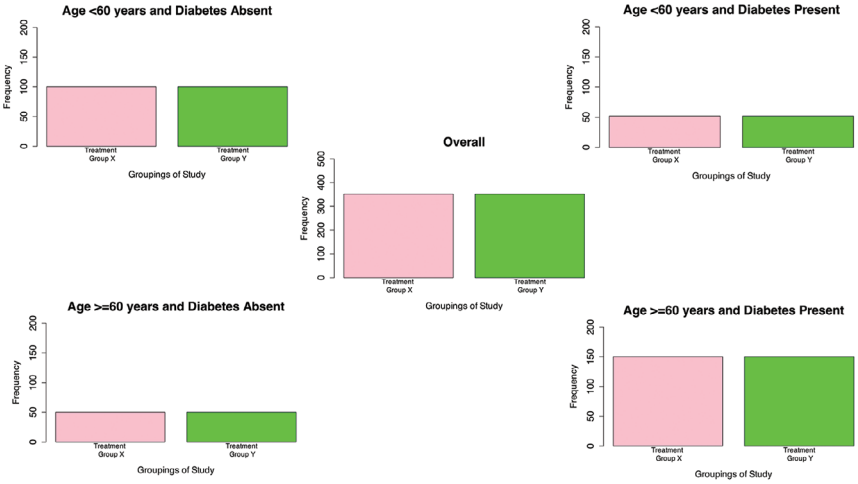

Fig. 14.3 Barplots for the assessment of the number of participants in different subgroups (blockrand package)

Step 4: Saving the Sequence into a File

```
#Saving of the randomization sequence in the case of `randomizeR` usage
saveRand(obj = g4a,file = "g4a.csv")
saveRand(obj = g4b,file = "g4b.csv")
saveRand(obj = g4c,file = "g4c.csv")
saveRand(obj = g4d,file = "g4d.csv")

#Saving of the randomization sequence in the case of `blockrand` usage
write.csv(FinalList,file = "FinalList_Startified.csv")
```

In the case of `randomizeR`, we need to add the stratum information after the generation of the CSV file. Meanwhile, the CSV file generated by `blockrand` already has information about the stratum.

Scenario 5

Biased randomization, or adaptive, may be intentionally used in certain research studies or clinical trials for specific reasons. While the term "biased" might suggest favoritism, it refers to a deliberate departure from complete randomness for particular purposes in the context of biased randomization. Biased randomization is sometimes used in adaptive clinical trial designs where the allocation of participants to treatment groups is adjusted based on interim analyses or accumulating data. This adaptive approach allows researchers to modify the randomization probabilities based on emerging results during the trial. They are typically used when a strong scientific rationale exists for intentionally favoring certain treatment allocations.

The readers are advised to read appropriate literature that discusses the scenario of usage.

Wei's Urn Design Randomization

Wei's urn design is a type of adaptive randomization technique used in clinical trials. Wei's urn design involves a metaphorical "urn" from which treatment allocations are drawn based on the current distribution of participants in the trial. A short explanation of the design is as follows:

- *Initialization*: Initially, the urn contains a specified number of balls representing the treatment arms (e.g., two colors for a 1:1 allocation ratio).
- *Treatment Assignment*: For each new participant enrolled in the trial, a ball is randomly selected from the urn. The color of the selected ball determines the treatment assignment for that participant.

- *Update Urn*: After each assignment, the selected ball is replaced along with additional balls of the same color. The number of balls added depends on the predefined allocation ratios and the current distribution of participants in each treatment group.
- *Adaptation*: Wei's urn design adapts to the changing distribution of participants over time. The current allocation ratio influences the probability of a participant being assigned to a particular treatment.

```
r5<-udPar(100, ini = 1, add = 1, groups = c("Group 1","Group 2"))
g5<-genSeq(r5)
g5@M
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12]
[,13] [,14]
## [1,]    1    0    1    1    0    1    0    0    0     0     1
1     0     1
##      [,15] [,16] [,17] [,18] [,19] [,20] [,21] [,22] [,23] [,24]
[,25] [,26]
## [1,]     0     0     0     1     1     0     0     1     0
1     0     1
##      [,27] [,28] [,29] [,30] [,31] [,32] [,33] [,34] [,35] [,36]
[,37] [,38]
## [1,]     1     1     0     1     0     0     1     1     0
0     0     0
##      [,39] [,40] [,41] [,42] [,43] [,44] [,45] [,46] [,47] [,48]
[,49] [,50]
## [1,]     1     1     0     1     0     0     0     0     0
1     1     0
##      [,51] [,52] [,53] [,54] [,55] [,56] [,57] [,58] [,59] [,60]
[,61] [,62]
## [1,]     1     1     1     0     0     1     0     0     1
1     1     0
##      [,63] [,64] [,65] [,66] [,67] [,68] [,69] [,70] [,71] [,72]
[,73] [,74]
## [1,]     0     0     1     0     1     0     0     1     1
0     1     0
##      [,75] [,76] [,77] [,78] [,79] [,80] [,81] [,82] [,83] [,84]
[,85] [,86]
## [1,]     1     1     0     1     1     1     0     1     0
1     0     0
##      [,87] [,88] [,89] [,90] [,91] [,92] [,93] [,94] [,95] [,96]
[,97] [,98]
## [1,]     1     1     0     1     0     0     1     1     1
0     1     0
##      [,99] [,100]
## [1,]     0      1
```

The "`udPar`" function is used for Wei's Urn Design of Randomization. The arguments include

(a) `ini`—takes in an integer for the initial urn composition.
(b) `add`—takes in an integer for the number of balls that are added to the urn in each step.
(c) `groups`—takes in a character vector for the labels of different treatments/ groups of study.

Other Biased Design

Efron's Biased Coin Randomization Design

Efron's Biased Coin Design, also known as the "minimization by Efron's biased coin design," is a biased randomization technique used in clinical trials. This method is a modification of the biased coin design, which minimizes imbalance between treatment arms.

In traditional biased coin randomization, the probability of assigning a participant to a particular treatment is influenced by the current imbalance in the treatment groups. Efron's modification introduces an additional feature called "probability matching," which aims to balance the current allocation and the expected allocation over the entire trial.

A simplified explanation of Efron's Biased Coin Design is as follows:

- *Initialization*: Begin with equal probabilities for each treatment arm.
- *Treatment Assignment*: For each new participant, calculate the expected imbalance based on the current allocation and assign the participant to the treatment arm that minimizes this expected imbalance.
- *Update Probabilities*: Adjust the probabilities for the next assignment based on the observed and expected imbalances.
- *Probability Matching*: Incorporate a probability matching component to account for the expected imbalance over the entire trial. Probability matching helps achieve balance at the current point in the trial and over the long term.

```
r6<-ebcPar(N = 99, p = 0.8,groups=LETTERS[1:2])
g6<-genSeq(r6)
g6
##
## Object of class "rEbcSeq"
##
## design = EBC(0.8)
## seed = 436332066
## N = 99
## groups = A B
## p = 0.8
##
## The sequence M:
##
## 1 B   A   A   B   A   B   A   B   A   A   ...
table(g6@M)
##
##  0  1
## 50 49
```

1. The "*ebcPar*" function is used to generate a randomization list for Efron's biased coin design. The arguments for the function are:
   N: integer for the total sample size of the trial.
   p: success probability of the biased coin (e.g. in Efron's Biased Coin Design).
   groups: a character vector of labels for the different treatments.
2. If both the treatments have equal probability, then a fair coin (i.e., p = 0.5) is tossed, else the probability is adjusted.

Accelerated Biased Coin Design

The Accelerated Biased Coin Design is an adaptive randomization technique used in clinical trials. It is an extension of the traditional biased coin design and is designed to minimize imbalance between treatment groups more rapidly than traditional methods. This design is particularly useful when there is a need to quickly balance covariates across treatment arms. The following things need to be considered when implementing randomization in such a design:

- *Initialization*: Begin with equal probabilities for each treatment arm.
- *Treatment Assignment*: For each new participant, calculate the expected imbalance based on the current allocation and assign the participant to the treatment arm that minimizes this expected imbalance.

- *Update Probabilities*: Adjust the probabilities for the next assignment based on the observed and expected imbalances. Unlike the traditional biased coin design, the probabilities are updated more aggressively to speed up the balancing process.
- *Adaptive Adjustments*: Incorporate adaptive adjustments to control the rate of acceleration in updating probabilities.

```
r7<-abcdPar(N = 100, a=0, groups = LETTERS[1:2])
g7<-genSeq(r7)
table(g7@M)
##
##  0  1
## 52 48
```

1. The "`abcdPar`" function is used for accelerated biased design randomization. The arguments of the function are:

   N—integer for the total sample size of the trial.

   a—nonnegative parameter which controls the degree of randomness: A smaller value for 'a' might imply that the randomization is less influenced by factors that could introduce variability. In extreme cases, it might lead to a more deterministic allocation where the treatment assignment becomes more predictable or systematic. Conversely, a larger value for 'a' could suggest increased influence or weighting towards certain factors in the randomization process. This could result in a more complete randomization where treatment assignments are less predictable and follow a pattern that approaches true randomness.

   groups—character vector of labels for the different treatments.

References

1. Uschner D, Schindler D, Hilgers R, Heussen N. randomizeR: an R package for the assessment and implementation of randomization in clinical trials. J Stat Softw. 2018;85(8):1–22. https://doi.org/10.18637/jss.v085.i08.
2. Snow G. blockrand: Randomization for block random clinical trials. R package version 1.5. 2020. https://CRAN.R-project.org/package=blockrand

Chapter 15
Development of Web-Based Interactive Servers Using R Shiny Package

Praveen Kumar-M and Archana Mishra

Introduction

Shiny package in R offers ready-to-use tools for deploying web-based tools [1]. Using the Shiny package, we can create user interface (UI) programs without requiring knowledge of front-end languages such as HTML, JavaScript, and PHP. The creation of ready-to-use tools for developed models helps maximize the translational capabilities of the research conducted, assists in its validation, and will also help avoid repetitive calculations. The developed Shiny web application can run on a local or remote server. Therefore, flexibility is present in its usage. This chapter will explain the core format of Shiny, the tools available, and develop an example program.

The Shiny web application has two files.

1. `ui.R`
2. `server.R`

ui.R

The file '`ui.R`' is used to create the front-end user interface (UI). The '`ui.R`' enables coding both input and output interfaces. Inputs are in the form of buttons, toolbars, and text boxes. There are provisions to apply various validations and

P. Kumar-M (✉)
Clinical Sciences, Nference, Bengaluru, Karnataka, India

A. Mishra
Department of Pharmacology, All India Institute of Medical Sciences, Bhubaneswar, Odisha, India

A. Srinivasan et al. (eds.), *R for Basic Biostatistics in Medical Research*,
https://doi.org/10.1007/978-981-97-6980-3_15

limits, thereby restricting the entries made through these interfaces. Output can be in the form of text, numbers, or graphs. The '`ui.R`' contains many HTML-related commands for modifying how the page should appear. The information consists of page layout, tabs, borders, font, and size of characters. A number of input widgets available help in the swift implementation of the web user interface for a web application.

server.R

The '`server.R`' file is used to create a back-end program to compute the information obtained through the input of '`ui.R`' and then give the output of the computed values to the output of '`ui.R`'.

We will be learning R shiny by constructing an example project.

Calculation of Creatinine Clearance (CrCl) Using the Cockcroft-Gault Formula

The Cockcroft-Gault formula was created to estimate creatinine clearance (CrCl) [2]. The formula is widely famous for estimating CrCl, and online tools are available for the same. The purpose of this exercise is to develop one such tool using the Shiny package of R. At the end of the exercise, readers will be mindful of the core concepts of Shiny.

$$\mathrm{CrCl} = \left[\left((140 - \mathrm{age}) \times \mathrm{weight}\right) / \left(72 \times \mathrm{serum\ creatinine}\right)\right] \times 0.85\left(\mathrm{if\ female}\right)$$

where CrCl is in mL/min, age is in years, weight in kg, and S. Creatinine in mg/dL.

Planning

The foremost step for creating a successful web application is to make a rough sketch of how the web application will look. This rough sketch will be the best guide for further steps in coding. Creating rough sketches will also help save valuable time spent on re-coding once the first draft is made. Also, we can document our progress once the rough sketch is made.

1. Since gender can take only two male or female values, we prefer to keep this option as a radio button. The radio button would enable us to pick only one option at a time. Simultaneously, the checker box is used in cases where more than one option needs to be selected.
2. The age and serum creatinine can also take only a specific set of values; hence, we resort to a slider with defined minimum and maximum limits. We should also have to explain the steps for the increase, which would be essential for determining the value that would increase by moving a single step in the slider.
3. For weight, we are keeping an input text box. To avoid the entry of wrong values, we incorporate validation in the input text box, such as double values alone, and limit the value to 25–150 kg alone. The entry of values should be in the kg units, which need to be written in either the helper box or the assistant with the text box. The user must follow the instructions given, and it cannot be validated if wrongly entered.
4. We can also assess the output values obtained after calculation based on the formula's inputs. If the output seems invalid, then warning messages are shown to the user.
5. We can also go ahead and interpret the CrCl value obtained.

Generation of Base Files

The base files for creating a shiny web app can be generated by selecting "New File" and then "Shiny Web APP" (Fig. 15.1). It can also be generated using the "File" option on the top left.

Following this, we need to name the application (Fig. 15.2). Here, we have the option to create either an application with a single file or multiple files. If we create a single file, the file generated will have the name '`app.R`'. If we select multiple files, we will have two files generated ('`ui.R`' and '`server.R`'). In either case, the fundamental coding pattern will remain the same. The '`app.R`' file will contain the components of both '`ui.R`' and '`server.R`'. For our exercise, let us select the multiple files option.

The generated files will have some pre-written codes and information (Fig. 15.3). We can delete those.

The codes of 'ui.R' and 'server.R' are mentioned below.

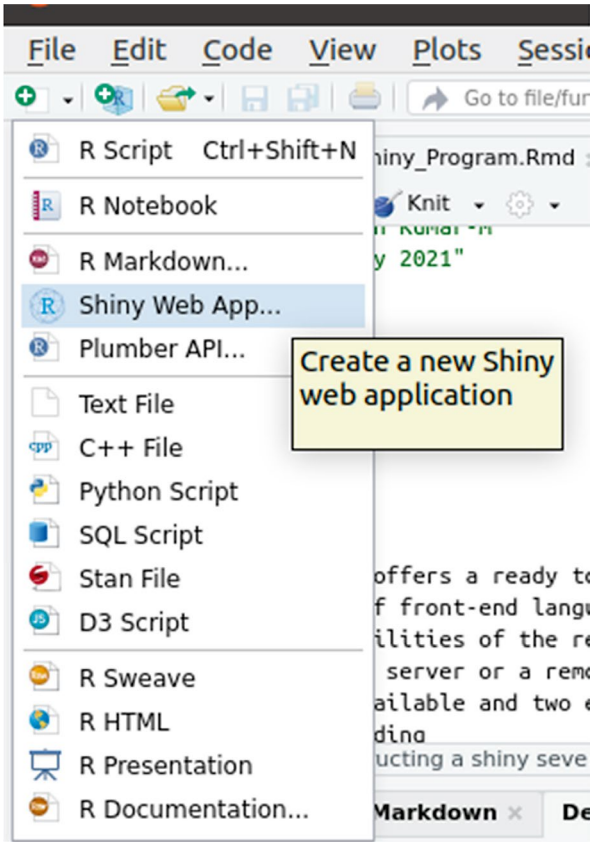

Fig. 15.1 Generation of a new Shiny web app file

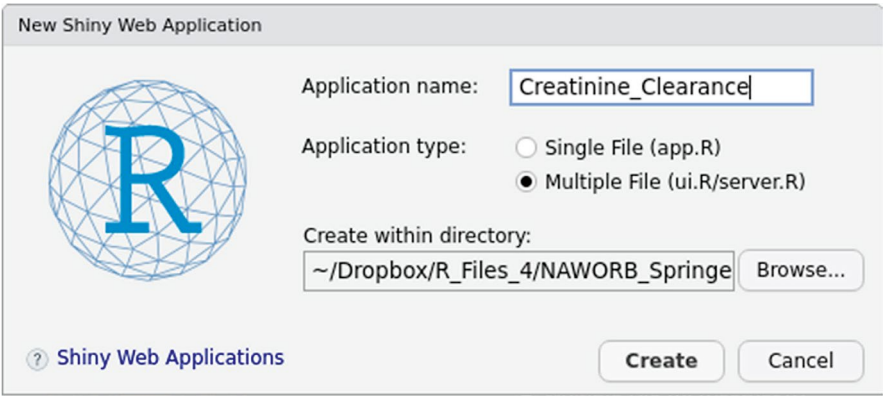

Fig. 15.2 Naming of the shiny web app and selection of 'Multiple File' option for new file generation

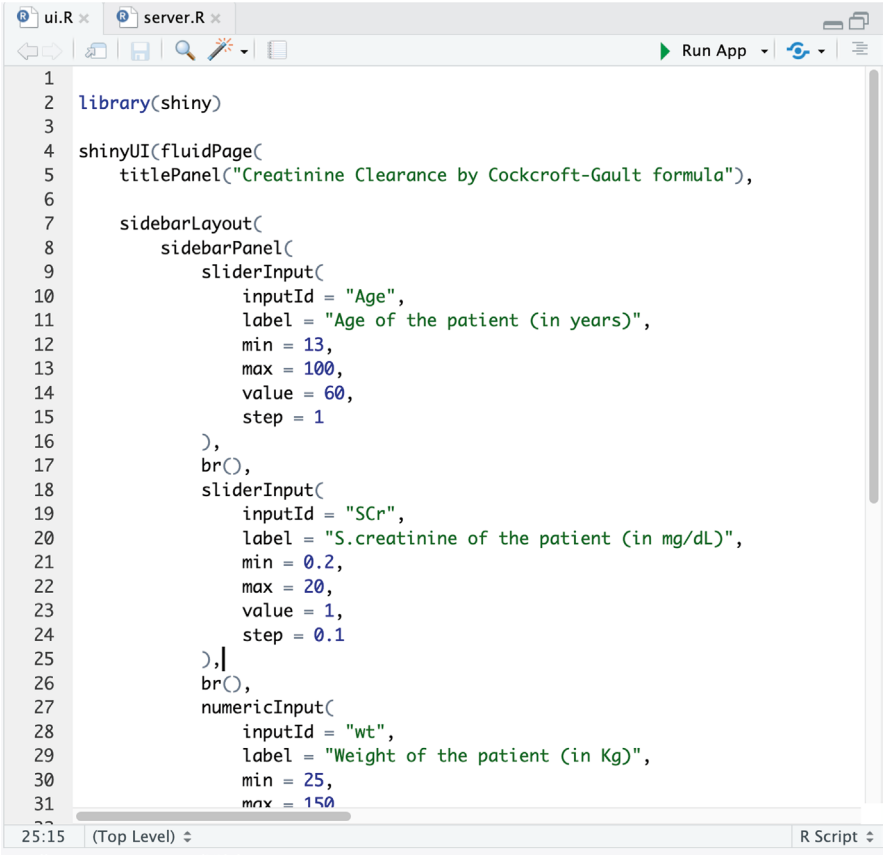

Fig. 15.3 Generation of 'ui.R' and 'server.R'

ui.R File

```
library(shiny)

shinyUI(fluidPage(
    titlePanel("Creatinine Clearance by Cockcroft-Gault formula"),

    sidebarLayout(
        sidebarPanel(
            sliderInput(
                inputId = "Age",
                label = "Age of the patient (in years)",
                min = 13,
                max = 100,
                value = 60,
                step = 1            ),
            br(),
            sliderInput(
                inputId = "SCr",
                label = "S.creatinine of the patient (in mg/dL)",
                min = 0.2,
                max = 20,
                value = 1,
                step = 0.1
            ),
            br(),
            numericInput(
                inputId = "wt",
                label = "Weight of the patient (in Kg)",
                min = 25,
                max = 150,
                value = 60,
                step = NA
            ),
            radioButtons(inputId = "gender",
                         label = "Gender",
                         choices = c("Male" = "male",
                           "Female" = "female"))
            ),

        mainPanel(hr(),
                  h3(textOutput("crcl")),
                  hr(),
                  h3(textOutput(
                      "crclinterpretation"
                  )),
                  hr())
        )
))
```

1. The subsequent codes must be separated by commas in shiny codes. And the outer function encompasses the inner function. So, we should be careful to match parenthesis. We start by importing the shiny package using the `library(shiny)` function.
2. The outermost function is known as *"ShinyUI"*.
3. Within this function, we have another function named *"fluidPage"*, which means a web application has been generated in the UI.
4. Now, we are giving a name to the web application using the function *"titlePanel"*.
5. Now is the time to decide the web application's layout. There are many types of layouts available, and a detailed list is present in Shiny's documentation. Here, we are using a `sidebar layout`. So, we are now adding the *"sidebarLayout"* function. Inside this layout, there is going to be one `sidebar panel` and one `main panel`.
6. So, we use two functions, namely *"sidebarPanel"* and *"mainPanel"*. Inside the *"sidebarPanel"*, we will include all the widgets of the side panel, and inside the *"mainPanel"*, we will include all the outputs that have been computed using the '`server.R`' code.
7. We will be including the *"sliderInput"* function for age and creatinine clearance, *"numericInput"* function for the weight of the patient, *"radioButtons"* for selecting gender.
8. The *"sliderInput"* and *"numericInput"* take the same set of arguments: *inputID, label, min, max, value,* and *step*. The difference between *inputID* and *label* is that *inputID* is used to refer to a widget in the '`server.R`' file, whereas a *label* is used to refer to the widget in the web application (that is, in the UI). The *min* and *max* refer to the minimum and maximum values allowed and are a form of validation. The *value* refers to the default value to be present in the widget. The *step* refers to the increment permitted in the widget.
9. The arguments of *"radioButtons"* are *inputID*, *label*, and *choices*. The *choices* take in a character vector as an argument with both "label of choices" and "value" mentioned simultaneously.
10. The *"mainPanel"* function contains two important functions, *"textOutput"* for the creatinine clearance value and another *"textOutput"* function for the interpretation. We have encased the functions with *"h3"* function to make the font look big.

server.R File

```
library(shiny)


shinyServer(function(input, output) {
  output$crcl <-
    renderPrint({
      Age <-
        input$Age
      Creatinine <- input$SCr
      Weight <- input$wt

      if (input$gender == "male") {
        crcl <- ((140 - Age) * Weight) / (72 * Creatinine)
      } else if (input$gender == "female")
      {
        crcl <- (((140 - Age) * Weight) / (72 * Creatinine)) * 0.85
      }

      crclfinal <-
        noquote(
          paste(
            "The CrCl of the patient, as calculated by Cockcroft-Gault
formula, is ",
            round(crcl, digits = 1),
            " mL/min"
          )
        )
      print(crclfinal)
    })

  output$crclinterpretation <-
    renderPrint({
      Age <-
        input$Age
      Creatinine <- input$SCr
      Weight <- input$wt

      if (input$gender == "male") {
        crcl <- ((140 - Age) * Weight) / (72 * Creatinine)
      } else if (input$gender == "female")
      {
        crcl <- (((140 - Age) * Weight) / (72 * Creatinine)) * 0.85
      }

      if (crcl > 90 & crcl < 200) {
        interpretation <- "kidney is normal for the patient"
      }
      if (crcl <= 89 & crcl >= 60) {
        interpretation <- "there is a mild reduction in GFR "
      } else if (crcl <= 59 & crcl >= 45) {
        interpretation <- "there is a moderate reduction in GFR"
```

```
      } else if (crcl <= 44 & crcl >= 30) {
        interpretation <- "there is a moderate reduction in GFR "
      } else if (crcl <= 29 & crcl >= 15) {
        interpretation <- "there is severe reduction in GFR "
      } else if (crcl < 15) {
        interpretation <- "there is kidney failure"
      } else {
        interpretation <- "interpretation is not available"
      }

      interpretationfinal <-
        noquote(paste("The interpretation based on report is that ",
interpretation))
      print(interpretationfinal)
    })
})
```

1. `shinyServer(function(input, output) { }` is the outermost function key for the 'server.R' document.
2. We can include the output functions in this. Here we have two output functions `output$crcl <-renderPrint({ })` and `output$crclinterpretation <-renderPrint({ })`. Notice the curly brackets that were present in both the previous functions.
3. We have used the *input$Age*, *input$SCr*, *input$wt* to extract the age, serum creatinine, and weight as entered in the input widget. *The value after '$'* depends on the *inputID* values which we have assigned for the widgets in the *'ui.R'*. After getting these values, we are storing the obtained values in a variable name. In this case, it is *Age, Creatinine,* and *Weight*.
4. Now is the time to compute the creatinine clearance value. We know that the males and females have mild variations in the creatinine clearance value. The *input$gender* can obtain the information on whether the patient is male or female. We are using this along with *if {} else if{}* to create an algorithm.
5. We input the formula per se. **crcl <- ((140 − Age) * Weight)/(72 * Creatinine)** in case of "male" and **crcl <- (((140 − Age) * Weight)/(72 * Creatinine)) * 0.85** in case of "female". Following this, we store the output as *crcl.*
6. To make the output more meaningful, we use the *"paste"* and *"noquote"* functions to remove quotes and store the obtained value to *crclfinal*.
7. We can then use the *"print"* function to print the value of *crclfinal*.
8. The reason for creating two output functions is to have an independent formattable value during output.
9. The *output$crclinterpretation* is similar to *output$crcl*. Here we are trying to interpret the values of *crcl* and making different interpretations ready utilizing the *if {} else if{}* algorithm.

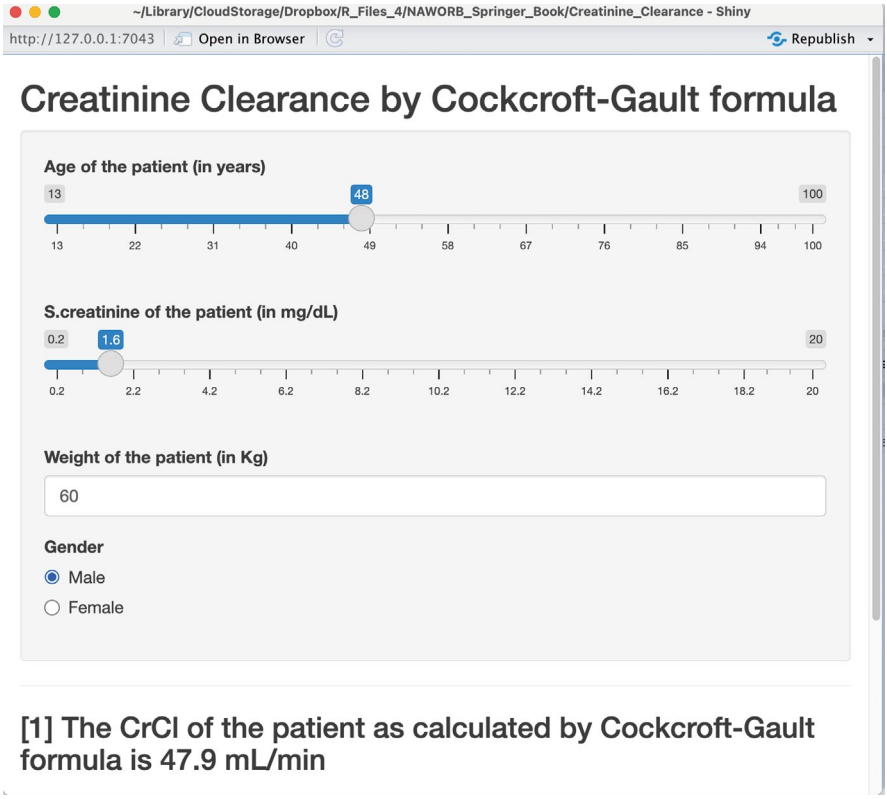

Fig. 15.4 Local rendering of shiny web app

10. To make the output more meaningful, we use the "`paste`" function along with "`noquote`" to remove quotes and then store the obtained value to *interpretationfinal*.
11. We use the "`print`" function to print the value of *interpretationfinal*.
12. If you click "Open in browser" then the web application opens in the web browser. (Fig. 15.4)

Publishing the Shiny Application to the Web Server

The shiny package offers tools for creating and deploying the developed program. There are two deployment formats: deployment at the local server or the remote server with a preinstalled R, and the developed web application runs on the server. The description of this methodology is beyond the scope of this book. Another standard way of deployment is to launch the developed program in the online portal of Shiny, *shinyapps.io*. The *shinyapps.io* offers both free and premium services.

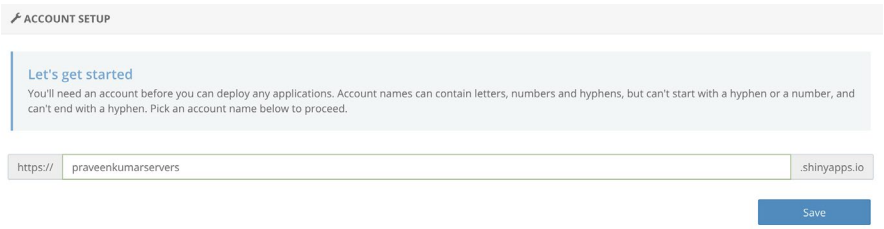

Fig. 15.5 Naming the account

Creation of Account in shinyapps.io

The foremost step for web publishing of the developed shiny application is registering and signing up for an account on the https://www.shinyapps.io/ website. (Fig. 15.5) Following this, we will be directed to a page where we will name our account. Remember that this name will be associated with all our uploads.

The on-screen instructions will ask us to do three essential steps. First is the installation of the `rsconnect` package.

```
# install.packages('rsconnect')
```

Next is the authorization of the account. There will be a code in the function "`rsconnect::setAccountInfo`". We need to copy the code from the website and paste it into the RStudio console. We can also use the *Copy to Clipboard* option on the right side. There will be three arguments required to customize the code, namely `name`, `token`, and `secret`. The `token` and `secret` are essential information and should not be shared. It is akin to the password we use for our web account. These credentials let the web server link our Rstudio to the appropriate account in the web server.

RStudio Instructions

To publish a web application, select the blue circle with a dot on the top right corner of the source window (Fig. 15.6). This shows a tab with the option "Publish Application", and it should be selected. If we have multiple accounts for web publishing, we can use the "manage accounts" option to select the required account.

The deployment of the shiny web application is effortless through the GUI interface. Here, we will select the files to be included in the final web application, the account from which the publication needs to be undertaken, and the web application's name. Following this, the "publish" button is selected. (Fig. 15.7).

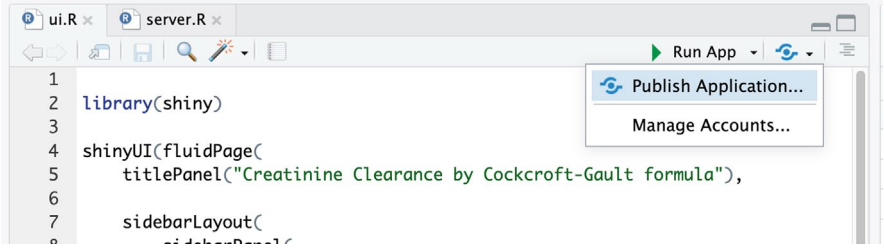

Fig. 15.6 Publishing a developed web application

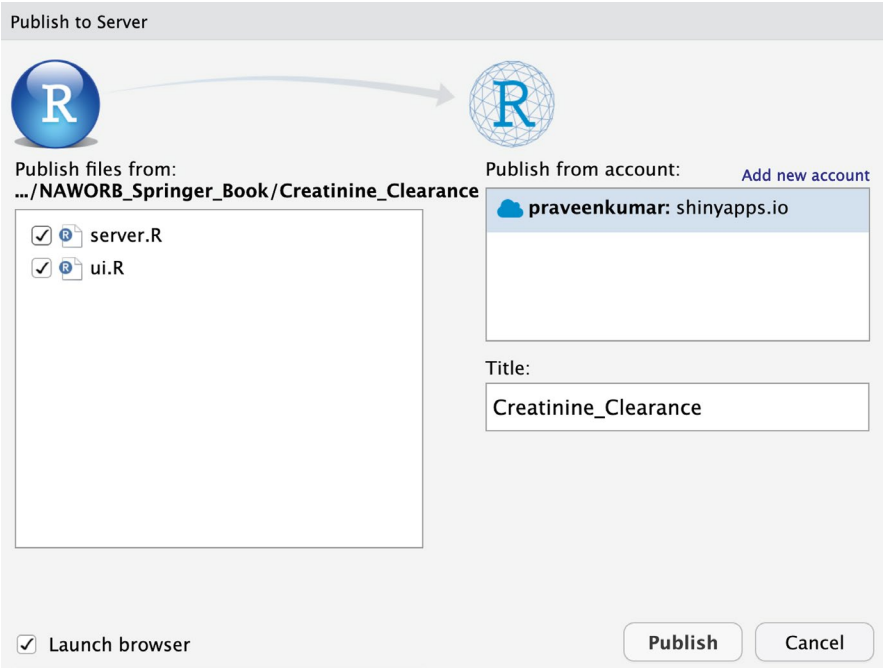

Fig. 15.7 Cloud upload details

The successful deployment will be associated with the deployment message (Fig. 15.8). The deployment process will include preparing, uploading, deploying, fetching packages, installing the image, starting a web instance, and activating the instance. If an application is already running on the same site, then the old application will be terminated (basically updated). Lastly, you will get the "Application successfully deployed to https://praveenkumar.shinyapps.io/Creatinine_Clearance/" message. The message also contains the web application URL. We can then visit the URL to see if the application works as intended.

The final version of the developed web shiny application is present in the URL: https://praveenkumar.shinyapps.io/Creatinine_Clearance/

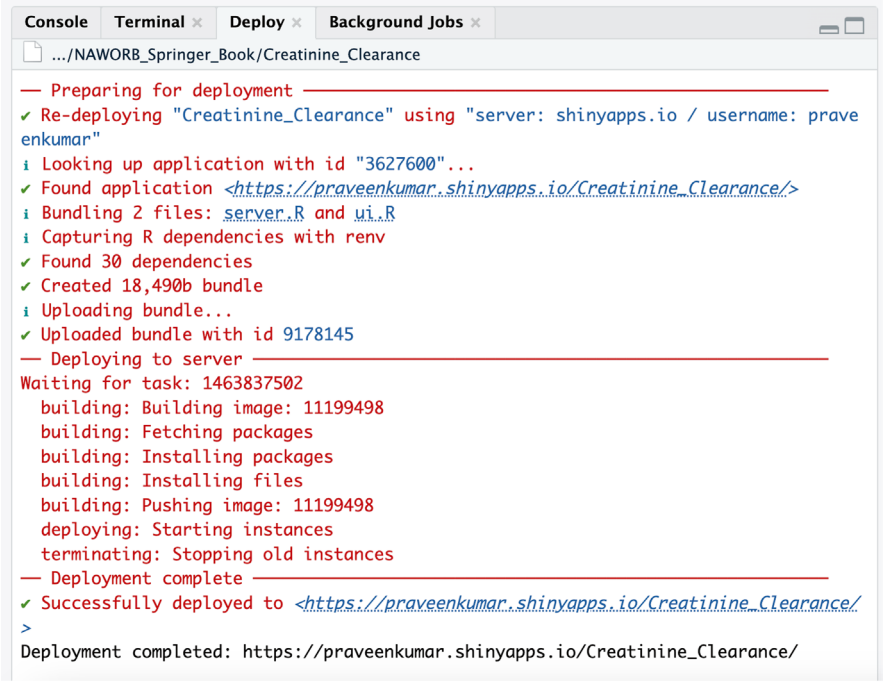

Fig. 15.8 Successful deployment message

Validation of the Developed Web Application

It is always essential that we validate the web application following development. There is standard literature available explaining different types of validation. The validation process helps to identify mistakes. Validation also gives users confidence that the application works as intended. However, it is always essential to add a disclaimer statement to the developed application so that users can be more cautious, and it also saves developers from litigation issues.

References

1. Chang W, Joe Cheng JJ Allaire CS, et al. Shiny: web application framework for R. R package version 1.6.0. 2021. https://CRAN.R-project.org/package=shiny
2. Cockcroft DW, Gault MH. Prediction of creatinine clearance from serum creatinine. Nephron. 1976;16(1):31–41. https://doi.org/10.1159/000180580.

Index

A. Srinivasan et al. (eds.), *R for Basic Biostatistics in Medical Research*,
https://doi.org/10.1007/978-981-97-6980-3