Learn R for Applied Statistics

With Data Visualizations, Regressions, and Statistics

Eric Goh Ming Hui

Apress®

Learn R for Applied Statistics

With Data Visualizations, Regressions, and Statistics

Eric Goh Ming Hui

Apress®

Learn R for Applied Statistics

Eric Goh Ming Hui
Singapore, Singapore

ISBN-13 (pbk): 978-1-4842-4199-8 ISBN-13 (electronic): 978-1-4842-4200-1
https://doi.org/10.1007/978-1-4842-4200-1

Library of Congress Control Number: 2018965216

While the advice and information in this book are believed to be true and accurate at the date of publication, neither the authors nor the editors nor the publisher can accept any legal responsibility for any errors or omissions that may be made. The publisher makes no warranty, express or implied, with respect to the material contained herein.

Managing Director, Apress Media LLC: Welmoed Spahr
Acquisitions Editor: Celestin Suresh John
Development Editor: Matthew Moodie
Coordinating Editor: Divya Modi

Cover designed by eStudioCalamar

Cover image designed by Freepik (www.freepik.com)

Distributed to the book trade worldwide by Springer Science+Business Media New York, 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com. Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a **Delaware** corporation.

For information on translations, please e-mail rights@apress.com, or visit www.apress.com/rights-permissions.

Apress titles may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Print and eBook Bulk Sales web page at www.apress.com/bulk-sales.

Any source code or other supplementary material referenced by the author in this book is available to readers on GitHub via the book's product page, located at www.apress.com/978-1-4842-4199-8. For more detailed information, please visit www.apress.com/source-code.

Printed on acid-free paper

Table of Contents

About the Author

Eric Goh Ming Hui is a data scientist, software engineer, adjunct faculty, and entrepreneur with years of experience in multiple industries. His varied career includes data science, data and text mining, natural language processing, machine learning, intelligent system development, and engineering product design. Eric Goh has led teams in various industrial projects, including the advanced product code classification system project which automates Singapore Custom's trade facilitation process and Nanyang Technological University's data science projects where he develop his own DSTK data science software. He has years of experience in C#, Java, C/C++, SPSS Statistics and Modeler, SAS Enterprise Miner, R, Python, Excel, Excel VBA, and more. He won the Tan Kah Kee Young Inventors' Merit Award and was a Shortlisted Entry for TelR Data Mining Challenge. Eric Goh founded the SVBook website to offer affordable books, courses, and software in data science and programming.

He holds a Masters of Technology degree from the National University of Singapore, an Executive MBA degree from U21Global (currently GlobalNxt) and IGNOU, a Graduate Diploma in Mechatronics from A*STAR SIMTech (a national research institute located in Nanyang Technological University), and a Coursera Specialization Certificate in Business Statistics and Analysis from Rice University. He possesses a Bachelor of Science degree in Computing from the University of Portsmouth after National Service. He is also an AIIM Certified Business Process Management Master (BPMM), GSTF certified Big Data Science Analyst (CBDSA), and IES Certified Lecturer.

About the Technical Reviewer

Preeti Pandhu has a Master of Science degree in Applied (Industrial) Statistics from the University of Pune. She is SAS certified as a base and advanced programmer for SAS 9 as well as a predictive modeler using SAS Enterprise Miner 7. Preeti has more than 18 years of experience in analytics and training. She started her career as a lecturer in statistics and began her journey into the corporate world with IDeaS (now a SAS company), where she managed a team of business analysts in the optimization and forecasting domain. She joined SAS as a corporate trainer before stepping back into the analytics domain to contribute to a solution-testing team and research/consulting team. She was with SAS for 9 years. Preeti is currently passionately building her analytics training firm, DataScienceLab (www.datasciencelab.in).

Acknowledgments

Let me begin by thanking Celestin Suresh John, the Acquisition Editor and Manager, for the LinkedIn message that triggered this project. Thanks to Amrita Stanley, project manager of this book, for her professionalism.

It took a team to make this book, and it is my great pleasure to acknowledge the hard work and smart work of Apress team. The following are a few names to mention: Matthew Moodie, the Development Editor; Divya Modi, the Coordinating Editor; Mary Behr for copy editing; Kannan Chakravarthy for proofreading; Irfanullah for indexing; eStudioCalamar and Freepik for image editing; Krishnan Sathyamurthy for managing the production process; and Parameswari Sitrambalam for composing. I am also thankful to Preeti Pandhu, the technical reviewer, for thoroughly reviewing this book and offering valuable feedback.

Introduction

Who is this book for?

This book is primarily targeted to programmers or learners who want to learn R programming for statistics. This book will cover using R programming for descriptive statistics, inferential statistics, regression analysis, and data visualizations.

How is this book structured?

The structure of the book is determined by following two requirements:

- This book is useful for beginners to learn R programming for statistics.
- This book is useful for experts who want to use this book as a reference.

Topic	Chapters
Introduction to R and R programming fundamentals	1 to 3
Descriptive statistics, data visualizations, inferential statistics, and regression analysis	4 to 6

Contacting the Author

More information about Eric Goh can be found at www.svbook.com. He can be reached at gohminghui88@svbook.com.

CHAPTER 1

Introduction

In this book, you will use R for applied statistics, which can be used in the data understanding and modeling stages of the CRISP DM (data mining) model. Data mining is the process of mining the insights and knowledge from data. R programming was created for statistics and is used in academic and research fields. R programming has evolved over time and many packages have been created to do data mining, text mining, and data visualizations tasks. R is very mature in the statistics field, so it is ideal to use R for the data exploration, data understanding, or modeling stages of the CRISP DM model.

What Is R?

According to Wikipedia, R programming is for statistical computing and is supported by the R Foundation for Statistical Computing. The R programming language is used by academics and researchers for data analysis and statistical analysis, and R programming's popularity has risen over time. As of June 2018, R is ranked 10th in the TIOBE index. The TIOBE Company created and maintains the TIOBE programming community index, which is the measure of the popularity of programming languages. TIOBE is the acronym for "The Importance of Being Earnest."

R is a GNU package and is available freely under the GNU General Public License. This means that R is available with source code, and you are free to use R, but you must adhere to the license. R is available in the

E. G. M. Hui, *Learn R for Applied Statistics*, https://doi.org/10.1007/978-1-4842-4200-1_1

command line, but there are many integrated development environments (IDEs) available for R. An IDE is software that has comprehensive facilities like a code editor, compiler, and debugger tools to help developers write R scripts. One famous IDE is RStudio, which assists developers in writing R scripts by providing all the required tools in one software package.

R is an implementation of the S programming language, which was created by Ross Ihahka and Robert Gentlemen at the University of Auckland. R and its libraries are made up of statistical and graphical techniques, including descriptive statistics, inferential statistics, and regression analysis. Another strength of R is that it is able to produce publishable quality graphs and charts, and can use packages like ggplot for advanced graphs.

According to the CRISP DM model, to do a data mining project, you must understand the business, and then understand and prepare the data. Then comes modeling and evaluation, and then deployment. R is strong in statistics and data visualization, so it is ideal to use R for data understanding and modeling.

Along with Python, R is used widely in the field of data science, which consists of statistics, machine learning, and domain expertise or knowledge.

High-Level and Low-Level Languages

A high-level programming language (HLL) is designed to be used by a human and is closer to the human language. Its programming style is easier to comprehend and implement than a lower-level programming language (LLL). A high-level programming language needs to be converted to machine language before being executed, so a high-level programming language can be slower.

A low-level programming language, on the other hand, is a lot closer to the machine and computer language. A low-level programming language can be executed directly on computer without the need to convert between languages before execution. Thus, a low-level programming language can be faster than a high-level programming language. Low-level programming languages like the assembly language are more inclined towards machine language that deals with bits 0 and 1.

R is a HLL because it shares many similarities to human languages. For example, in R programming code,

```
> var1 <- 1;
> var2 <- 2;
>
> result <- var1 + var2;
> print(result)
 [1] 3
>
```

The R programming code is more like human language. A low-level programming language like the assembly language is more towards the machine language, like 0011 0110:

```
0x52ac87:      movl7303445 (%ebx), %eax
0x52ac78:      calll        0x6bfb03
```

What Is Statistics?

Statistics is a collection of mathematics to deal with the organization, analysis, and interpretation of data. Three main statistical methods are used in the data analysis: descriptive statistics, inferential statistics, and regressions analysis.

Descriptive statistics summarizes the data and usually focuses on the distribution, the central tendency, and the dispersion of data. The distribution can be normal distribution or binomial distribution, and the central tendency is to describe the data with respect to the central of the data. The central tendency can be the mean, median, and mode of the data. The dispersion describes the spread of the data, and dispersion can be the variance, standard deviation, and interquartile range.

Inferential statistics tests the relationship between two data sets or two samples, and a hypothesis is usually set for the statistical relationships between them. The hypothesis can be a null hypothesis or alterative hypothesis, and rejecting the null hypothesis is done using tests like the T Test, Chi Square Test, and ANOVA. The Chi Square Test is more for categorical variables, and the T Test is more for continuous variables. The ANOVA test is for more complex applications.

Regression analysis is used to identify the relationships between two variables. Regressions can be linear regressions or non-linear regressions. The regression can also be a simple linear regression or multiple linear regressions for identifying relationships for more variables.

Data visualization is the technique used to communicate or present data using graphs, charts, and dashboards. Data visualizations can help us understand the data more easily.

What Is Data Science?

Data science is a multidisciplinary field that includes statistics, computer science, machine learning, and domain expertise to get knowledge and insights from data. Data science usually ends up developing a data product. A data product is the changing of the data of a company into a product to solve a problem.

For example, a data product can be the product recommendation system used in Amazon and Lazada. These companies have a lot of data based on shoppers' purchases. Using this data, Amazon and Lazada can identify the shopping patterns of shoppers and create a recommendation system or data product to recommend other products whenever a shopper buys a product.

The term “data science” has become a buzzword and is now used to represent many areas like data analytics, data mining, text mining, data visualizations, prediction modeling, and so on.

The history of data science started in November 1997, when C. F. Jeff Wu characterized statistical work as data collection, analysis, and decision making, and presented his lecture called “Statistics = Data Science?” In 2001, William S. Cleveland introduced data science as a field that comprised statistics and some computing in his article called “Data Science: An Action Plan for Expanding the Technical Area of the Field of Statistics.”

DJ Patil, who claims to have coined the term “data science” with Jeff Hammerbacher and who wrote the “Data Scientist: The Sexiest Job of the 21st Century” article published in the *Harvard Business Review*, says that there is a data scientist shortage in many industries, and data science is important in many companies because data analysis can help companies make many decisions. Every company needs to make decisions in strategic directions.

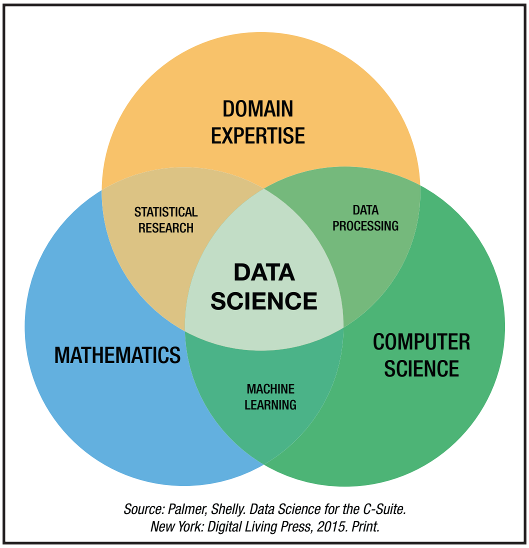
Statistics is important in data science because it can help analysts or data scientists analyze and understand data. Descriptive statistics assists in summarizing the data, inferential statistics tests the relationship between two data sets or samples, and regression analysis explores the relationships between multiple variables. Data visualizations can explore the data with charts, graphs, and dashboards. Regressions and machine learning algorithms can be used in predictive analytics to train a model and predict a variable.

Linear regression has the formula y = mx + c. You use historical data to train the formula to get the m and c. Y is the output variable and x is the input variable. Machine learning algorithms and regression or statistical learning algorithms are used to predict a variable like this approach.

Domain expertise is the knowledge of the data set. If the data set is business data, then the domain expertise should be business; if it is university data, education is the domain expertise; if the data set is healthcare data, healthcare is the domain knowledge. I believe that business is the most important knowledge because almost all companies use data analysis to make important strategic business decisions.

Adding in product design and engineering knowledge takes us into the fields of Internet of Things (IoT) and smart cities because data science and predictive analytics can be used on sensor data. Because data science is a multidisciplinary field, if you can master statistics, machine e-learning, and business knowledge, it is extremely hard to be replaced. You can also work with statisticians, machine learning engineers, or business experts to complete a data science project.

Figure 1-1 shows a data science diagram.



Source: Palmer, Shelly. Data Science for the C-Suite. New York: Digital Living Press, 2015. Print.

Figure 1-1. *Data science is an intersection*

What Is Data Mining?

Data mining is closely related to data science. Data mining is the process of identifying the patterns from data using statistics, machine learning, and data warehouses or databases.

Extraction of patterns from data is not very new, and early methods include the use of the Nayes theorem and regressions. The growth of technologies increases the ability in data collection. The growth of technologies also allows the use of statistical learning and machine learning algorithms like neural networks, fuzzy logic, decision trees, generic algorithms, and support vector machines to uncover the hidden patterns of data. Data mining combines statistics and machine learning, and usually results in the creation of models for making predictions based on historical data.

The cross-industry standard process of data mining, also known as CRISP-DM, is a process used by data mining experts and it is one of the most popular data mining models. See Figure 1-2.

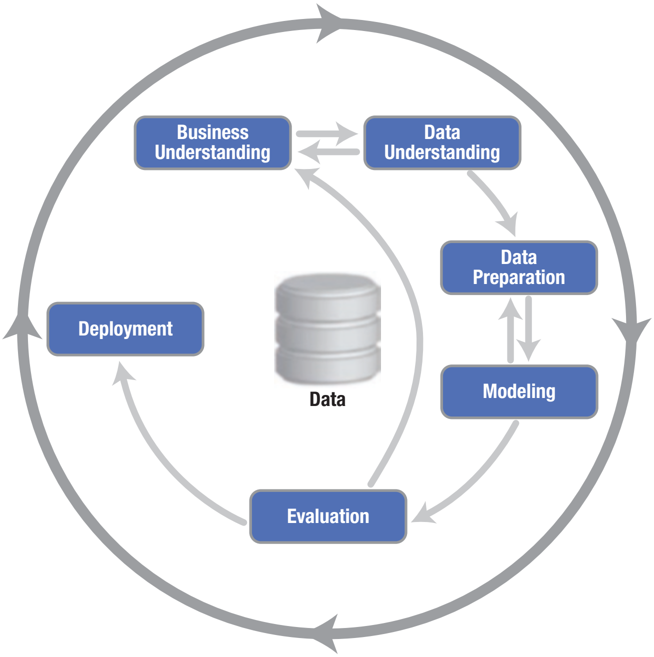


Figure 1-2. *Cross-industry standard process for data mining*

The CRISP-DM model was created in 1996 and involves SPSS, teradata, Daimler AG, NCR Corporation, and OHRA. The first version was depicted at the fourth CRISP-DM SIG Workshop in Brussels in 1999. Many practitioners use the CRISP-DM model, but IBM is the company that focuses on the CRISP-DM model and includes it in SPSS Modeler.

However, the CRISP-DM model is actually application neutral. The following sections explain its constituent parts.

Business Understanding

Business understanding is when you understand what your company wants and expects from the project. It is great to include key people in the discussions and documentation to produce a project plan.

Data Understanding

Data understanding involves the exploration of data that includes the use of statistics and data visualizations. Descriptive statistics can be used to summarize the data, inferential statistics can be used to test two data sets and samples, and regressions can be used to explore the relationships between multiple variables. Data visualizations use charts, graphs, and dashboards to understand the data. This phase allows you to understand the quality of data.

Data Preparation

Data preparation is one of the most important and time-consuming phases and can include selecting a sample subset or variables selection, imputing missing values, transforming attributes or variables including log transform and feature scaling transformation, and duplicates removal. Variables selection can be done with a correlation matrix in a data visualization.

Modeling

Modeling usually means the development of a prediction model to predict a variable in data. The prediction model can be developed using regression algorithms, statistical learning algorithms, and machine learning algorithms like neural networks, support vector machines, naïve Bayes, multiple linear regressions, decision trees, and more. You can also build prescriptive and descriptive models.

Evaluation

Evaluation is one of the phases where you may use ten-fold crossover validation techniques to evaluate the precision and recall of your model. You may improve your model accuracy by moving back to the previous phase to improve or prepare your data more. You may also select the most accurate model for your requirements. You may also evaluate the model using the business success criteria established in the beginning stage, which is the business understanding stage.

Deployment

Deployment is the process of using new insights and knowledge to improve your organization or make changes to improve your organization. You may use your prediction model to create a data product or to produce a final report based on your models.

What Is Text Mining?

While data mining is usually used to mine out patterns from numerical data, text mining is used to mine out patterns from textual data like Twitter tweets, blog postings, and feedback. Text mining, also known as text data mining, is the process of deriving high quality semantics and knowledge from textual data.

Text mining tasks may consist of text classification, text clustering, and entity extraction; text analytics may include sentiments analysis, TF-IDF, part-of-speech tagging, name entity recognizing, and text link analysis.

Text mining uses the same process as the data mining CRISP-DM model, with slight differences as shown in Figure 1-3.

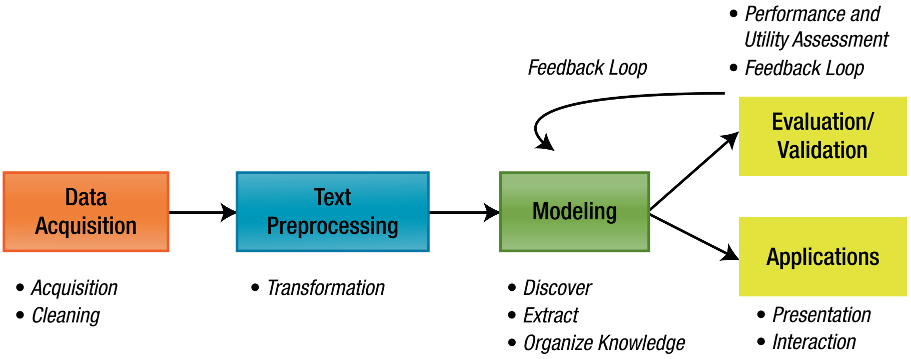


Figure 1-3. *Text mining*

Data Acquisition

Data acquisition is the process of gathering the textual data, combining the textual data, and doing some text cleaning. The business understanding stage may also be included here.

Text Preprocessing

Text preprocessing includes the process of porter stemming; stopwords removal; conversion of uppercase, lowercase, and propercase; extraction of words or tokens based on name entity or regular expressions; and transforming of text to vector or numerical forms. Text preprocessing is like the data preparation phase in CRISP-DM.

Modeling

Text analytics or text discovery is the use of part-of-speech tagging or name entity recognition to understand each document. It implements sentiment analysis to understand the sentiments of the documents and text link analysis to summarize all documents in text links. Some books may call text analytics as text mining; I think text analytics is similar to data understanding.

Modeling can also be the process of creating prediction models such as text classification. Some books may put the data mining process in this stage to create prediction models, descriptive models, and prescriptive models, after converting the text to vectors in the text preprocessing stage.

Evaluation/Validation

Evaluation or validation is the process of evaluating the accuracy of the model created. You can view this as the evaluation stage of the CRISP-DM model.

Applications

The applications stage is the deployment stage in the CRISP-DM model, where presentations or a full report are developed. You may also develop the model into a recommendation and classification system as a data product.

Natural Language Processing

Natural language processing (NLP) is an area of machine learning and computer science used to assist the computer in processing and understanding natural language. NLP can include part-of-speech tagging, parsing, porter stemming, name entity recognition, optical character recognition, sentiment analysis, speech recognition, and more. NLP works hand in hand with text analytics and text mining.

The history of NLP started in the 1950s when Alan Turing published an article called "Computing Machinery and Intelligence." Some notable natural language processing software was developed in the 1960s, such as ELIZA, which provided human-like interactions. In the 1970s, software was developed to write ontologies. In the 1980s, developers introduced Markov models and initiated research on statistical models to do POS tagging. Recent research has concentrated on supervised and semi-supervised algorithms and deep learning algorithms.

Three Types of Analytics

Selecting the type of analytics can be difficult and challenging; luckily, analytics can be categorized into descriptive analytics, predictive analytics, and prescriptive analytics. No analytic type is better than the others, but they can be combined with each other.

- **Descriptive Analytics**: Uses data analytics to know what happened.
- **Predictive Analytics**: Uses statistical learning and machine learning to predict the future.
- **Prescriptive Analytics**: Uses simulation algorithms to know what should be done.

Descriptive Analytics

Descriptive analytics uses statistics to summarize the data using descriptive statistics, inferential statistics to test the two data sets and samples, and regression analysis to study the relationships between multiple variables.

Predictive Analytics

Predictive analytics predicts a variable by implementing machine learning and statistical learning algorithms. In statistics, regressions can be used to predict a variable. For example, $y = mx + c$. You can determine m and c by training a linear regression model using historical data. Y is the variable to predict, x is the input variable. If you put in x value, you can predict the y.

Prescriptive Analytics

This is a field that allows a user to find the number of inputs to get a certain outcome. In simple form, this kind of analytics is used to provide advice. For example, $y = mx + c$. You have the m and c values. You want a y outcome, so what value should you put into x? To get the x value, what kind of things does your company need to do or what kind of advice do you need to give to the company? If you have multiple linear regressions, there are many x variables, so you need some simulation or evolutionary search algorithm to get the x values.

Big Data

Big data is data sets that are very big and complex for a computer to process. Big data has challenges that may include capturing data, data storage, data analysis, and data visualizations. There are three properties or characteristics of big data.

Volume

People are now more connected, so there are many more data sources, and as a consequence, the amount of data increased exponentially. The increase of data requires more computing power to process and analyze it. Traditional computing power is not able to process and analyze this data.

Velocity

The speed of data is increasing and the speed of data coming in is so fast that it is very difficult to process and analyze the data. Tradition computing methods can’t process and analyze at the speed of data coming in.

Variety

More sources means more data in different formats and types, such as images, videos, voice, speech, textual data, and numerical data, both unstructured and structured. Various data formats require different methods to extract the data from them. This means that the data is difficult to process and analyze, and traditional computing methods can’t process such data.

Data grows very quickly, due to IoT devices like mobile devices, wireless sensor networks, and RFID readers. Based on an IDC report, global data will increase from 4.4 zettabytes to 44 zettabytes from 2013 to 2020.

Relational databases and desktop statistics and data science software have challenges to process and analyze big data. Hence, big data requires parallel and distributed systems like Hadoop and Apache Spark to process and analyze the data.

Two popular systems or frameworks for big data are Apache Spark and Hadoop. Hadoop is a distributed data systems to store big data across different cluster and computers. One cluster can have many computers. The Hadoop storage system is known as the Hadoop Distributed System (HDFS). Hadoop has many ecosystems, such as mahout to do machine learning processing. Hadoop also has processing systems, such as MapReduce.

Apache Spark is a data processing system to process data on distributed data. Apache Spark does not have a file storage system, so it needs to integrate into a system like Hadoop. Apache Spark is a lot faster and completes full data analytics, data processing, and data prediction.

R, Python, and Java can interface with these Hadoop and Apache Spark systems.

Why R?

When learning data science, many people struggle with choosing which programming languages and data sciences to learn. There are many programming languages available for data science, like R, Python, SAS, Java, and more. There are many data science software packages to learn, such as SPSS Statistics, SPSS Modeler, SAS Enterprise Miner, Tableau, RapidMiner, Weka, GATE, and more.

I recommend learning R for statistics because it was developed for statistics in the first place. Python is a real programming language, so you can develop real applications and software via Python programming. Hence, if you want to develop a data product or data application, Python can be a better choice. R programming is very strong in statistics, so it is ideal for data exploration or data understanding using descriptive statistics, inferential statistics, regression analysis, and data visualizations. R is also ideal for modeling because you can use statistical learning like regressions for predictive analytics. R also has some packages for data mining, text mining, and machine learning like Rattle, CARET, and TM. R programming can also interface with big data systems like Apache Spark using Sparklyr. SAS programming is commercial, and Java has direct interfaces with GATE, Stanford NLP, and Weka. SPSS Statistics, SPSS Modeler, SAS Enterprise Miner, and Tableau are data science software packages with GUIs and are commercial. RapidMiner, Weka, and GATE are open source software packages for data science.

R is also heavily used in many of the companies that hire data scientists. Google and Facebook have data scientists who use R. R is also used in companies like Bank of America, Ford, Uber, Trulia, and more.

R is also heavily used in academia, and R is very popular among academic researchers, who can use R graphics for publications.

Scripts written in R can be used on different operating systems, including Linux, Apple, and Windows, as long as the R interpreter is installed. This is not possible with languages like C#.

Conclusion

In this chapter, you looked into R programming. You now know that R programming is a programming language for statistical computing and is supported by the R Foundation for Statistical Computing. The R language is used by researchers for statistical analysis, and R popularity has increased every year.

I also discussed high-level programming languages and low-level programming languages. HLLs are designed to be used by humans and are closer to the human language. LLLs, on the other hand, are a lot closer to the machine and computer languages. LLLs can be executed directly on a computer without the need to convert between languages, so they can be faster.

I also discussed statistics. Statistics is a collection of mathematics to deal with the organization, analysis, and interpretation of data. There are three main statistical methods used in data analysis: descriptive statistics, inferential statistics, and regressions analysis.

I also discussed data science. Data science is a multidisciplinary field that includes statistics, computer science, machine learning, and domain expertise to get knowledge and insights from data. Data science usually ends up with the development of a data product. A data product is the changing of the data of a company into a product to solve a problem.

Data mining is closely related to data science. Data mining is the process of identifying patterns from data using statistics, machine learning, and data warehouses or databases. Data mining consists of many models;

CRISP-DM is the most popular model for data mining. In CRISP-DM, data mining comprises business understanding, data understanding, data preparation, modeling, evaluation, and deployment.

While data mining is usually used to mine out patterns from numerical data, text mining is used to mine out patterns from textual data like Twitter tweets, blog postings, and feedback. Text mining, also known as text data mining, is the process of deriving high quality semantics and knowledge from textual data. Text mining consists of text classification, text clustering, and entity extraction; text analytics may include sentiments analysis, TF-IDF, part-of-speech tagging, name entity recognizing, and text link analysis. Text mining uses the same process as the data mining CRISP-DM model, with slight differences.

Natural language processing is an area of machine learning and computer science used to assist the computer in processing and understanding natural language. NLP can include part-of-speech tagging, parsing, porter stemming, name entity recognition, optical character recognition, sentiment analysis, speech recognition, and more. NLP works hand in hand with text analytics and text mining.

Selecting the types of analytics can be difficult and challenging. Luckily, analytics can be categorized into descriptive analytics, predictive analytics, and prescriptive analytics. No one type of analytics is better than the others, but they can be combined with each other.

Big data is data sets that are very big and complex for a computer to process. Big data has challenges that may include capturing data, data storage, data analysis, and data visualizations. There are three properties of big data: volume, velocity, and variety. There are two popular systems or frameworks for big data: Hadoop and Apache Spark.

When learning data science, there are many programming languages, like R, Python, SAS, and Java. There are many data science software packages, such as SPSS Statistics, SPSS Modeler, SAS Enterprise Miner, RapidMiner, and Weka. R was developed with statistics in mind, so it is best for the statistics portion of data mining, such as data understanding, modeling with statistical learning algorithms, and data visualizations. R has packages for machine learning, natural language processing, and text mining, and Apache Spark for big data. Python is a full programming language, and it is best for developing data product or software. The SAS programming language is commercial and not free. R has become very popular, according to the TIOBE ranking, and many companies like Facebook and Google have data scientists who use R. R is also very popular with academic researchers. R scripts or code can be run on different operating systems long as the R interpreter is installed.

References

Home. (2018, June 07). Retrieved from https://www.rstudio.com/.

Integrated development environment. (2018, August 22). Retrieved from https://en.wikipedia.org/wiki/Integrated_development_environment.

R (programming language). (2018, August 31). Retrieved from https://en.wikipedia.org/wiki/R_(programming_language).

RStudio. (2018, August 26). Retrieved from https://en.wikipedia.org/wiki/RStudio.

The R Project for Statistical Computing. (n.d.). Retrieved from https://www.r-project.org/.

What is R? (n.d.). Retrieved from https://www.r-project.org/about.html.

CHAPTER 2

Getting Started

R programming is a programming language with object-oriented features ideal for statistical computing and data visualizations. R programming can do descriptive statistics, inferential statistics, and regression analysis. R programming is a GNU package and is a command line application. RStudio is an integrated development environment (IDE) for R programming. An IDE offers features to help you write code more easily and more productively by providing a code editor, compiler, and debugger. The code editor usually has syntax highlighting and intelligent code completion.

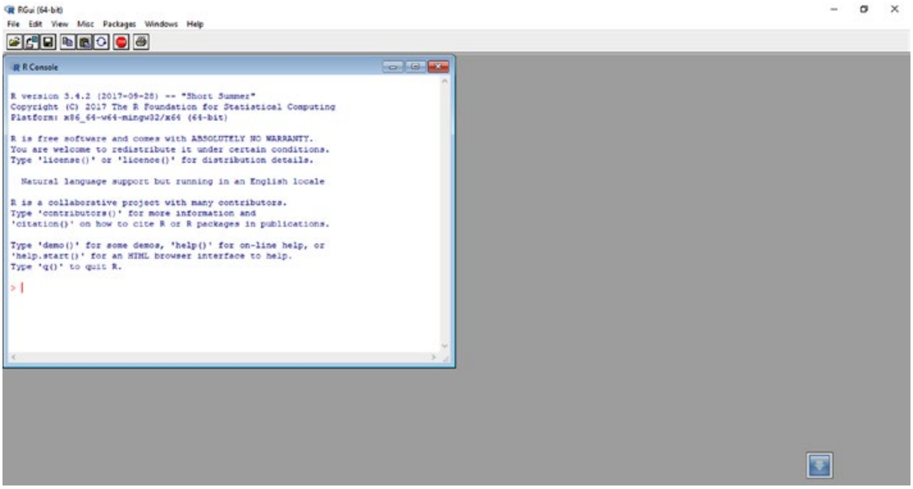
In this chapter, you will explore the R programming command line application and the RStudio IDE, and you will install R and RStudio on your computer. You will look into what an IDE is and you will explore the RStudio interface. RStudio and R can read a .csv file easily, perform some descriptive statistics, and plot simple graphs.

What Is R?

R programming is for statistical computing and is supported by the R Foundation for Statistical Computing. R programming is used by many academics and researchers for data and statistical analysis, and the popularity of R has risen over time.

E. G. M. Hui, *Learn R for Applied Statistics*, https://doi.org/10.1007/978-1-4842-4200-1_2

R is a GNU package and is available under the GNU General Public License, which can be assumed to be free to a certain extent and is open source. R is available in a command line application, as shown in Figure 2-1.



```
R version 3.4.2 (2017-09-28) -- "Short Summer"
Copyright (C) 2017 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> |
```

Figure 2-1. *The RGui interface*

R programming is an implementation of the S programming language, its libraries consist of statistical and data visualization techniques, and it can conduct descriptive statistics, inferential statistics, and regressions analysis. You will explore the differences between the R programming command line application and the RStudio IDE, as well as the basics of the descriptive statistics features and the data visualization features.

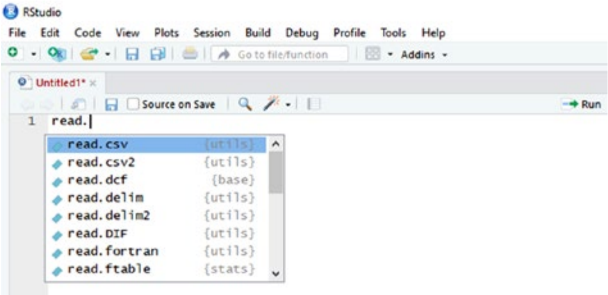
The Integrated Development Environment

An IDE is a software application that helps programmers develop software more easily and more productively. An IDE is made up of a code editor, compiler, and debugger tools. Code editors usually offer syntax highlighting and intelligent code completion.

Some IDEs, like NetBeans, also have an interpreter and others, like SharDevelop, don't. Some IDEs have a version control system and tools like a graphical user interface (GUI) builder, and many IDEs have class and object browsers.

IDEs are developed to increase the productivity of the developer by combining features like a code editor, compiler, debugger, and interpreter. This is different from a programming code text editor like VI and NotePad++, which offer syntax highlighting but usually don't communicate with the debugger and compiler.

The beginning of IDEs can be traced back to when punched cards were submitted to the compiler in early systems. Dartmouth BASIC was the first programming language to be created with an IDE. Maestro I was later created by Softlab Munich and can be considered the first full IDE between 1970s and 1980s. Maestro I can be found in the Museum of Information Technology at Arlington, Virginia. The Softbench IDE was later created to have plugins. Today, Visual Studio, NetBeans, and Eclipse are the most famous IDEs. The R programming IDE is RStudio, and Figure 2-2 shows its intelligent code completion.


RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
Go to file/function
Addins
Untitled1*
Source on Save
Run

```
read.
```

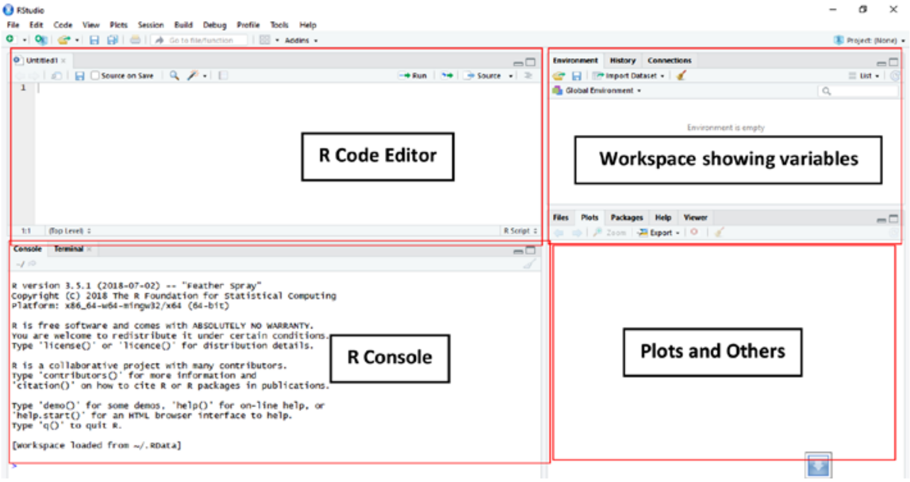
```
read.csv       {utils}
read.csv2      {utils}
read.dcf        {base}
read.delim     {utils}
read.delim2    {utils}
read.DIF       {utils}
read.fortran   {utils}
read.ftable    {stats}
```

Figure 2-2. *RStudio IDE intelligent code completion*

RStudio: The IDE for R

In R programming, RStudio is the most popular IDE. RStudio has a code editor that consists of syntax highlighting and intelligent code completion functions. RStudio also has a workspace showing all the variables and history. You may double-click the variables to view them using tables and other options.

The R console is in RStudio so you can view the results of the R scripts after running the scripts; you can also type into the R console with R code to do some simple computing. The Plots and Others portion is available in RStudio to let you view the charts and graphs plotted from R scripts. The Plots and Others portion allows you to easily save the graphs and charts. Figure 2-3 shows the RStudio IDE interface.

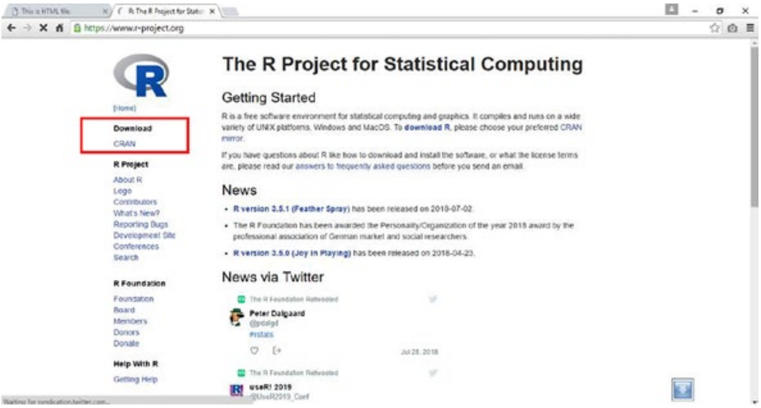


```
R version 3.5.1 (2018-07-02) -- "Feather Spray"
Copyright (C) 2018 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

[Workspace loaded from ~/.RData]

>
```

Figure 2-3. *RStudio IDE interface*

Installation of R and RStudio

In order to code R scripts, you must install the R programming command line application. You can download the R programming command line application from `www.r-project.org/`, as seen in Figure 2-4.



The R Project for Statistical Computing

Getting Started

R is a free software environment for statistical computing and graphics. It compiles and runs on a wide variety of UNIX platforms, Windows and MacOS. To **download R**, please choose your preferred CRAN mirror.

If you have questions about R like how to download and install the software, or what the license terms are, please read our answers to frequently asked questions before you send an email.

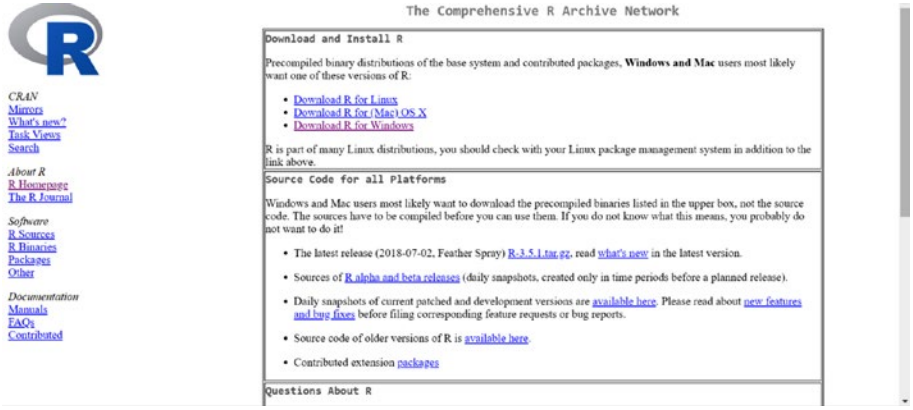
News

- **R version 3.5.1 (Feather Spray)** has been released on 2018-07-02.
- The R Foundation has been awarded the Personality/Organization of the year 2018 award by the professional association of German market and social researchers.
- **R version 3.5.0 (Joy in Playing)** has been released on 2018-04-23.

News via Twitter

Figure 2-4. *The R project website*

In this book, you will download R for Windows. You can also download for Linux and Mac OS, as seen in Figure 2-5.



The Comprehensive R Archive Network

Download and Install R

Precompiled binary distributions of the base system and contributed packages, **Windows and Mac** users most likely want one of these versions of R:

- Download R for Linux
- Download R for (Mac) OS X
- Download R for Windows

R is part of many Linux distributions, you should check with your Linux package management system in addition to the link above.

Source Code for all Platforms

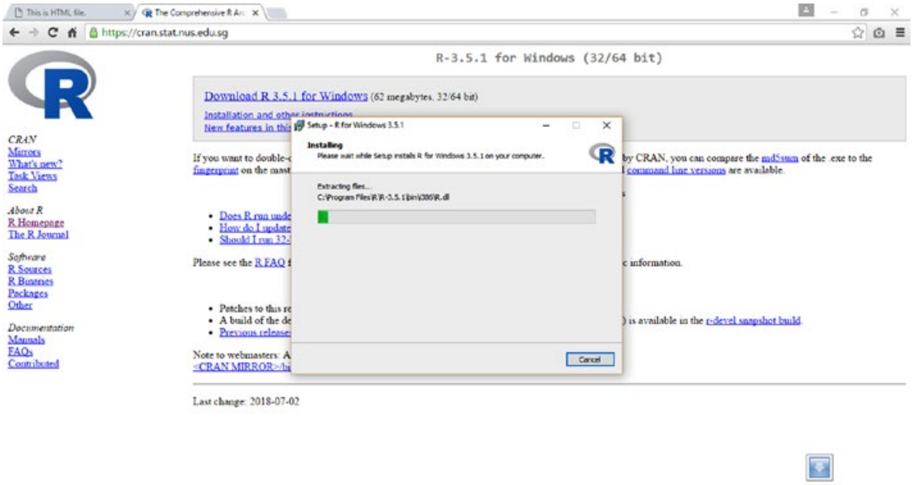
Windows and Mac users most likely want to download the precompiled binaries listed in the upper box, not the source code. The sources have to be compiled before you can use them. If you do not know what this means, you probably do not want to do it!

- The latest release (2018-07-02, Feather Spray) R-3.5.1.tar.gz, read what's new in the latest version.
- Sources of R alpha and beta releases (daily snapshots, created only in time periods before a planned release).
- Daily snapshots of current patched and development versions are available here. Please read about new features and bug fixes before filing corresponding feature requests or bug reports.
- Source code of older versions of R is available here.
- Contributed extension packages

Questions About R

Figure 2-5. *Downloading the R base for different OS options*

To install the software, double-click the download setup file and follow the instructions of the installer to install the R programming command line application, as seen in Figure 2-6.



R-3.5.1 for Windows (32/64 bit)

Download R 3.5.1 for Windows (62 megabytes, 32/64 bit)

Installation and other instructions

New features in this

Installing

Please wait while Setup installs R for Windows 3.5.1 on your computer.

Extracting files...

C:\Program Files\R\R-3.5.1\bin\i386\R.dll

If you want to double-c … by CRAN, you can compare the md5sum of the .exe to the fingerprint on the mast … command line versions are available.

- Does R run unde
- How do I update
- Should I run 32-

Please see the R FAQ f … c information.

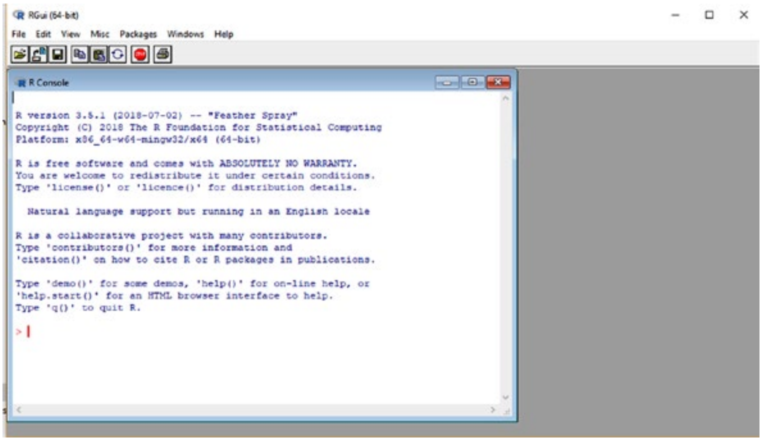
- Patches to this re
- A build of the de …) is available in the r-devel snapshot build.
- Previous releases

Note to webmasters: A
<CRAN MIRROR>/bi

Last change: 2018-07-02

Figure 2-6. *Installation of R*

After the R programming command line application is installed, you can start it, as seen in Figure 2-7.



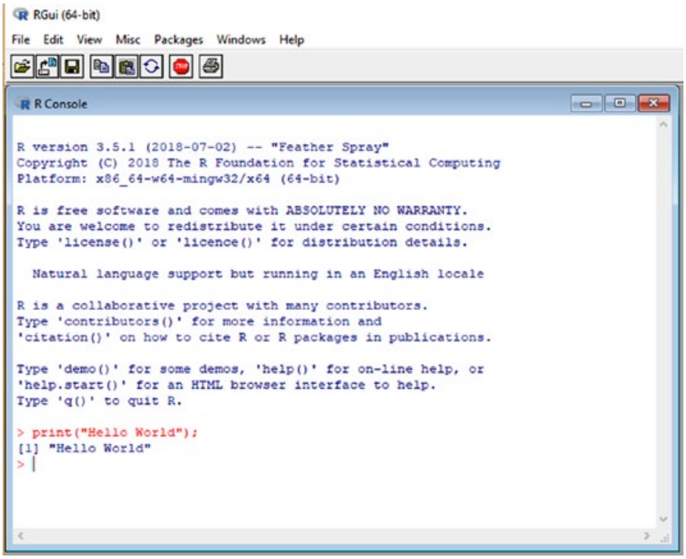
```
R version 3.5.1 (2018-07-02) -- "Feather Spray"
Copyright (C) 2018 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> |
```

Figure 2-7. *The RGui interface*

You can create your own Hello World application by using the `print()` function. The Hello World application is the standard first application to be developed when learning a programming language. Type the following code into the RGui:

```
print("Hello World");
```

The `print()` function is used to print some text on the console screen. You may print any text other than the "Hello World" shown in Figure 2-8.



```
R version 3.5.1 (2018-07-02) -- "Feather Spray"
Copyright (C) 2018 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> print("Hello World");
[1] "Hello World"
> |
```

Figure 2-8. *The R "Hello World" application*

RStudio is the most popular IDE for the R programming language. RStudio helps you write R programming code more easily and more productively. To download and install RStudio, visit `www.rstudio.com/`, as seen in Figure 2-9.

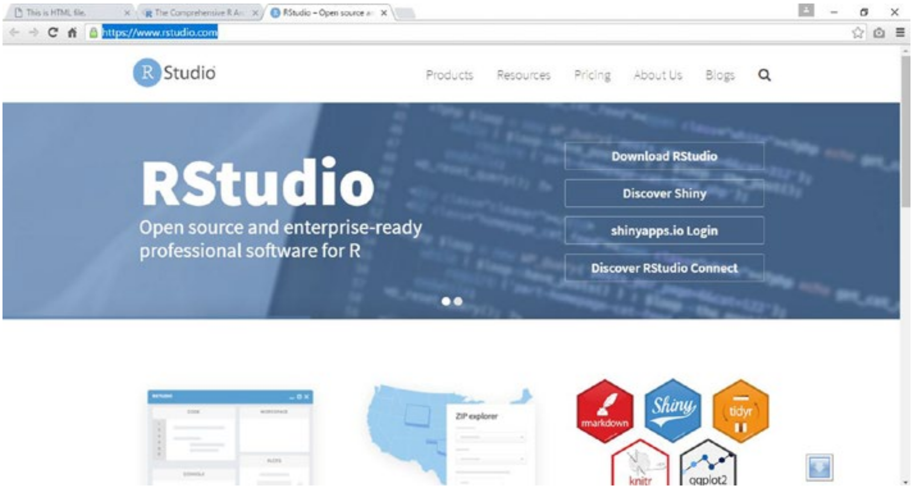
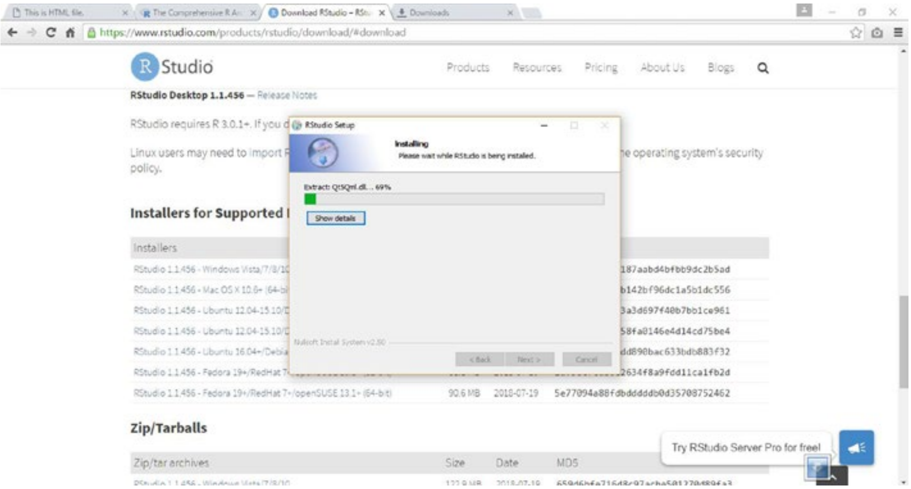


Figure 2-9. *The RStudio IDE website*

Download the latest version. For this book, you will download the 64-bit Windows version. After downloading the RStudio installer or setup file, double-click the file to install the RStudio IDE, as seen in Figure 2-10.



RStudio Desktop 1.1.456 — Release Notes

RStudio requires R 3.0.1+. If you d

Linux users may need to import R … e operating system's security policy.

Installing
Please wait while RStudio is being installed.

Extract: Qt5Qml.dll... 69%

Nullsoft Install System v2.50

Installers for Supported I

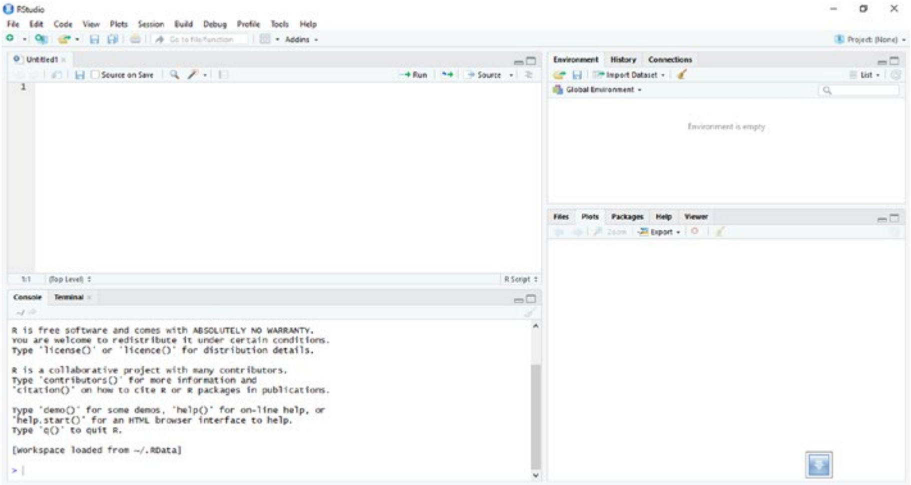
Installers
RStudio 1.1.456 - Windows Vista/7/8/10 … 187aabd4bfbb9dc2b5ad
RStudio 1.1.456 - Mac OS X 10.6+ (64-bi … b142bf96dc1a5b1dc556
RStudio 1.1.456 - Ubuntu 12.04-15.10/D … 3a3d697f40b7bb1ce961
RStudio 1.1.456 - Ubuntu 12.04-15.10/D … 58fa0146e4d14cd75be4
RStudio 1.1.456 - Ubuntu 16.04+/Debia … dd890bac633bdb883f32
RStudio 1.1.456 - Fedora 19+/RedHat 7 … 2634f8a9fdd11ca1fb2d
RStudio 1.1.456 - Fedora 19+/RedHat 7+/openSUSE 13.1+ (64-bit) 90.6 MB 2018-07-19 5e77094a88fdbddddb0d35708752462

Zip/Tarballs

Zip/tar archives	Size	Date	MD5
RStudio 1.1.456 - Windows Vista/7/8/10	122.9 MB	2018-07-19	659d6bfa716d8c97acba581270d89fa3

Figure 2-10. *Installation of the RStudio IDE*

After installing the RStudio IDE, you can run the RStudio IDE software, as seen in Figure 2-11.



```
R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

[Workspace loaded from ~/.RData]

> |
```

Figure 2-11. *The RStudio IDE interface*

Before running the script, you need to select the R programming command line application version to use. Click Tools ➤ Global Options, as seen in Figure 2-12.

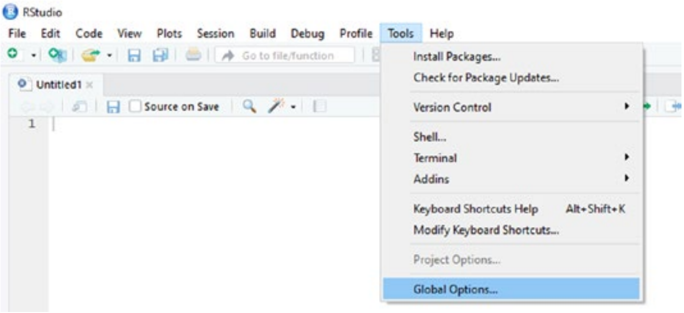
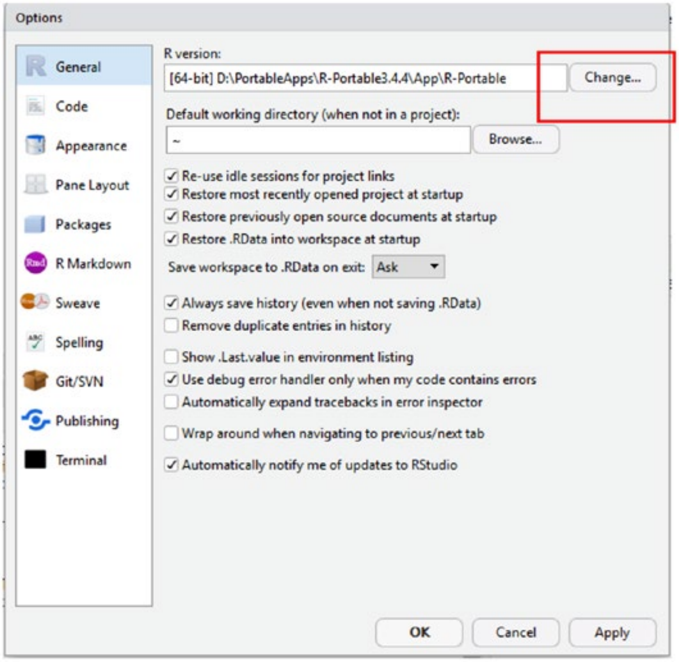

RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
Go to file/function
Untitled1
Source on Save
Install Packages...
Check for Package Updates...
Version Control
Shell...
Terminal
Addins
Keyboard Shortcuts Help Alt+Shift+K
Modify Keyboard Shortcuts...
Project Options...
Global Options...

Figure 2-12. *The RStudio IDE's Tools menu*

Click the Change button to select the R version, as seen in Figure 2-13.



R version:

[64-bit] D:\PortableApps\R-Portable3.4.4\App\R-Portable

Default working directory (when not in a project):

~

☑ Re-use idle sessions for project links
☑ Restore most recently opened project at startup
☑ Restore previously open source documents at startup
☑ Restore .RData into workspace at startup

Save workspace to .RData on exit: Ask

☑ Always save history (even when not saving .RData)
□ Remove duplicate entries in history

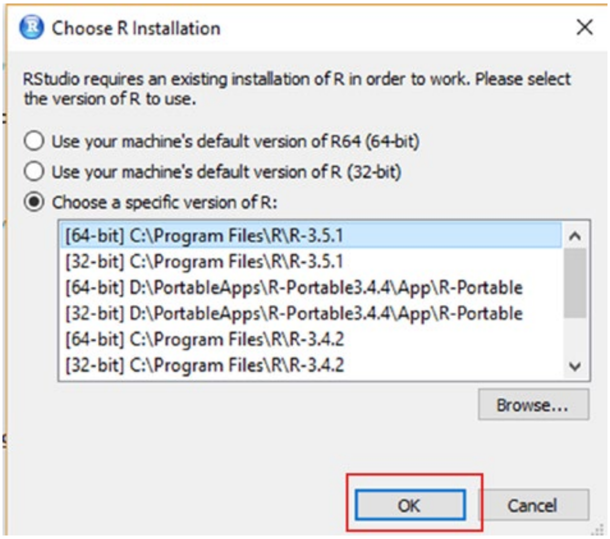
□ Show .Last.value in environment listing
☑ Use debug error handler only when my code contains errors
□ Automatically expand tracebacks in error inspector

□ Wrap around when navigating to previous/next tab

☑ Automatically notify me of updates to RStudio

Figure 2-13. *RStudio IDE options*

For the beginner, choose the R version shown in Figure 2-14. If you want to change the R version in the future, you can use this method to do so.


Choose R Installation

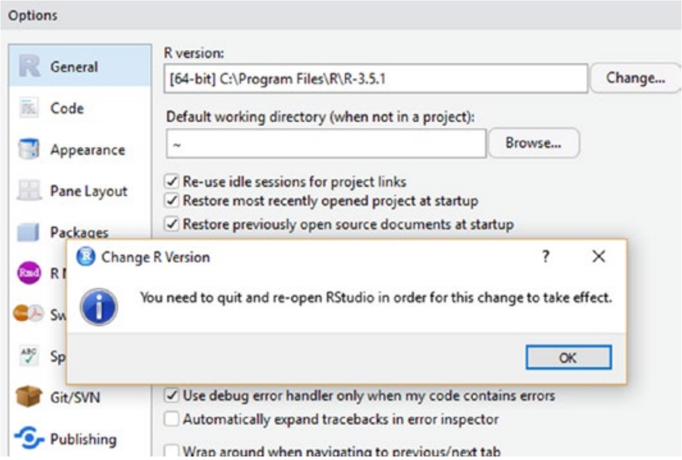
RStudio requires an existing installation of R in order to work. Please select the version of R to use.

○ Use your machine's default version of R64 (64-bit)
○ Use your machine's default version of R (32-bit)
● Choose a specific version of R:

```
[64-bit] C:\Program Files\R\R-3.5.1
[32-bit] C:\Program Files\R\R-3.5.1
[64-bit] D:\PortableApps\R-Portable3.4.4\App\R-Portable
[32-bit] D:\PortableApps\R-Portable3.4.4\App\R-Portable
[64-bit] C:\Program Files\R\R-3.4.2
[32-bit] C:\Program Files\R\R-3.4.2
```

Figure 2-14. *The Choose R Installation dialog*

After clicking OK and choosing the R version, you must restart the RStudio IDE, as depicted in Figure 2-15.



R version: [64-bit] C:\Program Files\R\R-3.5.1

Default working directory (when not in a project): ~

☑ Re-use idle sessions for project links
☑ Restore most recently opened project at startup
☑ Restore previously open source documents at startup

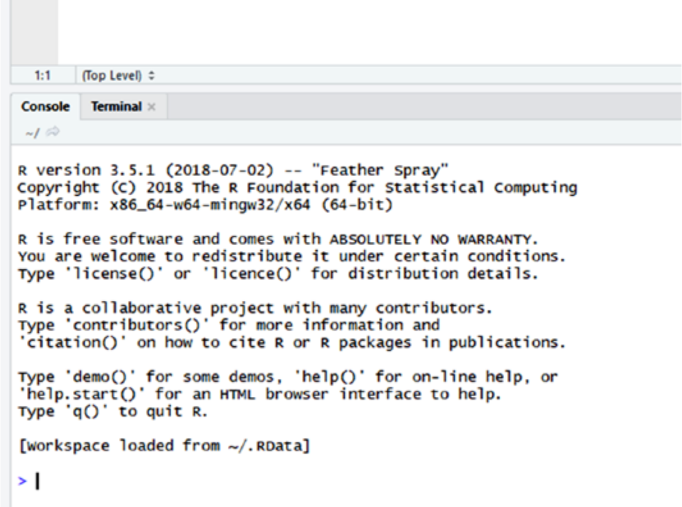
Change R Version

You need to quit and re-open RStudio in order for this change to take effect.

☑ Use debug error handler only when my code contains errors
□ Automatically expand tracebacks in error inspector
□ Wrap around when navigating to previous/next tab

Figure 2-15. *RStudio IDE R version changing*

After restarting RStudio, the Console tab should show the selected R version, as seen in Figure 2-16.



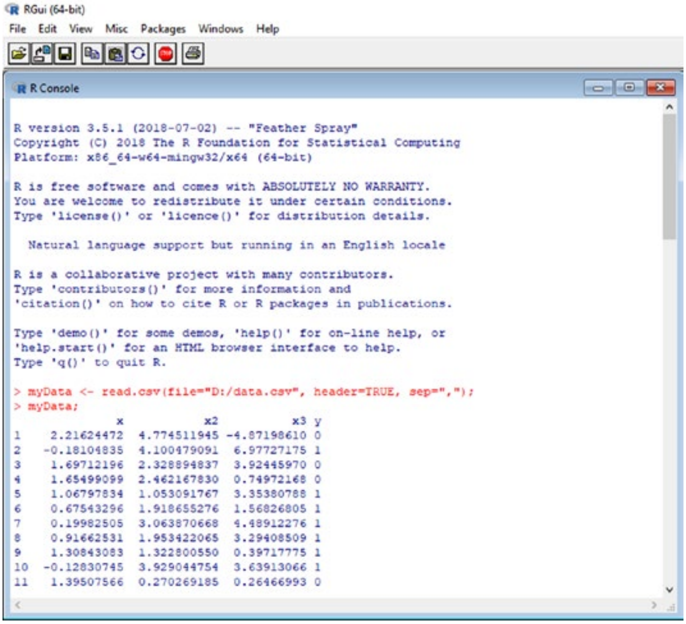
```
R version 3.5.1 (2018-07-02) -- "Feather Spray"
Copyright (C) 2018 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

[workspace loaded from ~/.RData]

>
```

Figure 2-16. *Changed R version*

Writing Scripts in R and RStudio

You can read a comma-separated values (CSV) file using the read.csv() function in the R programming language, as seen in Figure 2-17.

```
myData <- read.csv(file="D:/data.csv", header=TRUE, sep=",");
myData;
```

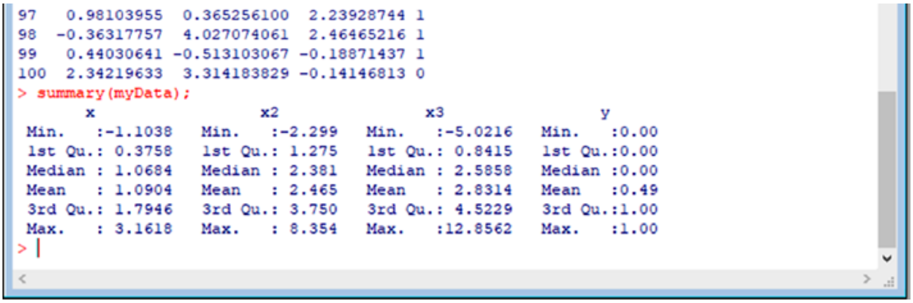


```
R version 3.5.1 (2018-07-02) -- "Feather Spray"
Copyright (C) 2018 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> myData <- read.csv(file="D:/data.csv", header=TRUE, sep=",");
> myData;
             x          x2          x3 y
1   2.21624472 4.774511945 -4.87198610 0
2  -0.18104835 4.100479091  6.97727175 1
3   1.69712196 2.328894837  3.92445970 0
4   1.65499099 2.462167830  0.74972168 0
5   1.06797834 1.053091767  3.35380788 1
6   0.67543296 1.918655276  1.56826805 1
7   0.19982505 3.063870668  4.48912276 1
8   0.91662531 1.953422065  3.29408509 1
9   1.30843083 1.322800550  0.39717775 1
10 -0.12830745 3.929044754  3.63913066 1
11  1.39507566 0.270269185  0.26466993 0
```

Figure 2-17. *RGui: Reading a CSV*

In the R programming language, you can use the `summary()` function to get the basic descriptive statistics for all the variables. I will discuss descriptive statistics, shown in Figure 2-18, in a future chapter of this book.

```
summary(myData);
```

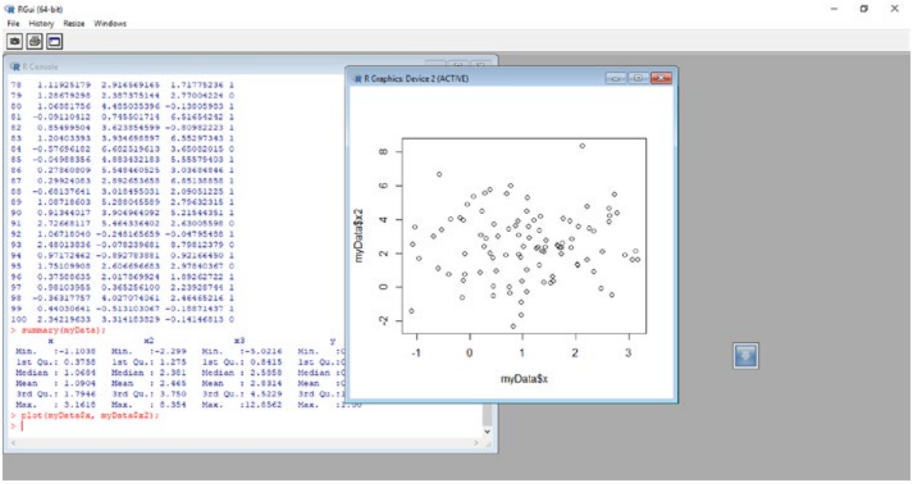


```
97   0.98103955  0.365256100  2.23928744 1
98  -0.36317757  4.027074061  2.46465216 1
99   0.44030641 -0.513103067 -0.18871437 1
100  2.34219633  3.314183829 -0.14146813 0
> summary(myData);
       x                 x2               x3                y       
 Min.   :-1.1038   Min.   :-2.299   Min.   :-5.0216   Min.   :0.00  
 1st Qu.: 0.3758   1st Qu.: 1.275   1st Qu.: 0.8415   1st Qu.:0.00  
 Median : 1.0684   Median : 2.381   Median : 2.5858   Median :0.00  
 Mean   : 1.0904   Mean   : 2.465   Mean   : 2.8314   Mean   :0.49  
 3rd Qu.: 1.7946   3rd Qu.: 3.750   3rd Qu.: 4.5229   3rd Qu.:1.00  
 Max.   : 3.1618   Max.   : 8.354   Max.   :12.8562   Max.   :1.00  
> 
```

Figure 2-18. *RGui: The summary() function*

In the R programming language, you can plot a scatterplot using the `plot()` function, as seen in Figure 2-19.

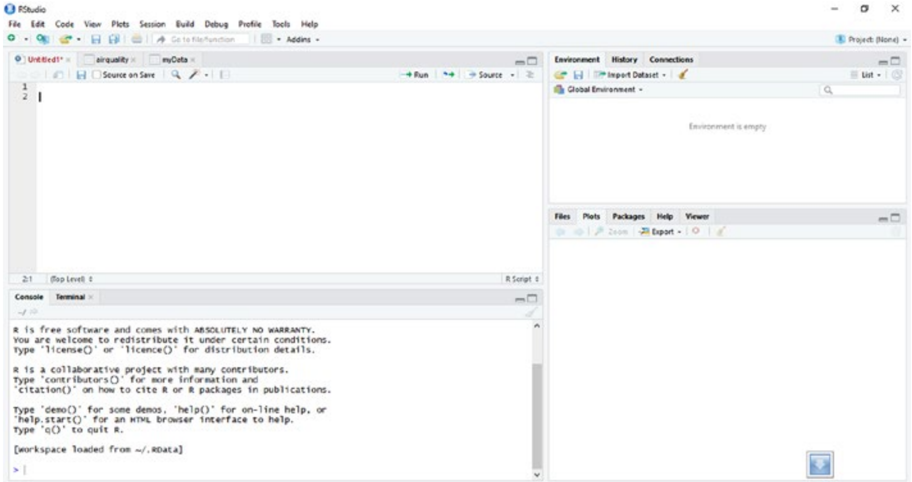
```
plot(myData$x, myData$x2);
```



```
78    1.11925179  2.916569165  1.71775236 1
79    1.28679298  2.387375144  2.77004224 0
80    1.06881756  4.485035396 -0.13805983 1
81   -0.09110412  0.745501714  6.51654242 1
82    0.85499504  3.623854599 -0.80982223 1
83    1.20403393  3.934698597  6.55297343 1
84   -0.57696102  6.602519613  3.65082015 0
85   -0.04988356  4.883432183  5.55579403 1
86    0.27860809  5.548460525  3.03684846 1
87    0.29924083  2.892653658  6.65138858 1
88   -0.68137641  3.018495031  2.09051225 1
89    1.08718603  5.288045589  2.79632315 1
90    0.91344017  3.906964092  5.21544351 1
91    2.72668117  5.464336402  2.63005598 0
92    1.06718040 -0.248165659 -0.04795488 1
93    2.48013836 -0.078239681  8.79812379 0
94    0.97172462 -0.892783881  0.92166450 1
95    1.75109908  2.606696683  2.97840367 0
96    0.37588635  2.017869924  1.89262722 1
97    0.98103985  0.365256100  2.23928744 1
98   -0.36317757  4.027074061  2.46465216 1
99    0.44030641 -0.513103067 -0.18871437 1
100   2.34219633  3.314183829 -0.14146813 0
> summary(myData);
       x                 x2                x3                 y
 Min.   :-1.1038   Min.   :-2.299   Min.   :-5.0216   Min.   :0
 1st Qu.: 0.3758   1st Qu.: 1.275   1st Qu.: 0.8415   1st Qu.:0
 Median : 1.0684   Median : 2.381   Median : 2.5858   Median :0
 Mean   : 1.0904   Mean   : 2.465   Mean   : 2.8314   Mean   :0
 3rd Qu.: 1.7946   3rd Qu.: 3.750   3rd Qu.: 4.5229   3rd Qu.:1
 Max.   : 3.1618   Max.   : 8.354   Max.   :12.8562   Max.   :1.00
> plot(myData$x, myData$x2);
> 
```

Figure 2-19. *RGui: Plotting a chart*

RStudio is an IDE that provides a GUI for the R programming command line application. RStudio provide word suggestions and syntax highlighting for the R programming language. The RStudio IDE for the R programming language is seen in Figure 2-20.



```
R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

[workspace loaded from ~/.RData]

>
```

***Figure 2-20.** The RStudio IDE interface*

With RStudio, you can write all the code into the code editor and run the script, as seen in Figures 2-22, 2-23, and 2-24.

```
myData <- read.csv(file="D:/data.csv", header=TRUE, sep=",");
myData;
summary(myData);
plot(myData$x, myData$x2);
```

As you type the code, RStudio shows the intelligent code completion, as shown in Figure 2-21.

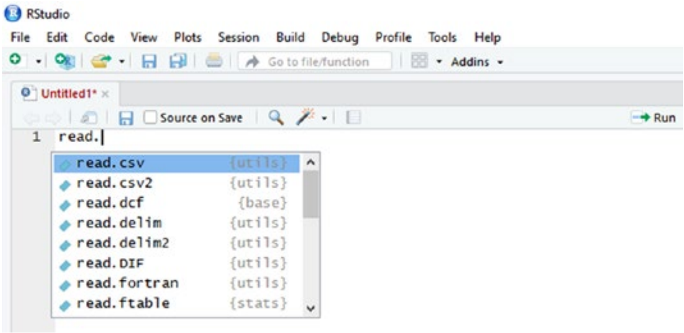
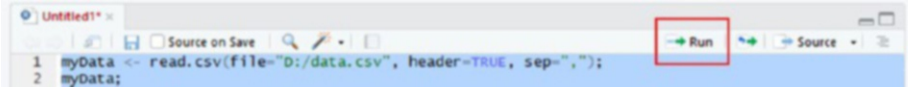


***Figure 2-21.** RStudio IDE intelligent code completion*

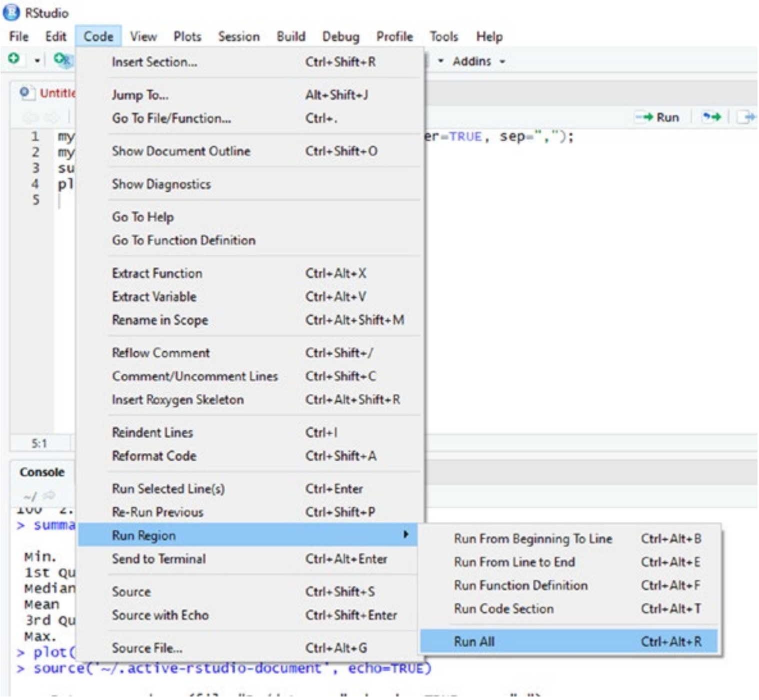
You must select all the R code in the code editor and click Run or Ctrl + Enter to run the script (Figure 2-22).



```
myData <- read.csv(file="D:/data.csv", header=TRUE, sep=",");
myData;
```

__Figure 2-22.__ RStudio IDE: Running a script

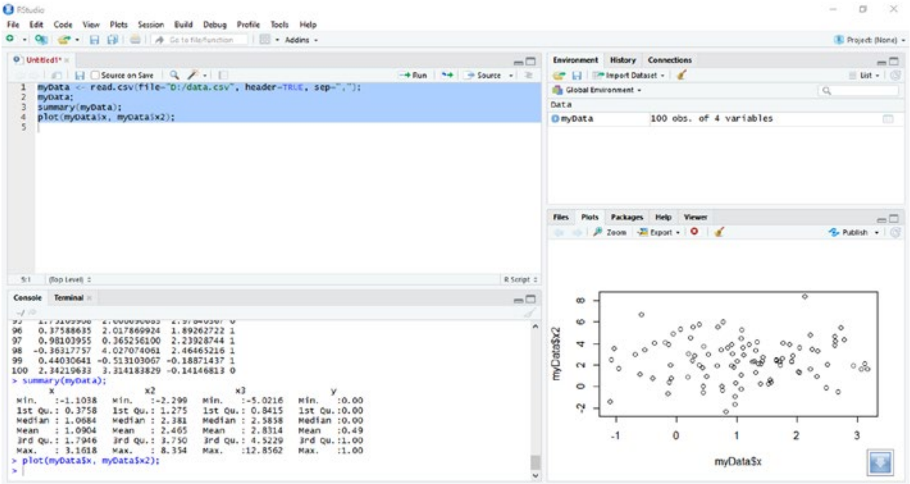
Or you can click Code ➤ Run Region ➤ Run All (Figure 2-23).


RStudio
File
Edit
Code
View
Plots
Session
Build
Debug
Profile
Tools
Help
Addins
Untitled
Run
Console
Insert Section... Ctrl+Shift+R
Jump To... Alt+Shift+J
Go To File/Function... Ctrl+.
Show Document Outline Ctrl+Shift+O
Show Diagnostics
Go To Help
Go To Function Definition
Extract Function Ctrl+Alt+X
Extract Variable Ctrl+Alt+V
Rename in Scope Ctrl+Alt+Shift+M
Reflow Comment Ctrl+Shift+/
Comment/Uncomment Lines Ctrl+Shift+C
Insert Roxygen Skeleton Ctrl+Alt+Shift+R
Reindent Lines Ctrl+I
Reformat Code Ctrl+Shift+A
Run Selected Line(s) Ctrl+Enter
Re-Run Previous Ctrl+Shift+P
Run Region
Send to Terminal Ctrl+Alt+Enter
Source Ctrl+Shift+S
Source with Echo Ctrl+Shift+Enter
Source File... Ctrl+Alt+G
Run From Beginning To Line Ctrl+Alt+B
Run From Line to End Ctrl+Alt+E
Run Function Definition Ctrl+Alt+F
Run Code Section Ctrl+Alt+T
Run All Ctrl+Alt+R

```
> source('~/.active-rstudio-document', echo=TRUE)
```

__Figure 2-23.__ RStudio IDE: Running a script

The results are shown in Figure 2-24.



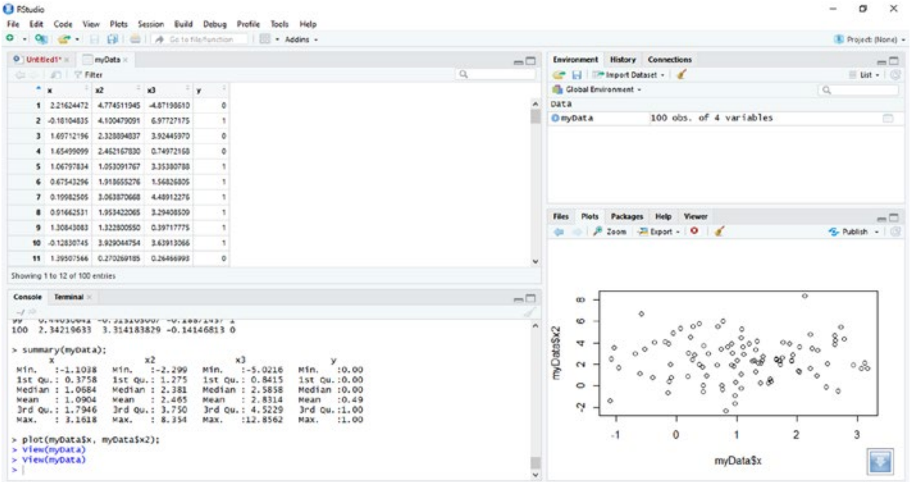
```
myData <- read.csv(file="D:/data.csv", header=TRUE, sep=",");
myData;
summary(myData);
plot(myData$x, myData$x2);
```

```
Data
myData          100 obs. of 4 variables
```

```
96    0.37588635  2.017869924  1.89262722 1
97    0.98103955  0.365256100  2.23928744 1
98   -0.36317757  4.027074061  2.46465216 1
99    0.44030641 -0.513103067 -0.18871437 1
100   2.34219633  3.314183829 -0.14146813 0
> summary(myData);
       x                  x2                x3                 y
 Min.   :-1.1038   Min.   :-2.299   Min.   :-5.0216   Min.   :0.00
 1st Qu.: 0.3758   1st Qu.: 1.275   1st Qu.: 0.8415   1st Qu.:0.00
 Median : 1.0684   Median : 2.381   Median : 2.5858   Median :0.00
 Mean   : 1.0904   Mean   : 2.465   Mean   : 2.8314   Mean   :0.49
 3rd Qu.: 1.7946   3rd Qu.: 3.750   3rd Qu.: 4.5229   3rd Qu.:1.00
 Max.   : 3.1618   Max.   : 8.354   Max.   :12.8562   Max.   :1.00
> plot(myData$x, myData$x2);
>
```

Figure 2-24. *RStudio IDE: Results after running the R script*

The RStudio IDE offers syntax highlighting features in the code editor. When you run the R script, you can view the results in the Console tab and see the scatterplot in the Plots tab. By double-clicking `myData` in the Global Environment tab, you can view the data loaded from the .csv file, as shown in Figure 2-25.



	x	x2	x3	y
1	2.21624472	4.774511945	-4.87198610	0
2	-0.18104835	4.100479091	6.97727175	1
3	1.69712196	2.328894837	3.92445970	0
4	1.65499099	2.462167830	0.74972168	0
5	1.06797834	1.053091767	3.35380788	1
6	0.67543296	1.918655276	1.56826805	1
7	0.19982505	3.063870668	4.48912276	1
8	0.91662531	1.953422065	3.29408509	1
9	1.30843083	1.322800550	0.39717775	1
10	-0.12830745	3.929044754	3.63913066	1
11	1.39507566	0.270269185	0.26466993	0

Showing 1 to 12 of 100 entries

Data
myData 100 obs. of 4 variables

```
100  2.34219633  3.314183829 -0.14146813 0

> summary(myData);
       x                 x2               x3                y
 Min.   :-1.1038   Min.   :-2.299   Min.   :-5.0216   Min.   :0.00
 1st Qu.: 0.3758   1st Qu.: 1.275   1st Qu.: 0.8415   1st Qu.:0.00
 Median : 1.0684   Median : 2.381   Median : 2.5858   Median :0.00
 Mean   : 1.0904   Mean   : 2.465   Mean   : 2.8314   Mean   :0.49
 3rd Qu.: 1.7946   3rd Qu.: 3.750   3rd Qu.: 4.5229   3rd Qu.:1.00
 Max.   : 3.1618   Max.   : 8.354   Max.   :12.8562   Max.   :1.00

> plot(myData$x, myData$x2);
> View(myData)
> View(myData)
>
```

Figure 2-25. *RStudio IDE: Viewing the loaded data*

Conclusion

In this chapter, you looked at R programming. You now understand what R programming is: it's a programming language for statistical computing and is supported by the R Foundation for Statistical Computing. R programming is used by researchers for statistical analysis, and R popularity has increased every year.

You also explored RStudio. RStudio is the IDE for the R programming language and it has syntax highlighting and intelligent code completion to assist you in writing R scripts more easily and more productively. You also looked at how R can read a .csv file and perform descriptive statistics and data visualizations, and you explored the differences between them.

You also installed R and the RStudio IDE, and you saw how to allow RStudio IDE to integrate with the R programming command line application.

You also learned than an IDE is software to help you write code more easily and more productively. IDEs usually offer syntax highlighting and intelligent code completion and have a code editor, a compiler, and a debugger.

For R programming, RStudio is the most popular IDE. RStudio has a code editor that consists of syntax highlighting and intelligent code completion. RStudio also has a workspace showing all the variables and history. You may double-click the variables to view them using tables and more. The R console is in RStudio so you can view the results of R scripts. The Plots and Others portion is available in RStudio to let you view the charts and graphs plotted from the R code.

References

Home. (2018, June 07). Retrieved from www.rstudio.com/.

Integrated development environment. (2018, August 22). Retrieved from https://en.wikipedia.org/wiki/Integrated_development_environment.

RStudio. (2018, August 26). Retrieved from https://en.wikipedia.org/wiki/RStudio.

The R Project for Statistical Computing. (n.d.). Retrieved from www.r-project.org/.

What is R? (n.d.). Retrieved from www.r-project.org/about.html.

CHAPTER 3

Basic Syntax

You will use R for applied statistics, which can be used in the data understanding and modeling stages of the CRISP-DM data mining model. R programming is a programming language with object-oriented programming features. R programming was created for statistics and is used in the academic and research fields. However, before you go into statistics, you need to learn to program R scripts.

In this chapter, you will explore the syntax of R programming. I will discuss the R console and code editor in RStudio, as well as R objects and the data structure of R programming, from variables and data types to lists, vectors, matrices, and data frames. I will also discuss conditional statements, loops, and functions. Then you will create a simple calculator after learning the basics.

Writing in R Console

As you saw in Chapter 2, the R console offers a fast and easy way to do statistical calculations and some data visualizations. The R console is also like a calculator, so you can always use the R console to calculate some math equations.

To do math calculations, you can just type in some math equations like

```
1 + 1
> 1 + 1
[1] 2
```

E. G. M. Hui, *Learn R for Applied Statistics*, https://doi.org/10.1007/978-1-4842-4200-1_3

```
1 - 3
> 1 - 3
[1] -2
```

```
1 * 5
> 1 * 5
[1] 5
```

```
1 / 6
> 1 / 6
[1] 0.1666667
```

```
tan(2)
> tan(2)
[1] -2.18504
```

To do some simple statistical calculations, you can so the following:

Standard deviation

```
sd(c(1, 2, 3, 4, 5, 6))
>sd(c(1, 2, 3, 4, 5, 6))
[1] 1.870829
```

Mean

```
mean(c(1, 2, 3, 4, 5, 6))
> mean(c(1, 2, 3, 4, 5, 6))
[1] 3.5
```

Min

```
min(c(1, 2, 3, 4, 5,6 ))
> min(c(1, 2, 3, 4, 5, 6))
[1] 1
```

To plot charts or graphs, type

```
plot(c(1, 2, 3, 4, 5, 6), c(2, 3, 4, 5, 6, 7))
> plot(c(1, 2, 3, 4, 5, 6), c(2, 3, 4, 5, 6, 7))
```

which is shown in Figure 3-1.

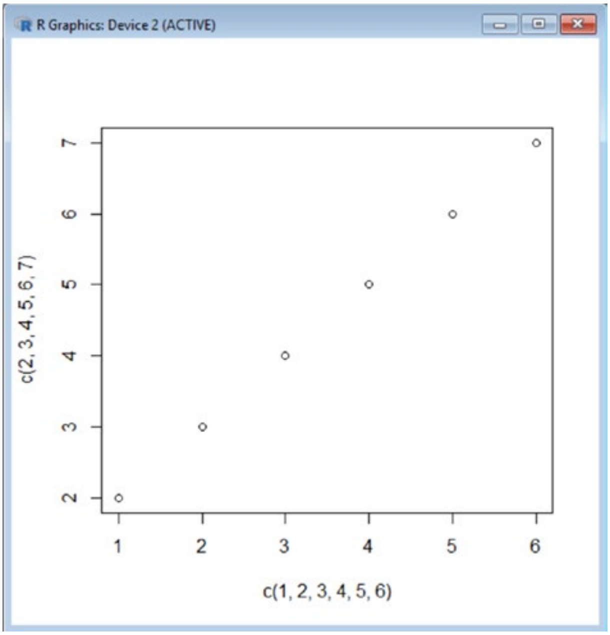


Figure 3-1. *Scatter plot*

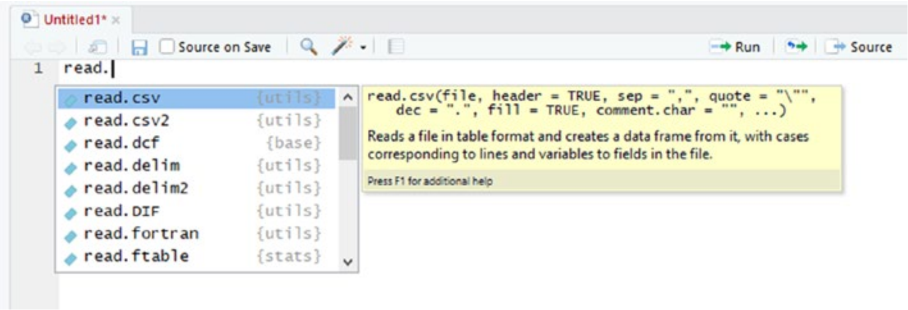
To sum up, the R console, despite being basic, offers the following advantages:

- High performance
- Fast prototyping and testing of your ideas and logic before proceeding further, such as when developing Windows Form applications

- Personally, I use the R console application to test algorithms and other code fragments when in the process of developing complex R scripts.

Using the Code Editor

The RStudio IDE offers features like a code editor, debugger, and compiler that communicate with the R command line application or R console. The R code editor offers features like intelligent code completion and syntax highlighting, shown in Figures 3-2 and 3-3, respectively.



```
read.
```

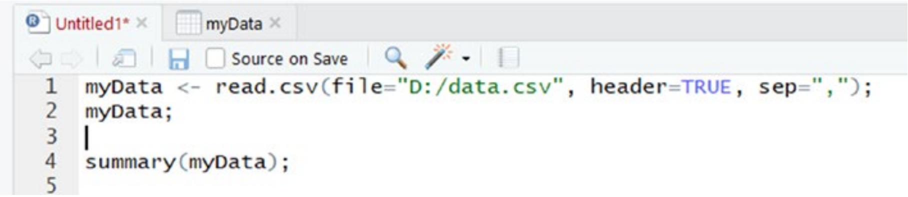
```
read.csv       {utils}
read.csv2      {utils}
read.dcf        {base}
read.delim     {utils}
read.delim2    {utils}
read.DIF       {utils}
read.fortran   {utils}
read.ftable    {stats}
```

```
read.csv(file, header = TRUE, sep = ",", quote = "\"",
    dec = ".", fill = TRUE, comment.char = "", ...)
```

Reads a file in table format and creates a data frame from it, with cases corresponding to lines and variables to fields in the file.

Press F1 for additional help

__Figure 3-2.__ Example of intelligent code completion

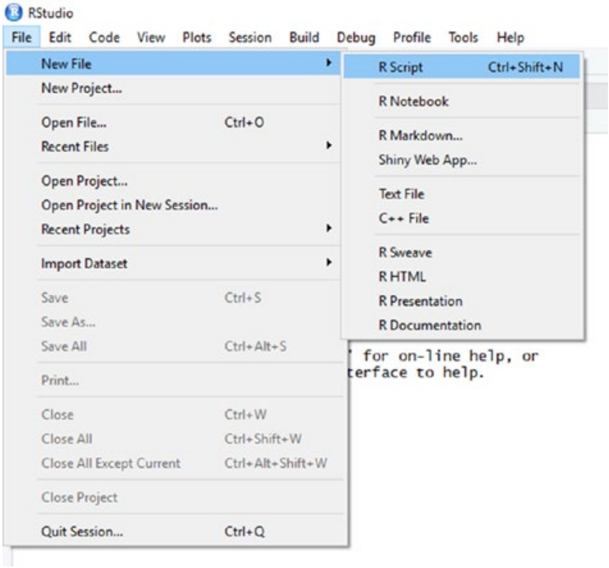


```
myData <- read.csv(file="D:/data.csv", header=TRUE, sep=",");
myData;

summary(myData);

```

__Figure 3-3.__ Example of syntax highlighting

To create a new script in RStudio, click

File ➤ New ➤ R Script, as shown in Figure 3-4.

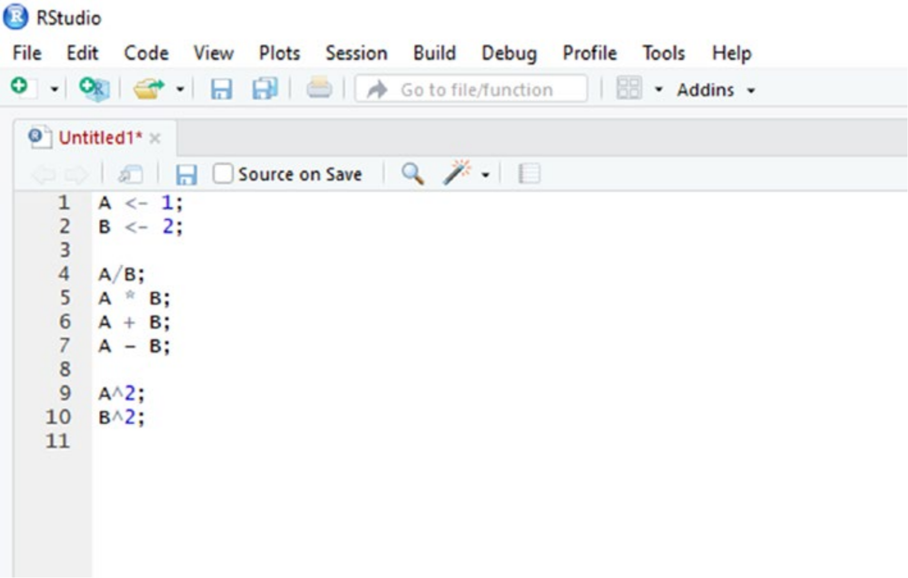


```
' for on-line help, or
erface to help.
```

Figure 3-4. *Creating a new script*

You can then code your R Script. For now, type in the following code, shown in Figure 3-5:

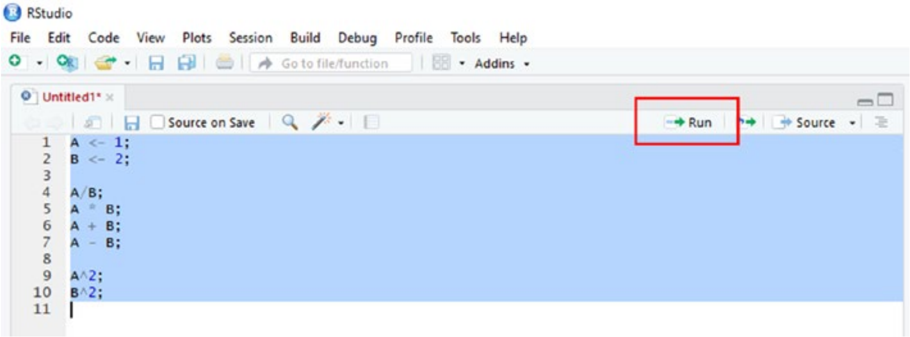
```
A <- 1;
B <- 2;

A/B;
A * B;
A + B;
A - B;

A^2;
B^2;
```


RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
Go to file/function
Addins
Untitled1*
Source on Save

```
A <- 1;
B <- 2;

A/B;
A * B;
A + B;
A - B;

A^2;
B^2;
```

Figure 3-5. *Code in a script*

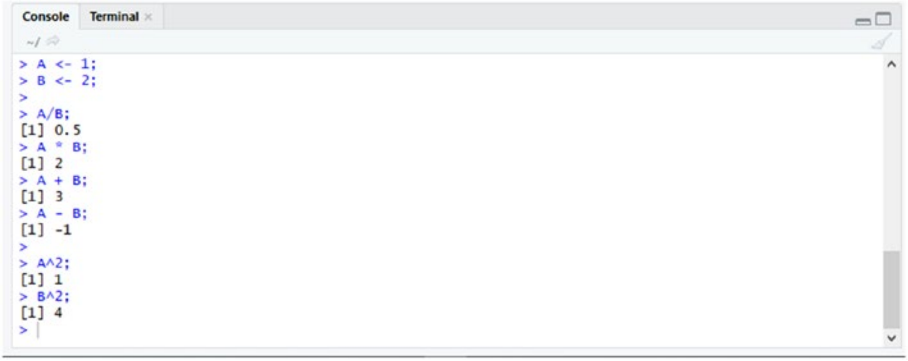
To run the R script, highlight the code in the code editor and click Run, as shown in Figure 3-6.



```
A <- 1;
B <- 2;

A/B;
A * B;
A + B;
A - B;

A^2;
B^2;
```

Figure 3-6. *Running the script*

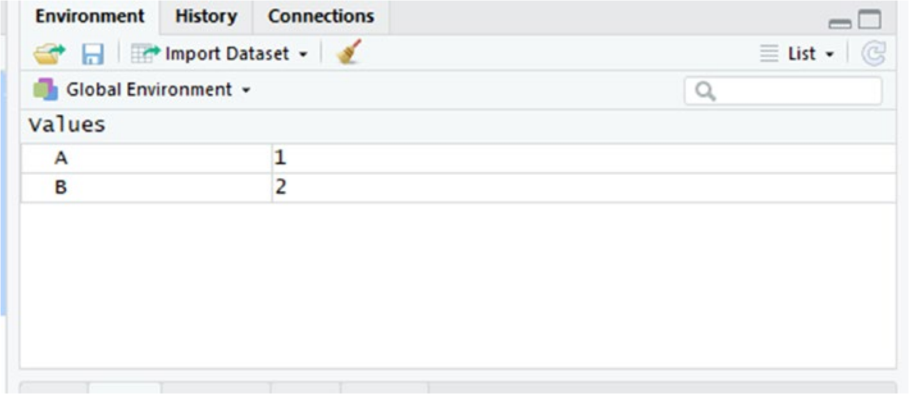
To view the results of the R script, look in the R console of RStudio, as shown in Figure 3-7.



```
> A <- 1;
> B <- 2;
>
> A/B;
[1] 0.5
> A * B;
[1] 2
> A + B;
[1] 3
> A - B;
[1] -1
>
> A^2;
[1] 1
> B^2;
[1] 4
>
```

Figure 3-7. *RStudio IDE console results*

You can also see that in the Environment tab, there are two variables, as shown in Figure 3-8.



Values

A	1
B	2

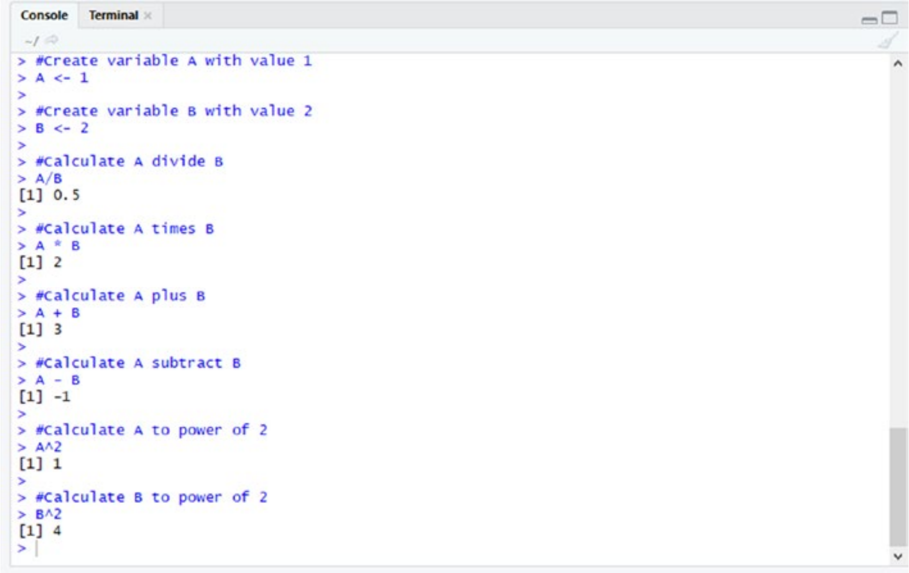
Figure 3-8. *RStudio IDE Environment tab*

Adding Comments to the Code

You can add comments to the code. Comments are text that will not be run by the R console. You can add in a comment by putting # in front of the text. The comment is for you to describe your code to let anyone read it more easily.

```
#Create variable A with value 1
A <- 1;

#Create variable B with value 2
B <- 2;

#Calculate A divide B
A/B;

#Calculate A times B
A * B;

#Calculate A plus B
A + B;

#Calculate A subtract B
A - B;

#Calculate A to power of 2
A^2;

#Calculate B to power of 2
B^2;
```

You can rerun the code and you should get the result shown in Figure 3-9.



```
> #Create variable A with value 1
> A <- 1
>
> #Create variable B with value 2
> B <- 2
>
> #Calculate A divide B
> A/B
[1] 0.5
>
> #Calculate A times B
> A * B
[1] 2
>
> #Calculate A plus B
> A + B
[1] 3
>
> #Calculate A subtract B
> A - B
[1] -1
>
> #Calculate A to power of 2
> A^2
[1] 1
>
> #Calculate B to power of 2
> B^2
[1] 4
> 
```

Figure 3-9. *RStudio IDE reruns the code with comments*

Variables

Let's look into the code and scripts you used previously. You actually created two variables, A and B, and assigned some values to the two variables.

```
A <- 1
B <- 2
```

In this code, A is a variable, and B is a variable also. <- means assign. A <- 1 means variable A is assigned a value of 1. 1 is a numeric type. B <- 2 means variable B is assigned a value of 2. 2 is a numeric type.

If you want to assign text or character values, you add quotations, like

```
A <- "Hello World"
```

Variable A is assigned a text value of "Hello World". Character and numeric are data types.

Data Types

Data types are the types or kind of information or data a variable is holding. A data type can be numeric and character.

For example,

```
A <- "abc"
B <- 1.2
```

In R, data types are automatically determined. Because of the quotations surrounding the values, variable A is of the character data type, while variable B is of the numeric data type.

R is also capable of storing other data types, as shown in Table 3-1.

Table 3-1. *Data Types*

Data Types	Values
Logical	TRUE FALSE
Numeric	12.3 2.55 1.0
Character	“a” “abc” “this is a bat”

For more information, please see www.tutorialspoint.com/r/r_data_types.htm.

You can also determine the data type of a variable by using the `class()` method. For example:

```
A <- "ABC";
print(class(A));
```

```
> A <- "ABC";
> print(class(A));
[1] "character"
```

```
A <- 123;
print(class(A));
```

```
> A <- 123;
> print(class(A));
[1] "numeric"
```

```
A <- TRUE;
print(class(A));
```

```
> A <- TRUE;
> print(class(A));
[1] "logical"
```

Why is the data type important? If you do a math calculation in R and one variable's data type is numeric and one variable's data type is non-numeric, you will get the following error:

```
> A <- 123;
> B <- "aaa";
> A + B;
Error in A + B : non-numeric argument to binary operator
```

You can also use `is.datatype()` to determine whether a variable is of a certain data type:

```
> A <- 123;
> print(is.numeric(A));
[1] TRUE
> print(is.character(A));
[1] FALSE
```

You can also use `as.datatype()` to convert between data types:

```
> A <- 12;
> B <- "56";
> A + B;
Error in A + B : non-numeric argument to binary operator
> B <- as.numeric(B);
> A + B;
[1] 68
```

`A <- 12` means that `A` is a numeric data type. `B <- "56"` means that `B` is a character data type. When `A` and `B` add together, you will get an error because you are adding a numeric data type to a character data type.

If you try to convert `B` to the numeric data type using `B <- as.numeric(B)`, you can add `A` and `B` together because `A` is a numeric data type and `B` is a numeric data type also.

Vectors

A vector is a basic data structure or R object for storing a set of values of the same data type. A vector is the most basic and common data structure in R. A vector is used when you want to store and modify a set of values. The data types can be logical, integer, double, and character. The integer data type is used to store number values without a decimal, and the double data type is used to store number values with a decimal. Vectors can be created using the c() function as follows:

```
variable = c(..., ..., ...)
> A <- c(1, 2, 3, 4, 5, 6);
> print(A);
[1] 1 2 3 4 5 6
```

You can check the data type of the vector using typeof() and class():

```
>typeof(A);
[1] "double"
> class(A);
[1] "numeric"
```

You can check the number of elements or values in a vector using the length() function:

```
> A <- c(1, 2, 3, 4, 5, 6);
> print(A);
[1] 1 2 3 4 5 6
> length(A);
[1] 6
```

You can also use the operator : to create a vector:

```
> A <- 1:8;
> print(A);
[1] 1 2 3 4 5 6 7 8
```

To retrieve the second element or value of a vector, use the [] brackets and put in the element number to retrieve:

```
> A <- 1:8;
> print(A);
[1] 1 2 3 4 5 6 7 8
```

```
> A[2];
[1] 2
```

You can also retrieve the elements in the vector using another vector, for example, to retrieve the second and fifth element:

```
> A <- 1:8;
> print(A);
[1] 1 2 3 4 5 6 7 8
> A[c(2, 5)];
[1] 2 5
```

To retrieve all elements except the second element, do this:

```
> A <- 1:8;
> print(A);
[1] 1 2 3 4 5 6 7 8
> A[-2];
[1] 1 3 4 5 6 7 8
```

You can also retrieve elements of a vector using a logical vector:

```
> A <- 1:8;
> print(A);
[1] 1 2 3 4 5 6 7 8
> A[c(FALSE, TRUE, FALSE, TRUE, FALSE, TRUE, FALSE, TRUE)];
[1] 2 4 6 8
```

You can also use more than or less than signs to retrieve elements:

```
> A <- 1:8;
> print(A);
[1] 1 2 3 4 5 6 7 8
> A[A > 5];
[1] 6 7 8
```

You can modify a vector as follows using assign, <-:

```
> A <- 1:8;
> print(A);
[1] 1 2 3 4 5 6 7 8
> A[3] <- 9;
> print(A);
[1] 1 2 9 4 5 6 7 8
```

Lists

A list is like a vector. It is an R object that can store a set of values or elements, but a list can store values of different data types. A list is also another common data structure in R. You use a list when you want to modify and store a set of values of different data types. A vector can only store values of the same data type. The syntax to create a list is as follows:

```
variable = list(..., ..., ...)
```

To create a list, do this:

```
> A <- list("a", "b", 1, 2);
> print(A);
[[1]]
[1] "a"

[[2]]
[1] "b"

[[3]]
[1] 1

[[4]]
[1] 2
```

To see the element data type or the data structure type of the list, you can use the `str()` and `typeof()` functions:

```
> str(A);
List of 4
 $ : chr "a"
 $ : chr "b"
 $ : num 1
 $ : num 2
>typeof(A);
[1] "list"
```

You can get the length or number of elements in the list by using the `length()` function:

```
> A <- list("a", "b", 1, 2);
> length(A);
[1] 4
```

You can retrieve the values in the list using an integer:

```
> A[1];
[[1]]
[1] "a"
```

You can retrieve the values in a list using an integer vector:

```
> A[c(1:2)];
[[1]]
[1] "a"

[[2]]
[1] "b"
```

You can retrieve the values using a negative integer:

```
> print(A);
[[1]]
[1] "a"

[[2]]
[1] "b"

[[3]]
[1] 1

[[4]]
[1] 2

> A[-1];
[[1]]
[1] "b"

[[2]]
[1] 1

[[3]]
[1] 2
```

You can also retrieve the values using a logical vector:

```
> print(A);
[[1]]
[1] "a"

[[2]]
[1] "b"

[[3]]
[1] 1
```

```
[[4]]
[1] 2

> A[c(TRUE, FALSE, FALSE, FALSE)];
[[1]]
[1] "a"
```

When you use only [ ] to retrieve a value, it will give you the sublist. If you want to get the content in the sublist, you need to use [[ ]].

```
> A[1];
[[1]]
[1] "a"

> A[[1]];
[1] "a"
```

You can modify a value or element in the list using

```
> print(A);
[[1]]
[1] "a"

[[2]]
[1] "b"

[[3]]
[1] 1

[[4]]
[1] 2

> A[[2]] <- "n";
> print(A);
[[1]]
[1] "a"
```

```
[[2]]
[1] "n"

[[3]]
[1] 1

[[4]]
[1] 2
```

To delete an element or value in a list:

```
> print(A);
[[1]]
[1] "a"

[[2]]
[1] "n"

[[3]]
[1] 1

[[4]]
[1] 2

> A[[2]] <- NULL;
> print(A);
[[1]]
[1] "a"

[[2]]
[1] 1

[[3]]
[1] 2
```

Matrix

A matrix is like a vector, but it has two dimensions. You usually use a matrix to modify and store values from a data set because a matrix has two dimensions. A matrix is good when you plan to do linear algebra types or mathematical operations. For a data set with different types, you need to use a data frame.

To create a matrix, you can use the following syntax:

```
variable <- matrix(vector, nrow=n, ncol=i)
> A <- matrix(c(1, 2, 3, 4, 6, 7, 8, 9, 0), nrow=3, ncol=3);
> print(A);
     [,1] [,2] [,3]
[1,]    1    4    8
[2,]    2    6    9
[3,]    3    7    0
```

You can use dimnames to rename the rows and columns:

```
> A <- matrix(c(1, 2, 3, 4, 6, 7, 8, 9, 0), nrow=3, ncol=3);
> print(A);
     [,1] [,2] [,3]
[1,]    1    4    8
[2,]    2    6    9
[3,]    3    7    0
> A <- matrix(c(1, 2, 3, 4, 6, 7, 8, 9, 0), nrow=3, ncol=3,
dimnames=list(c("X", "Y", "Z"), c("A", "S", "D")));
> print(A);
  A S D
X 1 4 8
Y 2 6 9
Z 3 7 0
```

You can check the dimension of the matrix using the `attributes()` function and whether a variable is a matrix using the `class()` function:

```
> class(A);
[1] "matrix"
> attributes(A);
$`dim`
[1] 3 3

$dimnames
$dimnames[[1]]
[1] "X" "Y" "Z"

$dimnames[[2]]
[1] "A" "S" "D"
```

You can get column names and row names using the `colnames()` and `rownames()` functions:

```
>colnames(A);
[1] "A" "S" "D"
>rownames(A);
[1] "X" "Y" "Z"
```

You can also create a matrix by using column binding and row binding functions:

```
> B <- cbind(c(1, 2, 3), c(4, 5, 6));
> print(B);
     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6
> C <- rbind(c(1, 2, 3), c(4, 5, 6));
> print(C);
```

```
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
```

To select the first row:

```
> print(A);
  A S D
X 1 4 8
Y 2 6 9
Z 3 7 0
> A[1,];
A S D
1 4 8
```

To select the first column:

```
> print(A);
  A S D
X 1 4 8
Y 2 6 9
Z 3 7 0
> A[,1];
X Y Z
1 2 3
```

To select all rows except the last row:

```
> print(A);
  A S D
X 1 4 8
Y 2 6 9
Z 3 7 0
> A[-3,];
```

```
  A S D
X 1 4 8
Y 2 6 9
```

To select the second row and second column:

```
> print(A);
  A S D
X 1 4 8
Y 2 6 9
Z 3 7 0
> A[2, 2];
[1] 6
```

Using a logical vector to select the first and last row:

```
> print(A);
  A S D
X 1 4 8
Y 2 6 9
Z 3 7 0
> A[c(TRUE, FALSE, FALSE),];
A S D
1 4 8
```

To select the elements and values based on more than and less than:

```
> print(A);
  A S D
X 1 4 8
Y 2 6 9
Z 3 7 0
> A[A>4];
[1] 6 7 8 9
```

To modify the second row and second column:

```
> print(A);
  A S D
X 1 4 8
Y 2 6 9
Z 3 7 0
> A[2, 2] <- 100;
> print(A);
  A   S D
X 1   4 8
Y 2 100 9
Z 3   7 0
```

To add a row, use the `rbind()` function:

```
> print(A);
  A   S D
X 1   4 8
Y 2 100 9
Z 3   7 0
> B <- rbind(A, c(1, 2, 3));
> print(B);
  A   S D
X 1   4 8
Y 2 100 9
Z 3   7 0
  1   2 3
```

To add a column, use the `cbind()` function:

```
> print(A);
  A   S D
X 1   4 8
```

```
Y 2 100 9
Z 3   7 0
> C <- cbind(A, c(1, 2, 3));
> print(C);
  A   S D
X 1   4 8 1
Y 2 100 9 2
Z 3   7 0 3
```

To transpose a matrix, use the `t()` function:

```
> print(A);
  A   S D
X 1   4 8
Y 2 100 9
Z 3   7 0
> A <- t(A);
> print(A);
  X   Y Z
A 1   2 3
S 4 100 7
D 8   9 0
```

Data Frame

A data frame is a special list or R object that is multidimensional and is usually used to store data read from an Excel or .csv file. A matrix can only store values of the same type, but a data frame can store values of different types. To declare a data frame, use the following syntax:

```
variable = data.frame(colName1 = c(..., ..., ...),
colName2 = c(..., ..., ...), ...)
```

```
> A <- data.frame(emp_id=c(1, 2, 3), names=c("John", "James",
"Mary"), salary=c(111.1, 222.2, 333.3));
> print(A);
emp_id names salary
1      1  John  111.1
2      2 James  222.2
3      3  Mary  333.3
```

You can use the `typeof()` and `class()` functions to check whether a variable is of the data frame type:

```
>typeof(A);
[1] "list"
> class(A);
[1] "data.frame"
```

To get the number of columns and rows, you can use the `ncol()` and `nrow()` functions:

```
>ncol(A);
[1] 3
>nrow(A);
[1] 3
```

To get the structure of the data frame, you can use the `str()` function:

```
> str(A);
'data.frame': 3 obs. of  3 variables:
 $ emp_id: num  1 2 3
 $ names : Factor w/ 3 levels "James","John",..: 2 1 3
 $ salary: num  111 222 333
```

You can also use the read.csv() function to read a .csv file as a data frame:

```
>myData<- read.csv(file="D:/data.csv", header=TRUE, sep=",");
>ncol(myData);
[1] 4
>nrow(myData);
[1] 100
> str(myData);
'data.frame': 100 obs. of  4 variables:
 $ x : num  2.216 -0.181 1.697 1.655 1.068 ...
 $ x2: num  4.77 4.1 2.33 2.46 1.05 ...
 $ x3: num  -4.87 6.98 3.92 0.75 3.35 ...
 $ y : int  0 1 0 0 1 1 1 1 1 1 ...
```

To select a column, use [], [[]], or $ to select:

```
> print(A);
emp_id names salary
1      1  John  111.1
2      2 James  222.2
3      3  Mary  333.3
> A["names"];
  names
1  John
2 James
3  Mary
>A$names;
[1] John  James Mary
Levels: James John Mary
> A[[2]];
[1] John  James Mary
Levels: James John Mary
```

To modify the first row and second column:

```
> print(A);
emp_id names salary
1      1  John  111.1
2      2 James  222.2
3      3  Mary  333.3
> A[1, 2] <- "James";
> print(A);
emp_id names salary
1      1 James  111.1
2      2 James  222.2
3      3  Mary  333.3
```

To add a row, use the `rbind()` function:

```
> print(A);
emp_id names salary
1      1 James  111.1
2      2 James  222.2
3      3  Mary  333.3
> B <- rbind(A, list(4, "John", 444.4));
> print(B);
emp_id names salary
1      1 James  111.1
2      2 James  222.2
3      3  Mary  333.3
4      4  John  444.4
```

To add a column, use the `cbind()` function:

```
> print(A);
emp_id names salary
1      1 James  111.1
```

```
2      2 James  222.2
3      3  Mary  333.3
> B <- cbind(A, state=c("NY", "NY", "NY"));
> print(B);
emp_id names salary state
1      1 James  111.1    NY
2      2 James  222.2    NY
3      3  Mary  333.3    NY
```

To delete a column:

```
> print(A);
emp_id names salary
1      1 James  111.1
2      2 James  222.2
3      3  Mary  333.3
>A$salary<- NULL;
> print(A);
emp_id names
1      1 James
2      2 James
3      3  Mary
```

Logical Statements

`if...else` statements are usually the logical fragments of your code in R. They give your program some intelligence and decision making by specifying the `if` rules:

```
if (Boolean expression) {
    #Codes to execute if Boolean expression is true
}else {
     #code to execute if Boolean expression is false }
```

Table 3-2 shows the Boolean operators that can be used in writing the Boolean expressions of the if... else statements.

Table 3-2. *Boolean Operators*

Boolean Operator	Definition
==	Equal to
>=	Greater than or equal to
<=	Lesser than or equal to
>	Greater than
<	Lesser than
!=	Not equal to

YouWe can put in more if...else statements using the else if to have more rules and intelligence in the program code:

```
if (Boolean expression 1) {
    #Codes to execute if Boolean expression 1 is true
} else if (Boolean expression2) {
#Codes to execute if Boolean expression 2 is true and Boolean
expression 1 is false
} else if(Boolean expression 3) {
#Codes to execute if Boolean expression 3 is true and Boolean
expression 1 and 2 are false
} else {
     #code to execute if all Boolean expressionsare false }
```

The following is an example of using else if:

```
> A <- c(1, 2);
> B <- sum(A); #1 + 2
>
```

```
> if(B >= 5) {
+   print("B is greater or equal to 5");
+ } else if(B >= 3) {
+   print("B is more than or equal to 3");
+ } else {
+   print("B is less than 3");
+ }
[1] "B is more than or equal to 3"
```

If B is more than or equal to 5, the R console will print "B is greater or equal to 5". Else if B is more than or equal to 3 but less than 5, the R console will print "B is more than or equal to 3". Otherwise, the R console will print "B is lesser than 5". The R console printed "B is more than or equal to 3" as B is 1 + 2 = 3.

Loops

Loops are used to repeat certain fragments of code. For example, if you want print the "This is R." message 100 times, it will be very tiresome to type `print("This is R. ");` 100 times. You can use loops to print the message 100 times more easily. Loops can usually be used to go through a set of vectors, lists, or data frames. In R, there are several loop options: both while loop, for loop, and repeat loop.

For Loop

Let's start with the syntax for a `for` loop:

```
for (value in vector) {
    #statements
}
```

Example:

```
> A <- c(1:5); #create a vector with values 1, 2, 3, 4, 5
>
> for(e in A) { #for each element and value in vector A
+   print(e); #print the element and value
+ }
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
```

In this example, you create vector A, with values 1 to 5. For each element in the vector, you print the element in the console. See Figure 3-10 for the logic.

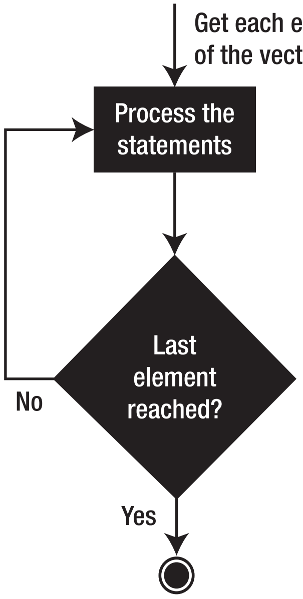


Figure 3-10. *For loop of R*

While Loop

You can also use while loop to loop until you meet a specific Boolean expression:

```
While (Boolean Expression) {
#Code to run or repeat until Bolean Expression is false
}
```

Example:

```
>i<- 1;
>
> while(i<= 10) {
+   print(i);
+   i<- i+1;
+ }
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
[1] 6
[1] 7
[1] 8
[1] 9
[1] 10
```

In this example, you use the while loop to print the text from 1 to 10. I is assigned a value of 1. While i is less than or equal to 10, the R console will print the value of i. After printing, I will be added to 1. It repeats printing until it is more than 10. See Figure 3-11 for the logic flow.

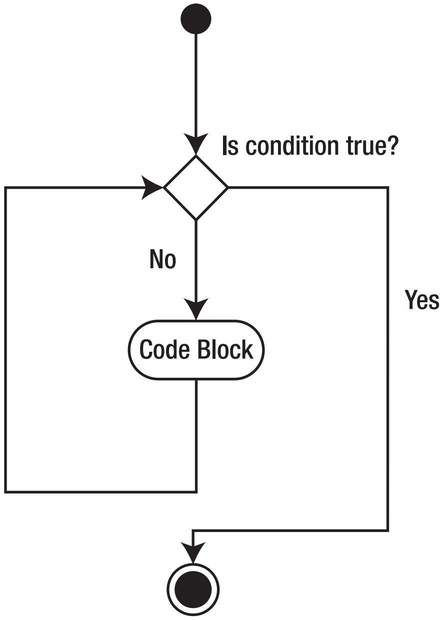


Figure 3-11. *While loop of R*

Break and Next Keywords

In loop statements, you can use the `break` keyword and the `next` keyword. The `break` keyword is to stop the iterations of the loop. The `next` keyword is to skip the current iteration of a loop.

Example for the `break` keyword:

```
> A <- c(1:10);
>
> for(e in A) {
+
+   if(e == 5) {
+     break;
+   }
+
+   print(e);
+ }
[1] 1
[1] 2
[1] 3
[1] 4
```

In this example, you create vector `A` with values from `1` to `10`. For each element in `A`, you print the element values. If the element value is `5`, you exit from the loop.

Example for the `next` keyword:

```
> A <- c(1:10);
>
> for(e in A) {
+
+   if(e == 5) {
+     next;
+   }
+
+   print(e);
+ }
[1] 1
[1] 2
[1] 3
[1] 4
[1] 6
[1] 7
[1] 8
[1] 9
[1] 10
```

In this example, you create vector A with values from 1 to 10. For each element in A, you print the element values. If the element value is 5, you skip the current iteration and go to the next iteration. Thus, you did not print the value 5. An example of when you want to use the break statement is when you use the loop statements to iterate through a set of values; when you find the values you want, you can break out from the loop.

Repeat Loop

The repeat loop repeats the code many times, and there is no Boolean expression to meet. To stop the loop, you must use the break keyword.

```
repeat {
#code to repeat
}
```

Example:

```
>i<- 1;
> repeat {
+   if(i> 10)
+     break;
+
+   print(i);
+   i<- i + 1;
+ }
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
[1] 6
[1] 7
```

```
[1] 8
[1] 9
[1] 10
```

In this example, you create variable `i` with a value of `1`. You repeat printing `i` and add `1` to `i` until `i` is more than `10`, and then you break out from the loop. If you forget or don't add a condition statement and the `break` keyword, you can end up in an infinite loop. An infinite loop is dangerous because it can consume your system resources and cause your program to keep on looping at the same place. The `for` loop is the preferable loop because the condition is defined in the first statement.

```
for(e in A) ....
```

The `while` loop is the next preferred loop. A condition statement is also stated in the first statement:

```
while(i<= ...) ...
```

If you forget to increment the `i`, `I <- I + 1;` , you will also have an infinite loop.

Functions

Functions help you organize your code and allow you to reuse code fragments whenever you need. To create a function, use the following syntax:

```
function_name<- function(arg1, arg2, ...) {
    # Codes fragments
    function_name = #value to return
}
```

Example:

```
> A <- c(1:5);
>
> productVect <- function(a) {
+
+   res <- 1;
+
+   for(e in a) {
+     res <- res * e;
+   }
+
+   productVect = res;
+ }
>
> print(productVect(A));
[1] 120
```

In this example, you create the productVect() function. This function is the same as prod() in R programming.

The productVect() function takes one argument. For every element in the argument (should be a vector), res will be equal to res times the element. After the loop is completed, the productVect() function will return the res value.

You can call the function using productVect(A). In the above code, you call the function using

```
A <- c(1:5);
print(productVect(A));
```

A is a vector with values from 1 to 5. You call the function by using productVect(A). The argument, a, in the function declaration is the formal argument. The argument, A, you passed to the function while calling the productVect() function is called the actual argument. When you call the function using `productVect(A)`, `a`, the formal argument, is assigned with `A`, the actual argument.

You can call `the productVect()` function a few times in your code:

```
>productVect<- function(a) {
+
+   res <- 1;
+
+   for(e in a) {
+     res <- res * e;
+   }
+
+   productVect = res;
+ }
>
> A <- c(1:5);
> print(productVect(A));
[1] 120
>
> B <- c(1:10);
> print(productVect(B));
[1] 3628800
```

You can also create default values for the argument in the function:

```
productVect<- function(a=c(1:5)) {

  res <- 1;

  for(e in a) {
    res <- res * e;
  }

productVect = res;
}
```

```
print(productVect());

A <- c(1:5);
print(productVect(A));

B <- c(1:10);
print(productVect(B));
```

When you call the function without argument, `print(productVect())`, you can still get the result:

```
>productVect<- function(a=c(1:5)) {
+
+   res <- 1;
+
+   for(e in a) {
+     res <- res * e;
+   }
+
+   productVect = res;
+ }
>
> print(productVect());
[1] 120
>
> A <- c(1:5);
> print(productVect(A));
[1] 120
>
> B <- c(1:10);
> print(productVect(B));
[1] 3628800
```

Instead of using productVect = res, you can also use the return() function to return the results:

```
>productVect<- function(a=c(1:5)) {
+
+   res <- 1;
+
+   for(e in a) {
+     res <- res * e;
+   }
+
+   return(res);
+ }
>
> print(productVect());
[1] 120
>
> A <- c(1:5);
> print(productVect(A));
[1] 120
>
> B <- c(1:10);
> print(productVect(B));
[1] 3628800
```

Create Your Own Calculator

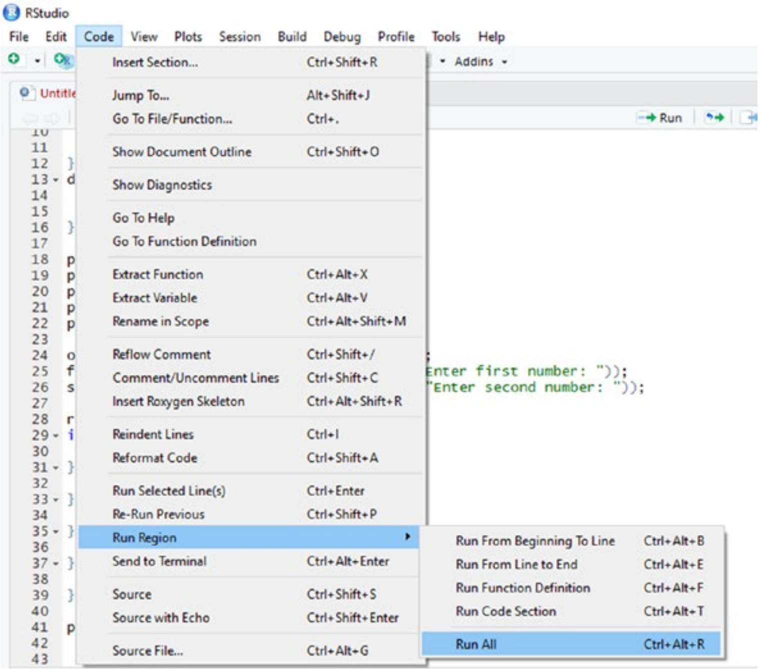
In this chapter, you learned many R programming syntaxes. You looked into variables, data types, matrix, vectors, lists, data frames, loops, and functions. In this section, you are going to put many things you just learned into a Calculator script:

```
add <- function(a, b) {
  res <- a + b;
  return(res);
}
subtract <- function(a, b) {
  res <- a - b;
  return(res);
}
product <- function(a, b) {
  res <- a * b;
  return(res);
}
division <- function(a, b) {
  res <- a / b;
  return(res);
}

print("Select your option: ");
print("1. Add");
print("2. Subtract");
print("3. Product");
print("4. Division");
```

```
opt <- as.integer(readline(prompt = "> "));
firstNum<- as.integer(readline(prompt="Enter first number: "));
secondNum<- as.integer(readline(prompt="Enter second number:
"));

res <- 0;
if(opt == 1) {
  res <- add(firstNum, secondNum);
} else if(opt == 2) {
  res <- subtract(firstNum, secondNum);
} else if(opt == 3) {
  res <- product(firstNum, secondNum);
} else if(opt == 4) {
  res <- division(firstNum, secondNum);
} else {
  print("Error. ");
}

print(res);
```

In this code, you create the add(), subtract(), product(), and division() functions. You then print the messages "Select your option:" and etc. You assign the opt variable with the user input. You get the user input using the readline() function, and you use as.integer() to convert the user input to the integer type. You use if statements to call the functions based on the user input. You then print the results.

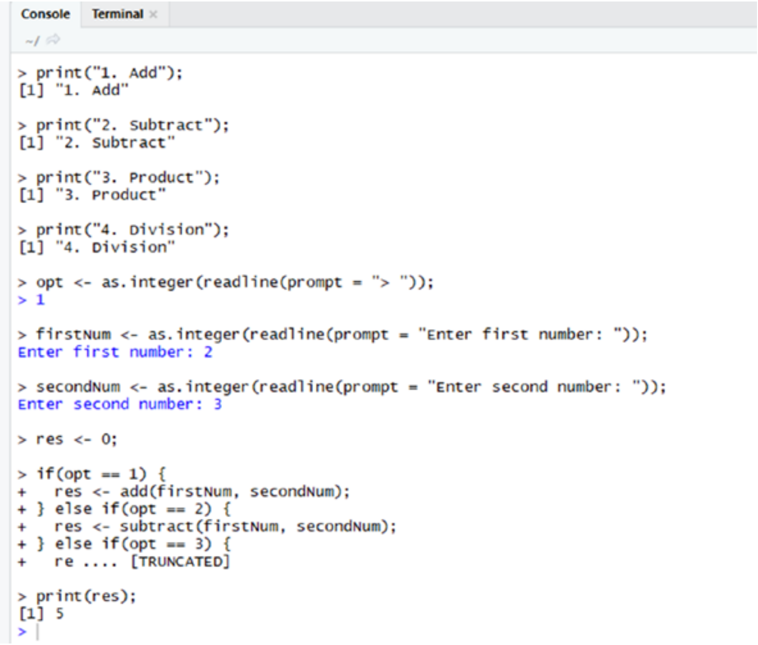
To run the code, go to Code ➤ Run Region ➤ Run All, as shown in Figure 3-12.


RStudio
File
Edit
Code
View
Plots
Session
Build
Debug
Profile
Tools
Help
Addins
Untitle
Run
Insert Section... Ctrl+Shift+R
Jump To... Alt+Shift+J
Go To File/Function... Ctrl+.
Show Document Outline Ctrl+Shift+O
Show Diagnostics
Go To Help
Go To Function Definition
Extract Function Ctrl+Alt+X
Extract Variable Ctrl+Alt+V
Rename in Scope Ctrl+Alt+Shift+M
Reflow Comment Ctrl+Shift+/
Comment/Uncomment Lines Ctrl+Shift+C
Insert Roxygen Skeleton Ctrl+Alt+Shift+R
Reindent Lines Ctrl+I
Reformat Code Ctrl+Shift+A
Run Selected Line(s) Ctrl+Enter
Re-Run Previous Ctrl+Shift+P
Run Region
Send to Terminal Ctrl+Alt+Enter
Source Ctrl+Shift+S
Source with Echo Ctrl+Shift+Enter
Source File... Ctrl+Alt+G
Run From Beginning To Line Ctrl+Alt+B
Run From Line to End Ctrl+Alt+E
Run Function Definition Ctrl+Alt+F
Run Code Section Ctrl+Alt+T
Run All Ctrl+Alt+R

```
;
Enter first number: "));
"Enter second number: "));
```

Figure 3-12. *Running the Calculator R script in the RStudio IDE*

The output is shown in Figure 3-13.



```
> print("1. Add");
[1] "1. Add"

> print("2. Subtract");
[1] "2. Subtract"

> print("3. Product");
[1] "3. Product"

> print("4. Division");
[1] "4. Division"

> opt <- as.integer(readline(prompt = "> "));
> 1

> firstNum <- as.integer(readline(prompt = "Enter first number: "));
Enter first number: 2

> secondNum <- as.integer(readline(prompt = "Enter second number: "));
Enter second number: 3

> res <- 0;

> if(opt == 1) {
+   res <- add(firstNum, secondNum);
+ } else if(opt == 2) {
+   res <- subtract(firstNum, secondNum);
+ } else if(opt == 3) {
+   re .... [TRUNCATED]

> print(res);
[1] 5
> |
```

Figure 3-13. *Running the results of the Calculator R script in the RStudio IDE*

Conclusion

In this chapter, you looked into R programming. You explored the R console and the RStudio code editor. The R console is for shorter code and the RStudio code editor is for longer R code or scripts.

You learned about variables. A variable is a container to store some values. A variable can have a name, which is called a variable name. Data types are the types or kind of information or data a variable is holding.

You also looked into vectors. A vector is a basic data structure or R object to store a set of values of the same data type. The data types can be logical, integer, double, character, and more. Vectors can be created using the `c()` function.

You also learned about lists. Lists are like vectors: they are R objects that can store a set of values or elements, but a list can store values of different data types.

You also learned about matrices. A matrix is an R object or two-dimensional data structure that is like vector, but has two dimensions.

You also learned about data frames. A data frame is a special list or R object that is multidimensional and is usually used to store data read from Excel or .csv files.

You also learned about conditional statements. `if...else` statements are usually the logical fragments of your code in R. They give your program some intelligence and decision making ability by specifying the `if` rules.

You also learned about loops. Loops are used to repeat certain fragments of code. For example, if you want print the "This is R." message 100 times, it will be very tiresome to type `print("This is R.");` 100 times. You can use loops to print the message 100 times more easily. R has both while loops, for loops, and repeat loops.

You also learned about functions. Functions help you organize your code and allow you to reuse code fragments whenever you need.

Finally, you created your own calculator based on what you learned.

References

Basic Data Types. (n.d.). Retrieved from `www.r-tutor.com/r-introduction/basic-data-types`.

Integrated development environment. (2018, August 22). Retrieved from `https://en.wikipedia.org/wiki/Integrated_development_environment`.

Kabacoff, R. (n.d.). Data Type Conversion. Retrieved from www.statmethods.net/management/typeconversion.html.

R Break and Next (With Syntax and Examples). (2018, April 11). Retrieved from www.datamentor.io/r-programming/break-next/.

R Data Frame (Create, Access, Modify, and Delete Data Frame in R). (2017, November 23). Retrieved from www.datamentor.io/r-programming/data-frame/.

T. (n.d.). R Data Types. Retrieved from www.tutorialspoint.com/r/r_data_types.htm.

T. (n.d.). R For Loop. Retrieved from www.tutorialspoint.com/r/r_for_loop.htm.

R For Loop (With Examples). (2018, April 11). Retrieved from www.datamentor.io/r-programming/for-loop/.

R Functions in Detail (With Examples). (2017, November 24). Retrieved from www.datamentor.io/r-programming/function/.

R if...else Statement (With Examples). (2018, April 11). Retrieved from www.datamentor.io/r-programming/if-else-statement/.

R Lists: Create, Append, and Modify List Components. (2017, November 23). Retrieved from www.datamentor.io/r-programming/list/.

R Matrix (Create and Modify Matrix, and Access Matrix Elements). (2017, November 24). Retrieved from www.datamentor.io/r-programming/matrix/.

R Program to Make a Simple Calculator. (2017, November 27). Retrieved from www.datamentor.io/r-programming/examples/simple-calculator/.

R Repeat Loop (With Syntax and Example). (2018, April 11). Retrieved from www.datamentor.io/r-programming/repeat-loop/.

R Return Value from Function. (2018, January 22). Retrieved from www.datamentor.io/r-programming/return-function/.

T. (n.d.). R Variables. Retrieved from www.tutorialspoint.com/r/r_variables.htm.

R Vector: Create, Modify, and Access Vector Elements. (2018, April 11). Retrieved from www.datamentor.io/r-programming/vector/.

T. (n.d.). R While Loop. Retrieved from www.tutorialspoint.com/r/r_while_loop.htm.

R While Loop (With Examples). (2018, April 11). Retrieved from www.datamentor.io/r-programming/while-loop/.

CHAPTER 4

Descriptive Statistics

Descriptive statistics is a set of math used to summarize data. Descriptive statistics can be distribution, central tendency, and dispersion of data. The distribution can be a normal distribution or binomial distribution. The central tendency can be mean, median, and mode. The dispersion or spreadness can be the range, interquartile range, variance, and standard deviation.

In this chapter, you will import a CSV file, Excel file, and SPSS file, and you will perform basic data processing. I will explain descriptive statistics, central tendency measurements, dispersion measurements, and distributions. You will look into how R programming can be used to calculate all these values, and how to test and see whether data is normally distributed.

What Is Descriptive Statistics?

Descriptive statistics summarizes the data and usually focuses on the distribution, the central tendency, and dispersion of the data. The distributions can be normal distribution, binomial distribution, and other distributions like Bernoulli distribution. Binomial distribution and normal distribution are the more popular and important distributions, especially normal distribution. When exploring data and many statistical tests, you will usually look for the normality of the data, which is how normal the data is or how likely it is that the data is normally distributed. The Central

E. G. M. Hui, *Learn R for Applied Statistics*, https://doi.org/10.1007/978-1-4842-4200-1_4

Limit Theorem states that the mean of a sample or subset of a distribution will be equal to the normal distribution mean when the sample size increases, regardless whether the sample is from a normal distribution. The central tendency, not the central limit theorem, is used to describe the data with respect to the center of the data. Central tendency can be the mean, median, and mode of the data. The dispersion describes the spread of the data, and dispersion can be the variance, standard deviation, and interquantile range.

Descriptive statistics summarizes the data set, lets us have a feel and understanding of the data and variables, and allows us to decide or determine whether we should use inferential statistics to identify the relationship between data sets or use regression analysis to identify the relationships between variables.

Reading Data Files

R programming allow you to import a data set, which can be comma-separated values (CSV) file, Excel file, tab-separated file, JSON file, or others. Reading data into the R console or R is important, since you must have some data before you can do statistical computing and understand the data.

Before you look into importing data into the R console, you must determine your workplace or work directory first. You should always set the current workspace directory to tell R the location of your current project folder. This allows for easier references to data files and scripts.

To print the current work directory, you use the `getwd()` function:

```
# get the current workspace location

print(getwd());
> print(getwd());
[1] "C:/Users/gohmi/Documents"
```

You can set the work directory using the setwd() function:

```
#set the current workspace location

setwd("D:/R");   #input your own file directory, for here
we use "D:/R"
> setwd("D:/R");
```

To get the new work directory location, you can use the getwd() function:

```
#get the new workspace
print(getwd());
> print(getwd());
[1] "D:/R"
```

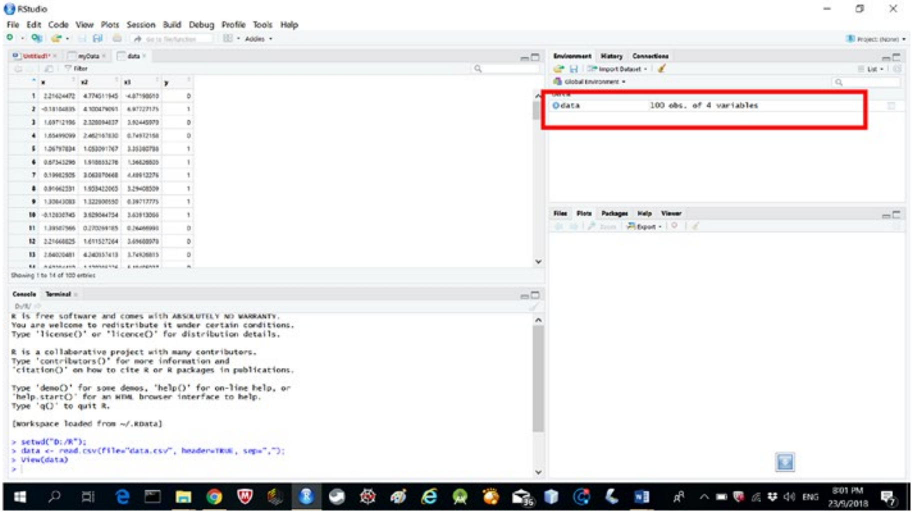
You can put the data.csv data set into D:/R folder.

Reading a CSV File

You can read the data CSV file into the R console using the read.csv() function:

```
> data <- read.csv(file="data.csv", header=TRUE, sep=",");
```

You can view the data by clicking the data in the Global Environment portion of the RStudio IDE. The data will be displayed in the table form. You can read the file using data.csv because you have set the work directory to D:/R, so file="data.csv" refers to D:/R/data.csv. See Figure 4-1.



	x	x2	x3	y
1	2.21624472	4.774511945	-4.87198010	0
2	-0.18104835	4.100479091	6.97727175	1
3	1.69712196	2.328094037	3.92445979	0
4	1.65499099	2.462167830	0.74972158	0
5	1.26797834	1.053091767	3.35380788	1
6	0.67343296	1.918853278	1.36628805	1
7	0.19982505	3.063870468	4.48912276	1
8	0.91662331	1.933422063	3.29408509	1
9	1.30843083	1.322906550	0.39717775	1
10	-0.12830745	3.629044754	3.63913066	1
11	1.39507966	0.270269185	0.26466990	0
12	2.21660825	1.611527264	3.69600978	0
13	2.84020481	4.240357613	3.74926815	0

Showing 1 to 14 of 100 entries

```
data                  100 obs. of 4 variables
```

```
R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

[Workspace loaded from ~/.RData]

> setwd("D:/R");
> data <- read.csv(file="data.csv", header=TRUE, sep=",");
> View(data)
>
```

Figure 4-1. *View data in table form in RStudio*

The data type of the data variable is data frame. You can determine whether the data type is data frame using the `class()` function:

```
> class(data);
[1] "data.frame"
```

The function you use to read a .csv file is

```
> data <- read.csv(file="data.csv", header=TRUE, sep=",");
```

`file` is the name or the data file path that you are going to read. `header` is a logical value to determine whether the names of the variables are in the first line. `sep` is the separator character, and `quote` is the quoting characters with "\"" for " and "\'" for '. You can add in the quote for double quotation as follows:

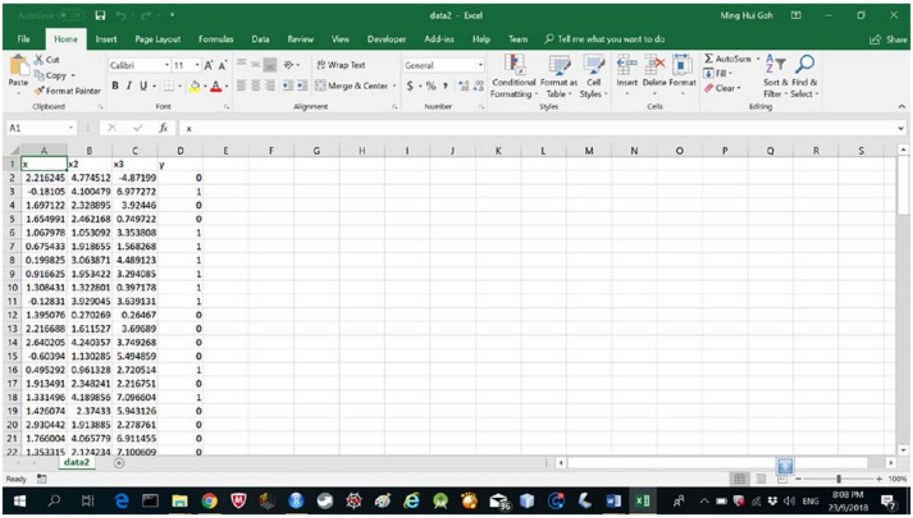
```
> data <- read.csv(file="data.csv", header=TRUE, sep=",");
> View(data);
```

Writing a CSV File

To write a CSV file, you can use the `write.csv()` function:

```
> write.csv(data, file="data2.csv", quote=TRUE, na="na", row.
names=FALSE);
```

The exported file is shown in Figure 4-2.



	A	B	C	D
1	x	x2	x3	y
2	2.216245	4.774512	-4.87199	0
3	-0.18105	4.100479	6.977272	1
4	1.697122	2.328895	3.92446	0
5	1.654991	2.462168	0.749722	0
6	1.067978	1.053092	3.353808	1
7	0.675433	1.918655	1.568268	1
8	0.199825	3.063871	4.489123	1
9	0.916625	1.953422	3.294085	1
10	1.308431	1.322801	0.397178	1
11	-0.12831	3.929045	3.639131	1
12	1.395076	0.270269	0.26467	0
13	2.216688	1.611527	3.69689	0
14	2.640205	4.240357	3.749268	0
15	-0.60394	1.130285	5.494859	0
16	0.495292	0.961328	2.720514	1
17	1.913491	2.348241	2.216751	0
18	1.331496	4.189856	7.096604	1
19	1.426074	2.37433	5.943126	0
20	2.930442	1.913885	2.278761	0
21	1.766004	4.065779	6.911455	0
22	1.353315	2.124234	7.100609	0

***Figure 4-2.** Exported csv file opened in Microsoft Excel*

In the `write.csv()` function you used previously to export the CSV file,

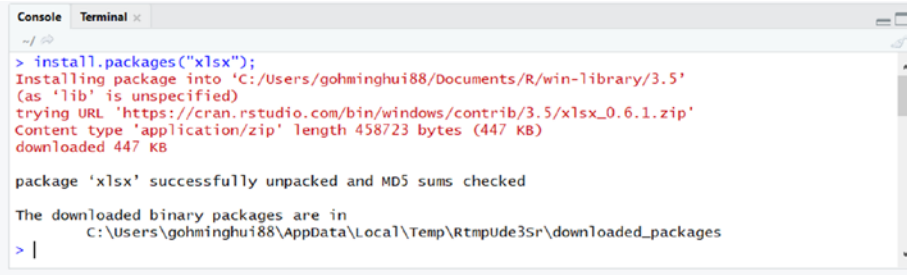
```
> write.csv(data, file="data2.csv", quote=TRUE, na="na", row.
names=FALSE);
```

`data` is the variable of the data frame type you would like to export, `file` is the file path or location to export, `quote` is a logical value to state whether to have quotations, `na` is the string value to use for missing values, and `row.names` is a logical value to indicate whether the row names should be written.

Reading an Excel File

The data set can also be in the Excel format or .xlsx format. To read an Excel file, you need to use the xlsx package. The xlsx package requires a Java runtime, so you must install it on your computer. To install the xlsx package, go to the R console and type the following, also shown in Figure 4-3:

```
> install.packages("xlsx");
```



```
> install.packages("xlsx");
Installing package into ‘C:/Users/gohminghui88/Documents/R/win-library/3.5’
(as ‘lib’ is unspecified)
trying URL 'https://cran.rstudio.com/bin/windows/contrib/3.5/xlsx_0.6.1.zip'
Content type 'application/zip' length 458723 bytes (447 KB)
downloaded 447 KB

package ‘xlsx’ successfully unpacked and MD5 sums checked

The downloaded binary packages are in
        C:\Users\gohminghui88\AppData\Local\Temp\RtmpUde3Sr\downloaded_packages
> |
```

Figure 4-3. *xlsx package installation in RStudio*

To use the xlsx package, use the `require()` function:

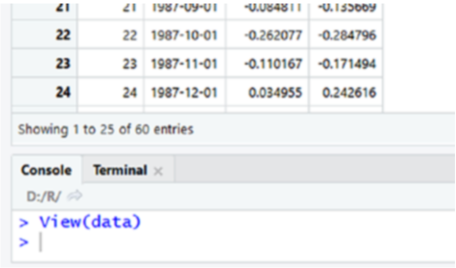
```
> require("xlsx");
Loading required package: xlsx
```

To read the Excel file, you can use the `read.xlsx()` function:

```
> data <- read.xlsx(file="data.xlsx", 1);
```

`file` is the location of the Excel file. `1` refers to sheet number 1.

To view the `data` variable, you can use the `View()` function or click the `data` variable in the Environment portion of RStudio, as shown in Figure 4-4.



21	21	1987-09-01	-0.084811	-0.135669
22	22	1987-10-01	-0.262077	-0.284796
23	23	1987-11-01	-0.110167	-0.171494
24	24	1987-12-01	0.034955	0.242616

Showing 1 to 25 of 60 entries

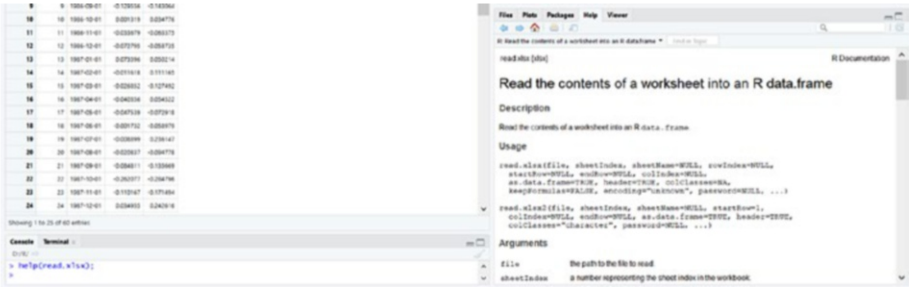
D:/R/

```
> View(data)
>
```

Figure 4-4. *View(data) in RStudio*

To look for the documentation of `read.xlsx()`, you can use the following code, as shown in Figure 4-5:

```
> help(read.xlsx);
```



9	9	1986-09-01	-0.129534	-0.143064
10	10	1986-10-01	0.001319	0.034776
11	11	1986-11-01	-0.033879	-0.060373
12	12	1986-12-01	-0.072795	-0.058735
13	13	1987-01-01	0.073396	0.050214
14	14	1987-02-01	-0.011618	0.111165
15	15	1987-03-01	-0.026032	-0.127492
16	16	1987-04-01	-0.042036	0.054322
17	17	1987-05-01	-0.047539	-0.072918
18	18	1987-06-01	-0.001732	-0.058979
19	19	1987-07-01	-0.038099	0.236147
20	20	1987-08-01	-0.020637	-0.094778
21	21	1987-09-01	-0.084811	-0.133669
22	22	1987-10-01	-0.262077	-0.284796
23	23	1987-11-01	-0.110167	-0.171494
24	24	1987-12-01	0.034955	0.242616

Showing 1 to 25 of 60 entries

D:/R/

```
> help(read.xlsx);
>
```

R: Read the contents of a worksheet into an R data.frame

read.xlsx {xlsx}   R Documentation

Read the contents of a worksheet into an R data.frame

Description

Read the contents of a worksheet into an R `data.frame`

Usage

```
read.xlsx(file, sheetIndex, sheetName=NULL, rowIndex=NULL,
  startRow=NULL, endRow=NULL, colIndex=NULL,
  as.data.frame=TRUE, header=TRUE, colClasses=NA,
  keepFormulas=FALSE, encoding="unknown", password=NULL, ...)

read.xlsx2(file, sheetIndex, sheetName=NULL, startRow=1,
  colIndex=NULL, endRow=NULL, as.data.frame=TRUE, header=TRUE,
  colClasses="character", password=NULL, ...)
```

Arguments

`file`	the path to the file to read
`sheetIndex`	a number representing the sheet index in the workbook.

Figure 4-5. *Documentation of read.xlsx()*

The data variable is of the data frame data type:

```
> class(data);
[1] "data.frame"
```

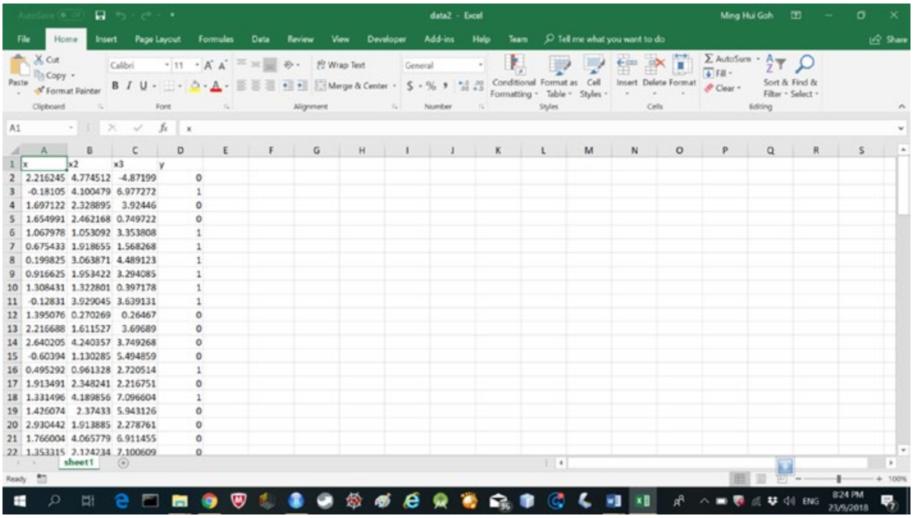
Writing an Excel File

To write a Excel file, you can use the `write.xlsx()` function:

```
> write.xlsx(data, file="data2.xlsx", sheetName="sheet1", col.
names=TRUE, row.names=FALSE);
```

data is the variable of data frame type to export to Excel file, file is the file location or path, sheetName is the sheet name, and col.names and row.names are logical values to state whether to export with column names or row names.

To view the documentation of the write.xlsx() function or any R function, you can use the help() function.



	A	B	C	D
1	x	x2	x3	y
2	2.216245	4.774512	-4.87199	0
3	-0.18105	4.100479	6.977272	1
4	1.697122	2.328895	3.92446	0
5	1.654991	2.462168	0.749722	0
6	1.067978	1.053092	3.353808	1
7	0.675433	1.918655	1.568268	1
8	0.199825	3.063871	4.489123	1
9	0.916625	1.953422	3.294085	1
10	1.308431	1.322801	0.397178	1
11	-0.12831	3.929045	3.639131	1
12	1.395076	0.270269	0.26467	0
13	2.216688	1.611527	3.69689	0
14	2.640205	4.240357	3.749268	0
15	-0.60394	1.130285	5.494859	0
16	0.495292	0.961328	2.720514	1
17	1.913491	2.348241	2.216751	0
18	1.331496	4.189856	7.096604	1
19	1.426074	2.37433	5.943126	0
20	2.930442	1.913885	2.278761	0
21	1.766004	4.065779	6.911455	0
22	1.353315	2.124234	7.100609	0

Figure 4-6. *Exported xlsx file*

Reading an SPSS File

To read an SPSS file, you need to use the foreign package. You can install the foreign package using the install.packages() function:

```
> install.packages("foreign");
Installing package into 'C:/Users/gohmi/Documents/R/win-
library/3.5'
(as 'lib' is unspecified)
trying URL 'https://cran.rstudio.com/bin/windows/contrib/3.5/
foreign_0.8-71.zip'
```

```
Content type 'application/zip' length 324526 bytes (316 KB)
downloaded 316 KB

package 'foreign' successfully unpacked and MD5 sums checked

The downloaded binary packages are in
    C:\Users\gohmi\AppData\Local\Temp\RtmpET26Hv\downloaded_
    packages
```

To use the foreign package, use the require() function:

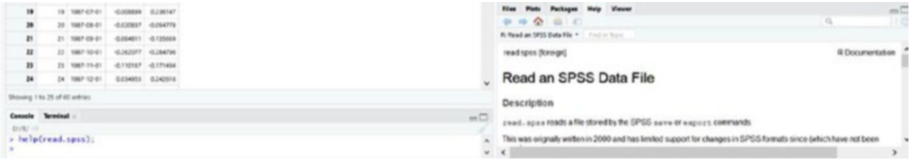
```
> require(foreign);
Loading required package: foreign
```

To read the SPSS file to a data frame type, you use the read.spss() function:

```
> data <- read.spss(file="data.spss", to.data.frame=TRUE);
```

file is the file path or location to read the SPSS file. to.data.frame is a logical value to state whether to read the SPSS file to a data frame type.

You can use the help() function to get the documentation of the read.spss() function, as shown in Figure 4-7.



```
> help(read.spss);
>
```

read.spss {foreign} R Documentation

Read an SPSS Data File

Description

read.spss reads a file stored by the SPSS save or export commands.

This was orignally written in 2000 and has limited support for changes in SPSS formats since (which have not been

Figure 4-7. *Documentation of read.spss()*

Writing an SPSS File

You can write the SPSS file using the write.foreign() function:

```
> write.foreign(data, "mydata.txt", "mydata.sps",
package="SPSS");
```

data is the variable to export to the SPSS data file, mydata.txt is the data in comma-delimited format, mydata.sps is the basic syntax file to read the data file into SPSS, and package determines the outputing or exporting to the SPSS format.

Reading a JSON File

JSON, or JavaScript Object Notation, is a very popular data interchange format that is easy for humans to write or read. A JSON file can be read by using the rjson package. To install the rjson package, you use the install.packages() function:

```
> install.packages("rjson");
Installing package into 'C:/Users/gohmi/Documents/R/win-
library/3.5'
(as 'lib' is unspecified)
trying URL 'https://cran.rstudio.com/bin/windows/contrib/3.5/
rjson_0.2.20.zip'
Content type 'application/zip' length 577826 bytes (564 KB)
downloaded 564 KB

package 'rjson' successfully unpacked and MD5 sums checked

The downloaded binary packages are in
    C:\Users\gohmi\AppData\Local\Temp\RtmpET26Hv\downloaded_
    packages
```

You can use the `require()` function to load the rjson package:

```
> require(rjson);
Loading required package: rjson
```

You can read JSON file using the `fromJSON()` function:

```
> data <- fromJSON(file="data.json");
```

To convert the data to a data frame type, you can use this code:

```
> data2 <- as.data.frame(data);
```

Basic Data Processing

After importing the data, you may need to do some simple data processing like selecting data, sorting data, filtering data, getting unique values, and removing missing values.

Selecting Data

You can select a few columns from the data using a vector:

```
> data;
              x          x2          x3 y
1    2.21624472 4.774511945 -4.87198610 0
2   -0.18104835 4.100479091  6.97727175 1
3    1.69712196 2.328894837  3.92445970 0
4    1.65499099 2.462167830  0.74972168 0
5    1.06797834 1.053091767  3.35380788 1
6    0.67543296 1.918655276  1.56826805 1
7    0.19982505 3.063870668  4.48912276 1
8    0.91662531 1.953422065  3.29408509 1
9    1.30843083 1.322800550  0.39717775 1
```

```
> data[, c("x", "x3")];
             x          x3
1    2.21624472 -4.87198610
2   -0.18104835  6.97727175
3    1.69712196  3.92445970
4    1.65499099  0.74972168
5    1.06797834  3.35380788
6    0.67543296  1.56826805
7    0.19982505  4.48912276
8    0.91662531  3.29408509
9    1.30843083  0.39717775
10  -0.12830745  3.63913066
11   1.39507566  0.26466993
12   2.21668825  3.69688978
13   2.64020481  3.74926815
14  -0.60394410  5.49485937
15   0.49529219  2.72051420
16   1.91349092  2.21675086
17   1.33149648  7.09660419
18   1.42607352  5.94312583
19   2.93044162  2.27876092
20   1.76600446  6.91145502
```

You can select a variable using the $ sign, as stated in a previous chapter:

```
> data$x3;
  [1] -4.87198610  6.97727175  3.92445970  0.74972168
  3.35380788  1.56826805  4.48912276
  [8]  3.29408509  0.39717775  3.63913066  0.26466993
  3.69688978  3.74926815  5.49485937
 [15]  2.72051420  2.21675086  7.09660419  5.94312583
 2.27876092  6.91145502  7.10060931
```

```
[22]  4.62416860  3.12633172  5.63667497  0.37028080
-0.11370995  2.27488863  0.43562110
[29]  0.46417756  3.44465146  4.14409404  3.78561287
1.86181693  8.10920939  0.87207093
[36]  0.55297962  4.26909037  1.01777720 12.85624593
4.79384178 -1.10646203  4.48442125
[43] -3.56106951  1.71246170  9.74478236  3.15799853
0.97278927  2.35670484  3.08804548
[50]  1.52772318 -5.02155267  5.64303286 -1.24622282
0.59864199  1.11359605  4.38302156
[57]  2.54163230  1.19193935 -0.57096625  7.49237946
6.88838713  5.37947543  0.72886289
[64]  2.20441458 -0.04416262  6.98441537  5.25116254
-0.15175665 -0.28652257  2.97419481
[71]  1.57454520  1.74898024  3.78645063  1.02328701
1.51030662 -1.46386054  5.65843587
[78]  1.71775236  2.77004224 -0.13805983  6.51654242
-0.80982223  6.55297343  3.65082015
[85]  5.55579403  3.03684846  6.85138858  2.09051225
2.79632315  5.21544351  2.63005598
[92] -0.04795488  8.79812379  0.92166450  2.97840367
1.89262722  2.23928744  2.46465216
[99] -0.18871437 -0.14146813
```

Sorting

You can sort the data in x3 in ascending order using

```
> data[order(data$x3), ];
              x          x2          x3 y
51   0.95505576 1.796297183 -5.02155267 1
1    2.21624472 4.774511945 -4.87198610 0
```

```
43   0.77142379  6.000455870 -3.56106951 0
76   1.43188297  2.758027392 -1.46386054 0
53   1.91399240  2.334445518 -1.24622282 0
41   0.69012311  5.527698064 -1.10646203 1
82   0.85499504  3.623854599 -0.80982223 1
59  -0.14645562 -0.629264301 -0.57096625 1
69   2.02211069  1.366558932 -0.28652257 0
99   0.44030641 -0.513103067 -0.18871437 1
68  -0.96007693  1.705067556 -0.15175665 0
100  2.34219633  3.314183829 -0.14146813 0
80   1.06881756  4.485035396 -0.13805983 1
```

You can sort the data in x3 in descending order using

```
> data[order(data$x3, decreasing=TRUE), ];
              x           x2          x3 y
39   0.44100266  0.041863046 12.85624593 1
45   0.45732586  3.706789430  9.74478236 1
93   2.48013836 -0.078239681  8.79812379 0
34   1.72284823  1.990169758  8.10920939 0
60   2.24529387  3.486295802  7.49237946 0
21   1.35331484  2.124233637  7.10060931 0
17   1.33149648  4.189856264  7.09660419 1
66   1.56980022  0.379400632  6.98441537 1
2   -0.18104835  4.100479091  6.97727175 1
20   1.76600446  4.065779075  6.91145502 0
61  -1.07591095  2.522409241  6.88838713 0
87   0.29924083  2.892653658  6.85138858 1
83   1.20403393  3.934698897  6.55297343 1
81  -0.09110412  0.745501714  6.51654242 1
```

You can sort the data by multiple variables:

```
> data[order(data$x3, data$x2), ];
             x           x2          x3 y
51  0.95505576  1.796297183 -5.02155267 1
1   2.21624472  4.774511945 -4.87198610 0
43  0.77142379  6.000455870 -3.56106951 0
76  1.43188297  2.758027392 -1.46386054 0
53  1.91399240  2.334445518 -1.24622282 0
41  0.69012311  5.527698064 -1.10646203 1
82  0.85499504  3.623854599 -0.80982223 1
59 -0.14645562 -0.629264301 -0.57096625 1
69  2.02211069  1.366558932 -0.28652257 0
99  0.44030641 -0.513103067 -0.18871437 1
68 -0.96007693  1.705067556 -0.15175665 0
```

Filtering

You can filter the data using Boolean expressions and statements:

```
> data[data$x > 0, ];
            x          x2          x3 y
1  2.21624472 4.774511945 -4.87198610 0
3  1.69712196 2.328894837  3.92445970 0
4  1.65499099 2.462167830  0.74972168 0
5  1.06797834 1.053091767  3.35380788 1
6  0.67543296 1.918655276  1.56826805 1
7  0.19982505 3.063870668  4.48912276 1
8  0.91662531 1.953422065  3.29408509 1
9  1.30843083 1.322800550  0.39717775 1
11 1.39507566 0.270269185  0.26466993 0
12 2.21668825 1.611527264  3.69688978 0
13 2.64020481 4.240357413  3.74926815 0
```

You can also filter the data with more complex expressions:

```
> data[data$x > 0 & data$x < 1, ];
            x          x2        x3 y
6  0.67543296  1.918655276 1.5682681 1
7  0.19982505  3.063870668 4.4891228 1
8  0.91662531  1.953422065 3.2940851 1
15 0.49529219  0.961328129 2.7205142 1
22 0.96711785 -1.656336500 4.6241686 0
24 0.22931395  4.513268166 5.6366750 1
29 0.92245691  3.066787671 0.4641776 1
32 0.26530020  2.407519006 3.7856129 1
35 0.74641160  0.007386508 0.8720709 1
```

Removing Missing Values

You can remove rows with NA values in any variables:

```
> na.omit(data);
              x          x2          x3 y
1    2.21624472  4.774511945 -4.87198610 0
2   -0.18104835  4.100479091  6.97727175 1
3    1.69712196  2.328894837  3.92445970 0
4    1.65499099  2.462167830  0.74972168 0
5    1.06797834  1.053091767  3.35380788 1
6    0.67543296  1.918655276  1.56826805 1
7    0.19982505  3.063870668  4.48912276 1
```

Removing Duplicates

You can remove duplicates based on the x variable using

```
> data[!duplicated(data$x), ];
             x          x2          x3 y
1   2.21624472 4.774511945 -4.87198610 0
2  -0.18104835 4.100479091  6.97727175 1
3   1.69712196 2.328894837  3.92445970 0
4   1.65499099 2.462167830  0.74972168 0
5   1.06797834 1.053091767  3.35380788 1
6   0.67543296 1.918655276  1.56826805 1
7   0.19982505 3.063870668  4.48912276 1
8   0.91662531 1.953422065  3.29408509 1
9   1.30843083 1.322800550  0.39717775 1
10 -0.12830745 3.929044754  3.63913066 1
11  1.39507566 0.270269185  0.26466993 0
12  2.21668825 1.611527264  3.69688978 0
13  2.64020481 4.240357413  3.74926815 0
14 -0.60394410 1.130285226  5.49485937 0
15  0.49529219 0.961328129  2.72051420 1
```

For more advanced duplicate removal for text data, you may use the Levenshtein similarity algorithm and others. It is beyond the scope of this book to cover the similarity duplicate removal. For each row of data, you can get the Levenshtein similarity between the current row and all other rows of the data and return the results with a similarity below a stated value. You can then check whether they are really similar and duplicated, and remove them as needed.

Some Basic Statistics Terms

The following are some of the most popular terms used in statistics.

- **Population**: Population is the total set of observations. A population is the whole, and it comprises every member or observation.
- **Sample**: A sample is a portion of a population. A sample can be extracted from the population using random sampling techniques and others.
- **Observations**: An observation is something you measure or count during a study or experiment. An observation usually means the row in data.
- **Variables**: A variable is a characteristics or quantity that can be counted and can be called a data item. Variables usually refer to the column.

Types of Data

Data can be numeric data or categorical data.

- Numeric data can have discrete data or continuous data. Discrete variables usually take integer values. Discrete variables have steps. Continuous variables can be any real number values.
- Categorical data are categories or non-numeric data. Categorical data can have nominal variables that have unordered categories. Ordinal variables can have ordered categories.

Mode, Median, Mean

Mean, median, and mode are the most common measures for central tendency. Central tendency is a measure that best summarizes the data and is a measure that is related to the center of the data set.

Mode

Mode is a value in data that has the highest frequency and is useful when the differences are non-numeric and seldom occur.

To get the mode in R, you start with data:

```
> A <- c(1, 2, 3, 4, 5, 5, 5, 6, 7, 8);
```

To get mode in a vector, you create a frequency table:

```
> y <- table(A);
> y;
A
1 2 3 4 5 6 7 8
1 1 1 1 3 1 1 1
```

You want to get the highest frequency, so you use the following to get the mode:

```
> names(y)[which(y==max(y))];
[1] "5"
```

Let's get the mode for a data set. First, you have the data:

```
> y <- table(data$x);
> print(y);
```

```
  -1.10380693563466   -1.07591095084206   -1.03307520058884   0.960076927640974
                  1                   1                   1                   1
 -0.681376413558189   -0.60394409792254  -0.576961817793324  -0.531319073429502
                  1                   1                   1                   1
 -0.393748947329461   -0.36317756860052  -0.181048347962983  -0.146455623720039
                  1                   1                   1                   1
 -0.128307452033018  -0.123388723621058  -0.101546995549492 -0.0911041245433437
                  1                   1                   1                   1
  -0.04988355797068   0.072199258362634   0.194009711645882   0.199825048055373
                  1                   1                   1                   1
  0.229313954917265   0.265300204243802   0.278608090170949   0.299240831235205
                  1                   1                   1                   1
  0.375685875108926   0.375886351624522   0.431165605395961   0.440306411318609
                  1                   1                   1                   1
  0.441002661190442   0.457325864971707   0.495292193747488   0.536273330307632
                  1                   1                   1                   1
  0.675432957349429   0.690123106678726   0.736279647441046   0.746411596302334
                  1                   1                   1                   1
  0.760336369473898   0.771423785187196   0.817889767078534   0.854995040060911
```

To get the mode in the data set, you must get the values of the highest frequency. Since all values are of frequency 1, your mode is

```
> names(y) [which(y==max(y))];
  [1] "-1.10380693563466"   "-1.07591095084206"
  "-1.03307520058884"   "-0.960076927640974"
  [5] "-0.681376413558189"  "-0.60394409792254"
  "-0.576961817793324"  "-0.531319073429502"
  [9] "-0.393748947329461"  "-0.363177568600528"
  "-0.181048347962983"  "-0.146455623720039"
 [13] "-0.128307452033018"  "-0.123388723621058"
 "-0.101546995549492"  "-0.0911041245433437"
 [17] "-0.04988355797068"   "0.072199258362634"
 "0.194009711645882"   "0.199825048055373"
 [21] "0.229313954917265"   "0.265300204243802"
 "0.278608090170949"   "0.299240831235205"
 [25] "0.375685875108926"   "0.375886351624522"
 "0.431165605395961"   "0.440306411318609"
 [29] "0.441002661190442"   "0.457325864971707"
 "0.495292193747488"   "0.536273330307632"
 [33] "0.675432957349429"   "0.690123106678726"
 "0.736279647441046"   "0.746411596302334"
 [37] "0.760336369473898"   "0.771423785187196"
 "0.817889767078534"   "0.854995040060911"
 [41] "0.913440165921528"   "0.916625307876996"
 "0.922456906688953"   "0.955055760366265"
 [45] "0.967117849282085"   "0.971724620868089"
 "0.981039549396608"   "1.00606498603125"
```

```
[49] "1.06718039898877"    "1.06797833714856"
"1.06881756289549"    "1.08718603297871"
[53] "1.10356948706272"    "1.11925179073556"
"1.2040339292453"     "1.25502822371368"
[57] "1.28679297901113"    "1.30790288496801"
"1.30843083022333"    "1.33149647893459"
[61] "1.35331484423307"    "1.39507565734447"
"1.40212267908162"    "1.42607352487406"
[65] "1.43188296915663"    "1.52346751351768"
"1.55472504319324"    "1.56980021618683"
[69] "1.65307570138401"    "1.65499098668696"
"1.69712196084468"    "1.72284823029622"
[73] "1.72661763645561"    "1.75109907720939"
"1.76600446229059"    "1.88048596823981"
[77] "1.91349091681916"    "1.91399240309242"
"1.92524009507999"    "2.02211068696315"
[81] "2.03853797041063"    "2.04518626417935"
"2.12978282462908"    "2.21624471542922"
[85] "2.21668824966669"    "2.24529386752873"
"2.33237405387913"    "2.34219633417921"
[89] "2.48013835527471"    "2.50480340549357"
"2.62246164503412"    "2.62566065094827"
[93] "2.64020480708033"    "2.68169620257743"
"2.72668117335907"    "2.77811502414916"
[97] "2.93044161969506"    "3.0710000206091"
"3.12515738750874"    "3.16181857813796"
```

Median

The median is the middle or midpoint of the data and is also the 50 percentile of the data. The median is affected by the outliers and skewness of the data. The median can be a better measurement for centrality than the mean if the data is skewed. The mean is the average, which is liable to be influenced by outliers, so median is a better measure when the data is skewed.

In R, to get the median, you use the `median()` function:

```
> A <- c(1, 2, 3, 4, 5, 5, 5, 6, 7, 8);
> median(A);
[1] 5
```

To get the median for dataset:

```
> median(data$x2);
[1] 2.380852
```

Mean

The mean is the average of the data. It is the sum of all data divided by the number of data points. The mean works best if the data is distributed in a normal distribution or distributed evenly. The mean represents the expected value if the distribution is random.

$$\overline{x} = \frac{1}{N}\sum_{i=1}^{N} x_i = \frac{x_1 + x_2 + \ldots + x_N}{N}$$

In R, to get the mean, you can use the `mean()` function:

```
> A <- c(1, 2, 3, 4, 5, 5, 5, 6, 7, 8);
> mean(A);
[1] 4.6
```

To get the mean of a data set:

```
> mean(data$x2);
[1] 2.46451
```

Interquartile Range, Variance, Standard Deviation

Measures of variability are the measures of the spread of the data. Measures of variability can be range, interquartile range, variance, standard deviation, and more.

Range

The range is the difference between the largest and smallest points in the data.

To find the range in R, you use the *range()* function:

```
> A <- c(1, 2, 3, 4, 5, 5, 5, 6, 7, 8);
> range(A);
[1] 1 8
```

To get the difference between the max and the min, you can use

```
> A <- c(1, 2, 3, 4, 5, 5, 5, 6, 7, 8);
> res <- range(A);
> diff(res);
[1] 7
```

You can use the `min()` and `max()` functions to find the range also:

```
> A <- c(1, 2, 3, 4, 5, 5, 5, 6, 7, 8);
> min(A);
[1] 1
> max(A);
```

```
[1] 8
> max(A) - min(A);
[1] 7
```

To get the range for a data set:

```
> res <- range(data$x2);
> diff(res);
[1] 10.65222
> res <- range(data$x2, na.rm=TRUE);
> diff(res);
[1] 10.65222
```

`na.rm` is a logical value to state whether to remove NA values.

Interquartile Range

The interquartile range is the measure of the difference between the 75 percentile or third quartile and the 25 percentile or first quartile.

To get the interquartile range, you can use the `IQR()` function:

```
> A <- c(1, 2, 3, 4, 5, 5, 5, 6, 7, 8);
> IQR(A);
[1] 2.5
```

You can get the quartiles by using the `quantile()` function:

```
> quantile(A);
  0%  25%  50%  75% 100%
1.00 3.25 5.00 5.75 8.00
```

You can get the 25 and 75 percentiles:

```
> quantile(A, 0.25);
 25%
3.25
```

```
> quantile(A, 0.75);
 75%
5.75
```

You can get the interquartile range and quartiles for data set using

```
> quantile(data$x2);
       0%        25%        50%        75%
-2.298551  1.274672  2.380852  3.750422
     100%
 8.353669
> IQR(data$x2);
[1] 2.47575
> IQR(data$x2, na.rm=TRUE);
[1] 2.47575
> help(IQR);
```

The `IQR()` and `quantile()` functions can have NA values removed using `na.rm = TRUE`.

Range measures the maximum and minimum data values, and the interquartile range measures where the majority value is.

Variance

The variance is the average of squared differences from the mean, and it is used to measure the spreadness of the data.

The variance of a population is

$$\sigma^2 = \frac{\Sigma\left(X - u\right)^2}{N}$$

where μ is the mean of the population and `N` is the number of data points.

The variance of a sample is

$$S^2 = \frac{\Sigma\left(X - \bar{X}\right)^2}{n-1}$$

where `n` is the number of data points. You use `n-1` for sample variance and sample standard deviation because of Bessel's correction to partially correct the bias on the estimation of the population variance and standard deviation.

To find the population variance, you use the `var()` function and `(N - 1)/N`:

```
> A <- c(1, 2, 3, 4, 5, 5, 5, 6, 7, 8);
> N <- length(A);
> N;
[1] 10
> var(A) * (N - 1) / N;
[1] 4.24
```

To get the sample variance, you can use the `var()` function:

```
> A <- c(1, 2, 3, 4, 5, 5, 5, 6, 7, 8);
> var(A);
[1] 4.711111
```

To get the population variance of a data set:

```
> N <- nrow(data);
> N;
[1] 100
> var(data$x) * (N - 1) / N;
[1] 1.062619
```

To get the sample variance of a data set:

```
> var(data$x);
[1] 1.073352
```

Standard Deviation

The standard deviation is the square root of a variance and it measures the spread of the data. Variances get bigger when there are more variations and get smaller when there are lesser variations, because the variance is a squared result. With standard deviation, the variance is the square root, so it is easier to picture and apply. The variance is the squared result, so the unit is different from the data. The standard deviation has the same unit as the data.

The population standard deviation is

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(x_i - \overline{x}\right)^2}$$

For the sample standard deviation, you use `n - 1` for sample variance and sample standard deviation because of Bessel's correction to partially correct the bias on the estimation of population variance and standard deviation.

$$s = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}\left(x_i - \overline{x}\right)^2}$$

To find the population standard deviation:

```
> A <- c(1, 2, 3, 4, 5, 5, 5, 6, 7, 8);
> N <- length(A);
> variance <- var(A) * (N - 1) / N;
> sqrt(variance);
[1] 2.059126
```

To find the population standard deviation for a data set:

```
> N <- nrow(data);
> variance <- var(data$x2) * (N - 1) / N;
> sqrt(variance);
[1] 1.908994
```

To find the sample standard deviation, you use the `sd()` function:

```
> A <- c(1, 2, 3, 4, 5, 5, 5, 6, 7, 8);
> sd(A);
[1] 2.170509
```

To find the sample standard deviation of a data set, you use the `sd()` function:

```
> sd(data$x2);
[1] 1.918611
```

Normal Distribution

Normal distribution is one of the more important theories because nearly all statistical tests require the data to be distributed normally. It describes how data looks when plotted. Normal distribution is also called the bell curve, shown in Figure 4-8.

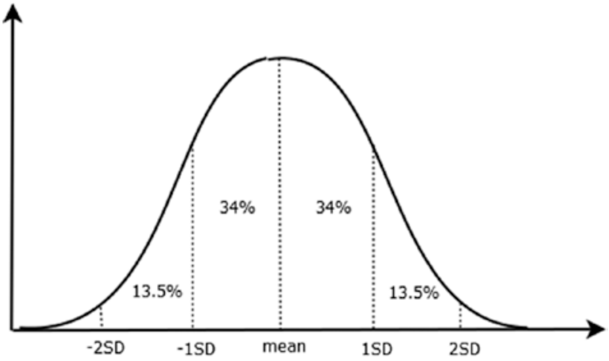


Figure 4-8. *Normal Distribution or Bell Curve*

You can plot a distribution in R using the hist() function:

```
> hist(data$x, breaks=15);
```

In R, breaks shows the number of bars in a histogram. bins is the class interval to sort the data. For example, in grade data, the bins can be 83-88, 89-94, 95-100, and each bin size should be the same. See Figure 4-9.

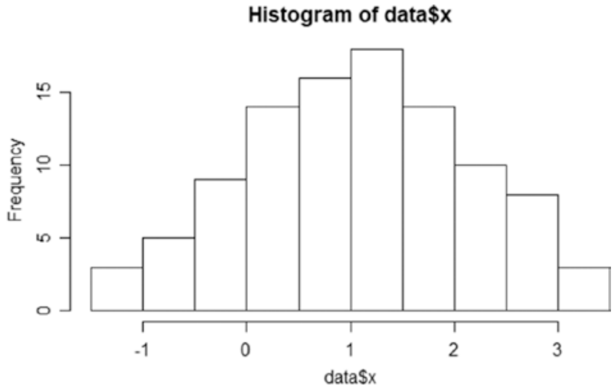


***Figure 4-9.** Histogram of x*

To see whether data is normally distributed, you can use the qqnorm() and qqline() functions:

```
> qqnorm(data$x);
> qqline(data$x);
```

In the Q-Q plot shown in Figure 4-10, if the points do not deviate away from the line, the data is normally distributed.

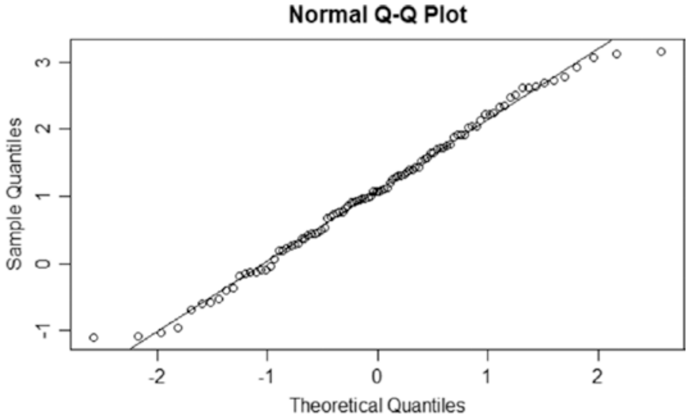


Figure 4-10. *QQPlot of x*

You can also use a Shapiro Test to test whether the data is normally distributed:

```
> shapiro.test(data$x);

    Shapiro-Wilk normality test

data:  data$x
W = 0.98698, p-value = 0.4363
```

If the `p-value` is more than 0.05, you can conclude that the data does not deviate from normal distribution.

In R, to generate random numbers from the normal distribution, you use `rnorm()` function:

```
> set.seed(123);
> A <- rnorm(50, 3, 0.5);
> hist(A, breaks=15);
```

3 is the mean and 0.5 is the standard deviation. In the above functions, you generated 50 random values from normal distribution. See Figure 4-11.

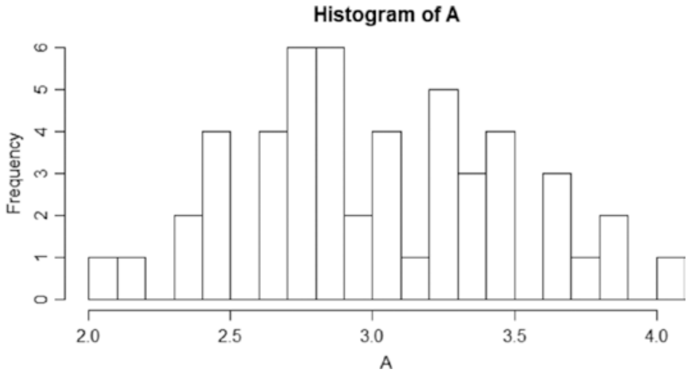


Figure 4-11. *Histogram of A*

In R, to calculate the cumulative distribution function (CDF), F(x) = P(X <= x) where X is normal, you use the `pnorm()` function:

```
> pnorm(1.9, 3, 0.5);
[1] 0.01390345
```

The above is a direct lookup for the probability P(X < 1.9) where X is a normal distribution with mean of 3 and standard deviation of 0.5. If you want P(X > 1.9), you use 1 - pnorm(1.9, 3, 0.5).

In R, if you want to calculate the inverse CDF and lookup for the p-th quantile of the normal distribution, you use

```
> qnorm(0.95, 3, 0.5);
[1] 3.822427
```

This code looks for the 95 percentile of the normal distribution with a standard deviation of 0.5 and a mean of 3. The value returned is an x value, not a probability.

Modality

The modality of a distribution can be seen by the number of peaks when we plot the histogram (see Figure 4-12):

```
> hist(data$x, breaks=15);
```

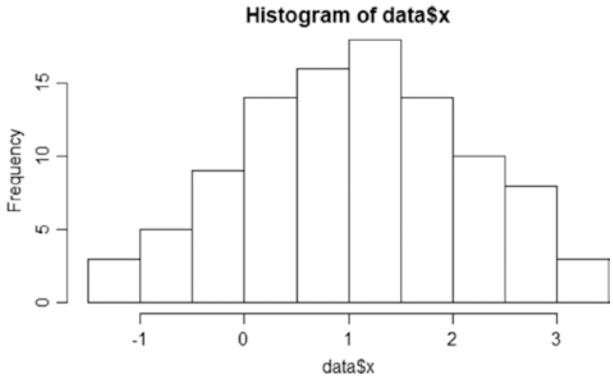


Figure 4-12. *Histogram of x*

Figure 4-13 shows the modality type. The distribution of the market variable can be argued as a unimodal type. The figure shows the unimodal, bimodal, and multimodal types.

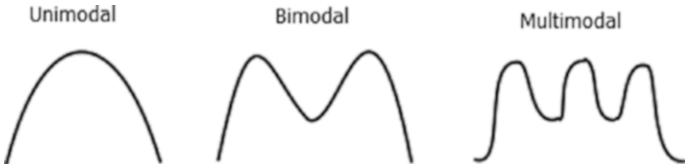


Figure 4-13. *Modality Type of Histogram or distribution*

Skewness

Skewness is a measure of how symmetric a distribution is and how much the distribution is different from the normal distribution. Figure 4-14 shows the types of skewness.

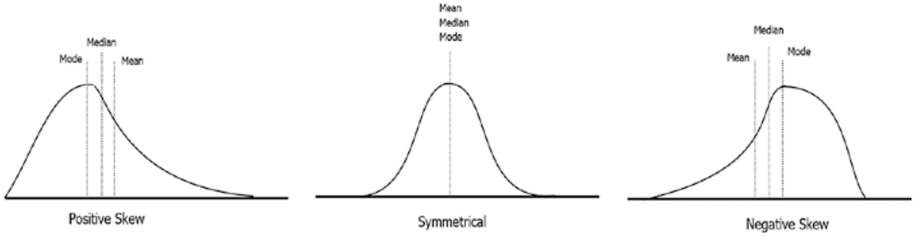


Figure 4-14. *Types of Skewness*

Negative skew is also known as left skewed, and positive skew is also known as right skewed. The histogram from the previous section has a negative skew.

The Kurtosis measure is used to see whether a dataset is heavy tailed or light tailed. High kurtosis means heavy tailed, so there are more outliers in the data. See Figure 4-15.

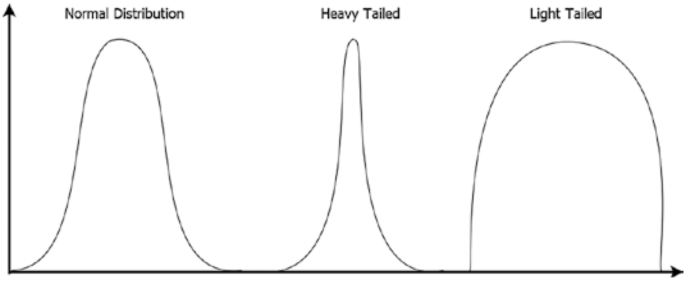


Figure 4-15. *Kurtosis Type or heavy tailed or light tailed distribution*

To find the kurtosis and skewness in R, you must install the `moments` package:

```
> install.packages("moments");
Installing package into 'C:/Users/gohmi/Documents/R/win-
library/3.5'
(as 'lib' is unspecified)
trying URL 'https://cran.rstudio.com/bin/windows/contrib/3.5/
moments_0.14.zip'
```

```
Content type 'application/zip' length 55827 bytes (54 KB)
downloaded 54 KB

package 'moments' successfully unpacked and MD5 sums checked

The downloaded binary packages are in
    C:\Users\gohmi\AppData\Local\Temp\RtmpET26Hv\downloaded_
    packages
```

You also need the `moments` package:

```
> require(moments);
Loading required package: moments
```

You then use the `skewness()` and `kurtosis()` functions to get the skewness and kurtosis:

```
> skewness(data$x);
[1] -0.06331548
> kurtosis(data$x);
[1] 2.401046
```

Binomial Distribution

Binomial distribution has two outcomes, success or failure, and can be thought of as the probability of success or failure in a survey that is repeated various times. The number of observations is fixed, and each observation or probability is independent, and the probability of success is the same for all observations.

To get the probability mass function, Pr(X=x), of binomial distribution, you can use the `dbinom()` function:

```
 > dbinom(32, 100, 0.5);
[1] 0.000112817
```

This code lookup is for P(X=30) where X is the binomial distribution with a size of 100 and a probability of success of 0.5

To get the cumulative distribution function, P(X <= x), of a binomial distribution, you can use the `pbinom()` function:

```
> pbinom(32, 100, 0.5);
[1] 0.0002043886
```

The above code lookup is for p(X <= 30) where X is the binomial distribution with a size of 100 and a probability of success of 0.5.

To get the p-th quantile of the binomial distribution, you can use the `qbinom()` function:

```
> qbinom(0.3, 100, 0.5);
[1] 47
```

The above code lookup is for the 30th quantile of the binomial distribution where the size is 100 and the probability of success is 0.5. The value is a cumulative value.

To generate random variables from a binomial distribution, you use the `rbinom()` function:

```
> set.seed(123);
> A <- rbinom(1000, 100, 0.5);
> hist(A, breaks=20);
```

You can use `rbinom()` or `rnorm()` to generate random variables to simulate a new dataset.

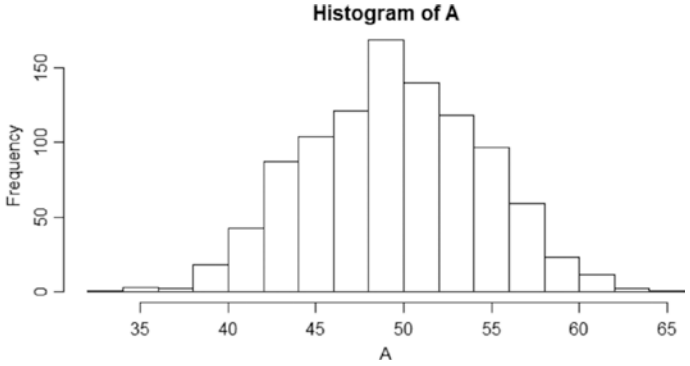


Figure 4-16. *Histogram of A*

The summary() and str() Functions

The `summary()` and `str()` functions are the fastest ways to get descriptive statistics of the data. The `summary()` function gives the basic descriptive statistics of the data. The `str()` function, as mentioned in a previous chapter, gives the structure of the variables.

You can get the basic descriptive statistics using the `summary()` function:

```
> summary(data);
       x                  x2                 x3
 y
 Min.   :-1.1038   Min.   :-2.299   Min.   :-5.0216
 Min.   :0.00
 1st Qu.: 0.3758   1st Qu.: 1.275   1st Qu.: 0.8415
 1st Qu.:0.00
 Median : 1.0684   Median : 2.381   Median : 2.5858
 Median :0.00
 Mean   : 1.0904   Mean   : 2.465   Mean   : 2.8314
 Mean   :0.49
 3rd Qu.: 1.7946   3rd Qu.: 3.750   3rd Qu.: 4.5229
 3rd Qu.:1.00
 Max.   : 3.1618   Max.   : 8.354   Max.   :12.8562
 Max.   :1.00
```

You can get the structure of the data using the `str()` function:

```
> str(data);
'data.frame':    100 obs. of  4 variables:
 $ x : num  2.216 -0.181 1.697 1.655 1.068 ...
 $ x2: num  4.77 4.1 2.33 2.46 1.05 ...
 $ x3: num  -4.87 6.98 3.92 0.75 3.35 ...
 $ y : num  0 1 0 0 1 1 1 1 1 1 ...
```

Conclusion

In this chapter, you looked into R programming. You now understand descriptive statistics. Descriptive statistics summarizes the data and usually focuses on the distribution, the central tendency, and the dispersion of the data.

You also looked into how R programming allows you to import a data set that can be a CSV file, Excel file, tab-separated file, JSON file, and others. Reading data into the R console or R is important because you must have some data before you can do statistical computing and understand the data.

You performed simple data processing like selecting data, sorting data, filtering data, getting unique values, and removing missing values.

You also learned some basic statistics terms such as population, sample, observations, and variables. You also learned about data types in statistics, which includes numeric data and categorical variables. Numeric data can have discrete and continuous data, and categorical data can have nominal and ordinal variables.

You also learned about mean, median, and mode, which are the most common measures for central tendency. Central tendency is a measure that best summarizes the data and is related to the center of the data set.

You also learned about measures of variability, which is the measure of the spread of the data. Measures of variability can be range, interquartile range, variance, standard deviation, and others.

You also learned about normal distribution, which is one of the more important theories because nearly all statistical testing requires the data to be distributed normally. You learned how to test normal distribution in data and the skewness and kurtosis of distributions.

You also learned about the summary() and str() functions, which are the fastest ways to get the descriptive statistics of the data. The summary() function gives the basic descriptive statistics of the data. The str() function, as mentioned in a previous chapter, gives the structure of the variables.

You learned more about two very popular distributions, where normal distribution is also known as bell curve and is a distribution that happens naturally in many data sets. Binomial distribution has two outcomes, success or failure, and can be thought of as the probability of success or failure in a survey that is repeated various times.

References

Probability Distributions in R (Stat 5101, Geyer). (n.d.). Retrieved from www.stat.umn.edu/geyer/old/5101/rlook.html.

Binomial Distribution: Formula, What it is and How to use it. (n.d.). Retrieved from www.statisticshowto.com/probability-and-statistics/binomial-theorem/binomial-distribution-formula/.

Biological data analysis, Tartu 2006/2007 (Tech.). (n.d.). Retrieved September 1, 2018, from www-1.ms.ut.ee/BDA/BDA4.pdf.

Calculate Standard Deviation. (n.d.). Retrieved from https://explorable.com/calculate-standard-deviation.

Descriptive Statistics. (n.d.). Retrieved from http://webspace.ship.edu/cgboer/descstats.html.

Descriptive statistics. (2018, August 22). Retrieved from https://en.wikipedia.org/wiki/Descriptive_statistics.

Donges, N. (2018, February 14). Intro to Descriptive Statistics – Towards Data Science. Retrieved from https://towardsdatascience.com/intro-to-descriptive-statistics-252e9c464ac9.

How to Make a Histogram with Basic R. (2017, May 04). Retrieved from www.r-bloggers.com/how-to-make-a-histogram-with-basic-r/.

How to open an SPSS file into R. (2016, January 15). Retrieved from www.milanor.net/blog/how-to-open-an-spss-file-into-r/.

How to Use Quantile Plots to Check Data Normality in R. (n.d.). Retrieved from www.dummies.com/programming/r/how-to-use-quantile-plots-to-check-data-normality-in-r/.

Interquartile Range. (n.d.). Retrieved from www.r-tutor.com/elementary-statistics/numerical-measures/interquartile-range.

Interquartile Range. (n.d.). Retrieved from www.r-tutor.com/elementary-statistics/numerical-measures/interquartile-range.

Is there a built-in function for finding the mode? (n.d.). Retrieved from https://stackoverflow.com/questions/2547402/is-there-a-built-in-function-for-finding-the-mode.

Kabacoff, R. (n.d.). Importing Data. Retrieved from www.statmethods.net/input/importingdata.html.

Normal Distributions: Definition, Word Problems. (n.d.). Retrieved from www.statisticshowto.com/probability-and-statistics/normal-distributions/.

Normal Distributions: Definition, Word Problems. (n.d.). Retrieved from www.statisticshowto.com/probability-and-statistics/normal-distributions/.

Normality Test in R. (n.d.). Retrieved from www.sthda.com/english/wiki/normality-test-in-r.

Q-Q plot. (n.d.). Retrieved from www.cookbook-r.com/Graphs/Q-Q_plot/.

Quartile. (n.d.). Retrieved from www.r-tutor.com/elementary-statistics/numerical-measures/quartile.

R - Data Processing. (n.d.). Retrieved from https://training-course-material.com/training/R_-_Data_Processing.

T. (n.d.). R JSON Files. Retrieved from www.tutorialspoint.com/r/r_json_files.htm.

T. (n.d.). R Mean, Median and Mode. Retrieved from www.tutorialspoint.com/r/r_mean_median_mode.htm.

Reading Data From TXT|CSV Files: R Base Functions. (n.d.). Retrieved from www.sthda.com/english/wiki/reading-data-from-txt-csv-files-r-base-functions.

Standard Deviation. (n.d.). Retrieved from www.r-tutor.com/elementary-statistics/numerical-measures/standard-deviation.

Statistical Variance. (n.d.). Retrieved from https://explorable.com/statistical-variance.

Variance. (n.d.). Retrieved from www.r-tutor.com/elementary-statistics/numerical-measures/variance.

CHAPTER 5

Data Visualizations

Descriptive statistics is a set of math used to summarize data. Data visualization is the equivalent of visual communication because creating graphics from data helps us understand the data. Humans distinguish differences in line, shape, and color without much processing effort, and data visualization can take advantage of this to create charts and graphs to help us understand the data more easily.

In this chapter, you will plot a bar chart, histogram, line chart, pie chart, scatterplot, boxplot, and scatterplot matrix using R. You will then look into plotting decision trees and social network analysis graphs. You will also use ggplot2 to create more advanced charts using the grammar of graphics, and then you will look into creating interactive charts using Plotly JS.

What Are Data Visualizations?

Descriptive statistics summarizes the data and usually focuses on the distribution, the central tendency, and the dispersion of the data. Data visualization, on the other hand, creates graphics from the data to help us understand it. The graphics or charts can also help us to communicate visually to our clients. A picture is worth a thousand words.

Data visualization can involve the plotting of a bar, histogram, scatterplot, boxplot, line chart, time series, and scatterplot matrix chart to help us analyze and reason about the data and understand the causality and relationship between variables. Data visualization can also be viewed

E. G. M. Hui, *Learn R for Applied Statistics*, https://doi.org/10.1007/978-1-4842-4200-1_5

as descriptive statistics to some. Humans distinguish differences in line, shape, and color without much processing effort, and data visualization can take advantage of this to create charts and graphs to help users understand the data more easily.

Bar Chart and Histogram

R programming allows us to plot bar charts and histograms. A bar chart represents data using bars, with y values being the value of the variable. R programming uses the `barplot()` function to create bar charts, and R can draw both horizontal and vertical bar charts. A histogram, on the other hand, represents the frequencies of the values within a variable and draws them into bars.

To plot a bar chart in R, you can use the `barplot()` function:

```
> data <- c(4, 6, 7, 9, 10, 20, 12, 8);
> barplot(data, xlab="X-axis", ylab="Y-axis", main="Bar Chart 1",
col="green");
```

`data` is the data to plot, `xlab` is the x-axis name, `ylab` is the y-axis name, `main` is the main title, and `col` is the color of the chart. See Figure 5-1.

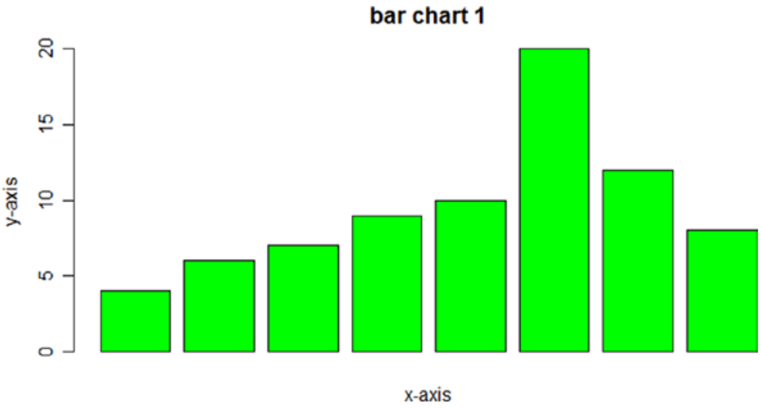


Figure 5-1. *Bar Chart of data*

To export the bar chart into an image file (see Figure 5-2), you can add in

```
> data <- c(4, 6, 7, 9, 10, 20, 12, 8);
> png(file="D:/barchart.png");
> barplot(data, xlab="x-axis", ylab="y-axis", main="bar chart 1",
col="green");
> dev.off();
RStudioGD
        2
```

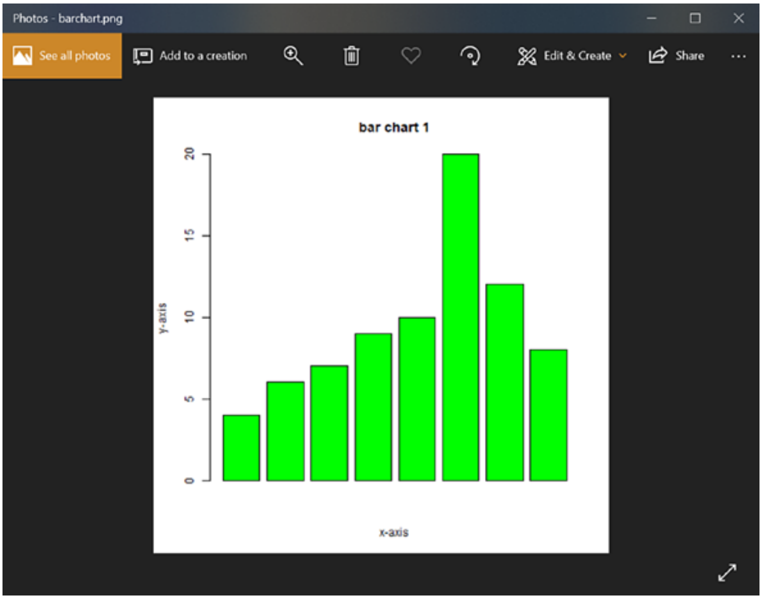


Figure 5-2. *Bar Chart PNG file Opened with Microsoft Photos*

To plot a horizontal bar chart (see Figure 5-3), you can use `horiz=TRUE`:

```
> data <- c(4, 6, 7, 9, 10, 20, 12, 8);
> png(file="D:/barchart.png");
> barplot(data, xlab="x-axis", ylab="y-axis", main="bar chart
1", col="green", horiz=TRUE);
> dev.off();
RStudioGD
        2
```

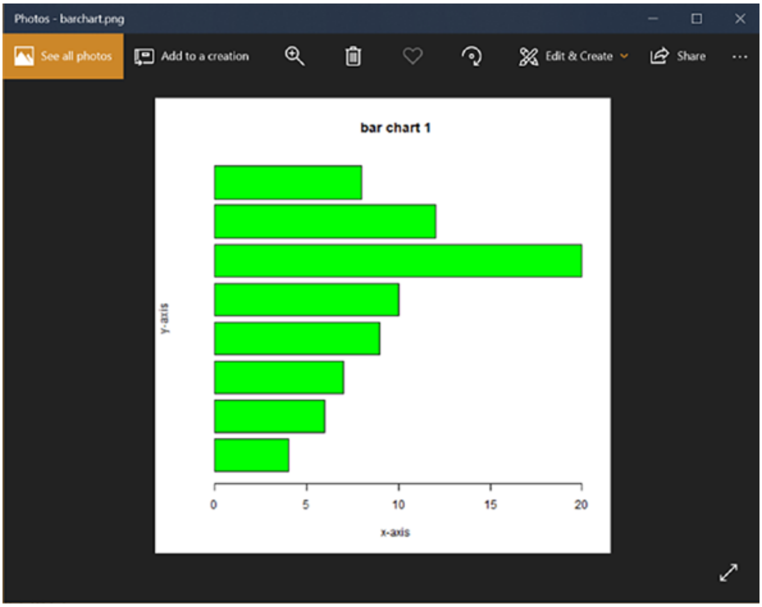


Figure 5-3. *Horizontal Bar Chart of data variable*

To plot a stacked bar plot, you can create the following data set:

```
> data(mtcars);
> data <- table(mtcars$gear, mtcars$carb);
> data;

    1 2 3 4 6 8
  3 3 4 3 5 0 0
  4 4 4 0 4 0 0
  5 0 2 0 1 1 1
```

To plot a stacked bar plot:

```
> png(file="D:/barchart.png");
> barplot(data3, xlab="x-axis", ylab="y-axis", main="bar chart
1", col=c("grey", "blue", "yellow"));
> dev.off();
RStudioGD
        2
```

In the data, 3 is the grey color, 4 is the blue color, and 5 is the yellow color. When the x-axis or x is 1, the grey color is 3 steps, the blue color is 4 steps, and the yellow color is 0 steps. See Figure 5-4.

```
> data;

  1 2 3 4 6 8
3 3 4 3 5 0 0
4 4 4 0 4 0 0
5 0 2 0 1 1 1
```

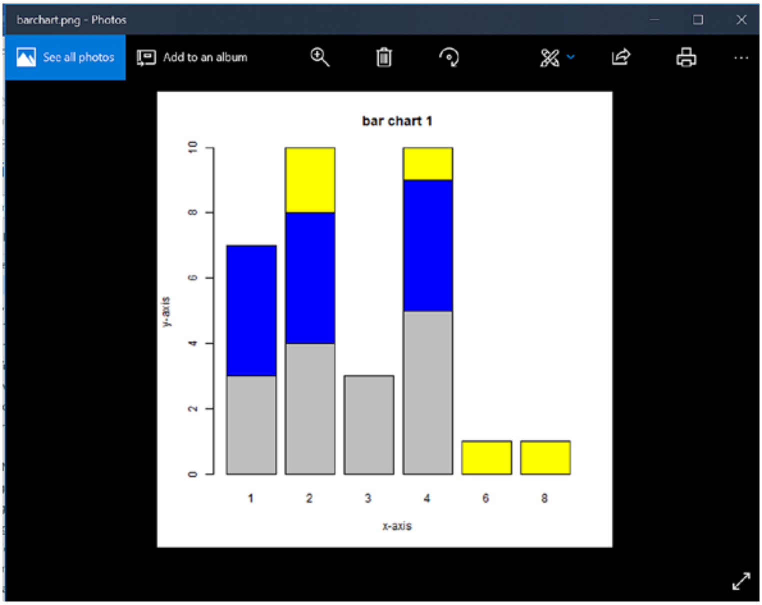


Figure 5-4. *Stacked bar plot of data*

To plot a grouped bar chart, you can use beside=TRUE:

```
> data(mtcars);
> data <- table(mtcars$gear, mtcars$carb);
> data;

    1 2 3 4 6 8
  3 3 4 3 5 0 0
  4 4 4 0 4 0 0
  5 0 2 0 1 1 1
> png(file="D:/barchart.png");
> barplot(data3, xlab="x-axis", ylab="y-axis", main="bar chart 1",
col=c("grey", "blue", "yellow"), beside=TRUE);
> dev.off();
RStudioGD
        2
```

In the data, 3 is the grey color, 4 is the blue color, and 5 is the yellow color. When the x-axis or x is 1, the grey color is 3 steps, the blue color is 4 steps, and the yellow color is 0 steps. See Figure 5-5.

```
> data;

    1 2 3 4 6 8
  3 3 4 3 5 0 0
  4 4 4 0 4 0 0
  5 0 2 0 1 1 1
```

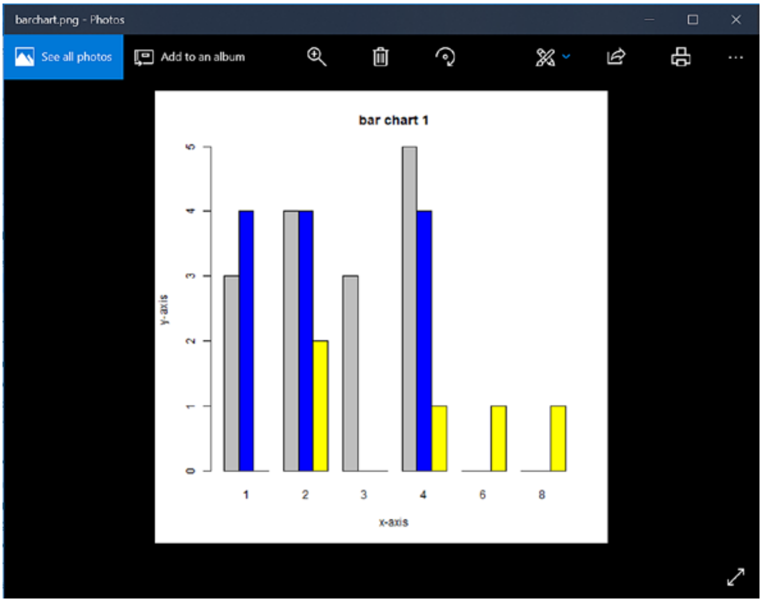


Figure 5-5. *Grouped bar chart of data*

To plot a histogram, you can use the `hist()` function (see Figure 5-6):

```
> set.seed(123);
> data1 <- rnorm(100, mean=5, sd=3);
> png(file="D:/histogram.png");
> hist(data1, main="histogram", xlab="x-axis", col="green",
  border="blue", breaks=10);
> dev.off();
RStudioGD
        2
```

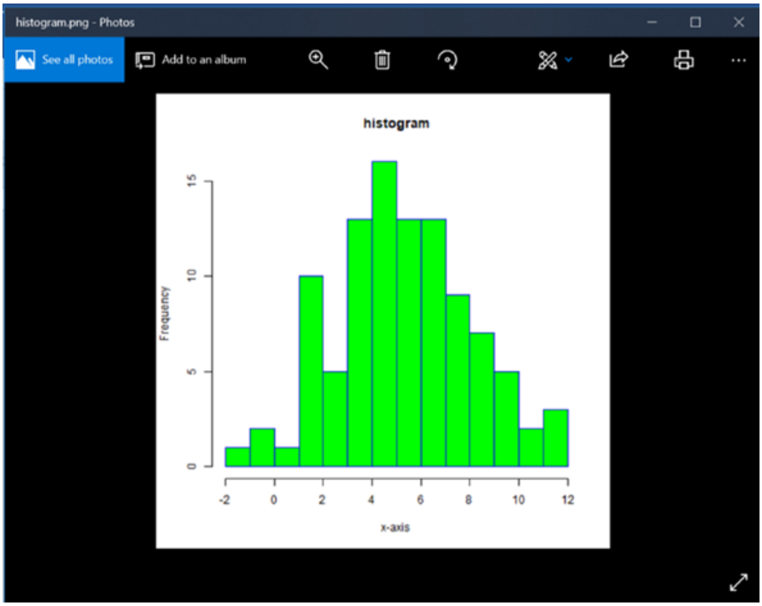


***Figure 5-6.** Histogram of data1*

`data1` is the data, `main` is the main title, `col` is the color, `border` is the color of the borders, `xlab` is the name of the x-axis, and `breaks` is the width of each bar.

To plot a histogram with a density line, you can change `freq=FALSE` so that the histogram is plotted based on probability, and you can use the `lines()` function to add the density line (see Figure 5-7):

```
> set.seed(123);
> data1 <- rnorm(100, mean=5, sd=3);
> hist(data1, main="histogram", xlab="x-axis", col="green",
  border="blue", breaks=10, freq=FALSE);
> lines(density(data1), col="red");
```

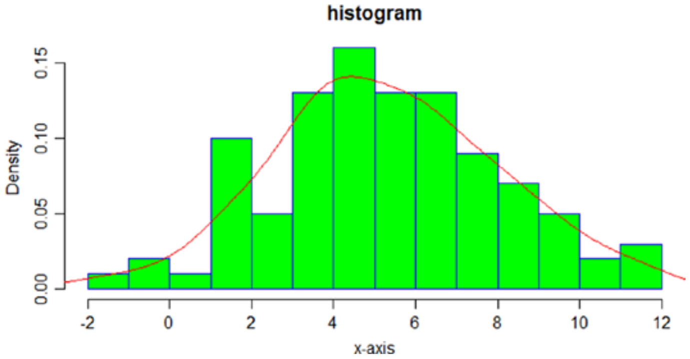


Figure 5-7. *Histogram with density line of data1*

Line Chart and Pie Chart

A line chart is a graph that has all the points connected together by drawing lines between them. A line chart is very useful for trend analysis and time series analysis. A pie chart, on the other hand, is the representation of data as slices of a circle with various colors.

You can plot a line graph using the `plot()` function (see Figure 5-8):

```
> x <- c(1, 2, 3, 4, 5, 6, 8, 9);
> y <- c(3, 5, 4, 6, 9, 8, 2, 1);
> png(file="D:/line.png");
> plot(x, y, type="l", xlab="x-axis", ylab="y-axis", main="line
  graph", col="blue");
> dev.off();
RStudioGD
        2
```

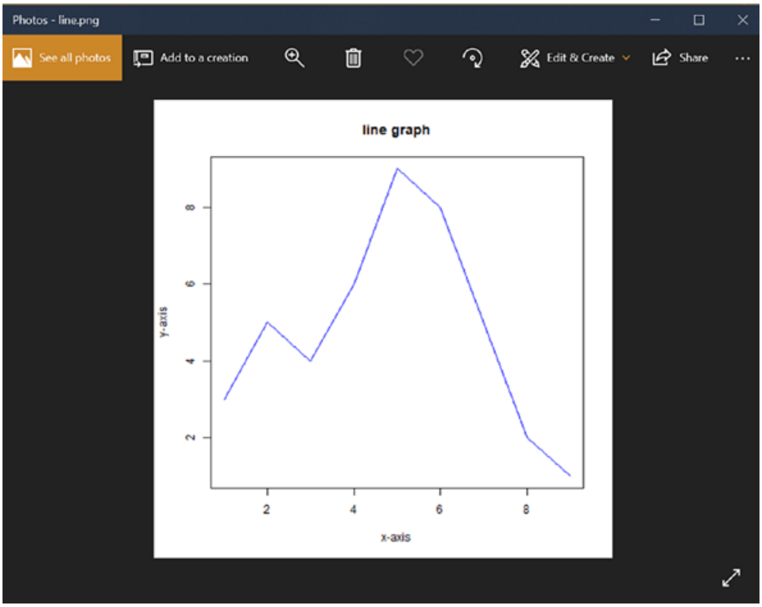


Figure 5-8. *Line chart*

You use `type="l"` when you want to plot a line chart and `type="p"` when you want to plot a point chart or scatter chart. `xlab` is the x-axis name, `ylab` is the y-axis name, `main` is the main title, and `col` is the color of the chart.

To plot multiple line graph, you can add in the `lines()` function:

```
> x <- c(1, 2, 3, 4, 5, 6, 8, 9);
> y <- c(3, 5, 4, 6, 9, 8, 2, 1);
> x.1 <- c(2, 3, 4, 6, 7, 8, 9, 10);
> y.1 <- c(6, 3, 5, 1, 5, 3, 4, 8);
> png(file="D:/line.png");
> plot(x, y, type="l", xlab="x-axis", ylab="y-axis", main="line
  graph", col="blue");
> lines(x.1, y.1, type="o", col="green");
> dev.off();
RStudioGD
        2
```

type="o" will give you a line graph with a point in it (see Figure 5-9).

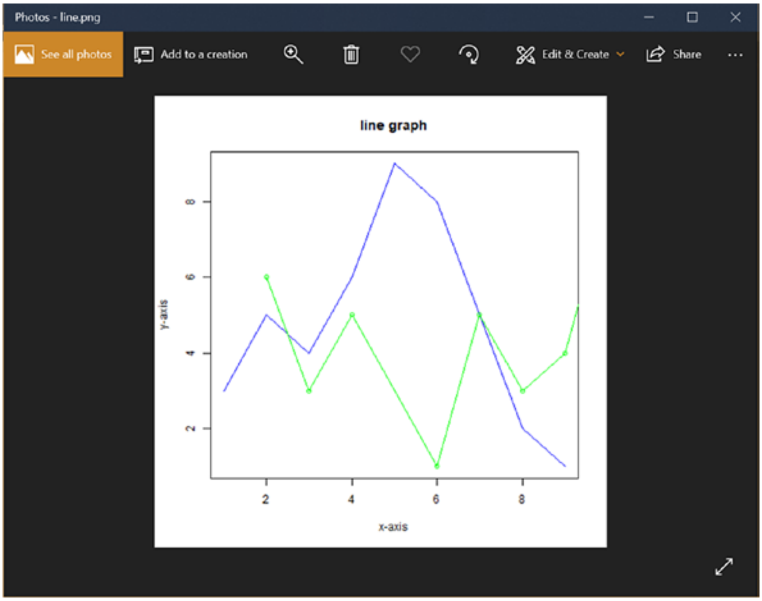


Figure 5-9. *Multiple Line Chart*

To plot a pie chart, you can use the pie() function (see Figure 5-10):

```
> x <- c(10, 30, 60, 10, 50);
> labels <- c("one", "two", "three", "four", "five");
> png(file="D:/pie.png");
> pie(x, labels, main="Pie Chart");
> dev.off();
RStudioGD
        2
```

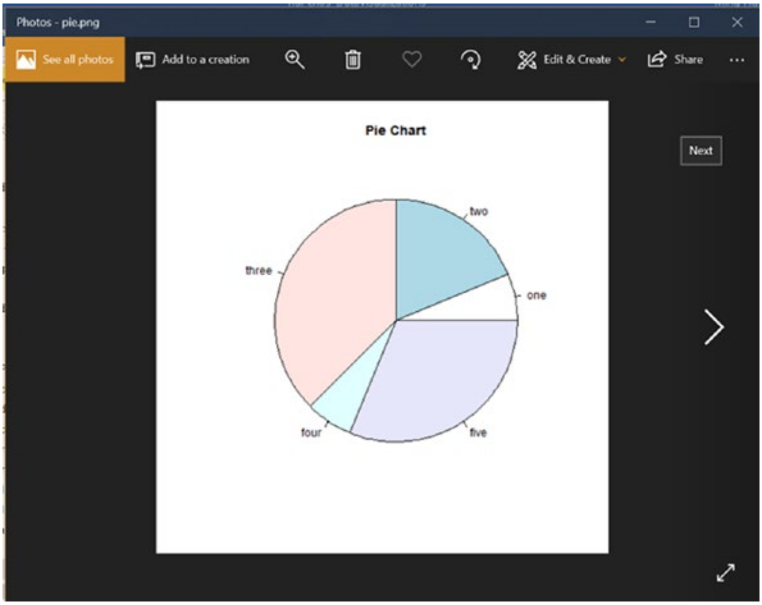


Figure 5-10. *Pie Chart*

To plot a 3D pie chart, you must install the `plotrix` library:

```
> install.packages("plotrix");
Installing package into 'C:/Users/gohmi/Documents/R/win-
library/3.5'
(as 'lib' is unspecified)
trying URL 'https://cran.rstudio.com/bin/windows/contrib/3.5/
plotrix_3.7-3.zip'
Content type 'application/zip' length 1055537 bytes (1.0 MB)
downloaded 1.0 MB

package 'plotrix' successfully unpacked and MD5 sums checked

The downloaded binary packages are in
    C:\Users\gohmi\AppData\Local\Temp\RtmpET26Hv\downloaded_
    packages
```

You can use the require() function or the library() function to call the plotrix library:

```
> library(plotrix);
```

You can use the pie3D() function to plot the 3D pie chart shown in Figure 5-11:

```
> x <- c(10, 30, 60, 10, 50);
> labels <- c("one", "two", "three", "four", "five");
> png(file="D:/pie.png");
> pie3D(x, labels=labels, explode=0.1, main="Pie Chart");
> dev.off();
RStudioGD
        2
```

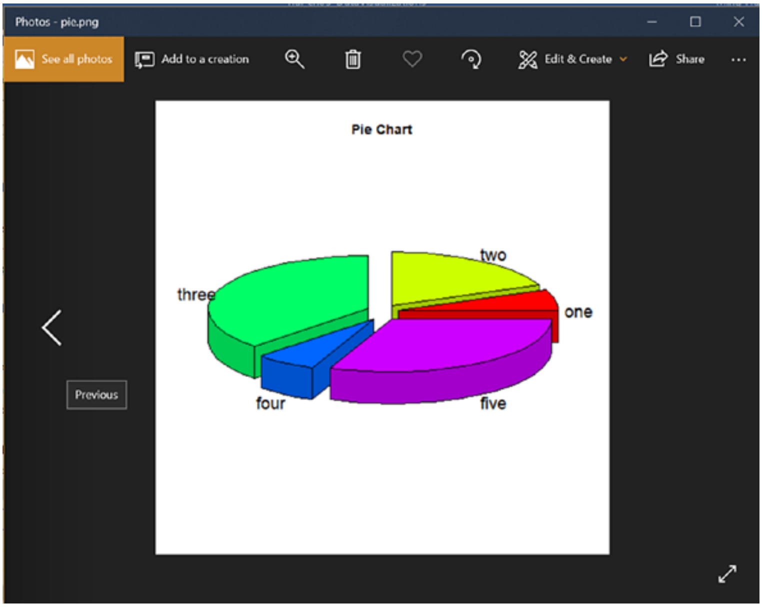


Figure 5-11. *3D Pie Chart*

Scatterplot and Boxplot

A scatterplot is a chart that represents data using points in the Cartesian plane. Each point is the value of two variables. A boxplot shows the statistics of the data.

You can plot a scatterplot using the plot() function:

```
> x <- c(1, 2, 3, 4, 5, 6, 8, 9);
> y <- c(3, 5, 4, 6, 9, 8, 2, 1);
> png(file="D:/scatter.png");
> plot(x, y, xlab="x-axis", ylab="y-axis", main="scatterplot");
> dev.off();
RStudioGD
        2
```

xlab is the x-axis name, ylab is the y-axis name, and main is the main title. See Figure 5-12.

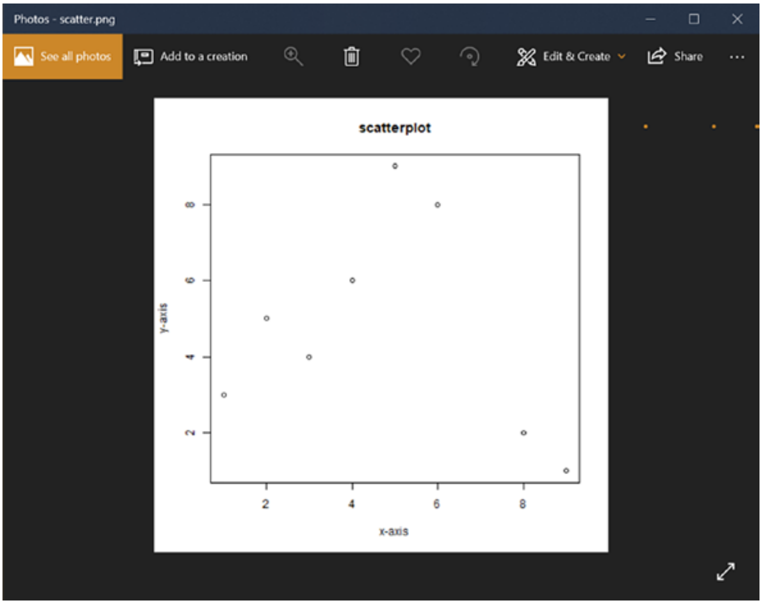


Figure 5-12. *Scatterplot*

A boxplot represents how well the data in a data set is distributed, depicting the minimum, maximum, median, first quartile, and third quartile. Figure 5-13 explains the boxplot.

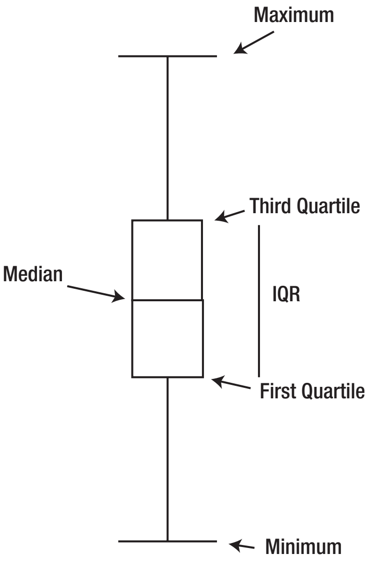


Figure 5-13. *Boxplot and its explanation*

To create the boxplot in Figure 5-14, you can use the `boxplot()` function:

```
> set.seed(12);
> var1 <- rnorm(100, mean=3, sd=3);
> var2 <- rnorm(100, mean=2, sd=2);
> var3 <- rnorm(100, mean=1, sd=3);
> data <- data.frame(var1, var2, var3);
> png(file="D:/boxplot.png");
> boxplot(data, main="boxplot", notch=FALSE, varwidth=TRUE,
  col=c("green", "purple", "blue"));
> dev.off();
RStudioGD
        2
```

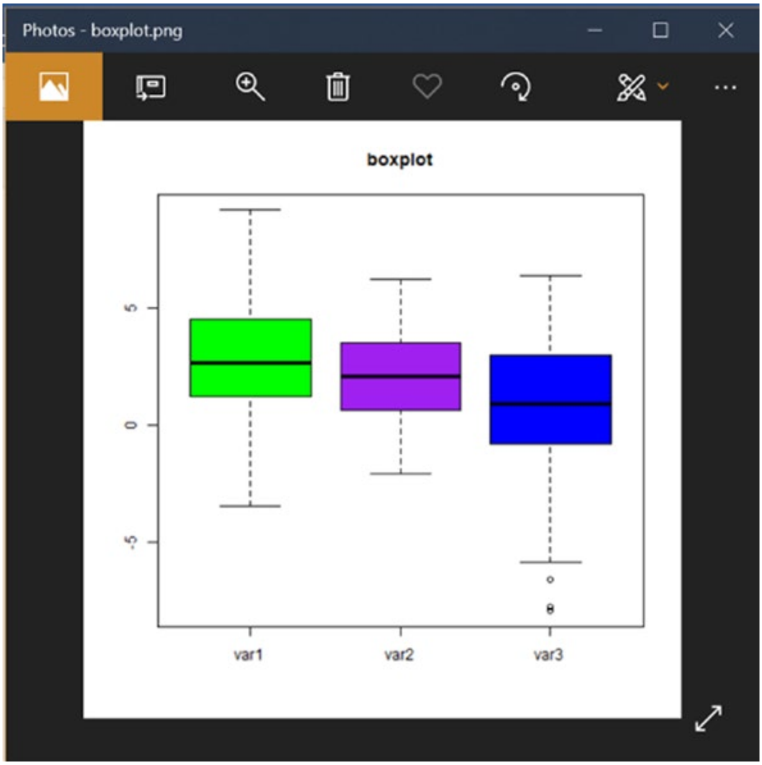


Figure 5-14. *Boxplots*

`data` is the data, `main` is the main title, `notch` is a logical value to state how medians of different groups match with each other, `varwidth` is a logical value to state whether to draw the width of box proportionate to the sample size, and `col` is the color of the boxplot.

You can draw a boxplot with a notch by setting `notch=TRUE` (see Figure 5-15):

```
> set.seed(12);
> var1 <- rnorm(100, mean=3, sd=3);
> var2 <- rnorm(100, mean=2, sd=2);
> var3 <- rnorm(100, mean=1, sd=3);
> data <- data.frame(var1, var2, var3);
> png(file="D:/boxplot.png");
> boxplot(data, main="boxplot", notch=TRUE, varwidth=TRUE,
  col=c("green", "purple", "blue"));
> dev.off();
RStudioGD
        2
```

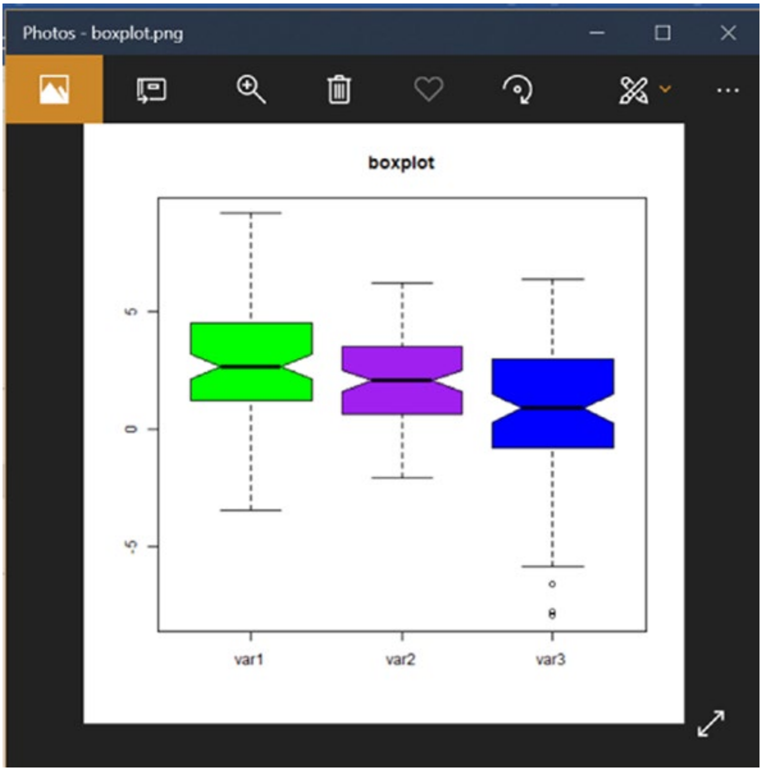


Figure 5-15. *Boxplots with notch*

Scatterplot Matrix

A scatterplot matrix is used to find the correlation between a variable and other variables, and you usually use it to select the important variables, which is also known as variable selection.

To plot the scatterplot matrix in Figure 5-16, you can use the pairs() function:

```
> set.seed(12);
> var1 <- rnorm(100, mean=1, sd=3);
> var2 <- rnorm(100, mean=1, sd=3);
> var3 <- rnorm(100, mean=1, sd=3);
> var4 <- rnorm(100, mean=2, sd=3);
> var5 <- rnorm(100, mean=2, sd=3);
> data <- data.frame(var1, var2, var3, var4, var5);
> png(file="D:/scatterplotMatrix.png");
> pairs(~var1+var2+var3+var4+var5, data=data, main="scatterplot
  matrix");
> dev.off();
RStudioGD
        2
```

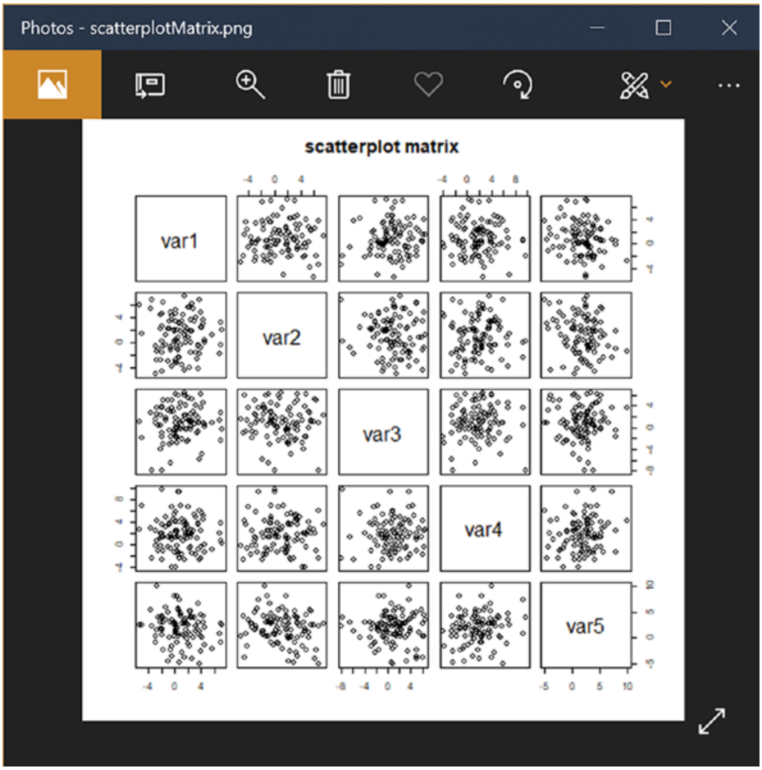


Figure 5-16. *Scatterplot Matrix*

Social Network Analysis Graph Basics

A social network analysis graph is an advanced data visualization; it is not related to statistics, but can be good for reference. The following is a fast introduction to social network analysis graphs. A social network analysis graph can help you understand the relationships between individuals or nodes. Social network analysis is usually used on social network data like Facebook and Weibo. Each node is an individual, and the social network graphs show us how each individual connects to others.

To plot a social network analysis graph, you must install the igraph package:

```
> install.packages("igraph");
Installing package into 'C:/Users/gohmi/Documents/R/win-
library/3.5'
(as 'lib' is unspecified)
trying URL 'https://cran.rstudio.com/bin/windows/contrib/3.5/
igraph_1.2.2.zip'
Content type 'application/zip' length 9148505 bytes (8.7 MB)
downloaded 8.7 MB

package 'igraph' successfully unpacked and MD5 sums checked

The downloaded binary packages are in
    C:\Users\gohmi\AppData\Local\Temp\RtmpET26Hv\downloaded_
    packages
```

To use the igraph package, you must use the `library()` function:

```
> library(igraph);

Attaching package: 'igraph'

The following objects are masked from 'package:stats':

    decompose, spectrum

The following object is masked from 'package:base':

    union
```

To plot the network graph for John ➤ James ➤ Mary ➤ John shown in Figure 5-17, you use

```
> g <- graph(edges=c("John", "James", "James", "Mary", "Mary",
"John"), directed=FALSE);
> plot(g);
```

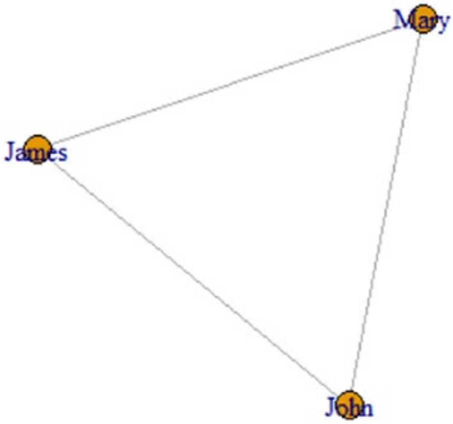


Figure 5-17. *Social Network Analysis with Undirected Network Graph*

To plot the directed network graph shown in Figure 5-18, you change `directed=FALSE` to `directed=TRUE`:

```
> g <- graph(edges=c("John", "James", "James", "Mary", "Mary",
"John"), directed=TRUE);
> plot(g);
```

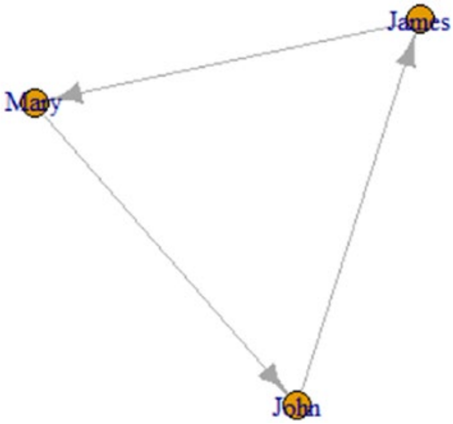


Figure 5-18. *Social Network Analysis with directed network graph*

Using ggplot2

ggplot2 is a package created by Hadley Wickham that offers a powerful graphics language for creating advanced charts. ggplot2 is very popular and famous in the R community, and it allows us to create charts that represent univariate, multivariate, and categorical data in a straightforward way. R's build-in functionality offer the plotting of charts, but ggplot allows us to plot more advanced charts using the grammar of graphics.

To use ggplot2, you must install the package:

```
> install.packages("ggplot2");
Installing package into 'C:/Users/gohmi/Documents/R/win-
library/3.5'
(as 'lib' is unspecified)
also installing the dependencies 'stringi', 'colorspace',
'stringr', 'labeling', 'munsell', 'RColorBrewer', 'digest',
'gtable', 'lazyeval', 'plyr', 'reshape2', 'scales',
'viridisLite', 'withr'

  There is a binary version available but
  the source version is later:
        binary source needs_compilation
stringi  1.1.7  1.2.4              TRUE

  Binaries will be installed
trying URL 'https://cran.rstudio.com/bin/windows/contrib/3.5/
stringi_1.1.7.zip'
Content type 'application/zip' length 14368013 bytes (13.7 MB)
downloaded 13.7 MB

trying URL 'https://cran.rstudio.com/bin/windows/contrib/3.5/
colorspace_1.3-2.zip'
Content type 'application/zip' length 527776 bytes (515 KB)
downloaded 515 KB
```

```
The downloaded binary packages are in
    C:\Users\gohmi\AppData\Local\Temp\RtmpET26Hv\downloaded_
    packages
```

What Is the Grammar of Graphics?

ggplot2 focuses on the grammar of graphics, which is the building blocks of the chart, such as

- Data
- Aesthetic mapping
- Geometric objects
- Statistical transformation
- Scales
- Coordinate systems
- Position adjustments
- Faceting

In this book, you are going to look into some of them. It is beyond this book to cover everything in ggplot2.

The Setup for ggplot2

To use ggplot2, you must call the library using the `library()` or `require()` functions. You also need to let ggplot know which data set to use, and you can use the `ggplot()` function:

```
> library(ggplot2);
> set.seed(12);
> var1 <- rnorm(100, mean=1, sd=1);
> var2 <- rnorm(100, mean=2, sd=1);
> var3 <- rnorm(100, mean=1, sd=2);
> data <- data.frame(var1, var2, var3);
> ggplot(data);
```

`ggplot()` can only take a data frame variable.

Aesthetic Mapping in ggplot2

In ggplot2, aesthetics are the things we can see, such as

- Position
- Color
- Fill
- Shape
- Line type
- Size

You can use aestheticsc in ggplot2 via the `aes()` function:

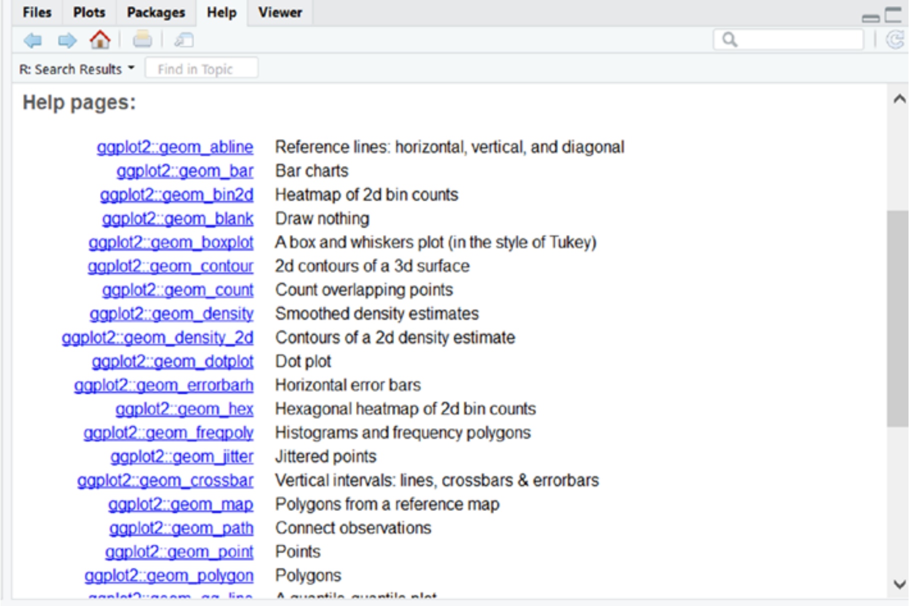
```
> ggplot(data, aes(x=var1, y=var2));
```

Geometry in ggplot2

Geometric objects are the plots or graphs you want to put in the chart. You can use `geom_point()` to create a scatterplot, `geom_line()` to create a line plot, and `geom_boxplot()` to create a boxplot in the chart.

You can see the available geometric objects (also shown in Figure 5-19) using

```
> help.search("geom_", package="ggplot2");
```



Help pages:

ggplot2::geom_abline	Reference lines: horizontal, vertical, and diagonal
ggplot2::geom_bar	Bar charts
ggplot2::geom_bin2d	Heatmap of 2d bin counts
ggplot2::geom_blank	Draw nothing
ggplot2::geom_boxplot	A box and whiskers plot (in the style of Tukey)
ggplot2::geom_contour	2d contours of a 3d surface
ggplot2::geom_count	Count overlapping points
ggplot2::geom_density	Smoothed density estimates
ggplot2::geom_density_2d	Contours of a 2d density estimate
ggplot2::geom_dotplot	Dot plot
ggplot2::geom_errorbarh	Horizontal error bars
ggplot2::geom_hex	Hexagonal heatmap of 2d bin counts
ggplot2::geom_freqpoly	Histograms and frequency polygons
ggplot2::geom_jitter	Jittered points
ggplot2::geom_crossbar	Vertical intervals: lines, crossbars & errorbars
ggplot2::geom_map	Polygons from a reference map
ggplot2::geom_path	Connect observations
ggplot2::geom_point	Points
ggplot2::geom_polygon	Polygons
ggplot2::geom_qq_line	A quantile-quantile plot

Figure 5-19. *Available Geometric Objects*

In ggplot2, geom is also the layers of the chart. You can add in one geom object after another, just like adding one layer after another layer.

You can add in a scatter plot (Figure 5-20) using the `geom_point()` function:

```
> ggplot(data, aes(x=var1, y=var2)) + geom_
  point(aes(color="red"));
```

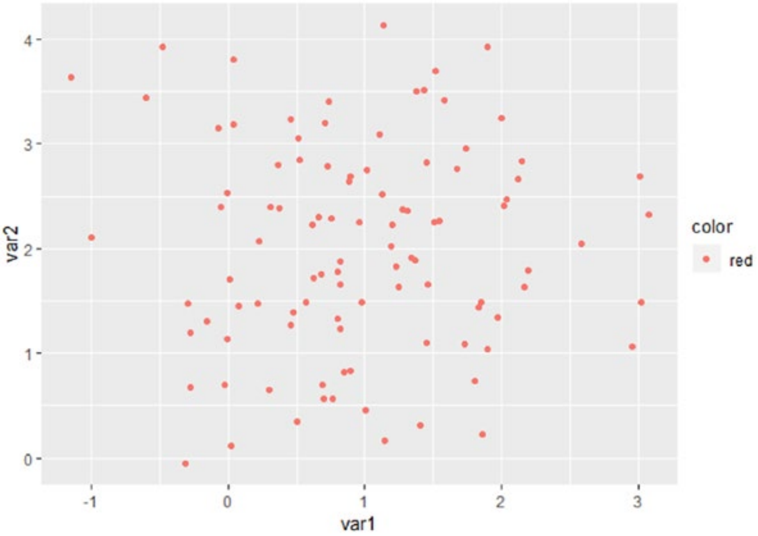


Figure 5-20. *Scatterplot*

You can add in a smoother that includes a line and a ribbon to the scatter plot (Figure 5-21) using another layer:

```
> ggplot(data, aes(x=var1, y=var2)) + geom_
  point(aes(color="red")) + geom_smooth();
`geom_smooth()` using method = 'loess' and formula 'y ~ x'
```

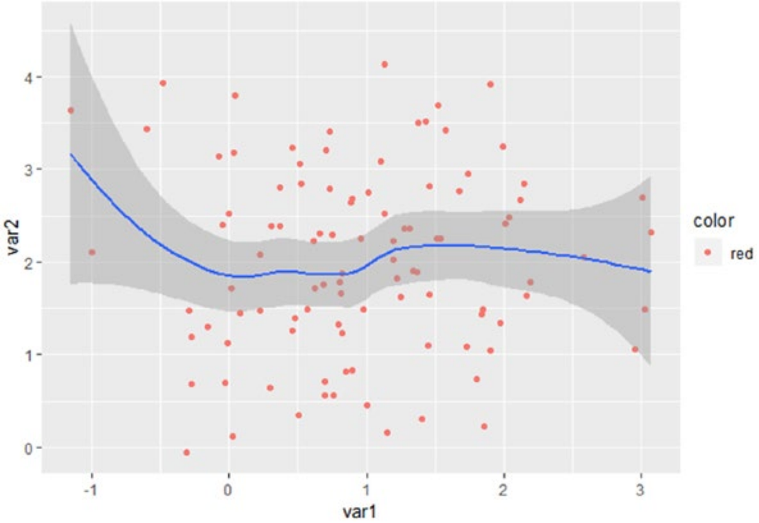


Figure 5-21. *Scatterplot with Smoother*

Labels in ggplot2

You have plotted the graphs in the charts, so now let's add in the main title and x- and y-axis titles. You can do this using the `labs()` layer to specify the labels.

To add in the x-axis title, y-axis title, and the main title into Figure 5-22, you can use the `labs()` function:

```
> ggplot(data, aes(x=var1, y=var2)) + geom_point
  (aes(color="red")) + geom_smooth() + labs(title="Scatter",
  x = "Xaxis", y = "Y-axis", color="Color");
`geom_smooth()` using method = 'loess' and formula 'y ~ x'
```

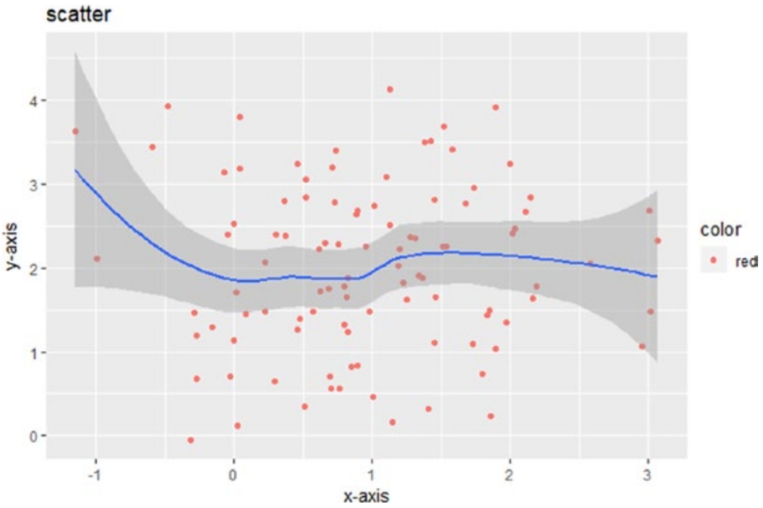


Figure 5-22. *Scatterplot with Smoother and labels*

Themes in ggplot2

If you want to change the size and style of the labels and legends, the `theme()` function can help. The `theme()` function in ggplot handles elements such as

- Axis labels
- Plot background
- Facet background
- Legend appearance

There are some built-in themes such as `theme_light()` and `theme_bw()`.

You can add the built-in themes shown in Figure 5-23 by using `theme_light()`:

```
> ggplot(data, aes(x=var1, y=var2)) + geom_point
  (aes(color="red")) + geom_smooth() + labs(title="scatter",
  x="x-axis", y="y-axis") + theme_light();
`geom_smooth()` using method = 'loess' and formula 'y ~ x'
```

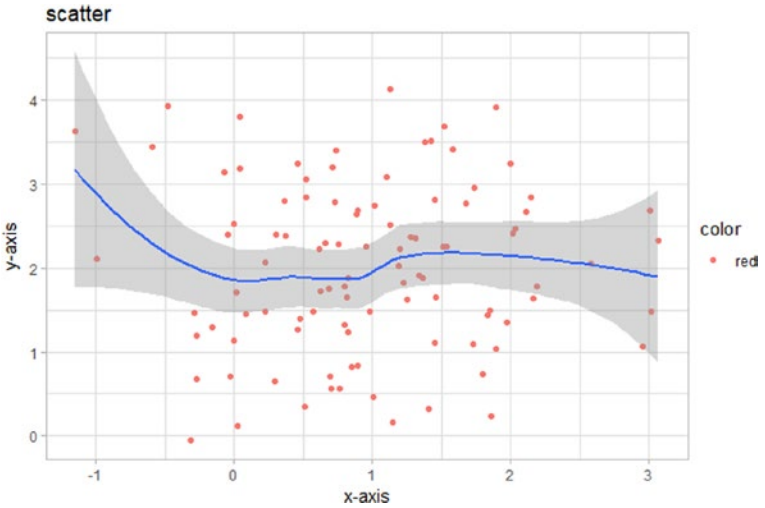


***Figure 5-23.** Scatterplot with smoothers and labels and themes*

You can add in your own theme, as shown in Figure 5-24, using

```
> ggplot(data, aes(x=var1, y=var2)) + geom_
  point(aes(color="red")) + geom_smooth() +
  labs(title="scatter", x="x-axis", y="y-axis") + theme(plot.
  title=element_text(size=30, face="bold"), axis.text.
  x=element_text(size=15, face="bold"), axis.text.y=element_
  text(size=15, face="bold"));
`geom_smooth()` using method = 'loess' and formula 'y ~ x'
```

plot.title is the title of the chart, axis.text.x is the x-axis label, axis.text.y is the y-axis label, axis.title.x is the title of x-axis, and axis.title.y is the title of y-axis. To change the size of the text, use the element_text() function. To remove the label, you can use element_blank().

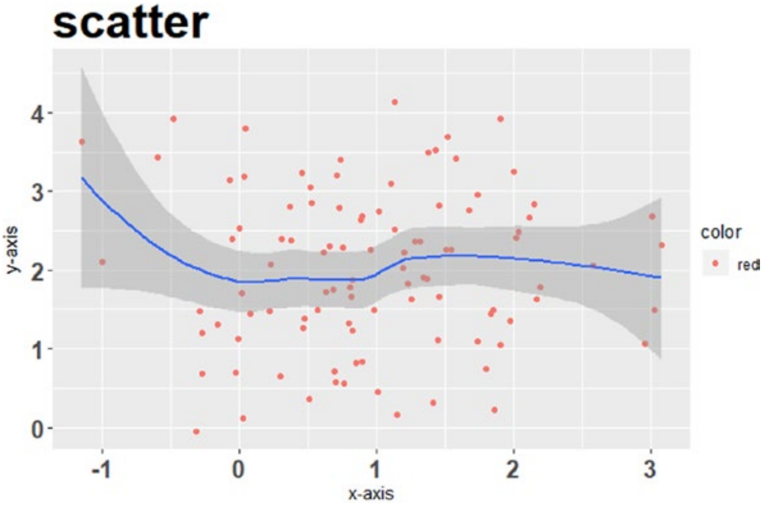


Figure 5-24. *Scatterplot with Smoother and customized themes*

ggplot2 Common Charts

After learning the ggplot basics and the grammar of graphics, the following sections cover some common charts that can be plotted with ggplot. The code is available also.

Bar Chart

A bar chart (Figure 5-25) is used when you want to compare things between groups:

```
> d <- c(1, 5, 8, 9, 8, 2, 1);
> df <- data.frame(d);
> ggplot(df) + geom_bar(aes(color="grey", x=d)) +
  labs(title="bar chart") + theme_light();
```

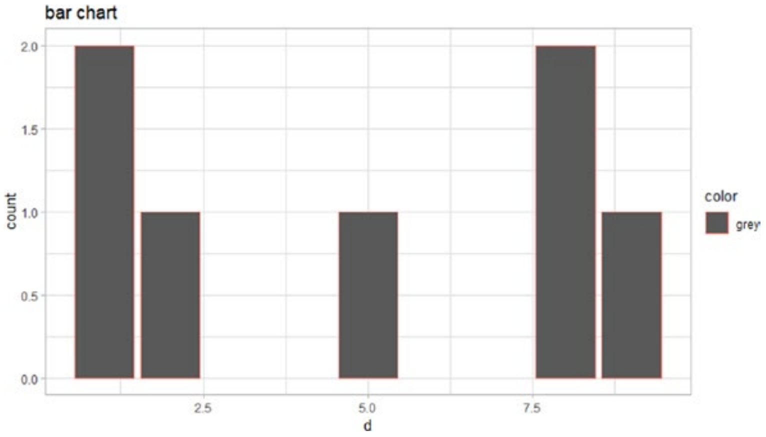


Figure 5-25. *Bar Chart*

To plot a horizontal bar chart (Figure 5-26), you can use the `coord_flip()` function:

```
> ggplot(df) + geom_bar(aes(color="grey", x=d)) + coord_flip()
+ labs(title="bar chart") + theme_light();
```

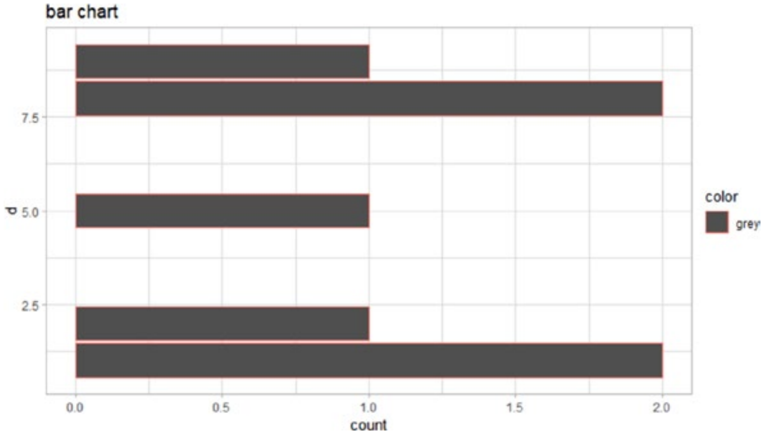


Figure 5-26. *Horizontal Bar Chart*

Histogram

A histogram (Figure 5-27) allows you to see whether the data is normally distributed.

```
> set.seed(12);
> var1 <- rnorm(100, mean=1, sd=1);
> var2 <- rnorm(100, mean=2, sd=1);
> var3 <- rnorm(100, mean=1, sd=2);
> data <- data.frame(var1, var2, var3);
> ggplot(data, aes(x=var1)) + geom_histogram(bins=10,
  color="black", fill="grey") + labs(title="histogram") +
  theme_light();
```

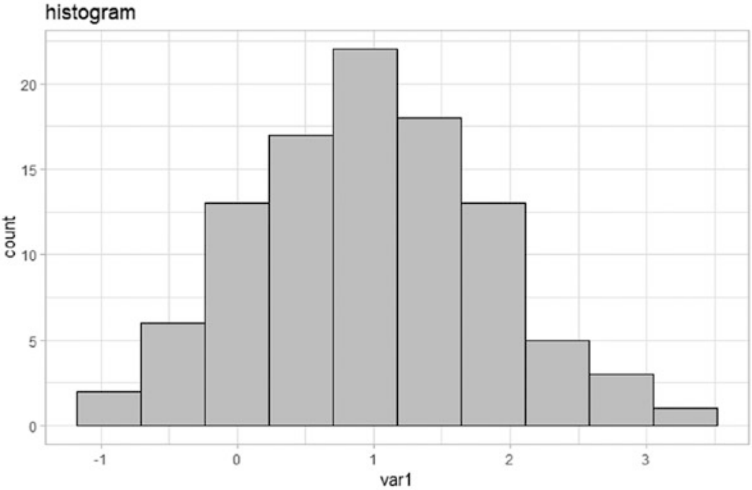


Figure 5-27. *Histogram*

Density Plot

A density plot (Figure 5-28) can also show you whether the data is normally distributed.

```
> ggplot(data, aes(x=var1)) + geom_density(fill="grey") +
  labs(title="density");
```

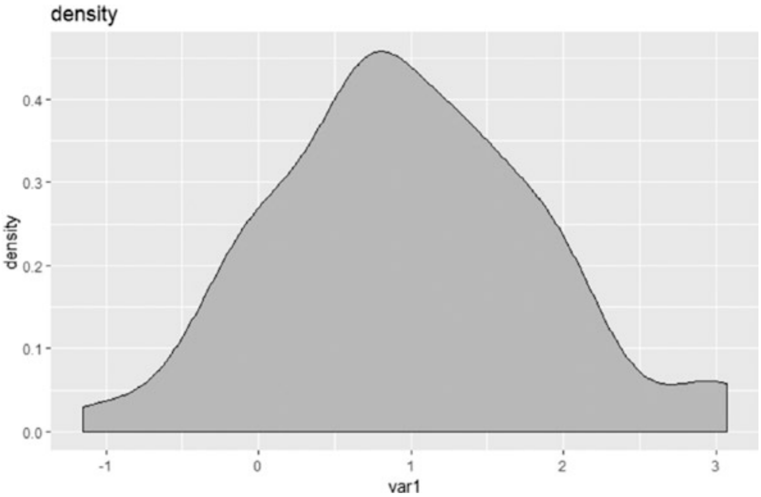


Figure 5-28. *Density Plot*

Scatterplot

A scatterplot (Figure 5-29) shows the relationships between two variables.

```
> ggplot(data) + geom_point(aes(color="red", x=var1, y=var2))
  + geom_point(aes(color="green", x=var1, y=var3)) +
  labs(title="scatter") + theme_light();
```

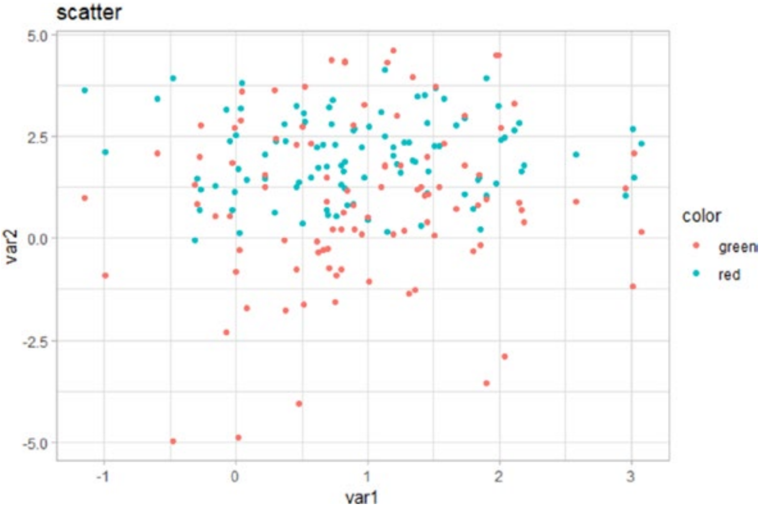


Figure 5-29. *Scatterplot*

Line chart

A line chart (Figure 5-30) also shows the relationship between two variables and can also be used for trend analysis.

```
> ggplot(data) + geom_line(aes(color="red", x=var1, y=var2))
  + geom_line(aes(color="green", x=var1, y=var3)) +
  labs(title="scatter") + theme_light();
```

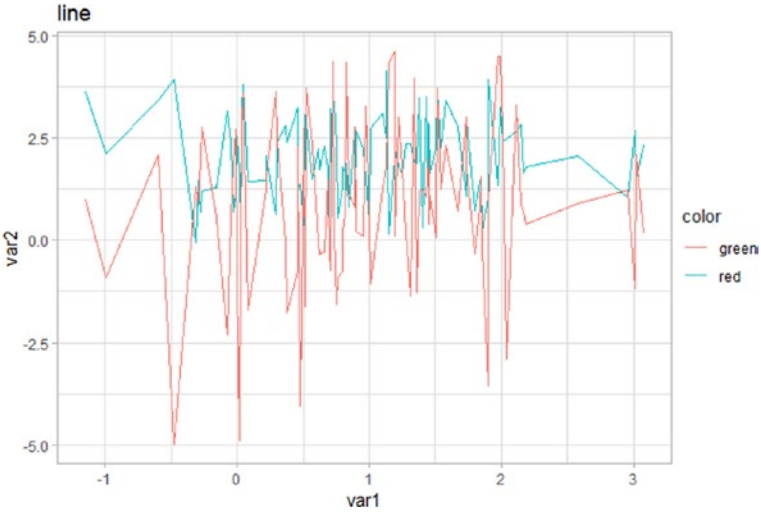


Figure 5-30. *Line Chart*

Boxplot

A boxplot (Figure 5-31) shows the statistics of the data.

```
> ggplot(data, aes(y=var2)) + geom_boxplot(fill="grey") +
  labs(title="boxplot");
```

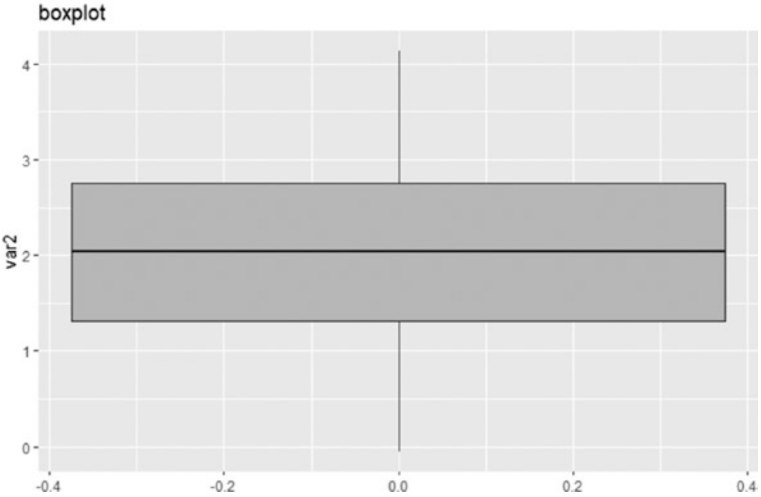


Figure 5-31. *Boxplot*

You can change the grids (Figure 5-32) using

```
> ggplot(data, aes(x=var1)) + geom_density(fill="grey") +
  labs(title="density") + theme(panel.background=element_
  rect(fill="yellow"), panel.grid.major=element_
  line(color="blue", size=2));
```

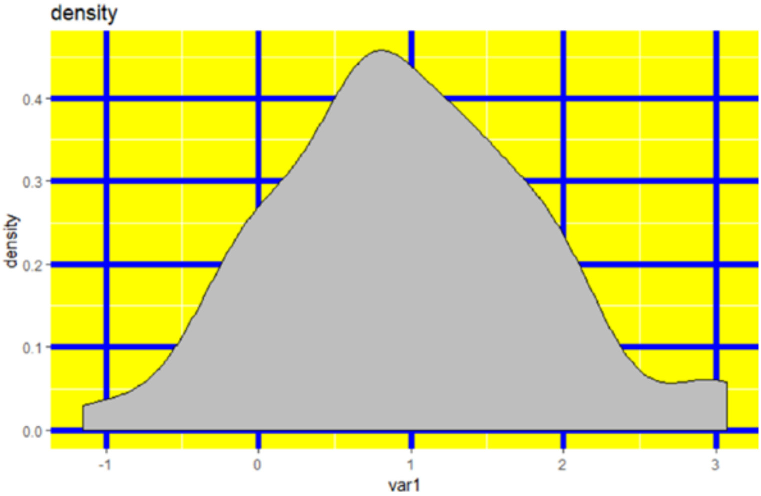


Figure 5-32. *Customised Themes*

You can change the background color using the `panel.background` and `element_rect()`, and you can change the grid using `panel.grid.major`, `panel.grid.minor`, and `element_line()`.

To save a chart, you can use the `ggsave()` function (Figure 5-33):

```
> ggsave("D:/density.png");
Saving 3.95 x 3.3 in image
```

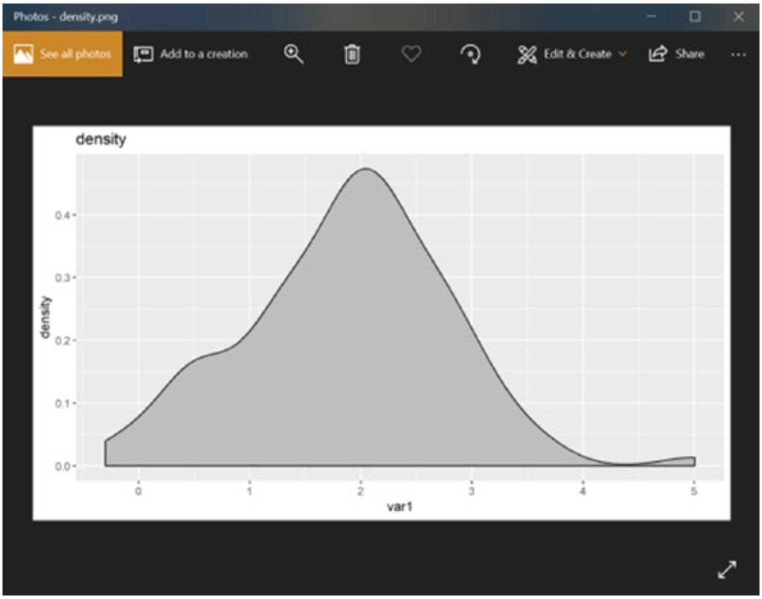


Figure 5-33. *Exported Chart to PNG file*

Interactive Charts with Plotly and ggplot2

Plotly JS allows you to create interactive, publication-quality charts. You can create a Plotly chart using ggplot. To use Plotly or to create Plotly chart, you must download the plotly package as follows:

```
> install.packages("plotly");
Installing package into 'C:/Users/gohmi/Documents/R/win-
library/3.5'
(as 'lib' is unspecified)
also installing the dependencies 'httpuv', 'xtable',
'sourcetools', 'mime', 'openssl', 'yaml', 'shiny', 'later',
'httr', 'jsonlite', 'base64enc', 'htmltools', 'htmlwidgets',
'tidyr', 'hexbin', 'crosstalk', 'data.table', 'promises'
```

```
  There is a binary version available but
  the source version is later:
            binary source needs_compilation
htmlwidgets    1.2    1.3             FALSE

trying URL 'https://cran.rstudio.com/bin/windows/contrib/3.5/
httpuv_1.4.5.zip'
Content type 'application/zip' length 1182084 bytes (1.1 MB)
downloaded 1.1 MB

trying URL 'https://cran.rstudio.com/bin/windows/contrib/3.5/
xtable_1.8-3.zip'
Content type 'application/zip' length 755944 bytes (738 KB)
downloaded 738 KB

trying URL 'https://cran.rstudio.com/bin/windows/contrib/3.5/
sourcetools_0.1.7.zip'
Content type 'application/zip' length 530521 bytes (518 KB)
downloaded 518 KB

trying URL 'https://cran.rstudio.com/bin/windows/contrib/3.5/
mime_0.5.zip'
Content type 'application/zip' length 46959 bytes (45 KB)
downloaded 45 KB

trying URL 'https://cran.rstudio.com/bin/windows/contrib/3.5/
openssl_1.0.2.zip'
Content type 'application/zip' length 3628608 bytes (3.5 MB)
downloaded 3.5 MB

trying URL 'https://cran.rstudio.com/bin/windows/contrib/3.5/
yaml_2.2.0.zip'
Content type 'application/zip' length 203553 bytes (198 KB)
downloaded 198 KB
```

To use Plotly, you must call the library using the library() or require() functions:

```
> library(plotly);

Attaching package: 'plotly'

The following object is masked from 'package:ggplot2':

    last_plot

The following object is masked from 'package:igraph':

    groups

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

    filter

The following object is masked from 'package:graphics':

    layout
```

To create a Plotly chart (Figure 5-34), you can use the ggplotly() function:

```
> set.seed(12);
> var1 <- rnorm(100, mean=1, sd=1);
> var2 <- rnorm(100, mean=2, sd=1);
> var3 <- rnorm(100, mean=1, sd=2);
> data <- data.frame(var1, var2, var3);
> gg <- ggplot(data) + geom_line(aes(x=var1, y=var2));
> g <- ggplotly(gg);
> g;
```

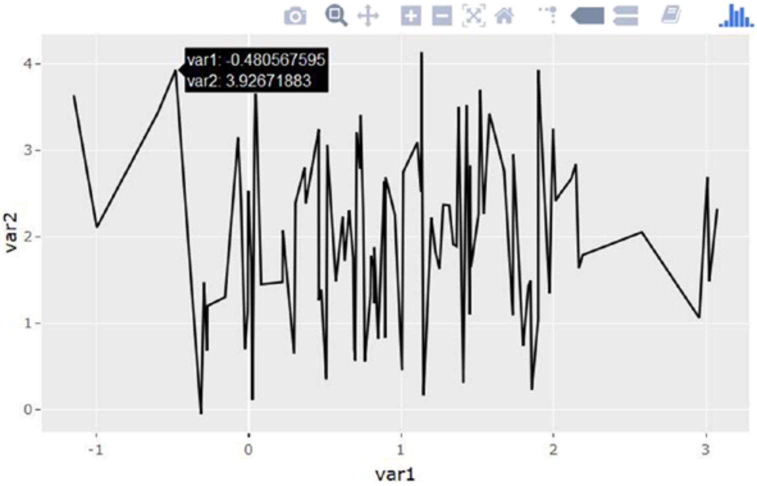


Figure 5-34. *Plotly Chart using R*

To save the Plotly chart, you must create a free Plotly account. Then you can use the following code to save the R session:

```
Sys.setenv("plotly_username"="your plotly username");
Sys.setenv("plotly_api_key"="your api key");
```

To publish the graph, you use

```
api_create(g, filename="");
```

Conclusion

In this chapter, you looked into R programming. You now understand that descriptive statistics summarizes the data and usually focuses on the distribution, the central tendency, and the dispersion of the data. Data visualization, on the other hand, creates graphics from the data to help us understand the data.

You learned how R programming allows you to plot line charts, bar charts, histograms, scatterplots, scatterplot matrices, pie charts, and box plots.

You also learned about decision trees, which are machine learning algorithms that perform both classification and regression tasks. They can be a graph to represent choices and results using a tree.

You also learned about social network analysis graphs, which can help us understand the relationships between individuals or nodes. Social network analysis is usually used on social network data like Facebook and Weibo. Each node is an individual, and the social network graphs show us how each individual connects to others.

You also explored ggplot2, a package created by Hadley Wickham that offers a powerful graphics language for creating of advanced charts. ggplot2 is very popular and famous in the R community, and it allows us to create charts that represents univariate, multivariate, and categorical data in a straightforward way.

You also learned the grammar of graphics. ggplot2 focuses on the grammar of graphics. The grammar of graphics is the building blocks of a chart.

You also explored Plotly JS. Plotly JS allows us to create interactive, publication-quality charts. You created a Plotly chart using ggplot.

References

An Introduction to ggplot2. (n.d.). Retrieved from https://uc-r.github.io/ggplot_intro.

Data visualization. (2018, August 27). Retrieved from https://en.wikipedia.org/wiki/Data_visualization.

Getting Started with Plotly and ggplot2. (n.d.). Retrieved from https://plot.ly/ggplot2/getting-started/.

ggplot2 . (n.d.). Retrieved September 1, 2018, from https://tutorials.iq.harvard.edu/R/Rgraphics/Rgraphics.html.

ggplot2 - Essentials. (n.d.). Retrieved from www.sthda.com/english/wiki/ggplot2-essentials.

Histogram Density Plot Combo in R. (2012, September 27). Retrieved from www.r-bloggers.com/histogram-density-plot-combo-in-r/.

Kabacoff, R. (n.d.). Bar Plots. Retrieved from www.statmethods.net/graphs/bar.html.

Kabacoff, R. (n.d.). Histograms and Density Plots. Retrieved from www.statmethods.net/graphs/density.html.

Kabacoff, R. (n.d.). Scatterplots. Retrieved from www.statmethods.net/graphs/scatterplot.html.

Kabacoff, R. (n.d.). Graphics with ggplot2. Retrieved from www.statmethods.net/advgraphs/ggplot2.html.

Network analysis with R and igraph: NetSci X Tutorial - Katya Ognyanova. (2018, July 11). Retrieved from http://kateto.net/networks-r-igraph.

T. (n.d.). R Bar Charts. Retrieved from www.tutorialspoint.com/r/r_bar_charts.htm.

T. (n.d.). R Boxplots. Retrieved from www.tutorialspoint.com/r/r_boxplots.htm.

T. (n.d.). R Line Graphs. Retrieved from www.tutorialspoint.com/r/r_line_graphs.htm.

T. (n.d.). R Pie Charts. Retrieved from www.tutorialspoint.com/r/r_pie_charts.htm.

T. (n.d.). R Scatterplots. Retrieved from www.tutorialspoint.com/r/r_scatterplots.htm.

R: Histograms. (n.d.). Retrieved from https://stat.ethz.ch/R-manual/R-devel/library/graphics/html/hist.html.

The Fundamentals of ggplot2 Explained. (n.d.). Retrieved from www.aridhia.com/technical-tutorials/the-fundamentals-of-ggplot-explained/.

Titanic: Getting Started With R - Part 3: Decision Trees. (2014, January 13). Retrieved from http://trevorstephens.com/kaggle-titanic-tutorial/r-part-3-decision-trees/.

CHAPTER 6

Inferential Statistics and Regressions

Inferential statistics and descriptive statistics are the main branches of statistics. Descriptive statistics derives a summary from the data set and makes use of central tendency, dispersion, and skewness. Inferential statistics describes and makes inferences about the population from the sampled data. In inferential statistics, you use hypothesis testing and estimating of parameters. Regression analysis is a set of statistical processes to estimate the relationships between all the variables.

In this chapter, you will look into the `apply()`, `lapply()`, and `sapply()` functions and then you'll sample data and perform correlations and covariances plus tests such as p-value, t-test, chi-square test, and ANOVA. You will then look into non-parametric tests, which include the Wilcoxon signed rank test, Wilcoxon-Mann-Whitney test, the Kruskal-Wallis test, and simple linear regression and multiple linear regression analysis.

What Are Inferential Statistics and Regressions?

Inferential statistics and descriptive statistics are the two main branches of statistics. Descriptive statistics derives a summary from the data by using central tendencies like mean and median, dispersions like variance and standard deviation, and skewness and kurtosis.

E. G. M. Hui, *Learn R for Applied Statistics*, https://doi.org/10.1007/978-1-4842-4200-1_6

Inferential statistics describes and makes inferences about the population and the sampled data. In inferential statistics, you use hypothesis testing and estimating of parameters. By estimating parameters, you try to answer the population parameters. In hypothesis testing, you try to answer a research question. In hypothesis testing, a research question is a hypothesis asked in question format. A research question can be, *Is there a significant difference between the grades of class 1 and class 2 for their engineering math exams?* A hypothesis can be, *There is a significant difference between the grades of class 1 and class 2 for their engineering math exams.* The research question begins with *Is there* and the hypothesis begins with *There is*. Based on the research question, the hypothesis can be a null hypothesis, H_0, and an alternate hypothesis, H_a. A null hypothesis, H_0, can be $\mu_1 = u_2$ and an alternate hypothesis, H_a, can be $\mu_1 \neq u_2$. So μ_1 is the mean of the grades of class 1 and μ_2 is the mean of the grades of class 2. You can then use inference tests to get the p-value. If the p-value is less than or equal to alpha, which is usually 0.05, you reject the null hypothesis and say that the alternate hypothesis is true at the 95% confidence interval. If the p-value is more than 0.05, you fail to reject the null hypothesis.

For estimating parameters, the parameters can be the mean, variance, standard deviation, and others. If you want to estimate the mean of heights of the whole population (and by the way, it is impossible to measure everyone in the population), you can use a sampling method to select some people from the population. Subsequently, you calculate the mean of the height of the samples and then make an inference on the mean of the height of the population. You can then construct the confidence intervals, which is the range in which the mean of the height of the population will fall. You construct a range because the sample cannot derive the exact mean of the height of the population.

Regression analysis is a set of statistical processes to estimate the relationships between all the variables. To be more specific, regression analysis is used to understand the relationships among independent variables and dependent variables and to explore the forms of the relationships.

apply(), lapply(), sapply()

The `apply()` function can perform a loop to go through the data and apply a function. The function can be the `mean()` function from R or it can be a customized function. The use of the `apply()` function is to avoid the use of loops. The `apply()` function can take list, matrix, or array.

To use the `apply()` function, you create random data:

```
> set.seed(123);
> var1 <- rnorm(100, mean=2, sd=1);
> var2 <- rnorm(100, mean=3, sd=1);
> var3 <- rnorm(100, mean=3, sd=2);
> data <- data.frame(var1, var2, var3);
> data;
           var1      var2        var3
1    1.43952435 2.2895934  7.39762070
2    1.76982251 3.2568837  5.62482595
3    3.55870831 2.7533081  2.46970989
4    2.07050839 2.6524574  4.08638812
5    2.12928774 2.0483814  2.17132010
6    3.71506499 2.9549723  2.04750621
7    2.46091621 2.2150955  1.42279432
8    0.73493877 1.3320581  1.81076547
9    1.31314715 2.6197735  6.30181493
10   1.55433803 3.9189966  2.89194375
```

You can use the `apply()` function with a `mean()` function:

```
> apply(data, 1, mean);
  [1] 3.708913 3.550511 2.927242 2.936451 2.116330 2.905848
      2.032935 1.292587 3.411579 2.788426
 [11] 2.962408 3.151718 3.082613 2.340998 1.992851 4.479820
      2.573733 1.315515 1.793035 1.311216
 [21] 1.967292 2.690174 2.901045 2.810064 2.830503 1.926952
      2.554660 2.265633 2.556436 2.383774
 [31] 4.593868 2.658598 3.121812 2.326194 1.873185 2.395315
      2.242476 3.172000 3.417250 1.537521
 [41] 2.143278 2.175163 2.718194 2.126560 2.416108 3.383570
      1.977803 1.833860 3.183512 2.533507
 [51] 2.763284 2.538914 2.533833 2.847073 3.617265 3.019648
      3.058612 3.157894 2.957849 1.591852
 [61] 2.796705 1.822920 2.166982 4.274287 3.699170 3.898973
      2.939493 1.352056 2.887190 3.532488
 [71] 2.994538 2.560393 3.446763 2.209567 2.756647 2.345487
      2.700874 2.412955 3.158053 2.484031
 [81] 1.202496 3.707153 2.683914 2.415853 2.593174 2.800894
      3.549599 3.933870 2.663340 2.995215
 [91] 3.848239 3.444024 3.541280 1.773932 4.018272 3.176785
      4.840582 1.859845 2.398367 2.762643
```

The above data is the result of the mean of every row. `data` is the data, `1` is the margin, and `mean` is the function. The margin means that the function will be applied to a column when it is 2 and a row when it is 1.

You can get the mean of each column by using `margin=2`:

```
> apply(data, 2, mean);
    var1     var2     var3
2.090406 2.892453 3.240930
```

apply() and lapply() are very similar, just that the output of lapply() is a list. You can use lapply() as follows:

```
> lapply(data$var1, mean);
[[1]]
[1] 1.439524

[[2]]
[1] 1.769823

[[3]]
[1] 3.558708

[[4]]
[1] 2.070508

[[5]]
[1] 2.129288
```

sapply() is very similar to lapply(), just that the output is a vector:

```
> sapply(data$var1, mean);
  [1]  1.43952435  1.76982251  3.55870831  2.07050839
       2.12928774  3.71506499  2.46091621
  [8]  0.73493877  1.31314715  1.55433803  3.22408180
       2.35981383  2.40077145  2.11068272
 [15]  1.44415887  3.78691314  2.49785048  0.03338284
       2.70135590  1.52720859  0.93217629
 [22]  1.78202509  0.97399555  1.27110877  1.37496073
       0.31330669  2.83778704  2.15337312
 [29]  0.86186306  3.25381492  2.42646422  1.70492852
       2.89512566  2.87813349  2.82158108
 [36]  2.68864025  2.55391765  1.93808829  1.69403734
       1.61952900  1.30529302  1.79208272
 [43]  0.73460365  4.16895597  3.20796200  0.87689142
       1.59711516  1.53334465  2.77996512
 [50]  1.91663093  2.25331851  1.97145324  1.95712954
       3.36860228  1.77422901  3.51647060
 [57]  0.45124720  2.58461375  2.12385424  2.21594157
       2.37963948  1.49767655  1.66679262
 [64]  0.98142462  0.92820877  2.30352864  2.44820978
       2.05300423  2.92226747  4.05008469
 [71]  1.50896883 -0.30916888  3.00573852  1.29079924
       1.31199138  3.02557137  1.71522699
 [78]  0.77928229  2.18130348  1.86110864  2.00576419
       2.38528040  1.62933997  2.64437655
 [85]  1.77951344  2.33178196  3.09683901  2.43518149
       1.67406841  3.14880762  2.99350386
 [92]  2.54839696  2.23873174  1.37209392  3.36065245
       1.39974041  4.18733299  3.53261063
 [99]  1.76429964  0.97357910
```

Sampling

Sampling is the selection of a subset of a population. The population is the data from every member. Sometimes a sample can be a subset from a full data set. The advantages of sampling are that the cost is lower and data collection is more efficient than collecting the data from every member in the population.

Simple Random Sampling

Simple random sampling (SRS) selects elements from the full data set randomly. Each element has the same probability of being selected, and each subset has the same probability of being selected as other subsets.

Stratified Sampling

Stratified sampling is when you divide the population into groups based on a factor or characteristics or some factors. These groups are called strata, and an individual group is called a stratum. In stratified sampling, you do the following:

1. Divide the population into groups.
2. Use simple random sampling on each group.
3. Collect data from each sampling unit.

Stratified sampling works well when a heterogeneous population is split into homogeneous groups.

Cluster Sampling

Cluster sampling is different from stratified sampling. It should be done in the following way:

1. Divide the population into clusters.
2. Use random sampling on clusters from all possible clusters.
3. Collect data from a sampling unit.

Unlike stratified sampling, cluster sampling all members of the selected clusters forms a sample, while in stratified sampling, a sample is based on the random selection of members from all the strata. For example, say 13 colleges are the strata and clusters. In stratified sampling, 30 students from each college or strata are selected using random sampling. In cluster sampling, all students in 5 out of 13 colleges are selected using random sampling.

You can do random sampling using the sample() function:

```
> set.seed(123);
> var1 <- rnorm(100, mean=2, sd=1);
> var2 <- rnorm(100, mean=3, sd=1);
> var3 <- rnorm(100, mean=3, sd=2);
> data <- data.frame(var1, var2, var3);
> sample(data$var1, 5, replace=TRUE);
[1] 1.27110877 2.92226747 0.97399555 1.70492852 0.03338284
```

data$var1 is the data, 5 is the number of items to select from, and replace=TRUE means that the chosen item can be repeated. If replace=FALSE, the chosen item cannot be repeated.

You can do stratified sampling using the dplyr library. You must install the dplyr library before using it:

```
> install.packages("dplyr");
Installing package into 'C:/Users/gohmi/Documents/R/win-
library/3.5'
(as 'lib' is unspecified)
also installing the dependencies 'fansi', 'utf8', 'bindr',
'cli', 'crayon', 'pillar', 'purrr', 'assertthat', 'bindrcpp',
'glue', 'magrittr', 'pkgconfig', 'R6', 'Rcpp', 'rlang',
'tibble', 'tidyselect', 'BH', 'plogr'
...
The downloaded binary packages are in
    C:\Users\gohmi\AppData\Local\Temp\RtmpaizS1C\downloaded_
    packages
```

You can load the iris data using

```
> data(iris);
> summary(iris);
```

```
Sepal.Length    Sepal.Width     Petal.Length    Petal.Width     Species
Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100   setosa    :50
1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300   versicolor:50
Median :5.800   Median :3.000   Median :4.350   Median :1.300   virginica :50
Mean   :5.843   Mean   :3.057   Mean   :3.758   Mean   :1.199
3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100   3rd Qu.:1.800
Max.   :7.900   Max.   :4.400   Max.   :6.900   Max.   :2.500
```

There are 50 setosa data, 50 versicolor data, and 50 virginica data.

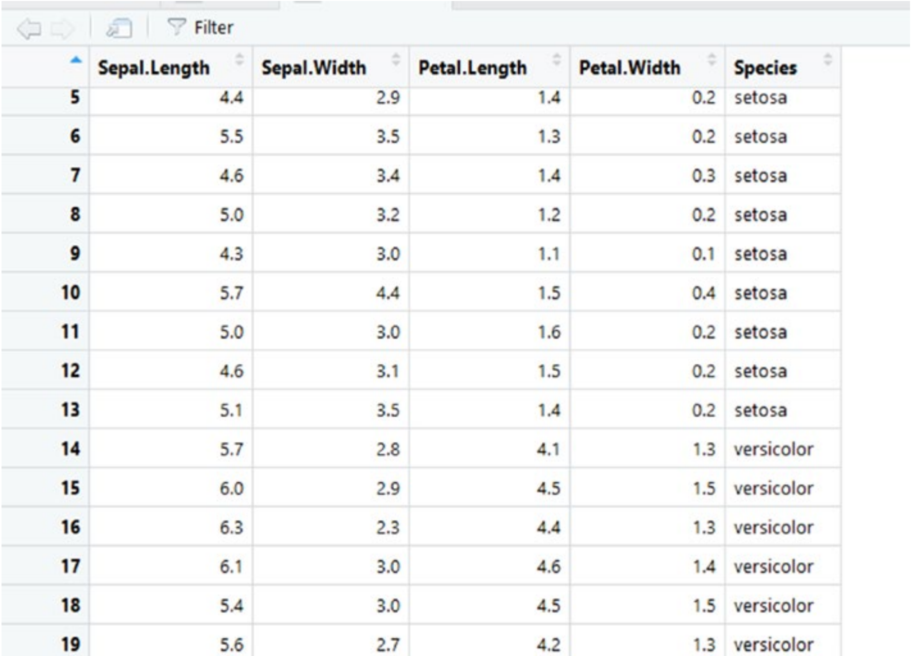
You can load the dplyr library using

```
> library(dplyr);

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
```

You can do stratified clustering using dplyr:

```
 > iris_sample <- iris %>%
+ group_by(Species) %>%
+ sample_n(13)
> iris_sample;
# A tibble: 39 x 5
# Groups:   Species [3]
   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
          <dbl>       <dbl>        <dbl>       <dbl> <fct>
 1          5           3.5          1.3         0.3 setosa
 2          5           3.4          1.5         0.2 setosa
 3          5.1         3.4          1.5         0.2 setosa
 4          5.7         4.4          1.5         0.4 setosa
 5          5.1         3.5          1.4         0.3 setosa
 6          5.2         3.4          1.4         0.2 setosa
 7          5           3.6          1.4         0.2 setosa
 8          5.1         3.5          1.4         0.2 setosa
 9          4.5         2.3          1.3         0.3 setosa
10          5.1         3.3          1.7         0.5 setosa
# ... with 29 more rows
> View(iris_sample);
```

sample_n(13) selects 13 items from each group. group_by(Species) means you group the data by species variable. See Figure 6-1.



	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
5	4.4	2.9	1.4	0.2	setosa
6	5.5	3.5	1.3	0.2	setosa
7	4.6	3.4	1.4	0.3	setosa
8	5.0	3.2	1.2	0.2	setosa
9	4.3	3.0	1.1	0.1	setosa
10	5.7	4.4	1.5	0.4	setosa
11	5.0	3.0	1.6	0.2	setosa
12	4.6	3.1	1.5	0.2	setosa
13	5.1	3.5	1.4	0.2	setosa
14	5.7	2.8	4.1	1.3	versicolor
15	6.0	2.9	4.5	1.5	versicolor
16	6.3	2.3	4.4	1.3	versicolor
17	6.1	3.0	4.6	1.4	versicolor
18	5.4	3.0	4.5	1.5	versicolor
19	5.6	2.7	4.2	1.3	versicolor

Figure 6-1. *Stratified Sampling - Selected 13 sample from each groups*

Correlations

Correlations are statistical associations to find how close two variables are and to derive the linear relationships between them. In predictive analytics, you can use correlation to find which variables are more related to the target variable and use this to reduce the number of variables. Correlation does not mean a causal relationship. Correlation finds how close two variables are, but does not tell you the how and why of the relationship. Causation tells you that one variable change will cause another variable to change.

The formula for correlation is

$$r_{xy} = \frac{\sum_{i=1}^{n}(x_i - \overline{x})(y_i - \overline{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \overline{x})^2 \sum_{i=1}^{n}(y_i - \overline{y})^2}}$$

where $(x_i - \overline{x})^2$ is the x value minus the mean and then square it, and $(y_i - \overline{y})$ $(y_i - \overline{y})^2$ is the y value minus the mean and then square it.

To get the correlation, you generate sample data first:

```
> View(iris_sample);
> set.seed(123);
> var1 <- rnorm(100, mean=2, sd=1);
> var2 <- rnorm(100, mean=3, sd=1);
> var3 <- rnorm(100, mean=3, sd=2);
> data <- data.frame(var1, var2, var3);
```

You can use the `cor()` function to get the correlation:

```
> cor(data$var1, data$var2);
[1] -0.04953215
```

The correlation has a range from -1.0 to 1.0. When the correlation is 0, there is no correlation or relationship. When the correlation is more than 0, it is a positive relationship. Positive correlation means that when one variable value increases, the other variable values also increase. When the correlation is less than tk, it is a negative relationship. Negative correlation means that when one variable increases, the other variables' values decrease. 1 is the perfect positive correlation and -1 is the perfect negative correlation. Hence, the larger the value towards 1, or the smaller the values towards -1, the better the relationship.

-0.04953215 means that the correlation is a negative relationship between var 1 and var2. The correlation is close to zero, so the relationship is not good.

Covariance

Covariance is a measure of variability between two variables. The greater the value of one variable and the greater of other variable means it will result in a covariance that is positive. The greater value of one variable to the lesser value of the other variable will result in a negative covariance. Covariance shows the linear relationship between both variables, but the covariance magnitude is difficult to interpret. Correlation is the normalized version of covariance so

$$\text{cov}\left(X,Y\right)=\sum_{i=1}^{N}\frac{\left(x_i-\overline{x}\right)\left(y_i-\overline{y}\right)}{N}$$

where $\left(x_i-\overline{x}\right)$ is the x value minus the mean and $\left(y_i-\overline{y}\right)$ is the y value minus the mean.

To get a covariance, create sample data:

```
> set.seed(123);
> var1 <- rnorm(100, mean=2, sd=1);
> var2 <- rnorm(100, mean=3, sd=1);
> var3 <- rnorm(100, mean=3, sd=2);
> data <- data.frame(var1, var2, var3);
```

You can use the cov() function to get the covariance:

```
> cov(data$var1, data$var2);
[1] -0.04372107
```

Correlation has a range of -1 to 1. Covariance does not have a range. Correlation is good for measuring how good the relationship between two variables is. When two variables have a positive covariance, when one variable increases, the other variable increases. When two variables have a negative covariance, when one variable increases, the other variable decreases. When two variables are independent of each other, the covariance is zero. -0.04372107 means the covariance is negative, and it is very close to zero, so the relationship between the two variables is not very good.

Correlation and covariance are usually within descriptive statistics.

Hypothesis Testing and P-Value

I mentioned hypothesis testing previously. In hypothesis testing, a research question is a hypothesis asked in question format. A research question can be, *Is there a significant difference between something?* A hypothesis can be, *There is a significant difference between something*. The research question begins with *Is there* and the hypothesis begins with *There is*. A hypothesis can also be a null hypothesis, H_0, and an alternate hypothesis, H_a. You can write the null hypothesis and alternate hypothesis as follows:

$$H_0\colon \mu_1 = u_2$$

$$H_a\colon \mu_1 \neq u_2$$

where μ_1 is the mean of one data, and μ_2 is the mean of another data.

You can use statistical tests to get your p-value. You use a t-test for continuous variables or data, and you use a chi-square test for categorical variables or data. For more complex testing, you use ANOVA. If data is not normally distributed, you use non-parametric tests. A P-value helps you determine the significance of your statistical test results. Your claim in the test is known as a null hypothesis and the alternate hypothesis means that you believe the null hypothesis is untrue.

- A small p-value <= alpha, which is usually 0.05, indicates that the observed data is sufficiently inconsistent with the null hypothesis, so the null hypothesis may be rejected. The alternate hypothesis is true at the 95% confidence interval.

- A larger p-value means that you failed to reject null hypothesis.

T-Test

A t-test is one of the more important tests in statistics. A t-test is used to determine whether the mean between two data points or samples are equal to each other. The null hypothesis means that the two means are equal, and the alternative means that the two means are different.

Types of T-Tests

Figure 6-2 shows the types of t-tests.

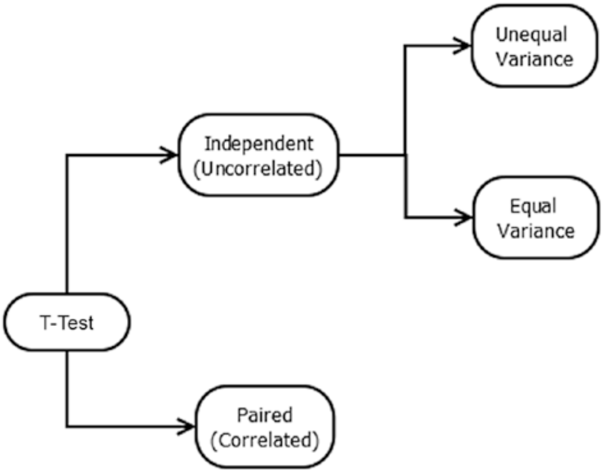


Figure 6-2. *Types of T-Tests*

Assumptions of T-Tests

Here are the assumptions:

- The samples are randomly sampled from their population.
- The population is normally distributed.

Type I and Type II Errors

A type I error is a rejection of the null hypothesis when it is really true. A type II error is a failure to reject a null hypothesis that is false.

One-Sample T-Test

A one-sample t-test is used to test whether the mean of a population is equal to a specified mean.

The formula of a one-sample t-test is

$$t = \frac{m - \mu}{s / \sqrt{n}}$$

where s is the standard deviation of the sample, n is the size of the sample, m is the mean of the sample, and u is the specified mean.

The degree of freedom formula is

$$df = n - 1$$

You can use the t statistics and the degree of freedom to estimate the p-value using a t-table.

To use a one-sample t-test in R, you can use the `t.test()` function:

```
> set.seed(123);
> var1 <- rnorm(100, mean=2, sd=1);
> var2 <- rnorm(100, mean=3, sd=1);
> var3 <- rnorm(100, mean=3, sd=2);
> data <- data.frame(var1, var2, var3);
> t.test(data$var1, mu=0.6);

    one sample t-test

data:  data$var1
t = 16.328, df = 99, p-value < 2.2e-16
alternative hypothesis: true mean is not equal to 0.6
95 percent confidence interval:
 1.909283 2.271528
sample estimates:
mean of x
 2.090406
```

In a one-sample t-test,

$$H_0: \mu = m$$

$$H_a: \mu \neq \mathrm{m}$$

`m` is 0.6 in the above R code. The p-value is 2.2e-16, so the p-value is less than 0.05, which is the alpha value. Therefore, the null hypothesis may be rejected. The alternate hypothesis, $\mu \neq 0.6$, is true at the 95% confidence interval.

Two-Sample Independent T-Test

The two-sample unpaired t-test is when you compare two means of two independent samples. The formula of the two-sample independent t-test is

$$t = \frac{\mu_A - \mu_B}{\sqrt{\frac{s^2}{n_A} + \frac{s^2}{n_B}}}$$

where

μ_A is the mean of one sample,

μ_B is the mean of the second sample,

n_A is the size of sample A, and

n_B is the size of sample B.

tk is the estimator of the common variance of the two samples, and the formula is

$$s^2 = \frac{\sum(x - \mu_A)^2 + \sum(x - \mu_B)^2}{n_A - n_B - 2}$$

The degrees of freedom formula is

$$df = n_A - n_B - 2$$

To use a two-sample unpaired t-test with a variance as equal in R:

```
> set.seed(123);
> var1 <- rnorm(100, mean=2, sd=1);
> var2 <- rnorm(100, mean=3, sd=1);
> var3 <- rnorm(100, mean=3, sd=2);
> data <- data.frame(var1, var2, var3);
> t.test(data$var1, data$var2, var.equal=TRUE, paired=FALSE);
```

```
    two-sample t-test

data:  data$var1 and data$var2
t = -6.0315, df = 198, p-value = 7.843e-09
alternative hypothesis: true difference in means is not equal
to 0
95 percent confidence interval:
 -1.0642808 -0.5398138
sample estimates:
mean of x mean of y
 2.090406  2.892453
```

The two-sample independent t-test:

$$H_0: \mu_A - \mu_B = 0$$

$$H_a: \mu_A - \mu_B \neq 0$$

The p-value is 7.843e-09, so it is less than 0.05, which is the alpha value. Therefore, the null hypothesis may be rejected. The alternate hypothesis, $\mu_A - \mu_B \neq 0$, is true at the 95% confidence interval.

In the two-sample unpaired t-test, when the variance is unequal, you use the Welch t-test. You can assume the two data variances are different, or you can calculate the variance of each data. The Welch t-test formula is as follows:

$$t = \frac{\mu_A - \mu_B}{\sqrt{\frac{s_A^2}{n_A} + \frac{s_B^2}{n_B}}}$$

where

μ_A is the mean of sample A,

μ_B is the mean of sample B,

n_A is the sample size of A,

n_B is the sample size of B,

s_A is the standard deviation of A, and

s_B is the standard deviation of B.

Unlike the normal t-test formula, the Welch t-test formula involves the variance of each sample.

The degrees of freedom formula of the Welch t-test formula is as follows:

$$df = {\left(\frac{s_A^2}{n_A} + \frac{s_B^2}{n_B^2} \right)} \Big/ {\left(\frac{s_A^4}{n_A^2 \left(n_B - 1 \right)} + \frac{s_B^4}{n_B^2 \left(n_B - 1 \right)} \right)}$$

To use the two-sample unpaired t-test with a variance as unequal in R:

```
> set.seed(123);
> var1 <- rnorm(100, mean=2, sd=1);
> var2 <- rnorm(100, mean=3, sd=1);
> var3 <- rnorm(100, mean=3, sd=2);
> data <- data.frame(var1, var2, var3);
> t.test(data$var1, data$var2, var.equal=FALSE, paired=FALSE);

    Welch two-sample t-test

data:  data$var1 and data$var2
t = -6.0315, df = 197.35, p-value = 7.88e-09
alternative hypothesis: true difference in means is not equal
to 0
95 percent confidence interval:
 -1.0642862 -0.5398084
sample estimates:
mean of x mean of y
 2.090406  2.892453
```

The two-sample unpaired t-test:

$$H_0: \mu_A - \mu_B = 0$$

$$H_a: \mu_A - \mu_B \neq 0$$

The p-value is 7.88e-09, so it less than 0.05, which is the alpha value. Therefore, the null hypothesis may be rejected. The alternate hypothesis, $\mu_A - \mu_B \neq 0$, is true at the 95% confidence interval.

Two-Sample Dependent T-Test

A two-sample paired t-test is used to test the mean of two samples that depend on each other. The t-test formula is

$$t = \frac{\bar{d}}{\sqrt{s^2 / n}}$$

where
$\bar{d}$ is the mean difference,
s is the sample variance, and
n is the sample size.
The degree of freedom formula is

$$df = n - 1$$

To use the two-sample paired t-test in R:

```
> set.seed(123);
> var1 <- rnorm(100, mean=2, sd=1);
> var2 <- rnorm(100, mean=3, sd=1);
> var3 <- rnorm(100, mean=3, sd=2);
> data <- data.frame(var1, var2, var3);
> t.test(data$var1, data$var2, paired=TRUE);
```

```
    Paired t-test

data:  data$var1 and data$var2
t = -5.8876, df = 99, p-value = 5.379e-08
alternative hypothesis: true difference in means is not equal
to 0
95 percent confidence interval:
 -1.0723482 -0.5317464
sample estimates:
mean of the differences
             -0.8020473
```

The two-sample paired t-test:

$$H_0: \mu_A - \mu_B = 0$$

$$H_a: \mu_A - \mu_B \neq 0$$

The p-value is 5.379e-08, so it is less than 0.05, which is the alpha value. Therefore, the null hypothesis may be rejected. The alternate hypothesis, $\mu_A - \mu_B \neq 0$, is true at the 95% confidence interval.

Chi-Square Test

The chi-square test is used to compare the relationships between two categorical variables. The null hypothesis means that there is no relationship between the categorical variables.

Goodness of Fit Test

When you have only one categorical variable from a population and you want to compare whether the sample is consistent with a hypothesized distribution, you can use the goodness of fit test.

The null hypothesis means that the data followed a specified distribution, and the alternate hypothesis means that data doesn't follow the specified distribution.

The goodness of fit formula is

$$X^2 = \sum_{i=1}^{n} \frac{\left(O_i - E_i\right)^2}{E_i}$$

where O_i is the observed frequency of bin *i* and E_i is the expected frequency of bin *i*.

To calculated the expected frequency, the formula is

$$E_i = N \cdot p_i$$

where N is the total sample size and p_i is the hypothesized proportion of the observations of bin *i*.

To use the goodness of fit chi-square test in R, you can use the `chisq.test()` function:

```
> data <- c(B=200, c=300, D=400);
> chisq.test(data);

    Chi-squared test for given probabilities

data:  data
X-squared = 66.667, df = 2, p-value = 3.338e-15
```

The goodness of fit chi-square test:

- H_0: No significant difference between the observed and expected values.
- H_a: There is a significant difference between the observed and expected values.

The p-value is 3.338e-15, so it is less than 0.05, which is the alpha value. Therefore, the null hypothesis may be rejected. The alternate hypothesis of a significant difference between the observed and expected values is true at the 95% confidence interval.

Contingency Test

If you have two categorical variables and you want to compare whether there is a relationship between two variables, you can use the contingency test.

The null hypothesis means that the two categorical variables have no relationship. The alternate hypothesis means that the two categorical variables have a relationship.

To calculate the expected value, use

$$E_{ij} = \frac{R_i C_j}{N}$$

where *R* is the row, *C* is the column, *N* is the total, *ith* is the row, and *jth* is the column.

The formula for X^2 statistics is

$$X^2 = \sum_{i=1}^{r} \sum_{j=1}^{c} \frac{\left(O_{ij} - E_{ij}\right)^2}{E_{ij}}$$

To use a contingency test in R, create your data first:

```
> var1 <- c("Male", "Female", "Male", "Female", "Male",
"Female", "Male", "Female", "Male", "Female");
> var2 <- c("chocolate", "strawberry", "strawberry",
"strawberry", "chocolate", "chocolate", "chocolate",
"strawberry", "strawberry", "strawberry");
```

```
> data <- data.frame(var1, var2);
> data;
     var1       var2
1    Male  chocolate
2  Female strawberry
3    Male strawberry
4  Female strawberry
5    Male  chocolate
6  Female  chocolate
7    Male  chocolate
8  Female strawberry
9    Male strawberry
10 Female strawberry
```

You can then create a table or frequency table for the variables to test:

```
> data.table <- table(data$var1, data$var2);
> data.table;

         chocolate strawberry
  Female         1          4
  Male           3          2
```

You can then use the `chisq.test()` function for the contingency test:

```
> chisq.test(data.table);

    Pearson's chi-squared test with Yates' continuity correction

data:  data.table
X-squared = 0.41667, df = 1, p-value = 0.5186

Warning message:
In chisq.test(data.table) : Chi-squared approximation may be
incorrect
```

The chi-square test:

- H_0: The two variables are independent.
- H_a: The two variables are not independent.

The p-value is 0.5186, so it is more than 0.05, which is the alpha value. Therefore, the null hypothesis fails to be rejected. The two variables are independent is true at the 95% confidence interval.

ANOVA

ANOVA is the process of testing the means of two or more groups. ANOVA also checks the impact of factors by comparing the means of different samples. In a t-test, you test the means of two samples; in a chi-square test, you test categorical attributes or variables; in ANOVA, you test more samples.

Grand Mean

In ANOVA, you use two kinds of means, sample means and a grand mean. A grand mean is the mean of all of the samples' means.

Hypothesis

In ANOVA, a null hypothesis means that the sample means are equal or do not have significant differences. The alternate hypothesis is when the sample means are not equal.

$$H_0 : \mu_1 = \mu_2 = \ldots = \mu_L \qquad \textit{Null hypothesis}$$

$$H_a : \mu_1 \neq \mu_m \qquad \textit{Alternate hypothesis}$$

Assumptions

You assume that the variables are sampled, independent, and selected or sampled from a population that is normally distributed with unknown but equal variances.

Between Group Variability

The distribution of two samples, when they overlap, their means are not significantly different. Hence, the difference between their individual mean and the grand mean is not significantly different. The group and level are different groups in the same independent variable. See Figure 6-3.

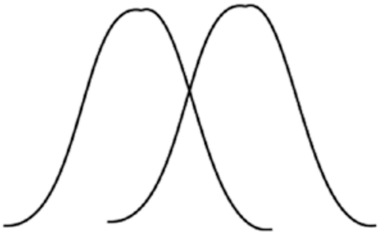


Figure 6-3. *Means are not Significantly Different*

For the two samples shown in Figure 6-4, their means are significantly different from each other. The difference between the individual means and the grand mean is significantly different.

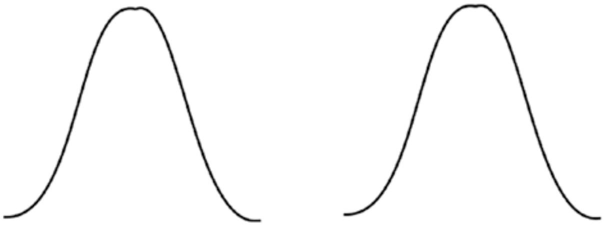


Figure 6-4. *Means are Significantly Different*

This variability is called the between-group variability, which refers to the variations between the distributions of the groups or levels. Figure 6-5 depicts the discrimination between the different groups.

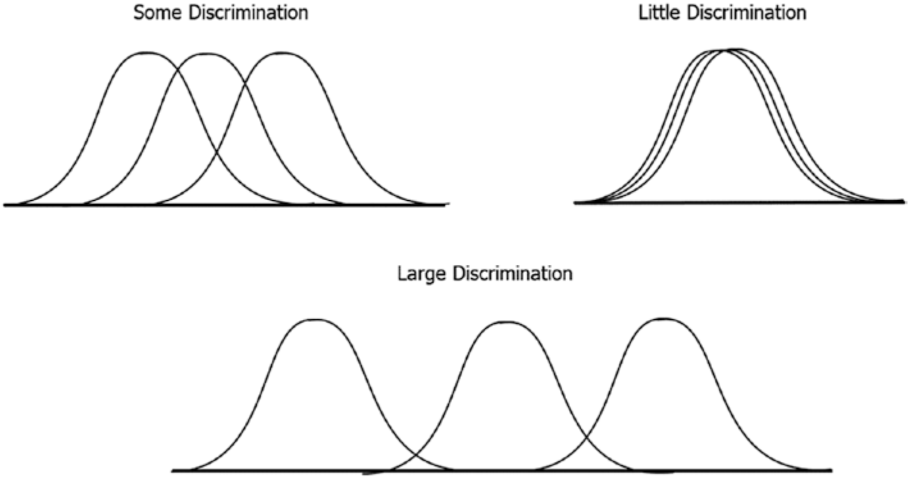


Figure 6-5. *Variations between the distributions of groups of levels*

To calculate the sum of the square of between the group variability, use

$$SS_{between} = n_1\left(\bar{x}_1 - \bar{x}_G\right)^2 + n_2\left(\bar{x}_2 - \bar{x}_G\right)^2 + n_3\left(\bar{x}_3 - \bar{x}_G\right)^2 + \ldots + n_k\left(\bar{x}_k - \bar{x}_k\right)^2$$

where
$\bar{x}_G$ is the grand mean,
$\bar{x}_1 \ldots \bar{x}_k$ is the mean of each sample, and
$n_1 \ldots n_k \ldots$ are the sample sizes.
To calculate the sum of each squared deviation, or mean square, use

$$MS_{between} = \frac{n_1\left(\bar{x}_1 - \bar{x}_G\right)^2 + n_2\left(\bar{x}_2 - \bar{x}_G\right)^2 + n_3\left(\bar{x}_3 - \bar{x}_G\right)^2 + \ldots + n_k\left(\bar{x}_k - \bar{x}_k\right)^2}{k-1}$$

You use the SS to divide by the degree of freedom, where the degree of freedom is the number of sample means(k) minus one.

Within Group Variability

For the following distributions of samples, as their variance increases, they overlap each other and become part of a population, as shown in Figure 6-6.

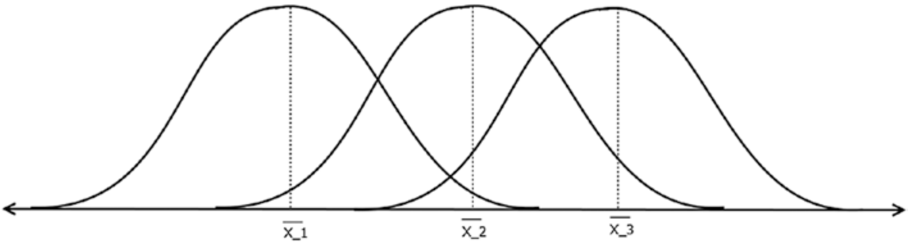


Figure 6-6. *Distributions of Samples*

Figure 6-7 shows another three samples with lesser variances, although the means are similar, they belong to different population.

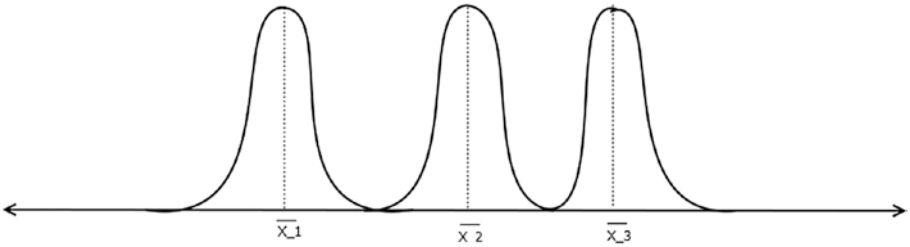


Figure 6-7. *Three samples with lesser variances*

Within-group variation refers to the variations caused by differences within individual groups or levels. To calculate the sum of squares of within-group variation, use

$$SS_{within} = \sum\left(x_{i1} - \bar{x}_1\right)^2 + \sum\left(x_{i2} - \bar{x}_2\right)^2 + \ldots + \sum\left(x_{ik} - \bar{x}_3\right)^2 = \sum\left(x_{ij} - \bar{x}_j\right)^2$$

where

x_{i1} is the *ith* value of first sample,

x_{i2} is the *ith* value of second sample, and

x_{ij} is the *jth* value from the jth sample.

The degree of freedom is

$$df_{within} = (n_1 - 1) + (n_2 - 1) + \ldots + (n_k - 1) = n_1 + n_2 + n_3 + \ldots + n_k - k(1) = N - k$$

To get the mean square of the within-group variability, you divide between group variability sum of the squares with degree of freedom within:

$$MS_{within} = \sum\left(x_{ij} - \overline{x_j}\right)^2 / (N - k)$$

The F-statistics are the measures if the means of samples are significantly different. The lower the F-statistics, the more the means are equal, so you cannot reject the null hypothesis.

$$F - statistics = \frac{Between - group\ variability}{Within - group\ variablility} = \frac{MS_{between}}{MS_{within}}$$

If the f-critical value is smaller than the f-value, reject the null hypothesis. The f-critical value can be found using F-statistics and the degree of freedom on the f distribution.

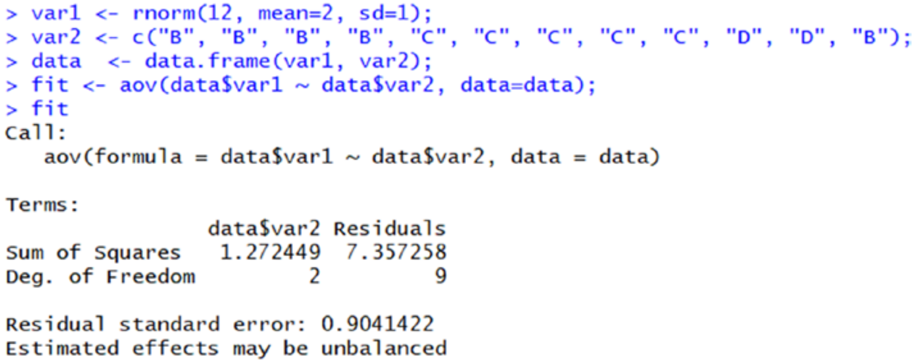
One-Way ANOVA

One-way ANOVA is used when you have only one independent variable.

In R, you can calculate the one-way ANOVA using

```
> set.seed(123);
> var1 <- rnorm(12, mean=2, sd=1);
> var2 <- c("B", "B", "B", "B", "C", "C", "C", "C", "C", "D",
  "D", "B");
```

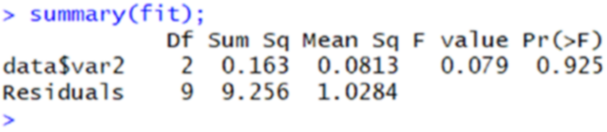
```
> data <- data.frame(var1, var2);
> fit <- aov(data$var1 ~ data$var2, data=data);
> fit;
Call:
   aov(formula = data$var1 ~ data$var2, data = data)

Terms:
                data$var2 Residuals
Sum of Squares   0.162695  9.255706
Deg. of Freedom         2         9

Residual standard error: 1.014106
Estimated effects may be unbalanced
```



```
> var1 <- rnorm(12, mean=2, sd=1);
> var2 <- c("B", "B", "B", "B", "C", "C", "C", "C", "C", "D", "D", "B");
> data  <- data.frame(var1, var2);
> fit <- aov(data$var1 ~ data$var2, data=data);
> fit
Call:
   aov(formula = data$var1 ~ data$var2, data = data)

Terms:
                data$var2 Residuals
Sum of Squares   1.272449  7.357258
Deg. of Freedom         2         9

Residual standard error: 0.9041422
Estimated effects may be unbalanced
```

To get the p-value, you use the summary() function:

```
> summary(fit);
            Df Sum Sq Mean Sq F value Pr(>F)
data$var2    2  0.163  0.0813   0.079  0.925
Residuals    9  9.256  1.0284
```



```
> summary(fit);
            Df Sum Sq Mean Sq F value Pr(>F)
data$var2    2  0.163  0.0813   0.079  0.925
Residuals    9  9.256  1.0284
>
```

$$H_0 : \mu_{var1} = \mu_{var2} \qquad \textit{Null hypothesis}$$

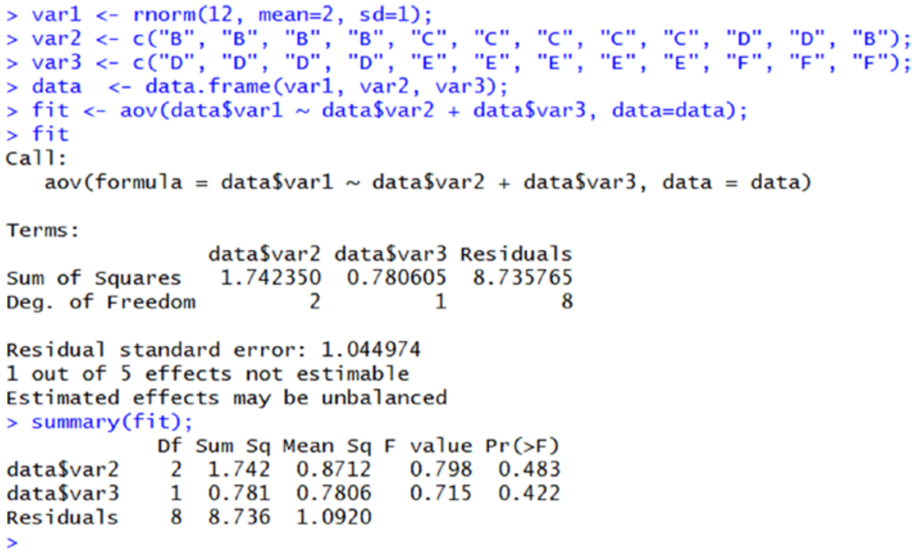
$$H_a : \mu_{var1} \neq \mu_{var2} \qquad \textit{Alternate hypothesis}$$

The p-value is more than 0.05, so you fail to reject the null hypothesis that the mean of var1 is the same as the mean of var2. The null hypothesis is true at the 95% confidence interval.

Two-Way ANOVA

Two-way ANOVA is used when you have two independent variables.

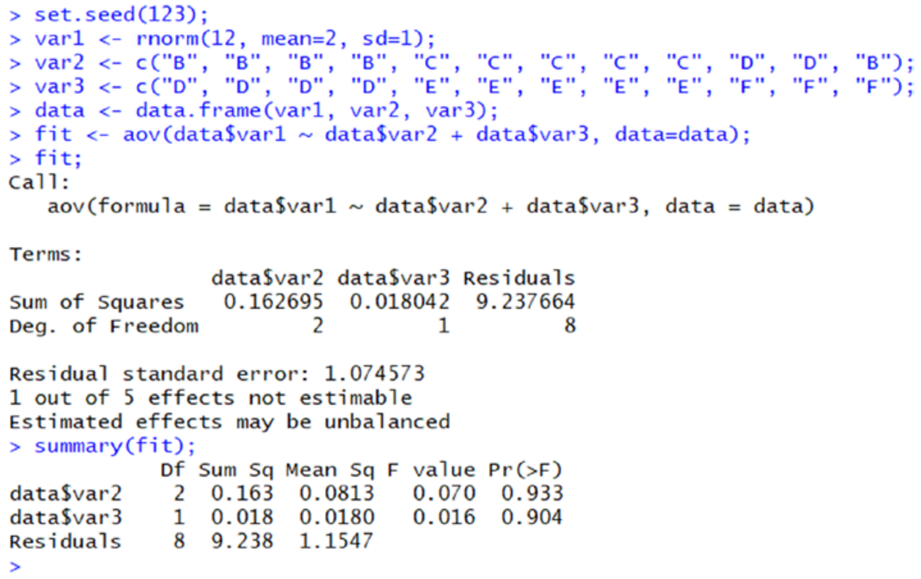
In R, you can calculate two-way ANOVA using



```
> var1 <- rnorm(12, mean=2, sd=1);
> var2 <- c("B", "B", "B", "B", "C", "C", "C", "C", "C", "D", "D", "B");
> var3 <- c("D", "D", "D", "D", "E", "E", "E", "E", "E", "F", "F", "F");
> data  <- data.frame(var1, var2, var3);
> fit <- aov(data$var1 ~ data$var2 + data$var3, data=data);
> fit
Call:
   aov(formula = data$var1 ~ data$var2 + data$var3, data = data)

Terms:
                data$var2 data$var3 Residuals
Sum of Squares   1.742350  0.780605  8.735765
Deg. of Freedom         2         1         8

Residual standard error: 1.044974
1 out of 5 effects not estimable
Estimated effects may be unbalanced
> summary(fit);
            Df Sum Sq Mean Sq F value Pr(>F)
data$var2    2  1.742  0.8712   0.798  0.483
data$var3    1  0.781  0.7806   0.715  0.422
Residuals    8  8.736  1.0920
>
```

```
> set.seed(123);
> var1 <- rnorm(12, mean=2, sd=1);
> var2 <- c("B", "B", "B", "B", "C", "C", "C", "C", "C", "D",
  "D", "B");
> var3 <- c("D", "D", "D", "D", "E", "E", "E", "E", "E", "F",
  "F", "F");
```

```
> data <- data.frame(var1, var2, var3);
> fit <- aov(data$var1 ~ data$var2 + data$var3, data=data);
> fit;
Call:
   aov(formula = data$var1 ~ data$var2 + data$var3, data = data)

Terms:
                data$var2 data$var3 Residuals
Sum of Squares   0.162695  0.018042  9.237664
Deg. of Freedom         2         1         8

Residual standard error: 1.074573
1 out of 5 effects not estimable
Estimated effects may be unbalanced
> summary(fit);
            Df Sum Sq Mean Sq F value Pr(>F)
data$var2    2  0.163  0.0813   0.070  0.933
data$var3    1  0.018  0.0180   0.016  0.904
Residuals    8  9.238  1.1547
```



```
> set.seed(123);
> var1 <- rnorm(12, mean=2, sd=1);
> var2 <- c("B", "B", "B", "B", "C", "C", "C", "C", "C", "D", "D", "B");
> var3 <- c("D", "D", "D", "D", "E", "E", "E", "E", "E", "F", "F", "F");
> data <- data.frame(var1, var2, var3);
> fit <- aov(data$var1 ~ data$var2 + data$var3, data=data);
> fit;
Call:
   aov(formula = data$var1 ~ data$var2 + data$var3, data = data)

Terms:
                data$var2 data$var3 Residuals
Sum of Squares   0.162695  0.018042  9.237664
Deg. of Freedom         2         1         8

Residual standard error: 1.074573
1 out of 5 effects not estimable
Estimated effects may be unbalanced
> summary(fit);
            Df Sum Sq Mean Sq F value Pr(>F)
data$var2    2  0.163  0.0813   0.070  0.933
data$var3    1  0.018  0.0180   0.016  0.904
Residuals    8  9.238  1.1547
>
```

$$H_0 : \mu_{var1} = \mu_{var2} = \mu_{var3} \qquad \textit{Null hypothesis}$$

$$H_a : \mu_{var1} \neq \mu_m \qquad \textit{Alternate hypothesis}$$

var1 does not depend on var2's mean and var3's mean. The var1 mean and var2 mean have p-values of 0.483, which is more than 0.05. Hence, you fail to reject the null hypothesis that the var1 mean is the same as the var2 mean. The null hypothesis is true at the 95% confidence interval. The var1 mean and the var3 mean have p-values of 0.422, which is more than 0.05. Hence, you fail to reject the null hypothesis that the var1 mean is the same as the var3 mean. The null hypothesis is true at the 95% confidence interval.

MANOVA

The multivariate analysis of variance is when there are multiple response variables that you want to test.

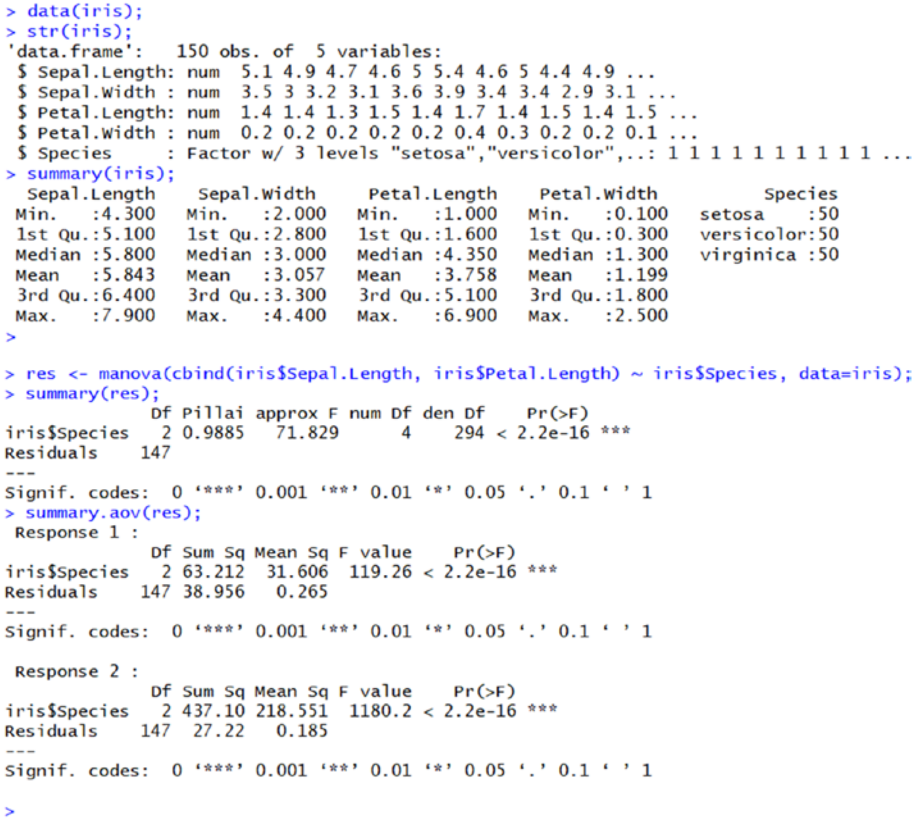
To use MANOVA in R, you can load the iris data:

```
> data(iris);
> str(iris);
'data.frame':    150 obs. of  5 variables:
 $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
 $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
 $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5
...
 $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1
...
 $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1
                 1 1 1 1 1 1 1 1 1 ...
> summary(iris);
```

```
  Sepal.Length    Sepal.Width     Petal.Length    Petal.Width          Species
 Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100   setosa    :50
 1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300   versicolor:50
 Median :5.800   Median :3.000   Median :4.350   Median :1.300   virginica :50
 Mean   :5.843   Mean   :3.057   Mean   :3.758   Mean   :1.199
 3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100   3rd Qu.:1.800
 Max.   :7.900   Max.   :4.400   Max.   :6.900   Max.   :2.500
```

```
> res <- manova(cbind(iris$Sepal.Length, iris$Petal.Length) ~
iris$Species, data=iris);
> summary(res);
              Df Pillai approx F num Df den Df    Pr(>F)
iris$Species   2 0.9885   71.829      4    294 < 2.2e-16 ***
Residuals    147
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
> summary.aov(res);
 Response 1 :
              Df Sum Sq Mean Sq F value    Pr(>F)
iris$Species   2 63.212  31.606  119.26 < 2.2e-16 ***
Residuals    147 38.956   0.265
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
 Response 2 :
                 Df Sum Sq Mean Sq F value    Pr(>F)
iris$Species      2 437.10 218.551  1180.2 < 2.2e-16 ***
Residuals       147  27.22   0.185
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```



```
> data(iris);
> str(iris);
'data.frame':	150 obs. of  5 variables:
 $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
 $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
 $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
 $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
 $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...
> summary(iris);
  Sepal.Length    Sepal.Width     Petal.Length    Petal.Width          Species  
 Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100   setosa    :50  
 1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300   versicolor:50  
 Median :5.800   Median :3.000   Median :4.350   Median :1.300   virginica :50  
 Mean   :5.843   Mean   :3.057   Mean   :3.758   Mean   :1.199                  
 3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100   3rd Qu.:1.800                  
 Max.   :7.900   Max.   :4.400   Max.   :6.900   Max.   :2.500                  
> 

> res <- manova(cbind(iris$Sepal.Length, iris$Petal.Length) ~ iris$Species, data=iris);
> summary(res);
              Df Pillai approx F num Df den Df    Pr(>F)    
iris$Species   2 0.9885   71.829      4    294 < 2.2e-16 ***
Residuals    147                                            
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
> summary.aov(res);
 Response 1 :
              Df Sum Sq Mean Sq F value    Pr(>F)    
iris$Species   2 63.212  31.606  119.26 < 2.2e-16 ***
Residuals    147 38.956   0.265                      
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

 Response 2 :
              Df Sum Sq Mean Sq F value    Pr(>F)    
iris$Species   2 437.10 218.551  1180.2 < 2.2e-16 ***
Residuals    147  27.22   0.185                      
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

> 
```

`cbind(iris$Sepal.Length, iris$Petal.Length) ~ iris$Species` is the formula, like `cbind(Sepal.Length, Petal.Length) = Species`. Hence, you have two response variables, `Sepal.Length` and `Petal.Length`.

$$H_0 : \mu_{Sepal.Length} = \mu_{Petal.Length} = \mu_{Species} \qquad \textit{Null hypothesis}$$

$$H_a : \mu_{Sepal.Length} \neq \mu_{Petal.Length} \neq \mu_{Species} \qquad \textit{Alternate hypothesis}$$

The p-value is 2.2e-16, which is less than 0.05. Hence, you reject the null hypothesis. The alternate hypothesis is true at the 95% confidence interval. There are significant differences in the means. The response variable `Sepal.Length` mean and the `Species` mean have p-values of 2.2e-16, which are less than 0.05. Hence, you reject the null hypothesis that the `Sepal.Length` mean is the same as the `Species` mean. The alternate hypothesis is true at the 95% confidence interval. The means for the response variables `Petal.Length` and `Species` have p-values of 2.2e-16, which are less than 0.05. Hence, you reject the null hypothesis that the `Petal.Length` mean is the same as the `Species` mean. The alternate hypothesis is true at the 95% confidence interval.

Nonparametric Test

The nonparametric test is a test that does not require the variable and sample to be normally distributed. Most of the time you should use parametric tests like the t-test, chi-square test, and ANOVA because they are more accurate. You use nonparametric tests when you do not have normally distributed data and the sample data is big.

Wilcoxon Signed Rank Test

The Wilcoxon signed rank test is used to replace the one-sample t-test.

a. For each x_i, for $i = 1, 2, ..., n$ the signed difference is d_i=x_i- u_0, where u_0 is the given median.

b. Ignore d_i= 0 and rank the rest of $|d_i|$, using r_i as rank. When there are tied values, assign the average of the tied ranks. For example, $|d_i|$ ranked as 3, 4, 5 are ties, so the rank should be $\frac{(3+4+5)}{3} = 4$.

c. The number of non-zero $d_i s$ is found.

d. To each rank of d_i, let $s_i = sign\,(d_i)r_i$.

e. The sum of a positive signed rank is calculated using

$$W = \sum_{s_i > 0} s_i$$

The test statistics calculated is W and the number n_1 of non-zero $d_i s$ is calculated.

The null hypothesis is that the population median has the specified value of μ_0.

- Null hypothesis: $H_0 : \mu = \mu_0$
- Alternate hypothesis: $H_a : \mu \neq \mu_0$

The normal test statistics formula is

$$z = \frac{\left(W - \frac{n_1(n_1+1)}{4}\right) - \frac{1}{2} \cdot sign\left(W - \frac{n_1(n_1+1)}{4}\right)}{\sqrt{\frac{1}{4} \cdot \sum_{i=1}^{n} S_i^2}}$$

You reject the null hypothesis when

$$|z| > Z_{\propto/2}, where\ Z \sim N(0,1)$$

The common alpha, ∝, value is 0.05.

To use the Wilcoxon signed rank test in R, you can first generate the data set using `random.org` packages, so that the variables are not normally distributed. To use `random.org` for random number generation, you must install the random packages:

```
> install.packages("random");
Installing package into 'C:/Users/gohmi/Documents/R/win-
library/3.5'
(as 'lib' is unspecified)
also installing the dependency 'curl'

trying URL 'https://cran.rstudio.com/bin/windows/contrib/3.5/
curl_3.2.zip'
Content type 'application/zip' length 2986409 bytes (2.8 MB)
downloaded 2.8 MB

trying URL 'https://cran.rstudio.com/bin/windows/contrib/3.5/
random_0.2.6.zip'
Content type 'application/zip' length 466978 bytes (456 KB)
downloaded 456 KB

package 'curl' successfully unpacked and MD5 sums checked
package 'random' successfully unpacked and MD5 sums checked

The downloaded binary packages are in
    C:\Users\gohmi\AppData\Local\Temp\RtmpaizS1C\downloaded_
    packages
```

To use the random package, you must load the package using the `library()` function:

```
> library(random);
```

To create random numbers from `random.org`, you can use

```
> library(random);
> var1 <- randomNumbers(n=100, min=1, max=1000, col=1);
> var2 <- randomNumbers(n=100, min=1, max=1000, col=1);
> var3 <- randomNumbers(n=100, min=1, max=1000, col=1);
```

`n` is the number of random numbers, `min` is the minimum value, `max` is the maximum value, and `col` is the number of columns for all the numbers. This is the method to generate true random numbers in R. Your data may be different because the data is generated randomly.

You can then create the data using

```
> data <- data.frame(var1[,1], var2[,1], var3[,1]);
> data;
    var1...1. var2...1. var3...1.
1         680         9       871
2         547       589       768
3         750       733       611
4         840       494        16
5         529       373       680
6          94       509       493
7         106        89       195
8         956       992       570
9         853       330       425
10        295       485       504
11        633       924       523
```

To use Wilcoxon signed rank test, you can use the `wilcox.test()` function:

```
> wilcox.test(data[,1], mu=0, alternatives="two.sided");

    Wilcoxon signed rank test with continuity correction

data:  data[, 1]
V = 5050, p-value < 2.2e-16
alternative hypothesis: true location is not equal to 0
```

$$H_0 : \mu = \mu_0$$

$$H_a : \mu \neq \mu_0$$

The p-value is 2.2e-16, which is less than 0.05. Hence, you reject the null hypothesis. There are significant differences in the median for the first variable median and the median of 0. The alternate hypothesis is true at the 95% confidence interval.

Wilcoxon-Mann-Whitney Test

The Wilcoxon-Mann-Whitney test is a nonparametric test to compare two samples. It is a powerful substitute to the two-sample t-test.

For two independent samples, $F(x)$ and $G(y)$, where their sample size is n_1 and n_2, and sample data can be $x_1, x_2, \ldots, x_{n_1}$ and $y_1, y_2, \ldots, y_{n_2}$, the hypothesis is

$$H_0 : F(x) = G(y)$$

$$H_a : F(x) \neq G(y)$$

To test the two samples,

1. Combine x_i and y_i as a group.
2. Rank the group in ascending order, where ties are the average of their rank. Let r_{1i} be the rank assigned for x_i for $i = 1,2,\ldots,n_1$ and r_{2j} be the rank assigned for y_i for $j = 1,2,\ldots,n_2$
3. Calculate the sum of ranks:

$$S_1 = \sum_{I=1}^{n_1} r_{1i}$$

$$S_2 = \sum_{I=1}^{n_2} r_{2j}$$

The test statistics U is calculated as follows:

$$U = S_1 - \frac{n_1\left(n_1 + 1\right)}{2}$$

The approximate normal statistics z is

$$z = \frac{U - M\left(U\right) \pm \frac{1}{2}}{\sqrt{\mathrm{var}\left(U\right)}}$$

where

$$M\left(U\right) = \frac{n_1 n_2}{2}$$

and the variance of U is

$$Var\left(U\right) = \frac{n_1 n_2\left(n_1 + n_2 + 1\right)}{12} - \frac{n_1 n_2}{\left(n_1 + n_2\right)\left(n_1 + n_2 - 1\right)} \times TS$$

where

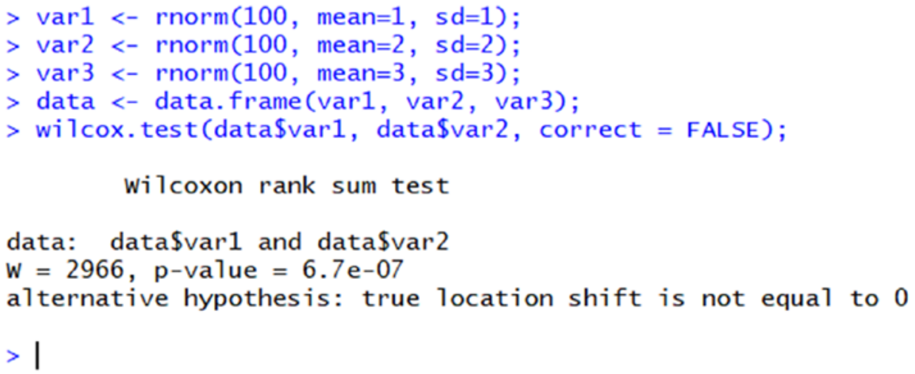
$$TS = \sum_{j=1}^{\tau} \frac{\left(t_j\right)\left(t_j - 1\right)\left(t_j + 1\right)}{12}$$

τ is the number of ties in sample and t_j is the number of ties in the jth group.

If there are no ties, the variance of U is

$$Var\left(U\right) = \frac{n_1 n_2\left(n_1 + n_2 + 1\right)}{12}$$

To use the Wilcoxon-Matt-Whitney test (or the Wilcoxon rank sum test or the Mann-Whitney test) in R, you can use the `wilcox.test()` function:



```
> var1 <- rnorm(100, mean=1, sd=1);
> var2 <- rnorm(100, mean=2, sd=2);
> var3 <- rnorm(100, mean=3, sd=3);
> data <- data.frame(var1, var2, var3);
> wilcox.test(data$var1, data$var2, correct = FALSE);

        Wilcoxon rank sum test

data:  data$var1 and data$var2
W = 2966, p-value = 6.7e-07
alternative hypothesis: true location shift is not equal to 0

> |
```

```
> var1 <- randomNumbers(n=100, min=1, max=1000, col=1);
> var2 <- randomNumbers(n=100, min=1, max=1000, col=1);
> var3 <- randomNumbers(n=100, min=1, max=1000, col=1);
> data <- data.frame(var1[,1], var2[,1], var3[,1]);
> wilcox.test(data[,1], data[,2], correct=FALSE);

    Wilcoxon rank sum test

data:  data[, 1] and data[, 2]
W = 5697.5, p-value = 0.08833
alternative hypothesis: true location shift is not equal to 0
```

$$H_0 : F(x) = G(y)$$

$$H_a : F(x) \neq G(y)$$

The p-value is 0.3351, which is more than 0.05. Hence, you fail to reject the null hypothesis. There are no significant differences in the median for first variable median and second variable median. The null hypothesis is true at the 95% confidence interval.

Kruskal-Wallis Test

The Kruskal-Wallis test is a nonparametric test that is an extension of the Mann-Whitney U test for three or more samples. The test requires samples to be identically distributed. Kruskal-Wallis is an alternative to one-way ANOVA. The Kruskal-Wallis test tests the differences between scores of *k* independent samples of unequal sizes with the *i*th sample containing l_i rows. The hypothesis is

$$H_0 : \mu_0 = \mu_1 = \mu_2 = \ldots = \mu_k$$

$$H_a : \mu_0 \neq \mu_k$$

where $\mu$is the median. The alternate hypothesis is that at least one median is different.

The algorithm is as follows:

1. Rank all rows in ascending order. Tied scores will have average ranks.
2. Sum up the ranks of rows in each sample to give rank sum R_i for $i = 1,2,\ldots,k$.
3. The Kruskal-Wallis test statistics is calculated as

$$H = \frac{12}{N(N+1)} \sum_{i=1}^{k} \frac{R_i^2}{l_i} - 3(N+1)$$

where

$$N = \sum_{i=1}^{k} l_i$$

N is the total number of rows. If there are tied scores, *H* is divided by

$$1 - \frac{\sum(t^3 - t)}{N^3 - N}$$

Where *t* is the number of tied scores in a group.

To use Kruskal-Wallis test in R:

```
> data("airquality");
> str(airquality);
'data.frame':    153 obs. of  6 variables:
 $ Ozone  : int  41 36 12 18 NA 28 23 19 8 NA ...
 $ Solar.R: int  190 118 149 313 NA NA 299 99 19 194 ...
 $ Wind   : num  7.4 8 12.6 11.5 14.3 14.9 8.6 13.8 20.1 8.6
...
 $ Temp   : int  67 72 74 62 56 66 65 59 61 69 ...
 $ Month  : int  5 5 5 5 5 5 5 5 5 5 ...
 $ Day    : int  1 2 3 4 5 6 7 8 9 10 ...
> summary(airquality);
Ozone               Solar.R              Wind
   Temp                Month                Day
Min.   :  1.00      Min.   :  7.0        Min.   : 1.700
   Min.   :56.00       Min.   :5.000        Min.   : 1.0
1st Qu.: 18.00      1st Qu.:115.8        1st Qu.: 7.400
   1st Qu.:72.00       1st Qu.:6.000        1st Qu.: 8.0
Median : 31.50      Median :205.0        Median : 9.700
   Median :79.00       Median :7.000        Median :16.0
Mean   : 42.13      Mean   :185.9        Mean   : 9.958
   Mean   :77.88       Mean   :6.993        Mean   :15.8
3rd Qu.: 63.25      3rd Qu.:258.8        3rd Qu.:11.500
   3rd Qu.:85.00       3rd Qu.:8.000        3rd Qu.:23.0
Max.   :168.00      Max.   :334.0        Max.   :20.700
   Max.   :97.00       Max.   :9.000        Max.   :31.0
NA's   :37          NA's   :7
> kruskal.test(airquality$Ozone ~ airquality$Month);
```

```
    Kruskal-Wallis rank sum test

data:  airquality$Ozone by airquality$Month
Kruskal-Wallis chi-squared = 29.267, df = 4, p-value = 6.901e-06
```

$$H_0 : \mu_0 = \mu_1 = \mu_2 = \ldots = \mu_k$$

$$H_a : \mu_0 \neq \mu_k$$

The p-value is 6.901e-06, which is less than 0.05. Hence, you reject the null hypothesis. There are significant differences in the median for the first variable median and the second variable median. The alternate hypothesis is true at the 95% confidence interval.

Linear Regressions

Regression analysis is a form of predictive modelling techniques that identify the relationships between dependent and independent variables(s). The technique is used to find causal effect relationships between variables.

The benefit of using regression analysis is that it identifies the significant relationships between dependent and independent variables and the strength of the impact of multiple independent variables on independent variables.

Linear regression finds the relationship between one dependent variable and one independent variable using a regression line.

The linear regression equation is $y = b_0+b_1x$

y is the dependent variable, x is the independent variable, b_0 is the intercept, and b_1 is the slope. See Figure 6-8.

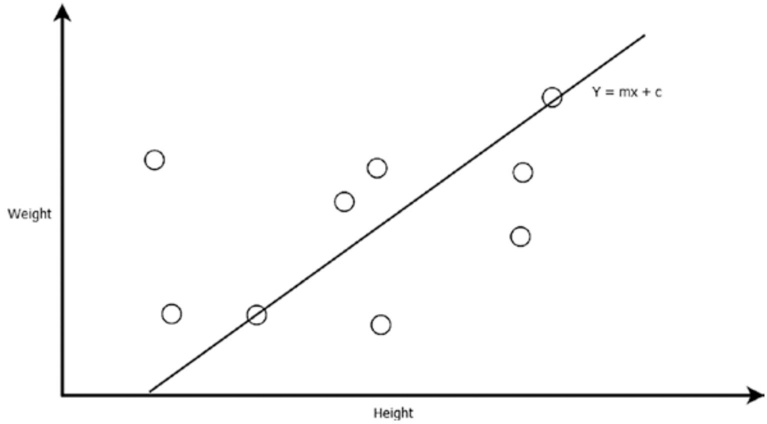


Figure 6-8. *LInear Regressions*

To calculate the slope, you can use

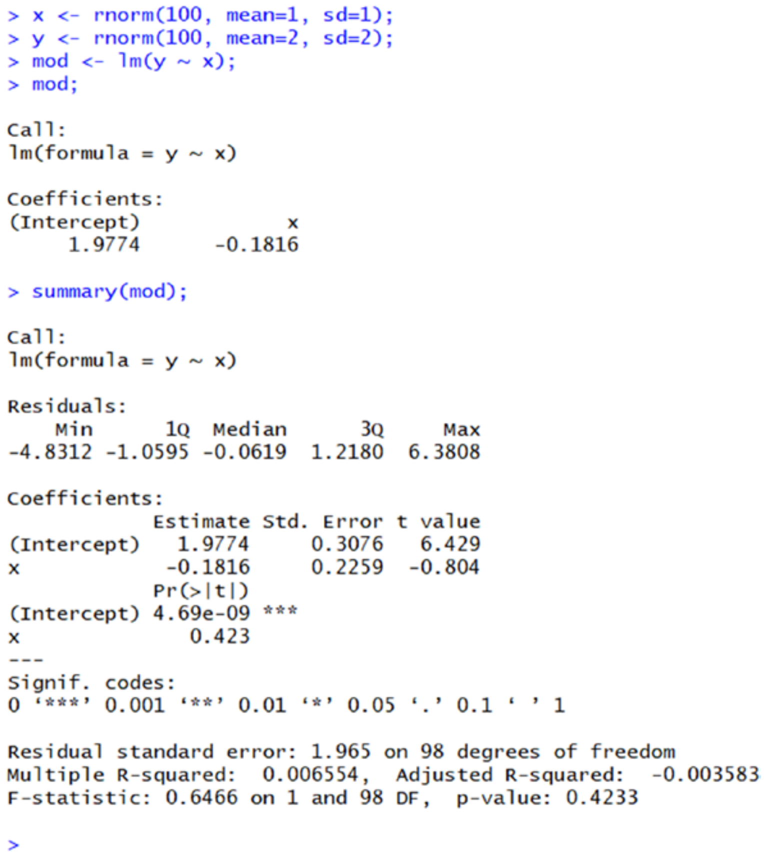
$$b_1 = \frac{\sum_{i=1}^{n}(x_i - \overline{x})(y_i - \overline{y})}{\sum_{i=1}^{n}(x_i - \overline{x})^2}$$

To calculate the intercept, you can use

$$b_0 = \overline{y} - b_1\overline{x}$$

If $b_1 > 0$, x and y have a positive relationship.
If $b_1 < 0$, x and y have a negative relationship.

To use linear regression in R, you use the `lm()` function:



```
> x <- rnorm(100, mean=1, sd=1);
> y <- rnorm(100, mean=2, sd=2);
> mod <- lm(y ~ x);
> mod;

Call:
lm(formula = y ~ x)

Coefficients:
(Intercept)            x
     1.9774      -0.1816

> summary(mod);

Call:
lm(formula = y ~ x)

Residuals:
    Min      1Q  Median      3Q     Max
-4.8312 -1.0595 -0.0619  1.2180  6.3808

Coefficients:
            Estimate Std. Error t value
(Intercept)   1.9774     0.3076   6.429
x            -0.1816     0.2259  -0.804
            Pr(>|t|)
(Intercept) 4.69e-09 ***
x              0.423
---
Signif. codes:
0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 1.965 on 98 degrees of freedom
Multiple R-squared:  0.006554,  Adjusted R-squared:  -0.003583
F-statistic: 0.6466 on 1 and 98 DF,  p-value: 0.4233

>
```

```
> set.seed(123);
> x <- rnorm(100, mean=1, sd=1);
> y <- rnorm(100, mean=2, sd=2);
> data <- data.frame(x, y);
> mod <- lm(data$y ~ data$x, data=data);
> mod;

Call:
lm(formula = data$y ~ data$x, data = data)
```

```
Coefficients:
(Intercept)       data$x
     1.8993      -0.1049

> summary(mod);

Call:
lm(formula = data$y ~ data$x, data = data)

Residuals:
   Min     1Q Median     3Q    Max
-3.815 -1.367 -0.175  1.161  6.581

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   1.8993     0.3033   6.261 1.01e-08 ***
data$x       -0.1049     0.2138  -0.491    0.625
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 1.941 on 98 degrees of freedom
Multiple R-squared:  0.002453,    Adjusted
R-squared:  -0.007726
F-statistic: 0.241 on 1 and 98 DF,  p-value: 0.6246
```

The output depicts that the linear equation is

$$y = -0.1049x + 1.8993$$

The p-values of 1.01e-08, 0.625, and 0.6246 tell you the significance of the linear model. When the p-value is less than 0.05, the model is significant.

- H_0 : : Coefficient associated with the variable is equal to zero
- H_a : : Coefficient is not equal to zero (there is a relationship)

The intercept has a p-value of 1.01e-08, which is smaller than 0.05, so there is a significance with the y variable. The significance is indicated with the number of *. The x has a p-value of 0.625, which is more than 0.05, so there is no significance with the y variable. The null hypothesis is true at the 95% confidence interval.

R-square depicts the proportion of the variation in the dependent variable, and the formula is

$$R^2 = 1 - \frac{SSE}{SST}$$

where SSE is the sum of squared errors

$$SSE = \sum_i^n \left(y_i - \hat{y}_i\right)^2$$

and SST is the sum of the squared total

$$SST = \sum_i^n \left(y_i - \overline{y}_i\right)^2$$

$\overline{\text{y}}$ is the mean of Y and $\hat{y}$ is the fitted value for row i.

$\hat{y}$ is the fitted value, which mean that in y = -0.1049x + 1.8993, you fit in x to get y. The y is the $\hat{y}$. $\left(y - \hat{y}\right)$ means that you use the original y values minus the $\hat{y}$ predicted values, which is the error. Hence, $\sum_i \left(y_i - \hat{y}_i\right)^2$ is the sum of the square error (SSE). In order for $\frac{SSE}{SST}$ to be small, SSE must be small. Nevertheless, $1 - \frac{SSE}{SST}$ is large when $\frac{SSE}{SST}$ is small.

Hence, the higher the R-squared and the adjusted R-squared, the better the linear model. The lower the standard error, the better the model.

Multiple Linear Regressions

A simple linear regression is for a single response variable, y, and a single independent variable, x. The equation for a simple linear regression is

$$y = b_0 + b_1 x$$

Multiple linear regression is built from a simple linear regression. Multiple linear regression is used when you have more than one independent variable. The equation of a multiple linear regression is

$$y = b_0 + b_1 x_1 + b_2 x_2 + \ldots + b_k x_k + \epsilon$$

When you have n observations or rows in the data set, you have the following model:

$$y_1 = b_0 + b_1 x_{11} + b_2 x_{12} + \ldots + b_k x_{1k} + \epsilon_1$$

$$y_2 = b_0 + b_1 x_{21} + b_2 x_{22} + \ldots + b_k x_{2k} + \epsilon_2$$

$$y_3 = b_0 + b_1 x_{31} + b_2 x_{32} + \ldots + b_k x_{3k} + \epsilon_3$$

$$\ldots$$

$$y_n = b_0 + b_1 x_{n1} + b_2 x_{n2} + \ldots + b_k x_{nk} + \epsilon_n$$

Using a matrix, you can represent the equations as

$$y = Xb + \epsilon$$

where

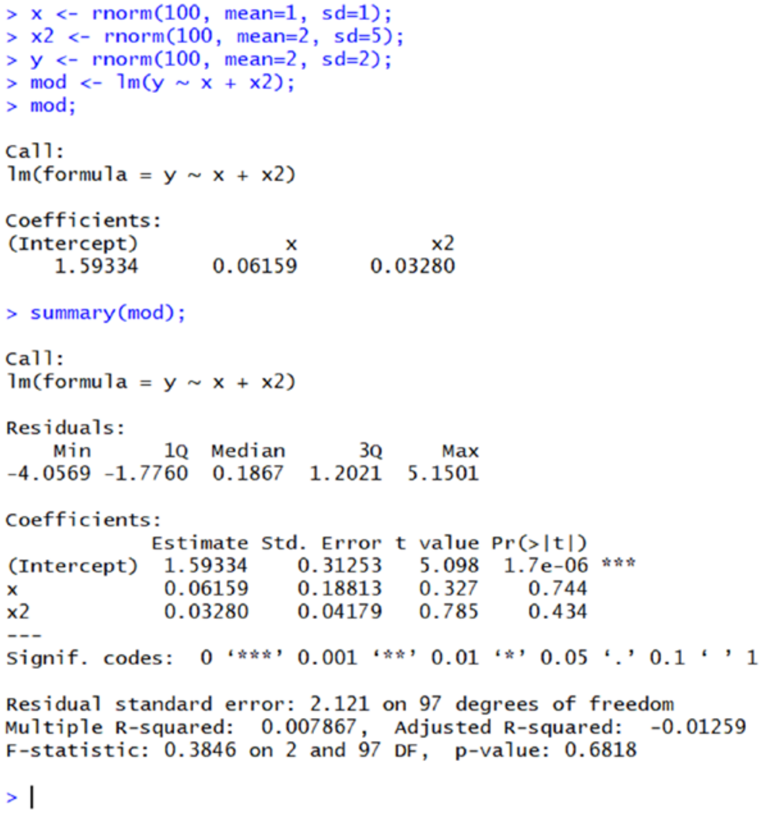
$$y = \begin{bmatrix} y_1 \\ y_2 \\ . \\ . \\ . \\ y_n \end{bmatrix} \quad X = \begin{bmatrix} 1 & x_{11} & x_{12} & . & . & . & x_{1k} \\ 1 & x_{21} & x_{22} & . & . & . & x_{2k} \\ . & . & . & & & & . \\ . & . & . & & & & . \\ . & . & . & & & & . \\ 1 & x_{n1} & x_{n2} & . & . & . & x_{nk} \end{bmatrix}$$

$$b = \begin{bmatrix} b_1 \\ b_2 \\ . \\ . \\ . \\ b_n \end{bmatrix} \quad and \quad \epsilon = \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ . \\ . \\ . \\ \epsilon_n \end{bmatrix}$$

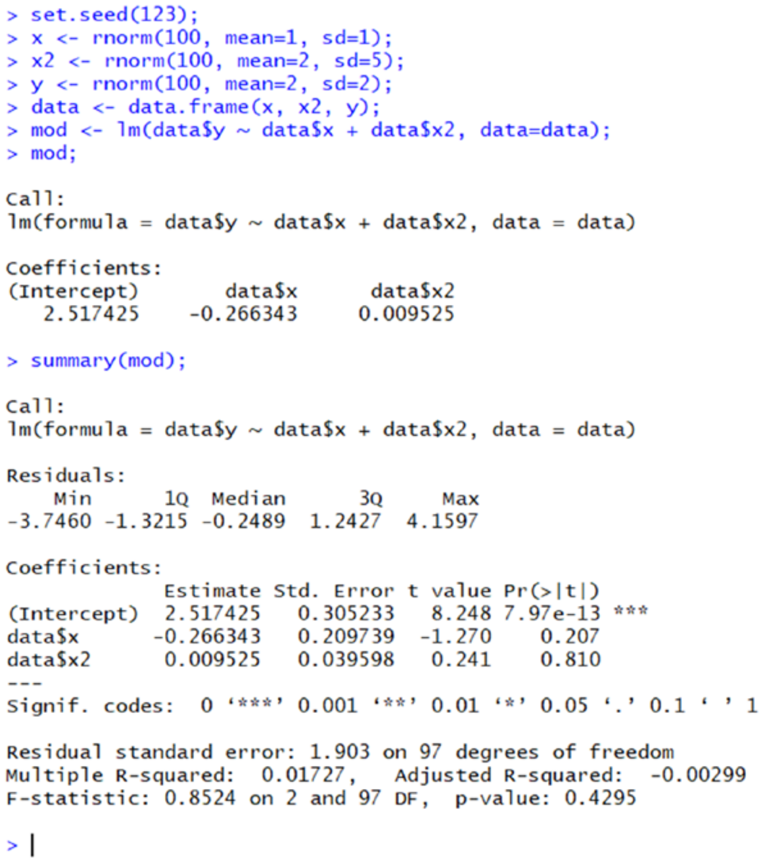
To calculate the coefficients:

$$\hat{b} = \left(X'X\right)^{-1} X'y$$

You can use the multiple linear regression in R:



```
> x <- rnorm(100, mean=1, sd=1);
> x2 <- rnorm(100, mean=2, sd=5);
> y <- rnorm(100, mean=2, sd=2);
> mod <- lm(y ~ x + x2);
> mod;

Call:
lm(formula = y ~ x + x2)

Coefficients:
(Intercept)            x           x2
    1.59334      0.06159      0.03280

> summary(mod);

Call:
lm(formula = y ~ x + x2)

Residuals:
    Min      1Q  Median      3Q     Max
-4.0569 -1.7760  0.1867  1.2021  5.1501

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  1.59334    0.31253   5.098  1.7e-06 ***
x            0.06159    0.18813   0.327    0.744
x2           0.03280    0.04179   0.785    0.434
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 2.121 on 97 degrees of freedom
Multiple R-squared:  0.007867,  Adjusted R-squared:  -0.01259
F-statistic: 0.3846 on 2 and 97 DF,  p-value: 0.6818

> |
```



```
> set.seed(123);
> x <- rnorm(100, mean=1, sd=1);
> x2 <- rnorm(100, mean=2, sd=5);
> y <- rnorm(100, mean=2, sd=2);
> data <- data.frame(x, x2, y);
> mod <- lm(data$y ~ data$x + data$x2, data=data);
> mod;

Call:
lm(formula = data$y ~ data$x + data$x2, data = data)

Coefficients:
(Intercept)       data$x      data$x2
   2.517425    -0.266343     0.009525

> summary(mod);

Call:
lm(formula = data$y ~ data$x + data$x2, data = data)

Residuals:
    Min      1Q  Median      3Q     Max
-3.7460 -1.3215 -0.2489  1.2427  4.1597

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  2.517425   0.305233   8.248 7.97e-13 ***
data$x      -0.266343   0.209739  -1.270    0.207
data$x2      0.009525   0.039598   0.241    0.810
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 1.903 on 97 degrees of freedom
Multiple R-squared:  0.01727,   Adjusted R-squared:  -0.00299
F-statistic: 0.8524 on 2 and 97 DF,  p-value: 0.4295

> |
```

```
> set.seed(123);
> x <- rnorm(100, mean=1, sd=1);
> x2 <- rnorm(100, mean=2, sd=5);
> y <- rnorm(100, mean=2, sd=2);
> data <- data.frame(x, x2, y);
> mod <- lm(data$y ~ data$x + data$x2, data=data);
> mod;

Call:
lm(formula = data$y ~ data$x + data$x2, data = data)
```

```
Coefficients:
(Intercept)       data$x      data$x2  
   2.517425    -0.266343     0.009525  

> summary(mod);


Call:
lm(formula = data$y ~ data$x + data$x2, data = data)

Residuals:
    Min      1Q  Median      3Q     Max 
-3.7460 -1.3215 -0.2489  1.2427  4.1597 

Coefficients:
             Estimate Std. Error t value Pr(>|t|)    
(Intercept)  2.517425   0.305233   8.248 7.97e-13 ***
data$x      -0.266343   0.209739  -1.270    0.207    
data$x2      0.009525   0.039598   0.241    0.810    
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 1.903 on 97 degrees of freedom
Multiple R-squared:  0.01727,    Adjusted R-squared:  -0.00299 
F-statistic: 0.8524 on 2 and 97 DF,  p-value: 0.4295
```

To create multiple linear regression in R, you must first create data:

```
> set.seed(123);
> x <- rnorm(100, mean=1, sd=1);
> x2 <- rnorm(100, mean=2, sd=5);
> y <- rnorm(100, mean=2, sd=2);
> data <- data.frame(x, x2, y);
```

You create a multiple linear regression model using the `lm()` function:

```
> mod <- lm(data$y ~ data$x + data$x2, data=data);
> mod;

Call:
lm(formula = data$y ~ data$x + data$x2, data = data)

Coefficients:
(Intercept)       data$x      data$x2
   2.517425    -0.266343     0.009525
data$y ~ data$x + data$x2 is y = x + x2
```

To get the summary of the model, you can use the `summary()` function:

```
 > summary(mod);

Call:
lm(formula = data$y ~ data$x + data$x2, data = data)

Residuals:
    Min      1Q  Median      3Q     Max
-3.7460 -1.3215 -0.2489  1.2427  4.1597

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  2.517425   0.305233   8.248 7.97e-13 ***
data$x      -0.266343   0.209739  -1.270    0.207
data$x2      0.009525   0.039598   0.241    0.810
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 1.903 on 97 degrees of freedom
Multiple R-squared:  0.01727,    Adjusted R-squared:  -0.00299
F-statistic: 0.8524 on 2 and 97 DF,  p-value: 0.4295
```

The linear model from the output is

```
Y = -0.266343x + 0.009525x2 + 2.517425
```

The p-values are 7.97e-13, 0.207, 0.810, 0.4295. The intercept is significant because the p-value is 7.97e-13, which is smaller than 0.05.

$$R^2 = 1 - \frac{SSE}{SST} = 1 - \frac{\sum_i \left(y_i - \hat{y}_i\right)^2}{\sum_i \left(y_i - \hat{y}_i\right)^2}$$

$\bar{y}$ is the mean of Y and $\hat{y}$ is the fitted value for row i.

$\hat{y}$ is the fitted value, which means that in y = -0.266343x + 0.009525x2 + 2.517425, you fit in x and x2 to get y. The y is the $\hat{y}$. $\left(y - \hat{y}\right)$ means that you use the original y values minus the $\hat{y}$ predicted values, which is the error. Hence, $\sum_i \left(y_i - \hat{y}_i\right)^2$ is the SSE. In order for $\frac{SSE}{SST}$ to be small, SSE must be small. Nevertheless, $1 - \frac{SSE}{SST}$ is large when $\frac{SSE}{SST}$ is small. The R-squared is 0.01727 and the adjusted R-squared is -0.00299. The higher the R-squared value, the better, as SSE is smaller.

Conclusion

In this chapter, you looked into R programming. You now know that inferential statistics and descriptive statistics are the main branches of statistics. Descriptive statistics derives a summary from the data set and uses central tendency, dispersion, and skewness. Inferential statistics describes and makes inferences about the population and the sampled data. In inferential statistics, you use hypothesis testing and estimating of parameters.

You learned that the `apply()` function can perform a loop to go through the data and apply a function. The function can be a `mean()` function from R or it can be a customized function.

You also found out that sampling is the selection of a subset of a population. The population is the data from everyone. Sometimes a sample can be a subset from a full data set. The advantages of sampling are that the cost is lower and the data collection is more efficient than collecting the data from everyone in the population.

You also learned that correlation is a statistical association to find how close two variables are and derive a linear relationship between them.

You also learned that covariance is a measure of variability between two variables. The greater value of one variable and the greater of another variable means or will result in a covariance that is positive. The greater values of one variable to the lesser values of the other variable will result in a negative covariance.

You also learned how p-values help you determine the significance of your statistical tests results. Your claim in the test is known as a null hypothesis and the alternate hypothesis means that you believe the null hypothesis is untrue.

You also learned that a t-test is one of the more important tests in statistics. A t-test is used to determine whether the mean between two data points or samples are equal to each other. The null hypothesis means that the two means are equal, and the alternative means that the two means are different.

You also learned that the chi-square test is used to compare the relationship between two categorical variables. The null hypothesis means that there is no relationship between the categorical variables.

You also learned that ANOVA is the process of testing the means of two or more groups. ANOVA also checks the impact of factors by comparing the means of different samples. In a t-test, you test the means of two samples; in a chi-square test, you test categorical attributes or variables; and in ANOVA, you test more samples.

You also learned that nonparametric tests are tests that do not require the variable and sample to be normally distributed. Most of the time you should use parametric tests like t-tests, chi-square tests, and ANOVA because they are more accurate. You use nonparametric tests when you do not have normally distributed data, and the sample data is big.

You also learned that regression analysis is some form of a predictive modeling technique that identifies the relationships between dependent and independent variables(s). The technique is used to find causal effect relationships between variables.

References

(n.d.). Manuscript submitted for publication, Columbia University. Retrieved September 6, 2018, from www.stat.columbia.edu/~martin/W2024/R8.pdf.

17.5.1.2 Algorithms (One-sample Wilcoxon signed rank test). (n.d.). Retrieved from www.originlab.com/doc/Origin-Help/SignRank1-Algorithm.

17.5.4.2 Algorithms (Mann-Whitney Test). (n.d.). Retrieved from www.originlab.com/doc/Origin-Help/MW-Test-Algorithm.

17.5.6.2 Algorithms (Kruskal-Wallis ANOVA). (n.d.). Retrieved from www.originlab.com/doc/Origin-Help/KW-ANOVA-Algorithm.

Analysis of variance. (2018, September 03). Retrieved from https://en.wikipedia.org/wiki/Analysis_of_variance.

ANOVA Test: Definition, Types, Examples. (n.d.). Retrieved from www.statisticshowto.com/probability-and-statistics/hypothesis-testing/anova/.

Apply(), sapply(), tapply() in R with Examples. (n.d.). Retrieved from www.guru99.com/r-apply-sapply-tapply.html.

Chi-Square Statistic: How to Calculate It/Distribution. (n.d.). Retrieved from www.statisticshowto.com/probability-and-statistics/chi-square/.

Chi-Square Test of Independence in R. (n.d.). Retrieved from www.sthda.com/english/wiki/chi-square-test-of-independence-in-r.

Correlation. (n.d.). Retrieved from www.mathsisfun.com/data/correlation.html.

Covariance. (n.d.). Retrieved from http://mathworld.wolfram.com/Covariance.html.

Das, S. (2018, June 06). Data Sampling Methods in R - DZone AI. Retrieved from https://dzone.com/articles/data-sampling-methods-in-r.

Department of Statistics. (n.d.). Retrieved from https://statistics.berkeley.edu/computing/r-t-tests.

P. (n.d.). Eval(ez_write_tag([[728,90],'r_statistics_co-box-3','ezslot_4']));Linear Regression. Retrieved from http://r-statistics.co/Linear-Regression.html.

Evaluation of Means for small samples - The t-test. (n.d.). Retrieved from www.chem.utoronto.ca/coursenotes/analsci/stats/ttest.html.

F Statistic/F Value: Simple Definition and Interpretation. (n.d.). Retrieved from www.statisticshowto.com/probability-and-statistics/f-statistic-value-test/.

Galili, T. (n.d.). Tag: Iris data set. Retrieved from www.r-statistics.com/tag/iris-data-set/.

Ghosh, B. (2017, August 28). One-way ANOVA in R. Retrieved from https://datascienceplus.com/one-way-anova-in-r/.

How to Take Samples from Data in R. (n.d.). Retrieved from www.dummies.com/programming/r/how-to-take-samples-from-data-in-r/.

Introduction. (n.d.). Retrieved from http://sphweb.bumc.bu.edu/otlt/MPH-Modules/BS/BS704_Nonparametric/BS704_Nonparametric_print.html.

Kabacoff, R. (n.d.). Correlations. Retrieved from www.statmethods.net/stats/correlations.html.

Kabacoff, R. (n.d.). ANOVA. Retrieved from www.statmethods.net/stats/anova.html.

Kabacoff, R. (n.d.). Nonparametric Tests of Group Differences. Retrieved from www.statmethods.net/stats/nonparametric.html.

Kabacoff, R. (n.d.). Multiple (Linear) Regression. Retrieved from www.statmethods.net/stats/regression.html.

Kruskal-Wallis Test. (n.d.). Retrieved from www.r-tutor.com/elementary-statistics/non-parametric-methods/kruskal-wallis-test.

Kruskal-Wallis Test in R. (n.d.). Retrieved from www.sthda.com/english/wiki/kruskal-wallis-test-in-r.

Linear Regression Analysis using SPSS Statistics. (n.d.). Retrieved from https://statistics.laerd.com/spss-tutorials/linear-regression-using-spss-statistics.php.

Mann-Whitney-Wilcoxon Test. (n.d.). Retrieved from www.r-tutor.com/elementary-statistics/non-parametric-methods/mann-whitney-wilcoxon-test.

MANOVA Test in R: Multivariate Analysis of Variance. (n.d.). Retrieved from www.sthda.com/english/wiki/manova-test-in-r-multivariate-analysis-of-variance.

Multiple Linear Regression Analysis. (n.d.). Retrieved from http://reliawiki.org/index.php/Multiple_Linear_Regression_Analysis.

Non Parametric Data and Tests (Distribution Free Tests). (n.d.). Retrieved from www.statisticshowto.com/parametric-and-non-parametric-data/.

One-Sample Wilcoxon Signed Rank Test in R. (n.d.). Retrieved from www.sthda.com/english/wiki/one-sample-wilcoxon-signed-rank-test-in-r.

One-Way ANOVA Test in R. (n.d.). Retrieved from www.sthda.com/english/wiki/one-way-anova-test-in-r.

One-Way ANOVA Test in R. (n.d.). Retrieved from www.sthda.com/english/wiki/one-way-anova-test-in-r.

P-value in Statistical Hypothesis Tests: What is it? (n.d.). Retrieved from www.statisticshowto.com/p-value/.

Paired t Test. (n.d.). Retrieved from www.statsdirect.com/help/parametric_methods/paired_t.htm.

R ANOVA Tutorial: One-way & Two-way [with Examples]. (n.d.). Retrieved from www.guru99.com/r-anova-tutorial.html.

R Tutorial Series: Multiple Linear Regression. (2016, October 02). Retrieved from www.r-bloggers.com/r-tutorial-series-multiple-linear-regression/.

R: Wilcoxon Rank Sum and Signed Rank Tests. (n.d.). Retrieved from https://stat.ethz.ch/R-manual/R-devel/library/stats/html/wilcox.test.html.

Ray, S., & Business Analytics and Intelligence. (2018, April 06). 7 Types of Regression Techniques you should know. Retrieved from www.analyticsvidhya.com/blog/2015/08/comprehensive-guide-regression/.

Regression Analysis: Step by Step Articles, Videos, Simple Definitions. (n.d.). Retrieved from www.statisticshowto.com/probability-and-statistics/regression-analysis/.

Sample rows of subgroups from dataframe with dplyr. (n.d.). Retrieved from https://stackoverflow.com/questions/21255366/sample-rows-of-subgroups-from-dataframe-with-dplyr.

Sampling in Statistics: Different Sampling Methods, Types & Error. (n.d.). Retrieved from www.statisticshowto.com/probability-and-statistics/sampling-in-statistics/.

SIGNED RANK TEST. (n.d.). Retrieved from www.itl.nist.gov/div898/software/dataplot/refman1/auxillar/signrank.htm.

Simple Random Sampling and Other Sampling Methods. (n.d.). Retrieved September 6, 2018, from https://onlinecourses.science.psu.edu/stat100/node/18/.

Singh, G., H., & Budding Data Scientist. (2018, January 15). A Simple Introduction to ANOVA (with applications in Excel). Retrieved from www.analyticsvidhya.com/blog/2018/01/anova-analysis-of-variance/.

Swaminathan, S. (2018, February 26). Linear Regression - Detailed View – Towards Data Science. Retrieved from https://towardsdatascience.com/linear-regression-detailed-view-ea73175f6e86.

T Test. (n.d.). Retrieved from https://researchbasics.education.uconn.edu/t-test/#.

T test formula. (n.d.). Retrieved from www.sthda.com/english/wiki/t-test-formula.

The chi-square test. (n.d.). Retrieved September 6, 2018, from https://web.stanford.edu/class/psych252/cheatsheets/chisquare.html.

Two sample Student's t-test #1. (2010, September 06). Retrieved from www.r-bloggers.com/two-sample-students-t-test-1/.

Two-way Anova. (n.d.). Retrieved from https://rcompanion.org/rcompanion/d_08.html.

Two-way ANOVA. (n.d.). Retrieved September 6, 2018, from https://onlinecourses.science.psu.edu/stat500/node/216/.

Unpaired Two-Samples T-test in R. (n.d.). Retrieved from www.sthda.com/english/wiki/unpaired-two-samples-t-test-in-r.

Using apply, sapply, lapply in R. (2012, December 22). Retrieved from www.r-bloggers.com/using-apply-sapply-lapply-in-r/.

Using Chi-Square Statistic in Research. (n.d.). Retrieved from www.statisticssolutions.com/using-chi-square-statistic-in-research/.

Using R for statistical analyses - ANOVA. (n.d.). Retrieved from www.gardenersown.co.uk/Education/Lectures/R/anova.htm.

Welch t-test. (n.d.). Retrieved from www.sthda.com/english/wiki/welch-t-test.

Wetherill, C. (2015, August 17). How to Perform T-tests in R. Retrieved from https://datascienceplus.com/t-tests/.

What a p-value Tells You about Statistical Data. (n.d.). Retrieved from www.dummies.com/education/math/statistics/what-a-p-value-tells-you-about-statistical-data/.

Wilcoxon-Mann-Whitney rank sum test (or test U). (2009, August 05). Retrieved from www.r-bloggers.com/wilcoxon-mann-whitney-rank-sum-test-or-test-u/.

Index

E. G. M. Hui, *Learn R for Applied Statistics*, https://doi.org/10.1007/978-1-4842-4200-1