Community Experience Distilled

R Data Mining Blueprints

Learn about data mining with real-world datasets

Pradeepta Mishra

[PACKT] PUBLISHING open source* community experience distilled

R Data Mining Blueprints

Learn about data mining with real-world datasets

Pradeepta Mishra

BIRMINGHAM - MUMBAI

R Data Mining Blueprints

First published: July 2016

Production reference: 1250716

Published by Packt Publishing Ltd.

Livery Place

35 Livery Street

Birmingham

B3 2PB, UK.

ISBN 978-1-78398-968-3

www.packtpub.com

Credits

Author
Pradeepta Mishra

Reviewer
Alexey Grigorev

Commissioning Editor
Priya Singh

Acquisition Editor
Tushar Gupta

Content Development Editor
Deepti Thore

Technical Editor
Tanmayee Patil

Copy Editors
Vikrant Phadke
Alpha Singh

Project Coordinator
Shweta H. Birwatkar

Proofreader
Safis Editing

Indexer
Hemangini Bari

Graphics
Jason Monteiro

Production Coordinator
Melwyn Dsa

About the Author

Pradeepta Mishra is a data scientist, predictive modeling expert, deep learning and machine learning practitioner, and an econometrician. He is currently leading the data science and machine learning practice for Ma Foi Analytics, Bangalore, India. Ma Foi Analytics is an advanced analytics provider for Tomorrow's Cognitive Insights Ecology, using a combination of cutting-edge artificial intelligence, proprietary big data platform, and data science expertise. He holds a patent for enhancing planogram design for the retail industry. Pradeepta has published and presented research papers at IIM Ahmedabad, India. He is a visiting faculty at various leading B-schools and regularly gives talks on data science and machine learning.

Pradeepta has spent more than 10 years in his domain and has solved various projects relating to classification, regression, pattern recognition, time series forecasting, and unstructured data analysis using text mining procedures, spanning across domains such as healthcare, insurance, retail and e-commerce, manufacturing, and so on.

If you have any questions, don't hesitate to look me up on Twitter via `@mishra1_PK`, I will be more than glad to help a fellow web professional wherever, whenever.

I would like to thank my wife, Prajna, and daughter, Aarya, for putting up with my late-night writing sessions. I would also like to thank my friends and colleagues at work for their continuous encouragement in writing.

About the Reviewer

Alexey Grigorev is a skilled data scientist and software engineer with more that 5 years of professional experience. Now he works as a data scientist at Searchmetrics Inc. In his day-to-day job he actively uses R and Python for data cleaning, data analysis, and modeling. He is a reviewer of other Packt Publishing books on data analysis such as *Test-Driven Machine Learning* and *Mastering Data Analysis with R*.

www.PacktPub.com

eBooks, discount offers, and more

Did you know that Packt offers eBook versions of every book published, with PDF and ePub files available? You can upgrade to the eBook version at `www.PacktPub.com` and as a print book customer, you are entitled to a discount on the eBook copy. Get in touch with us at `customercare@packtpub.com` for more details.

At `www.PacktPub.com`, you can also read a collection of free technical articles, sign up for a range of free newsletters and receive exclusive discounts and offers on Packt books and eBooks.

`https://www2.packtpub.com/books/subscription/packtlib`

Do you need instant solutions to your IT questions? PacktLib is Packt's online digital book library. Here, you can search, access, and read Packt's entire library of books.

Why subscribe?

- Fully searchable across every book published by Packt
- Copy and paste, print, and bookmark content
- On demand and accessible via a web browser

Table of Contents

Preface

With the growth of data in volume and type, it is becoming very essential to perform data mining in order to extract insights from large datasets. This is because organizations feel the need to a get return on investment (ROI) from large-scale data implementations. The fundamental reason behind data mining is to find out hidden treasure in large databases so that the business stakeholders can take action about future business outcomes. Data mining processes not only help the organizations reduce cost and increase profit but also help them find out new avenues.

In this book, I am going to explain the fundamentals of data mining using an open source tool and programming language known as R. R is a freely available language and environment for performing statistical computation, graphical data visualization, predictive modeling, and integration with other tools and platforms. I am going to explain the data mining concepts by taking example datasets using the R programming language.

In this book, I am going to explain the topics, their mathematical formulation, their implementation in a software environment, and also how the topics help in solving a business problem. The book is designed in such a way that the user can start from data management techniques, exploratory data analysis, data visualization, and modeling up to creating advanced predictive modeling such as recommendation engines, neural network models, and so on. It also gives an overview of the concept of data mining, its various facets with data science, analytics, statistical modeling, and visualization.

So let's have a look at the chapters briefly!

What this book covers

`Chapter 1`, *Data Manipulation Using In-built R Data*, gives a glimpse of programming basics using R, how to read and write data, programming notations, and syntax understanding with the help of a real-world case study. It also includes R scripts for practice to get hands-on experience of the concepts, terminologies, and underlying reasons for performing certain tasks. The chapter is designed in such a way that any reader with little programming knowledge should be able to execute R commands to perform various data mining tasks. We will discuss in brief the meaning of data mining and its relations with other domains such as data science, analytics, and statistical modeling; apart from this, we will start the data management topics using R.

`Chapter 2`, *Exploratory Data Analysis with Automobile Data*, helps the learners to understand exploratory data analysis. It involves numerical as well as graphical representation of variables in a dataset for easy understanding and quick conclusion about a dataset. It is important to get an understanding of the dataset, type of variables considered for analysis, the association between various variables, and so on. Creating cross-tabulations to understand the relationship between categorical variables and performing classical statistical tests on the data to verify various different hypotheses about the data can be tested out.

`Chapter 3`, *Visualize Diamond Dataset*, covers the basics of data visualization along with how to create advanced data visualization using existing libraries in the R programming language. While looking at numbers and statistics, it may tell a similar story for the variables we are looking at by different cuts; however, when we visually look at the relationship between variables and factors, it shows a different story altogether. Hence, data visualization tells you a message that numbers and statistics fail to do.

`Chapter 4`, *Regression with Automobile Data*, helps you to know the basics of predictive analytics using regression methods, including various linear and nonlinear regression methods using R programming. In this chapter, you will get to know the basics of predictive analytics using regression methods, including various linear and nonlinear regression methods using R programming. You will be able to understand the theoretical background as well as get practical hands-on experience on all the regression methods using R.

`Chapter 5`, *Market Basket Analysis with Groceries Data*, shows the second method of product recommendation, popularly known as Market Basket Analysis (MBA) and also known as association rules. This is about associating items purchased at transaction level, finding out the sub-segments of users having similar products and hence, recommending the products. MBA can also be used to form upsell and cross-sell strategies.

`Chapter 6`, *Clustering with E-commerce Data*, teaches the following things: what segmentation is, how clustering can be applied to perform segmentation, what are the methods used for clustering, and a comparative view of the various methods for segmentation. In this chapter, you will know the basics of segmentation using various clustering methods.

`Chapter 7`, *Building a Retail Recommendation Engine*, covers the following things and their implementation using the R programming language: what recommendation is and how it works, types and methods for performing recommendation, and implementation of product recommendation using R.

`Chapter 8`, *Dimensionality Reduction*, implements dimensionality reduction techniques such as PCA, singular value decomposition (SVD), and iterative feature selection methods using a practical dataset and R. With the growth of data in volumes and variety, dimensions of data have been continuously on the rise. Dimensionality reduction techniques have many applications in different industries, such as in image processing, speech recognition, recommendation engines, text processing, and so on.

`Chapter 9`, *Applying Neural Networks to Healthcare Data*, teaches you various types of neural networks, methods, and variants of neural networks with different functions to control the training of artificial neural networks in performing standard data mining tasks such as these: prediction of real-valued output using regression-based methods, prediction of output levels in a classification-based task, forecasting future values of a numerical attribute based on historical data, and compressing features to recognize important ones in order to perform prediction or classification.

What you need for this book

To follow the examples and code shared along with this book, you need to have R software downloaded from `https://cran.r-project.org/` (it is optional to download RStudio from `https://www.rstudio.com/`) and have it installed on the machine. There are no specific hardware requirements; you can have any computer with more than 2 GB RAM and it works on all platforms, including Mac, Linux, and Windows.

Who this book is for

This book is for readers who are starting their career in data mining, data science, or predictive modeling, or they are at some intermediate level with some degree of statistical knowledge and programming knowledge. Basic statistical knowledge is a must to understand the data mining concepts covered in this book. Having prior programming knowledge is not mandatory as first couple of chapters, I am going to cover data management and basic statistical analysis using R. This book is also for students, professionals, and experienced people aspiring to become data analysts.

Conventions

In this book, you will find a number of text styles that distinguish between different kinds of information. Here are some examples of these styles and an explanation of their meaning.

Code words in text, database table names, folder names, filenames, file extensions, pathnames, dummy URLs, user input, and Twitter handles are shown as follows: "In the current scenario from the `ArtPiece` dataset, we are trying to predict whether a work of art is a good purchase, or not, by taking a few business-relevant variables."

Any command-line input or output is written as follows:

```
>fit<- neuralnet(formula = CurrentAuctionAveragePrice ~ Critic.Ratings +
Acq.Cost + CollectorsAverageprice + Min.Guarantee.Cost, data = train,
hidden = 15, err.fct = "sse", linear.output = F)
> fit
Call: neuralnet(formula = CurrentAuctionAveragePrice ~ Critic.Ratings +
Acq.Cost + CollectorsAverageprice + Min.Guarantee.Cost, data = train,
hidden = 15, err.fct = "sse", linear.output = F)
1 repetition was calculated.
Error Reached Threshold Steps
1 54179625353167 0.004727494957 23
```

Warnings or important notes appear in a box like this.

Tips and tricks appear like this.

Reader feedback

Feedback from our readers is always welcome. Let us know what you think about this book-what you liked or disliked. Reader feedback is important for us as it helps us develop titles that you will really get the most out of. To send us general feedback, simply e-mail `feedback@packtpub.com`, and mention the book's title in the subject of your message. If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, see our author guide at `www.packtpub.com/authors`.

Customer support

Now that you are the proud owner of a Packt book, we have a number of things to help you to get the most from your purchase.

Downloading the example code

You can download the example code files for this book from your account at `http://www.packtpub.com`. If you purchased this book elsewhere, you can visit `http://www.packtpub.com/support` and register to have the files e-mailed directly to you.

You can download the code files by following these steps:

1. Log in or register to our website using your e-mail address and password.
2. Hover the mouse pointer on the **SUPPORT** tab at the top.
3. Click on **Code Downloads & Errata**.
4. Enter the name of the book in the **Search** box.
5. Select the book for which you're looking to download the code files.
6. Choose from the drop-down menu where you purchased this book from.
7. Click on **Code Download**.

Once the file is downloaded, please make sure that you unzip or extract the folder using the latest version of:

- WinRAR / 7-Zip for Windows
- Zipeg / iZip / UnRarX for Mac
- 7-Zip / PeaZip for Linux

The code bundle for the book is also hosted on GitHub at `https://github.com/PacktPublishing/R-Data-Mining-Blueprints`. We also have other code bundles from our rich catalog of books and videos available at `https://github.com/PacktPublishing/`. Check them out!

Downloading the color images of this book

We also provide you with a PDF file that has color images of the screenshots/diagrams used in this book. The color images will help you better understand the changes in the output. You can download this file from `https://www.packtpub.com/sites/default/files/downloads/RDataMiningBlueprints_ColorImages.pdf`.

Errata

Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you find a mistake in one of our books-maybe a mistake in the text or the code-we would be grateful if you could report this to us. By doing so, you can save other readers from frustration and help us improve subsequent versions of this book. If you find any errata, please report them by visiting `http://www.packtpub.com/submit-errata`, selecting your book, clicking on the **Errata Submission Form** link, and entering the details of your errata. Once your errata are verified, your submission will be accepted and the errata will be uploaded to our website or added to any list of existing errata under the Errata section of that title.

To view the previously submitted errata, go to `https://www.packtpub.com/books/content/support` and enter the name of the book in the search field. The required information will appear under the **Errata** section.

Piracy

Piracy of copyrighted material on the Internet is an ongoing problem across all media. At Packt, we take the protection of our copyright and licenses very seriously. If you come across any illegal copies of our works in any form on the Internet, please provide us with the location address or website name immediately so that we can pursue a remedy.

Please contact us at `copyright@packtpub.com` with a link to the suspected pirated material.

We appreciate your help in protecting our authors and our ability to bring you valuable content.

Questions

If you have a problem with any aspect of this book, you can contact us at `questions@packtpub.com`, and we will do our best to address the problem.

1
Data Manipulation Using In-built R Data

The book *R Data Mining Blueprints* focuses mainly on learning methods and steps in performing data mining using the R programming language as a platform. Since R is an open source tool, learning data mining using R is very interesting for learners at all levels. The book is designed in such a way that the user can start from data management techniques, exploratory data analysis, data visualization, and modeling up to creating advanced predictive modeling such as recommendation engines, neural network models, and so on. This chapter gives an overview of the concept of data mining, its various facets with data science, analytics, statistical modeling, and visualization. This chapter gives a glimpse of programming basics using R, how to read and write data, programming notations, and syntax understanding with the help of a real-world case study. This chapter includes R scripts for practice to get hands-on experience of the concepts, terminologies, and underlying reasons for performing certain tasks. The chapter is designed in such a way that any reader with little programming knowledge should be able to execute R commands to perform various data mining tasks.

In this chapter, we will discuss in brief the meaning of data mining and its relations with other domains such as data science, analytics, and statistical modeling; apart from this, we will start the data management topics using R so that you can achieve the following objectives:

- Understanding various data types used in R, including vector and its operations
- Indexing of data frames and factors sequences
- Sorting and merging dataframes and data type conversion
- String manipulation and date object formatting
- Handling missing values and NAs and missing value imputation techniques
- Flow control, looping constructs, and the use of apply functions

What is data mining?

Data mining can be defined as the process of deciphering meaningful insights from existing databases and analyzing results for consumption by business users. Analyzing data from various sources and summarizing it into meaningful information and insights is that part of statistical knowledge discovery that helps not only business users but also multiple communities such as statistical analysts, consultants, and data scientists. Most of the time, the knowledge discovery process from databases is unexpected and the results can be interpreted in many ways.

The growing number of devices, tablets, smartphones, computers, sensors, and various other digital devices helps in generating and collecting data at a much faster rate than ever before. With the ability of modern-day computers, the increased data can be preprocessed and modeled to answer various questions related to any business decision-making process. Data mining can also be defined as a knowledge-intensive search across discrete databases and information repositories using statistical methodologies, machine learning techniques, visualization, and pattern recognition technologies.

The growth of structured and unstructured data, such as the existence of bar codes in all products in a retail store, attachment of RFID-based tags on all assets in a manufacturing plant, Twitter feeds, Facebook posts, integrated sensors across a city to monitor the changing weather conditions, video analysis, video recommendation based on viewership statistics, and so on creates a conducive ecosystem for various tools, technologies, and methodologies to splurge. Data mining techniques applied to the variety of data discussed previously not only provide meaningful information about the data structure but also recommend possible future actions to be taken by businesses.

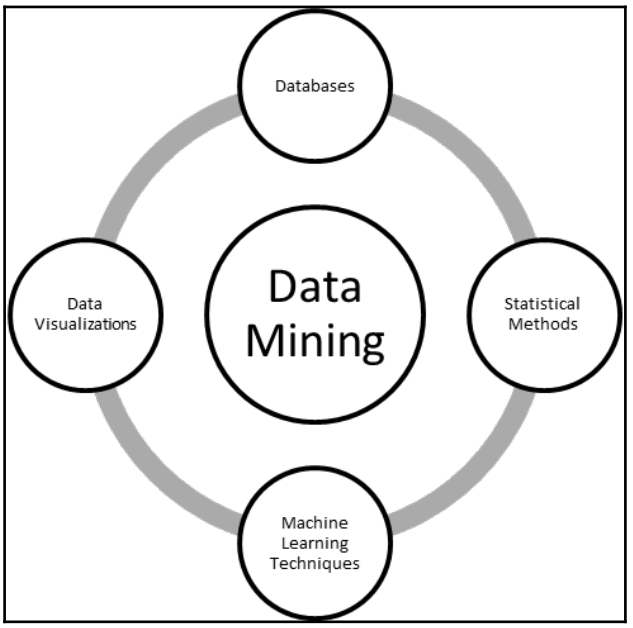


Figure 1: Data Mining – a multi-disciplinary subject

The process of data mining involves various steps:

1. Extract the required data from databases and data warehouses.
2. Perform a sanity check on the data to remove redundant characters and irrelevant information.
3. At times, it is important to combine information from various other disjoint databases. Hence, look for common attributes to combine databases.
4. Apply data transformation techniques. Sometimes, it is required to include a few attributes and features in a model.
5. Pattern recognition among the input features, where any of the pattern recognition methods can be applied.

6. Knowledge representation. This includes representation of knowledge mined from the databases in a visual form to various business stakeholders.

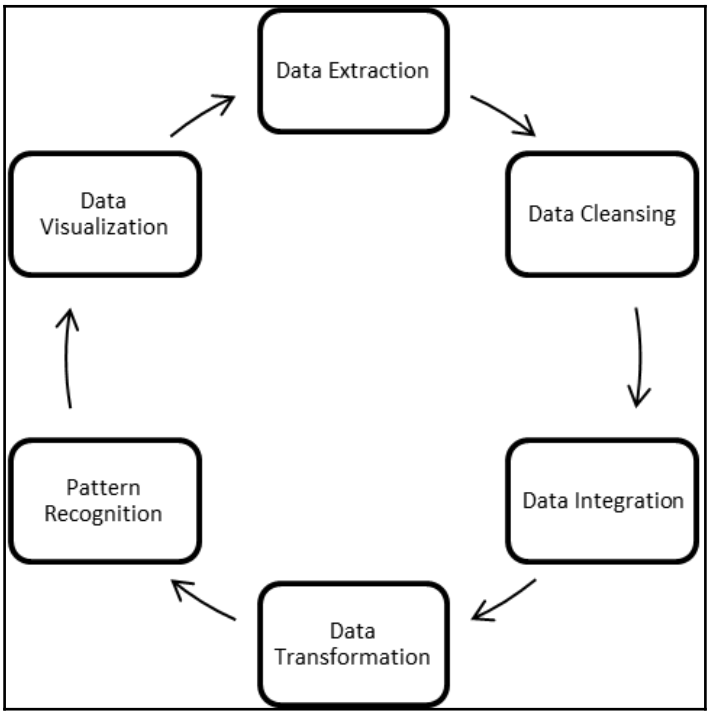


Figure 2: A typical data mining process flow

Having discussed the process flow of data mining and the core components, it is also important to look at a few challenges that one may encounter in data mining, such as computational efficiency, unstructured databases and their confluence with structured databases, high-dimensional data visualization, and so on. These issues can be resolved using innovative approaches. In this book, we are going to touch upon a few solutions while performing practical activities on our projects.

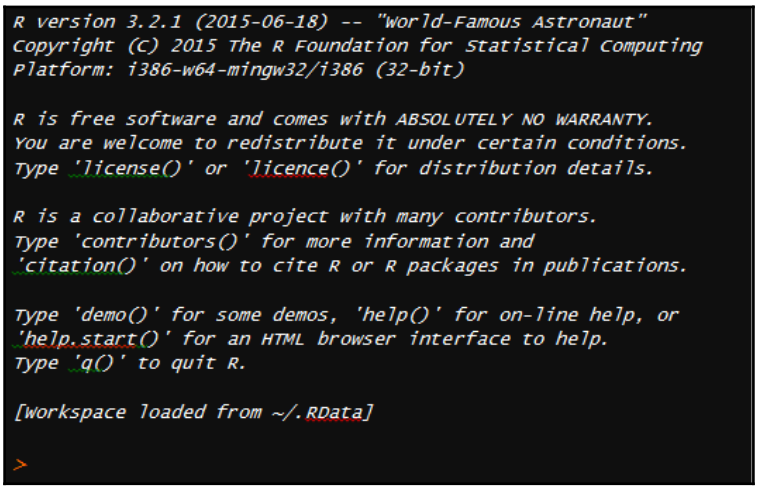
How is it related to data science, analytics, and statistical modeling?

Data science is a broader topic under which the data mining concept resides. Going by the aforementioned definition of data mining, it is a process of identifying patterns hidden in data and some interesting correlations that can provide useful insights. Data mining is a subset in data science projects that involves techniques such as pattern recognition, feature selection, clustering, supervised classification, and so on. Analytics and statistical modeling involve a wide range of predictive models-classification-based models to be applied on datasets to solve real-world business problems. There is a clear overlap between the three terminologies – data science, analytics, statistical modeling, and data mining. The three terminologies should not be looked at in isolation. Depending upon the project requirements and the kind of business problem, the overlap position might change, but at a broad level, all the concepts are well associated. The process of data mining also includes statistical and machine learning-based methods to extract data and automate rules and also represent data using good visualizations.

Introduction to the R programming language

In this chapter, we are going to start with basic programming using R for data management and data manipulation; we are also going to cover a few programming tips. R can be downloaded from`https://cran.r-project.org/`. Based on the operating system, anyone can download and install R binary files on their systems. The R programming, language which is an extension of the S language, is a statistical computing platform. It provides advanced predictive modeling capability, machine learning algorithms implementation capability, and better graphical visualization. R has various other plugins such as R.Net, rJava, SparkR, and RHadoop, which increases the usability of R in big data scenarios. The user can integrate R scripts with other programming platforms. Detailed information about R can be accessed using the following link:

`https://cran.r-project.org/`.



Getting started with R

The opening message when starting R would be as shown in the preceding screenshot. Everything entered in the R console is an object, many objects created in an active R session have different attributes associated with them, and one common attribute associated with an object is called its class. There are two popular approaches to object-oriented programming in R, which are S3 classes and S4 classes. The basic difference between the S3 and S4 approaches is that the former is a more flexible approach; however, the latter is a more structured approach to object-oriented programming. Both S3 and S4 approaches recognize any symbol, character, and number as an object in R sessions and provide functionality where the object can be used for further computations.

Data types, vectors, arrays, and matrices

There are two broad sets of data types: atomic vectors and compound vectors. There are basically five data types in R programming under the atomic vector category: numeric or numbers, characters or strings, factors, logical, and complex. And there are four compound data types: data frame, lists, array, and matrix. The primary data object in R is a vector; even when we assign a single-digit number to any alphabet, it is a single element vector in R. All data objects contain a mode and a length. The mode determines the kind of data stored in the object, and the length determines the number of elements contained in that object. The `c()` function in R implies concatenating of various elements in a vector.

Let's take a look at various examples showing different data types in R:

```
> x1<-c(2.5,1.4,6.3,4.6,9.0)
> class(x1)
[1] "numeric"
> mode(x1)
[1] "numeric"
> length(x1)
[1] 5
```

In the preceding script, the `x1` vector is a numeric vector and the number of elements is `5`. Both `class()` and `mode()` return the same results, hence both determine the type of vector:

```
> x2<-c(TRUE,FALSE,TRUE,FALSE,FALSE)
> class(x2)
[1] "logical"
> mode(x2)
[1] "logical"
> length(x2)
[1] 5
```

The `x2` vector is a logical vector having five elements. The logical vector elements or values can be written as either `T/F` or `TRUE/FALSE`.

```
    > x3<-
c("DataMining","Statistics","Analytics","Projects","MachineLearning")
    > class(x3)
    [1] "character"
    > length(x3)
    [1] 5
```

Object `x3` represents a character vector of length 25. All elements of the vector can be mentioned within double quote (`" "`) or single quote (`' '`).

The factor is another form of data where various categories listed in the vector are known as levels, in the preceding example; vector `a` is a character vector with two levels or categories, which are repeated with some frequency. The `as.factor()` command is used to convert a character vector into a factor data type. After applying that, it indicates there are five levels such as `Analytics`, `DataMining`, `MachineLearning`, `Projects`, and `Statistics`. The `table()` command indicates the frequency table computed from the factor variable:

```
> x<-data.frame(x1,x2,x3)
> class(x)
[1] "data.frame"
> print(x)
  x1    x2              x3
1 12  TRUE       Analytics
```

```
2 13 FALSE      DataMining
3 24  TRUE MachineLearning
4 54 FALSE        Projects
5 29  TRUE      Statistics
```

Dataframes are another popular form of data type in the R programming language that include all different data types. A dataframe is a list that contains multiple vectors of the same length and different types of data. If you simply import a dataset from a spreadsheet, the data type by default becomes dataframe. Later on, the data type for individual variables can be changed. So, dataframe can be defined as a matrix that contains columns of different data types. In the preceding script, the dataframe `x` contains three different data types: numeric, logical, and character. Most real-world datasets contain different data types; for example, in a retail store, information about customers is stored in a database. This includes customer ID, purchase date, amount purchased, whether part of any loyalty program or not, and so on.

One important point about vectors: all elements of a vector should be of the same type. If not, R will forcibly convert that by coercion. For example, in a numeric vector, if one element contains a character value, the vector type will change from numeric to character. The script is given as follows:

```
> x1<-c(2.5,1.4,6.3,4.6,9.0)
> class(x1)
[1] "numeric"
> x1<-c(2.5,1.4,6.3,4.6,9.0,"cat")
> class(x1)
[1] "character"
```

R is case sensitive, so `"cat"` is different from `"Cat"`. Hence, please be careful while assigning object names to the vectors. At times, it would be difficult to remember the object names:

```
> ls()
 [1] "a"              "centers"        "df"             "distances"
 [5] "dt2"            "i"              "indexes"        "km"
 [9] "kmd"            "kmeans.results" "log_model"      "mtcars"
[13] "outliers"       "pred"           "predict.kmeans" "probs"
[17] "Smarket"        "start"          "sumsq"          "t"
[21] "test"           "Titanic"        "train"          "x"
[25] "x1"             "x2"             "x3"             "x4"
[29] "x5"             "y"              "z"
```

To know what all objects are active in the current R session, the `ls()` command can be used; the `list` command will print all the active objects in the current session. Let's take a look at what a list is, how to retrieve the elements of a list, and how the list function can be used.

List management, factors, and sequences

A list is an ordered collection of objects that can contain arbitrary objects. Elements of a list can be accessed using the double square bracket. Those collections of objects are not necessarily of the same type. They can be of different types:

```
> mylist<-list(custid=112233, custname="John R", mobile="989-101-1011",
+ email="JohnR@gmail.com")
> mylist
$custid
[1] 112233
$custname
[1] "John R"
$mobile
[1] "989-101-1011"
$email
[1] "JohnR@gmail.com"
```

In the preceding example, the customer ID and mobile number are of numeric data type; however, the customer name and e-mail ID are of character data type. There are basically four elements in the preceding list. To extract elements from a list, we use double square brackets, and if we need to extract only a sublist from the list, we can use a single square bracket:

```
> mylist[[2]]
[1] "John R"
> mylist[2]
$custname
[1] "John R"
```

The next thing related to lists is how to combine more than one list. Lists can be combined using the `cbind()` function, that is, the column bind function:

```
    > mylist1<-list(custid=112233, custname="John R",
mobile="989-101-1011",
    + email="JohnR@gmail.com")
    > mylist2<-list(custid=443322, custname="Frank S",
mobile="781-101-6211",
    + email="SFranks@hotmail.com")
    > mylist<-cbind(mylist1,mylist2)
```

```
> mylist
         mylist1           mylist2
custid   112233            443322
custname "John R"          "Frank S"
mobile   "989-101-1011"    "781-101-6211"
email    "JohnR@gmail.com" "SFranks@hotmail.com"
```

Factors can be defined as various levels occurring in a categorical or nominal variable with a certain frequency. In other words, levels that are repetitive in a categorical variable are known as factors. In the following sample script, a character vector "domains" contains many levels; using the `factor` command, the frequency for each level can be estimated.

Sequences are repeated number of iterations, either numerical values or categorical or nominal values that can be part of a dataset. Numeric sequences can be created using a colon operator. To generate sequences using factor variables, the `gl()` function can be used. This is a very useful function while computing quantiles and graphical functions. Also, there are various other possible scenarios where you can use the function:

```
> seq(from=1,to=5,length=4)
[1] 1.000000 2.333333 3.666667 5.000000
> seq(length=10,from=-2,by=.2)
 [1] -2.0 -1.8 -1.6 -1.4 -1.2 -1.0 -0.8 -0.6 -0.4 -0.2
> rep(15,10)
 [1] 15 15 15 15 15 15 15 15 15 15
> gl(2,5,labels=c('Buy','DontBuy'))
 [1] Buy     Buy     Buy     Buy     Buy     DontBuy DontBuy DontBuy
DontBuy
[10] DontBuy
Levels: Buy DontBuy
```

The first line of code generates the sequence in ascending order, the second line creates a reverse sequence, and the last line creates a sequence for the factor data type.

Import and export of data types

If the Windows directory path is set, to import a file into the R system, it is not required to write the complete path where the file resides. If the Windows directory path is set to some other location in your system and still you want to access the file, then the complete path needs to be given to read the file:

```
> getwd()
[1] "C:/Users/Documents"
> setwd("C:/Users/Documents")
```

Any file in the documents folder can be read without mentioning the detailed path. Hence it is always suggested to change the Windows directory to the folder where the file resides.

There are different file formats; among them, CSV or text format is the best for the R programming platform. However, we can import from other file formats:

```
> dt<-read.csv("E:/Datasets/hs0.csv")
> names(dt)
 [1] "X0"      "X70"     "X4"      "X1"      "X1.1"    "general" "X57"
 [8] "X52"     "X41"     "X47"     "X57.1"
```

If you are using the `read.csv` command, there is no need to write the header `True` and separator as comma, but if you are using the `read.table` command, it is mandatory to use. Otherwise, it will read the first variable from the dataset:

```
> data<- read.table("E:/Datasets/hs0.csv",header=T,sep=",")
> names(data)
 [1] "X0"      "X70"     "X4"      "X1"      "X1.1"    "general" "X57"
 [8] "X52"     "X41"     "X47"     "X57.1"
```

While mentioning paths to extract the files, you can use either `/` or `\\`; both ways will work. In real-life projects, typically data is stored in Excel format. How to read data from Excel format is a challenge. It is not always convenient to store data in CSV format and then import. The following script shows how we can import Excel files in R. Two additional libraries are required to import an RDBMS file such as Excel. Those libraries are mentioned in the script and the sample data snippets are given as well:

```
> library(xlsx)
Loading required package: rJava
Loading required package: xlsxjars
> library(xlsxjars)
> dat<-read.xlsx("E:/Datasets/hs0.xls","hs0")
> head(dat)
  gender  id race ses schtyp  prgtype read write math science socst
1      0  70    4   1      1  general   57    52   41      47    57
2      1 121    4   2      1   vocati   68    59   53      63    61
3      0  86    4   3      1  general   44    33   54      58    31
4      0 141    4   3      1   vocati   63    44   47      53    56
5      0 172    4   2      1 academic   47    52   57      53    61
6      0 113    4   2      1 academic   44    52   51      63    61
```

Importing data from SPSS files is explained as follows. Legacy enterprise-based software systems generate data in either SPSS format or SAS format. The syntax for importing data from SPSS and SAS files needs additional packages or libraries. To import SPSS files, the `Hmisc` package is used, and to import SAS files, the `sas7bdat` library is used:

```
> library(Hmisc)
> mydata <- spss.get("E:/Datasets/wage.sav", use.value.labels=TRUE)
> head(mydata)
   HRS  RATE ERSP ERNO NEIN ASSET  AGE   DEP RACE SCHOOL
1 2157 2.905 1121  291  380  7250 38.5 2.340 32.1   10.5
2 2174 2.970 1128  301  398  7744 39.3 2.335 31.2   10.5
3 2062 2.350 1214  326  185  3068 40.1 2.851   NA    8.9
4 2111 2.511 1203   49  117  1632 22.4 1.159 27.5   11.5
5 2134 2.791 1013  594  730 12710 57.7 1.229 32.5    8.8
6 2185 3.040 1135  287  382  7706 38.6 2.602 31.4   10.7
> library(sas7bdat)
> mydata <- read.sas7bdat("E:/Datasets/sales.sas7bdat")
> head(mydata)
  YEAR NET_SALES PROFIT
1 1990       900    123
2 1991       800    400
3 1992       700    300
4 1993       455     56
5 1994       799    299
6 1995       666    199
```

Exporting a dataset from R to any external location can be done by changing the read command to the write command and changing the directory path where you want to store the file.

Data type conversion

There are various types of data such as numeric, factor, character, logical, and so on. Changing one data type to another if the formatting is not done properly is not difficult at all using R. Before changing the variable type, it is essential to look at the data type it currently is. To do that, the following command can be used:

```
> is.numeric(x1)
[1] TRUE
> is.character(x3)
[1] TRUE
> is.vector(x1)
[1] TRUE
> is.matrix(x)
[1] FALSE
> is.data.frame(x)
[1] TRUE
```

When we are checking a numeric variable as numeric or not, the resultant output will display `TRUE` or `FALSE`. The same holds for other data types as well. If any data type is not right, that can be changed by the following script:

```
> as.numeric(x1)
[1] 2.5 1.4 6.3 4.6 9.0
> as.vector(x2)
[1]  TRUE FALSE  TRUE FALSE FALSE
> as.matrix(x)
     x1    x2      x3                x4  x5
[1,] "2.5" " TRUE" "DataMining"      "1" "1+ 0i"
[2,] "1.4" "FALSE" "Statistics"      "2" "6+ 5i"
[3,] "6.3" " TRUE" "Analytics"       "3" "2+ 2i"
[4,] "4.6" "FALSE" "Projects"        "4" "4+ 1i"
[5,] "9.0" "FALSE" "MachineLearning" "5" "6+55i"
> as.data.frame(x)
   x1    x2              x3 x4    x5
1 2.5  TRUE      DataMining  1 1+ 0i
2 1.4 FALSE      Statistics  2 6+ 5i
3 6.3  TRUE       Analytics  3 2+ 2i
4 4.6 FALSE        Projects  4 4+ 1i
5 9.0 FALSE MachineLearning  5 6+55i
> as.character(x2)
[1] "TRUE"  "FALSE" "TRUE"  "FALSE" "FALSE"
```

When using `as.character()`, even a logical vector is changed from logical to character vector. For a numeric variable, nothing is changed as the `x1` variable was already in numeric form. A logical vector can also be changed from logical to factor using this command:

```
> as.factor(x2)
[1] TRUE  FALSE TRUE  FALSE FALSE
Levels: FALSE TRUE
```

Sorting and merging dataframes

Sorting and merging are two importantconcepts in data management. The object can be a single vector or it can be a data frame or matrix. To sort a vector in R, the `sort()` command is used. An decreasing order option can be used to change the order to ascending or descending. For a data frame such as `ArtPiece.csv`, the order command is used to sort the data, where ascending or descending order can be set for multiple variables. Descending order can be executed by putting a negative sign in front of a variable name. Let's use the dataset to explain the concept of sorting in R as shown in the following script:

```
> # Sorting and Merging Data
> ArtPiece<-read.csv("ArtPiece.csv")
> names(ArtPiece)
 [1] "Cid"                          "Critic.Ratings"
"Acq.Cost"
 [4] "Art.Category"                 "Art.Piece.Size"
"Border.of.art.piece"
 [7] "Art.Type"                     "Prominent.Color"
"CurrentAuctionAveragePrice"
[10] "Brush"                        "Brush.Size"
"Brush.Finesse"
[13] "Art.Nationality"              "Top.3.artists"
"CollectorsAverageprice"
[16] "Min.Guarantee.Cost"
> attach(ArtPiece))
```

In the `ArtPiece` dataset, there are 16 variables: 10 numeric variables, and 6 categorical variables. Using the `names` command, all the names of the variables in the dataset can be printed. The `attach` function helps in keeping all the variable names in the current session of R, so that every time the user will not have to type the dataset name before the variable name in the code:

```
> sort(Critic.Ratings)
 [1] 4.9921 5.0227 5.2106 5.2774 5.4586 5.5711 5.6300 5.7723 5.9789
5.9858 6.5078 6.5328
[13] 6.5393 6.5403 6.5617 6.5663 6.5805 6.5925 6.6536 6.8990 6.9367
7.1254 7.2132 7.2191
[25] 7.3291 7.3807 7.4722 7.5156 7.5419 7.6173 7.6304 7.6586 7.7694
7.8241 7.8434 7.9315
[37] 7.9576 8.0064 8.0080 8.0736 8.0949 8.1054 8.2944 8.4498 8.4872
8.6889 8.8958 8.9046
[49] 9.3593 9.8130
```

By default, the sorting is done by ascending order. To sort the vector based on descending order, it is required to put a negative sing before the name of the variable. The `Critic.Ratings` variable can also be sorted based on descending order, which is shown as follows. To sort in descending order, `decreasing` is true and it can be set within the command:

```
> sort(Critic.Ratings, decreasing = T)
 [1] 9.8130 9.3593 8.9046 8.8958 8.6889 8.4872 8.4498 8.2944 8.1054
8.0949 8.0736 8.0080
[13] 8.0064 7.9576 7.9315 7.8434 7.8241 7.7694 7.6586 7.6304 7.6173
7.5419 7.5156 7.4722
[25] 7.3807 7.3291 7.2191 7.2132 7.1254 6.9367 6.8990 6.6536 6.5925
6.5805 6.5663 6.5617
[37] 6.5403 6.5393  6.5328 6.5078 5.9858 5.9789 5.7723 5.6300 5.5711
```

```
5.4586 5.2774 5.2106
    [49] 5.0227 4.9921
```

Instead of sorting a single numeric vector, most of the times, it is required to sort a dataset based on some input variables or attributes present in the dataframe. Sorting a single variable is quite different from sorting a dataframe. The following script shows how a dataframe can be sorted using the `order` function:

```
> i2<-ArtPiece[order(Critic.Ratings,Acq.Cost),1:5]
> head(i2)
   Cid Critic.Ratings Acq.Cost          Art.Category Art.Piece.Size
9    9         4.9921    39200             Vintage I  26in. X 18in.
50  50         5.0227    52500         Portrait Art I  26in. X 24in.
26  26         5.2106    31500            Dark Art II    1in. X 7in.
45  45         5.2774    79345              Gothic II   9in. X 29in.
21  21         5.4586    33600  Abstract Art Type II  29in. X 29in.
38  38         5.5711    35700 Abstract Art Type III   9in. X 12in.
```

The preceding code shows critic ratings and acquisition cost sorted in ascending order. The `order` command is used instead of the `sort` command. The `head` command prints the first six observations by default from the sorted dataset. The number `1:5` in the second argument after the `order` command implies that we want to print the first six observations and five variables from the `ArtPiece` dataset. If it is required to print 10 observations from the beginning, `head(i2, 10)` can be executed. The dataset does not have any missing values; however, the existence of NA or missing values cannot be ruled out from any practical dataset. Hence, with the presence of NA or missing values, sorting a data frame can be tricky. So by including some arbitrary NA values in the dataset, the order command produced the following result:

```
> i2<-ArtPiece[order(Border.of.art.piece, na.last = F),2:6]
> head(i2)
   Critic.Ratings Acq.Cost         Art.Category Art.Piece.Size
Border.of.art.piece
    18         7.5156    34300          Vintage III   29in. X 6in.
    43         6.8990    59500 Abstract Art Type II  23in. X 21in.
    1          8.9046    49700  Abstract Art Type I  17in. X 27in.
Border 1
    12         7.5419    37100        Silhoutte III   28in. X 9in.
Border 10
    14         7.1254    54600           Vintage II   9in. X 12in.
Border 11
    16         7.2132    23100           Dark Art I  10in. X 22in.
Border 11
```

The `NA.LAST` command is used to separate the missing values and NAs from the dataset. They can be either placed at the bottom of the dataset using `NA.LAST` is TRUE or at the

beginning of the data set as `NA.LAST` is FALSE. By keeping NAs separate from the order function, data sanity can be maintained.

The `merge` function helps in combining two data frames. To combine two data frames, at least one column name should be identical. The two data frames can also be joined by using a function called column bind. To display the difference between the column bind and `merge` functions, we have taken `audit.CSV` dataset. There are two small datasets, A and B, prepared out of the `audit` dataset:

```
> A<-audit[,c(1,2,3,7,9)]
> names(A)
[1] "ID"          "Age"         "Employment" "Income"     "Deductions"
> B<-audit[,c(1,3,4,5,6)]
> names(B)
[1] "ID"          "Employment" "Education"  "Marital"    "Occupation"
```

Two columns, `ID` and `Employment`, are common in both datasets `A` and `B`, which can be used as a primary key for merging two data frames. Using the `merge` command, the common columns are taken once in the `result` dataset from the `merge` function. The merged data frame contains all the rows that have complete entries in both the data frames:

```
    > head(merge(A,B),3)
          ID Employment Age   Income Deductions Education   Marital
Occupation
    1 1004641    Private  38  81838.0          0   College Unmarried
Service
    2 1010229    Private  35  72099.0          0 Associate    Absent
Transport
    3 1024587    Private  32 154676.7          0    HSgrad  Divorced
Clerical
```

The `merge` function allows four different ways of combining data: natural join, full outer join, left outer join, and right outer join. Apart from these joins, two data frames can also be merged using any specific column or multiple columns. Natural join helps to keep only rows that match from the data frames, which can be specified by the `all=F` argument:

```
    > head(merge(A,B, all=F),3)
          ID Employment Age   Income Deductions Education Marital
Occupation
    1 1044221    Private  60  7568.23          0   College Married
Executive
    2 1047095    Private  74 33144.40          0    HSgrad Married
Service
    3 1047698    Private  43 43391.17          0  Bachelor Married
Executive
```

Full outer join allows us to keep all rows from both data frames, and this can be specified by the `all=T` command. This performs the complete merge and fills the columns with NA values where there is no matching data in both the data frames. The default setting of the `merge` function drops all unmatched cases from both the data frames; to keep all the cases in new data frame, we need to specify `all=T`:

```
    > head(merge(A,B, all=T),3)
           ID Employment Age   Income Deductions Education Marital
Occupation
    1 1004641    Private  38  81838.0          0      <NA>    <NA>
<NA>
    2 1010229    Private  35  72099.0          0      <NA>    <NA>
<NA>
    3 1024587    Private  32 154676.7          0      <NA>    <NA>
<NA>
```

Left outer join allows us to include all the rows of data frame one (`A`) and only those from data frame two (`B`) that match; to perform this, we need to specify `all.x=T`:

```
    > head(merge(A,B, all.x = T),3)
           ID Employment Age   Income Deductions Education Marital
Occupation
    1 1004641    Private  38  81838.0          0      <NA>    <NA>
<NA>
    2 1010229    Private  35  72099.0          0      <NA>    <NA>
<NA>
    3 1024587    Private  32 154676.7          0      <NA>    <NA>
<NA>
```

Right outer join allows us to include all the rows of data frame `B` and only those from `A` that match; specify `all.y=T`:

```
    > head(merge(A,B, all.y = T),3)
           ID Employment Age   Income Deductions Education Marital
Occupation
    1 1044221    Private  60  7568.23          0   College Married
Executive
    2 1047095    Private  74 33144.40          0    HSgrad Married
Service
    3 1047698    Private  43 43391.17          0  Bachelor Married
Executive
```

In data frames `A` and `B`, two columns are common; they are `ID` and `Employment`. Using the `merge` command, if we select by one common variable, the other common variable would appear on the result data frame. If we select multiple data frames as the criteria to merge, then all duplicate columns from the resulting data frame will disappear:

```
    > head(merge(A,B,by="ID"),3)
          ID Age Employment.x   Income Deductions Employment.y Education
Marital Occupation
    1 1044221  60      Private  7568.23          0      Private   College
Married  Executive
    2 1047095  74      Private 33144.40          0      Private    HSgrad
Married    Service
    3 1047698  43      Private 43391.17          0      Private  Bachelor
Married  Executive
    > head(merge(A,B,by=c("ID","Employment")),3)
          ID Employment Age   Income Deductions Education Marital
Occupation
    1 1044221    Private  60  7568.23          0   College Married
Executive
    2 1047095    Private  74 33144.40          0    HSgrad Married
Service
    3 1047698    Private  43 43391.17          0  Bachelor Married
Executive
```

The `merge` function works when two data frames contain at least one common column, if both the data frames contain disjoint columns or there is no common column between the two data frames, then to combine both data frames, the column bind function can be used. The column bind function prints all the columns in data frame `A` and data frame `B` and puts them side by side:

```
> A<-audit[,c(2,7,9)]
> names(A)
[1] "Age"        "Income"     "Deductions"
> B<-audit[,c(4,5,6)]
> names(B)
[1] "Education"  "Marital"    "Occupation"
> head(cbind(A,B),3)
  Age   Income Deductions Education   Marital Occupation
1  38  81838.0          0   College Unmarried    Service
2  35  72099.0          0 Associate    Absent  Transport
3  32 154676.7          0    HSgrad  Divorced   Clerical
```

Indexing or subsetting dataframes

While working on a client dataset with a large number of observations, it is required to subset the data based on some selection criteria and with or without replacement-based sampling. Indexing is the process of extracting the subset of data from the dataframe based on some logical conditions. The `subset` function helps in extracting elements from the data frame like indexing:

```
    > newdata <- audit[ which(audit$Gender=="Female" & audit$Age > 65), ]
    > rownames(newdata)
     [1] "49"   "537"  "552"  "561"  "586"  "590"  "899"  "1200" "1598"
"1719"
```

The preceding code explains: select those observations from the `audit` dataset where the gender is female and the age is more than 65 years. Which command is used to select that subset of data `audit` based on the preceding two criteria? There are 10 observations satisfying the preceding condition; the row numbers of the data frame are printed previously. A similar result can be obtained by using the `subset` function as well. Instead of the `which` function, the `subset` function should be used, as the latter is more efficient in passing multiple conditions. Let's take a look at the way the `subset` function is used:

```
    > newdata <- subset(audit, Gender=="Female" & Age > 65,
select=Employment:Income)
    > rownames(newdata)
     [1] "49"   "537"  "552"  "561"  "586"  "590"  "899"  "1200" "1598"
"1719"
```

The additional argument in the `subset` function makes the function more efficient as it provides the additional benefit of selecting specific columns from the dataframe where the logical condition is satisfied.

Date and time formatting

The date functions return a `Date` class that represents the number of days since January 1, 1970. The `as.numeric()` function can be used to create a numeric variable with the number of days since 1/1/1970. The return value of `as.Date()` is a `Date` class object:

```
> Sys.time()
[1] "2015-11-10 00:43:22 IST"
> dt<-as.Date(Sys.time())
> class(dt)
[1] "Date"
```

The system time function captures the date and time with the time zone. When we convert the system time using the `as.Date` function and stored as a new object in R, we find that the class of that object is `Date`. The `weekdays` function returns the name of the day such as `"Monday"` or `"Wednesday"`. The `months` function returns the name of the month from the `date` variable. The `quarters` function returns the name of the quarter for the date object and year value also can be extracted using the `substr()` command:

```
> weekdays(as.Date(Sys.time()))
```

```
[1] "Monday"
> months(as.Date(Sys.time()))
[1] "November"
> quarters(as.Date(Sys.time()))
[1] "Q4"
> substr(as.POSIXct(as.Date(Sys.time())),1,4)
[1] "2015"
```

If the date variable given in the dataset is not in proper format for further computations, it can be formatted using the `format` function:

```
> format(Sys.time(),format = "%m %d %y")
[1] "11 10 15"
```

There are various options that can be passed to the format argument based on the user requirement:

Option	What it does
`#%d`	Means day as a number from (`0-31`) `01-31`
`#%a`	Means abbreviated weekday as Mon
`#%`	A means unabbreviated weekday, Monday
`#%m`	Month (`00-12`)
`#%b`	Abbreviated month
`#%B`	Unabbreviated month January
`#%y`	Two-digit year (`13`)
`#%Y`	Four-digit year (`2013`)

Table 1: Formatting date options

Practical datasets contain date fields such as the transaction date in retail, visit date in healthcare, and processing date in BFSI; and any time series data contains at least one time element. To include the date variable in any statistical model, data transformation is required, such as calculating the vintage of a customer in a retail scenario. The data transformation can be done using the aforementioned options.

Creating new functions

There are two different types of functions in R, user-defined functions and built-in functions.

User-defined functions

A user-defined function provides customization and flexibility to users to write their functions to perform computations. It has a general notation shown as follows:

```
newFunc <- function(x){define function}
> int<-seq(1:20)
> int
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
> myfunc<-function(x){x*x}
> myfunc(int)
 [1]   1   4   9  16  25  36  49  64  81 100 121 144 169 196 225 256
289 324
 361 400
```

In the preceding script, we are creating a small sequence of numbers from 1 to 20 and a user-defined function to calculate the square of each integer. Using the new function, we can calculate the square of any number. So the user can define and create his or her own custom function.

Built-in functions

Built-in functions, such as mean, median, standard deviation, and so on, provide the user the ability to compute basic statistics using R. There are many built-in functions; the following table displays a few important built-in functions:

Function	**Description**
`abs(x)`	Absolute value
`sqrt(x)`	Square root
`ceiling(x)`	Rounding up the number
`floor(x)`	Rounding down the number
`trunc(x)`	`trunc(5.99)` is `5`
`round(x, digits=n)`	`round(3.475, digits=2)` is `3.48`
`signif(x, digits=n)`	`signif(3.475, digits=2)` is `3.5`

`cos(x),sin(x),tan(x)`	Also `acos(x),cosh(x),acosh(x)`, and so on
`log(x)`	Natural logarithm
`log10(x)`	Common logarithm
`exp(x)`	`e^x`

Table 2: Some built-in functions

Loop concepts – the for loop

The `for` loop is the most popular looping construct in R. Using a `for` loop, a similar task can be performed many times iteratively, let's look at a sample example where the `for` loop concept is applied. In the following code, a series of numbers from 10 to 25 is created. The null vector `v` is acting like a storage unit. If the condition mentioned in the following code is not met, the loop is never executed:

```
x<-100:200
y <- NULL # NULL vector as placeholder
for(i in seq(along=x)) {
   if(x[i] < 150) {
   y <- c(y, x[i] - 50)
   } else {
   y <- c(y, x[i] + 50)
   }
   }
print(y)
```

Loop concepts – the repeat loop

The `repeat` loop is used to iterate a certain calculation over a vector or dataframe. There is no provision to check the condition to exit the loop; generally a `break` statement is used to exit the loop. If you fail to provide any break condition within the `repeat` loop, you will end up running the `repeat` loop infinitely. Let's look at the code showing how to write a `repeat` loop. The `break` condition used in the following code is if `x > 2.6`:

```
x <- 100
repeat {
  print(x)
  x = sqrt(x)+10
  if (x > 2.6){
    break
```

```
  }
}
```

Loop concepts – while conditions

The structure of the `while` loop in R is simple; it starts with a desired result that the user wants to see from the experiment. As the condition is entered in the beginning, the body of the loop will start iteration and go on as long the condition is being met. The skeleton structure of a `while` loop consists of a constraint condition to start with; here is an example:

```
x <- 10
while (x < 60) {
  print(x)
  x = x+10
}
```

If we compare different types of loops in the R programming language, `for` loop and `while` loop are very frequently used; `repeat` loop is not used that frequently because of the time it takes to complete the run. If we compare the loops with the apply group of functions, the latter set of functions is quite effective in handling different tasks in R. Let's look at the apply group of functions.

Apply concepts

The `apply` function uses an array, a matrix, or a dataframe as an input and returns the result in an array format. The calculation or operation is defined by the user's custom function or using any built-in functions. The margin argument is used to specify which margin we want to apply to the function and which margin we wish to keep. If the array we are using is a matrix, then we can specify the margin to be either `1` (apply the function to the rows) or `2` (apply the function to the columns). The function can be any function such as `mean`, `median`, `standard deviation`, `variance`, and so on that is built in or user defined. Here we are going to use iris dataset to perform the task:

```
> apply(ArtPiece[,2:3],2,mean)
Critic.Ratings       Acq.Cost
      7.200416   44440.900000
> apply(ArtPiece[,2:3],1,mean)
 [1] 24854.45 26604.68 17153.69 14353.28 14003.47 19604.05 14703.27
15753.29 19602.50
 [10] 26954.24 19254.00 18553.77 18903.97 27303.56 24153.74 11553.61
23804.04 17153.76
```

```
    [19] 19953.30 24854.22 16802.73 20303.33 14354.91 26952.99 24503.28
15752.61 28004.45
    [28] 30803.81 29403.27 19604.00 29053.88 17152.81 33253.91 24502.89
37453.92 12604.15
    [37] 21353.82 17852.79 28703.83 29753.25 23453.27 18204.34 29753.45
27654.05 39675.14
    [46] 24853.61 16102.99 13653.98 14353.66 26252.51
```

The `lapply` function is useful when dealing with (applying any function) dataframes. In R, the dataframe is considered a list and the variables in the dataframe are the elements of the list. We can, therefore, apply a function to all the variables in a dataframe using the `lapply` function:

```
> lapply(ArtPiece[,2:3],mean)
$Critic.Ratings
[1] 7.200416
$Acq.Cost
[1] 44440.9
```

The `sapply` function applies to elements in a list and returns the results in a vector, matrix, or list. When the argument is `simplify=F`, then the `sapply` function returns the results in a list just like the `lapply` function. However, when the argument is `simplify=T`, which is the default argument, the `sapply` function returns the results in a simplified form if at all possible:

```
> sapply(ArtPiece[,2:3],mean)
Critic.Ratings       Acq.Cost
      7.200416   44440.900000
```

When we want to apply a function to subsets of a vector and the subsets are defined by some other vector, usually a factor. The output from `tapply` is a matrix/array, where an element in the matrix/array is the value of `f` at a grouping `g` of the vector, and `g` gets pushed to the row/col names:

```
> head(tapply(Critic.Ratings,Acq.Cost,summary),3)
$`23100`
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  7.213   7.213   7.213   7.213   7.213   7.213
$`25200`
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  8.294   8.294   8.294   8.294   8.294   8.294
$`27300`
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  7.958   7.958   7.958   7.958   7.958   7.958
```

There are other functions in the `apply` family of functions, such as:

- `eapply`: Apply a function over values in an environment
- `mapply`: Apply a function to multiple list or vector arguments
- `rapply`: Recursively apply a function to a list

String manipulation

String manipulation or character manipulation is an important aspect of any data management system. In a typical real-world dataset, names of customers for example are written in different ways, such as J H Smith, John h Smith, John h smith, and so on. Upon verifying, it is observed that all three names belong to the same person. In typical data management, it is important to standardize the text columns or variables in a dataset because R is case sensitive and it reads any discrepancy as a new data point. There can be many other variables such as the name/model of a vehicle, product description, and so on. Let's look how the text can be standardized using some functions:

```
    > x<-"data Mining is not a difficult subject, anyone can master the
subject"
    > class(x)
    [1] "character"
    > substr(x, 1, 12)
    [1] "data Mining "
```

The object `X` in the preceding script is a string or character object. The `substr` command is used to pull a `sub` string from the string with the position defined in the function. If certain patterns or texts need to be altered or changed, then the `sub` command can be used. There are four important arguments that the user needs to pass: the string in which a pattern needs to be searched, the pattern, the modified pattern that needs to be replaced, and whether case sensitivity is acceptable or not. Let's look at a sample script:

```
    > sub("data mining", "The Data Mining", x, ignore.case =T, fixed=FALSE)
    [1] "The Data Mining is not a difficult subject, anyone can master the
subject"
    > strsplit(x, "")
    [[1]]
     [1] "d" "a" "t" "a" " " "M" "i" "n" "i" "n" "g" " " "i" "s" " " "n"
"o" "t" " " "a" " "
    [22] "d" "i" "f" "f" "i" "c" "u" "l" "t" " " "s" "u" "b" "j" "e" "c"
"t" "," " " "a" "n"
    [43] "y" "o" "n" "e" " " "c" "a" "n" " " "m" "a" "s" "t" "e" "r" " "
"t" "h" "e" " " "s"
    [64] "u" "b" "j" "e" "c" "t"
```

The `strsplit` function helps in expanding the letters from a string. The `sub` command is used to alter a pattern that is not right in the string. The `ignore.Case` option provides the user the chance to keep the case sensitivity on or off while searching for the pattern in the defined string.

NA and missing value management

Missing value treatment is an important task in standard data mining literature. In the R programming language, missing values are represented as `NA`. NAs are not string or numeric values; they are considered as an indicator for missing values. After importing a dataset into the R programming platform, it is important to check whether, for any variable, missing values exist or not; to check that, the `is.na()` command is used. Please see the example given here:

```
> x<-c(12,13,14,21,23,24,NA,25,NA,0,NA)
> is.na(x)
 [1] FALSE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE  TRUE FALSE  TRUE
> mean(x,na.rm=TRUE)
[1] 16.5
> mean(x)
[1] NA
```

Object `x` is a numeric vector that contains some NA values, to verify that `is.na()` can be used, wherever it is satisfied the result would be `TRUE`. If we compute anything with the presence of `NA`, we end up getting an error or no result. Either we can replace the data set by altering the NA values, or we can remove those NA values while performing any computation. As in the preceding script, it is `na.rm=T` that is used to remove the NAs from the mean computation for object `x`.

Missing value imputation techniques

To delete missing values from the dataset, `na.omit()` can be used. It removes the entire row even if the data is missing for a single variable. There are various missing value imputation methods:

- **Mean imputation**: The missing values in a data vector is replaced by the mean or median value of that vector, excluding the NA

- **Local average method**: Taking the local average for the missing value, by taking into account 3 or 5 periods moving average, that is for a 3 period the average of missing data prior value and posterior value can decide what the missing value should be
- **Keeping that separate**: Sometimes the imputation is simply not possible, and may be the client would be interested in keeping the missing values separate to understand the missing behavior separately
- Model-based: There are some model-based missing value imputation techniques such as the regression-based missing value prediction method
- **Clustering**: Similar to regression-based prediction to the missing value imputation, k-means clustering can be used to impute the missing values from the dataset

Summary

In the light of the preceding discussion, it can be summarized that data manipulation and data management is an important step in performing data mining on various live projects. Since R provides a better statistical programming platform and visualization, the use of R to explain various data mining concepts to the readers makes sense. In this chapter, we looked at an introduction to data mining and R with concepts, a few programming basics, R data types, and so on. We also covered importing and exporting of various external file formats using R, sorting and merging concepts, and missing data management techniques.

In the next chapter, we are going to learn more about performing exploratory data analysis using R and how to understand univariate, bivariate and multivariate datasets. First of all, we are going to understand the concepts, practical interpretation, and then R implementation to gain knowledge on exploratory data analysis.

2

Exploratory Data Analysis with Automobile Data

Exploratory data analysis is an integral part of data mining. It involves numerical as well as graphical representation of variables in a dataset for easy understanding and quick conclusion about a dataset. It is important to get an understanding about the dataset, type of variables considered for analysis, association between various variables, and so on. Creating cross tabulations to understand the relationship between categorical variables and performing classical statistical tests on the data to verify various different hypotheses about the data can be tested out.

You will now get an understanding about the following things:

- How to use basic statistics to know properties of a single and multiple variables
- How to calculate correlation and association between two or more variables
- Performing multivariate data analysis
- Statistical properties of various probability functions for any dataset
- Applying statistical tests on data to conclude hypotheses
- Comparing two or more samples

Univariate data analysis

To generate univariate statistics about a dataset, we have to follow two approaches, one for continuous variables and the other for discrete or categorical variables. Univariate statistics for continuous variables includes numerical measures such as averages (mean), variance, standard deviation, quantiles, median quartiles, and so on. The mean represents each and every point in the dataset; the variance indicates the fluctuation/deviation of the individual data points from the mean, which is the center of the distribution. Quantiles are also known as percentiles, which divide the distribution into 100 equal parts. The 10^{th} percentile value is equivalent to 1 decile, the 25^{th} percentile value is equivalent to the 1^{st} quartile, and 75^{th} percentile value is equivalent to the 3^{rd} quartile.

There are other statistical measures of central tendency for understanding the univariate characteristics of a dataset. Median and mode are referred to as positional values, but still mode can be looked at for a continuous variable to check whether it is a bimodal series. In case of bimodal series, it is difficult to compute the center of the distribution. For ordinal data or rank data calculation of mean, representation is a good idea; it is always suggested to represent univariate statistics using median or mode. Comparison of mean, median, and mode values along with skewness, kurtosis, and standard deviation gives a clear picture about the shape of the data distribution. All of these measures of central tendency and measures of dispersion together can be calculated using a single command and also using different individual commands, which are given as follows.

Here, we are going to use two datasets, `diamonds.csv` and `Cars93.csv`. Both belong to two libraries that are inbuilt in the R software for practical demo purposes.

Let's use a few commands in R to understand the data better:

```
> names(Cars93)
[1] "Manufacturer" "Model" "Type" "Min.Price"
[5] "Price" "Max.Price" "MPG.city" "MPG.highway"
[9] "AirBags" "DriveTrain" "Cylinders" "EngineSize"
[13] "Horsepower" "RPM" "Rev.per.mile" "Man.trans.avail"
[17] "Fuel.tank.capacity" "Passengers" "Length" "Wheelbase"
[21] "Width" "Turn.circle" "Rear.seat.room" "Luggage.room"
[25] "Weight" "Origin" "Make"
```

The `Cars93.csv` dataset contains the previously mentioned variable names and it has 27 variables and 93 observations. The variable type can be printed using the `str()` function:

```
> str(Cars93)
'data.frame': 93 obs. of 27 variables:
```

```
$ Manufacturer : Factor w/ 32 levels "Acura","Audi",..: 1 1 2 2 3 4 4 4 4 5
...
$ Model : Factor w/ 93 levels "100","190E","240",..: 49 56 9 1 6 24 54 74
73 35 ...
$ Type : Factor w/ 6 levels "Compact","Large",..: 4 3 1 3 3 3 2 2 3 2 ...
$ Min.Price : num 12.9 29.2 25.9 30.8 23.7 14.2 19.9 22.6 26.3 33 ...
$ Price : num 15.9 33.9 29.1 37.7 30 15.7 20.8 23.7 26.3 34.7 ...
$ Max.Price : num 18.8 38.7 32.3 44.6 36.2 17.3 21.7 24.9 26.3 36.3 ...
$ MPG.city : int 25 18 20 19 22 22 19 16 19 16 ...
$ MPG.highway : int 31 25 26 26 30 31 28 25 27 25 ...
$ AirBags : Factor w/ 3 levels "Driver & Passenger",..: 3 1 2 1 2 2 2 2 2 2
...
$ DriveTrain : Factor w/ 3 levels "4WD","Front",..: 2 2 2 2 3 2 2 3 2 2 ...
$ Cylinders : Factor w/ 6 levels "3","4","5","6",..: 2 4 4 4 2 2 4 4 4 5
...
$ EngineSize : num 1.8 3.2 2.8 2.8 3.5 2.2 3.8 5.7 3.8 4.9 ...
$ Horsepower : int 140 200 172 172 208 110 170 180 170 200 ...
$ RPM : int 6300 5500 5500 5500 5700 5200 4800 4000 4800 4100 ...
$ Rev.per.mile : int 2890 2335 2280 2535 2545 2565 1570 1320 1690 1510 ...
$ Man.trans.avail : Factor w/ 2 levels "No","Yes": 2 2 2 2 2 1 1 1 1 1 ...
$ Fuel.tank.capacity: num 13.2 18 16.9 21.1 21.1 16.4 18 23 18.8 18 ...
$ Passengers : int 5 5 5 6 4 6 6 6 5 6 ...
$ Length : int 177 195 180 193 186 189 200 216 198 206 ...
$ Wheelbase : int 102 115 102 106 109 105 111 116 108 114 ...
$ Width : int 68 71 67 70 69 69 74 78 73 73 ...
$ Turn.circle : int 37 38 37 37 39 41 42 45 41 43 ...
$ Rear.seat.room : num 26.5 30 28 31 27 28 30.5 30.5 26.5 35 ...
$ Luggage.room : int 11 15 14 17 13 16 17 21 14 18 ...
$ Weight : int 2705 3560 3375 3405 3640 2880 3470 4105 3495 3620 ...
$ Origin : Factor w/ 2 levels "USA","non-USA": 2 2 2 2 2 1 1 1 1 1 ...
$ Make : Factor w/ 93 levels "Acura Integra",..: 1 2 4 3 5 6 7 9 8 10 ...
```

Calculating the univariate statistics for a few continuous (`Price`, `MPG.city`, and `MPG.highway`) and discrete variables (`Type`, `AirBags`, and manual transmission available) can be displayed here. You can practice the rest of the variable to get a complete understanding of the dataset.

The `max()` command estimates the maximum value for a variable. `min()` computes the minimum value. `sum()` calculates the total of all the values. The `mean()` function calculates the arithmetic average of the values, the `median()` function calculates the median value, and `range()` calculates the vector of `min()` and `max()`. `var()` computes the sample variance and `cor()` correlation between two vectors. `rank()` calculates the vector of the ranks of the values in a vector. The `quantile()` function computes a vector containing the minimum, lower quartile, median, upper quartile, and maximum of a vector.

Using the `summary()` function for univariate:

```
> summary(Cars93$Price)
Min. 1st Qu. Median Mean 3rd Qu. Max.
7.40 12.20 17.70 19.51 23.30 61.90
> summary(Cars93$MPG.city)
Min. 1st Qu. Median Mean 3rd Qu. Max.
15.00 18.00 21.00 22.37 25.00 46.00
> summary(Cars93$MPG.highway)
Min. 1st Qu. Median Mean 3rd Qu. Max.
20.00 26.00 28.00 29.09 31.00 50.00
> summary(Cars93$Type)
Compact Large Midsize Small Sporty Van
16 11 22 21 14 9
> summary(Cars93$AirBags)
Driver & Passenger Driver only None
16 43 34
> summary(Cars93$Man.trans.avail)
No Yes
32 61
```

Now let's look at the results of the summary function on the data frame. For a continuous variable, the numerical measures of central tendency are computed, and, for a categorical variable, the class frequencies are computed:

```
> summary(Cars93)
Manufacturer Model Type Min.Price
Chevrolet: 8 100 : 1 Compact:16 Min. : 6.70
Ford : 8 190E : 1 Large :11 1st Qu.:10.80
Dodge : 6 240 : 1 Midsize:22 Median :14.70
Mazda : 5 300E : 1 Small :21 Mean :17.13
Pontiac : 5 323 : 1 Sporty :14 3rd Qu.:20.30
Buick : 4 535i : 1 Van : 9 Max. :45.40
(Other) :57 (Other):87
Price Max.Price MPG.city MPG.highway
Min. : 7.40 Min. : 7.9 Min. :15.00 Min. :20.00
1st Qu.:12.20 1st Qu.:14.7 1st Qu.:18.00 1st Qu.:26.00
Median :17.70 Median :19.6 Median :21.00 Median :28.00
Mean :19.51 Mean :21.9 Mean :22.37 Mean :29.09
3rd Qu.:23.30 3rd Qu.:25.3 3rd Qu.:25.00 3rd Qu.:31.00
Max. :61.90 Max. :80.0 Max. :46.00 Max. :50.00
AirBags DriveTrain Cylinders EngineSize
Driver & Passenger:16 4WD :10 3 : 3 Min. :1.000
Driver only :43 Front:67 4 :49 1st Qu.:1.800
None :34 Rear :16 5 : 2 Median :2.400
6 :31 Mean :2.668
8 : 7 3rd Qu.:3.300
rotary: 1 Max. :5.700
```

```
Horsepower RPM Rev.per.mile Man.trans.avail
Min. : 55.0 Min. :3800 Min. :1320 No :32
1st Qu.:103.0 1st Qu.:4800 1st Qu.:1985 Yes:61
Median :140.0 Median :5200 Median :2340
Mean :143.8 Mean :5281 Mean :2332
3rd Qu.:170.0 3rd Qu.:5750 3rd Qu.:2565
Max. :300.0 Max. :6500 Max. :3755
Fuel.tank.capacity Passengers Length Wheelbase
Min. : 9.20 Min. :2.000 Min. :141.0 Min. : 90.0
1st Qu.:14.50 1st Qu.:4.000 1st Qu.:174.0 1st Qu.: 98.0
Median :16.40 Median :5.000 Median :183.0 Median :103.0
Mean :16.66 Mean :5.086 Mean :183.2 Mean :103.9
3rd Qu.:18.80 3rd Qu.:6.000 3rd Qu.:192.0 3rd Qu.:110.0
Max. :27.00 Max. :8.000 Max. :219.0 Max. :119.0
Width Turn.circle Rear.seat.room Luggage.room
Min. :60.00 Min. :32.00 Min. :19.00 Min. : 6.00
1st Qu.:67.00 1st Qu.:37.00 1st Qu.:26.00 1st Qu.:12.00
Median :69.00 Median :39.00 Median :27.50 Median :14.00
Mean :69.38 Mean :38.96 Mean :27.83 Mean :13.89
3rd Qu.:72.00 3rd Qu.:41.00 3rd Qu.:30.00 3rd Qu.:15.00
Max. :78.00 Max. :45.00 Max. :36.00 Max. :22.00
NA's :2 NA's :11
Weight Origin Make
Min. :1695 USA :48 Acura Integra: 1
1st Qu.:2620 non-USA:45 Acura Legend : 1
Median :3040 Audi 100 : 1
Mean :3073 Audi 90 : 1
3rd Qu.:3525 BMW 535i : 1
Max. :4105 Buick Century: 1
(Other) :87
```

The summary command for continuous variables such as RPM, horsepower, and so on shows the minimum, 1st quartile, mean, median, 3rd quartile, and maximum values. The univariate statistics for the categorical variable, which are car type, airbags, manual transmission availability, and so on, are represented as frequency tables. The class that has the highest frequency is considered to be the modal class.

Similar summary statistics can be generated using a few more functions such as `fivenum()` and `describe()`, which provides more information than the summary function:

```
> fivenum(Cars93$Price)
[1] 7.4 12.2 17.7 23.3 61.9
> fivenum(Cars93$MPG.city)
[1] 15 18 21 25 46
> fivenum(Cars93$MPG.highway)
[1] 20 26 28 31 50
```

The `describe()` function from the library (`Hmisc`) can be used to get a better understanding of the data description:

```
> library(Hmisc)
> describe(Cars93)
Cars93
27 Variables 93 Observations
Price
n missing unique Info Mean .05 .10 .25 .50 .75 .90
93 0 81 1 19.51 8.52 9.84 12.20 17.70 23.30 33.62
.95
36.74
lowest : 7.4 8.0 8.3 8.4 8.6, highest: 37.7 38.0 40.1 47.9 61.9
MPG.city
n missing unique Info Mean .05 .10 .25 .50 .75 .90
93 0 21 0.99 22.37 16.6 17.0 18.0 21.0 25.0 29.0
.95
31.4
lowest : 15 16 17 18 19, highest: 32 33 39 42 46
---------------------------------------------------------------------------
--------------
MPG.highway
n missing unique Info Mean .05 .10 .25 .50 .75 .90
93 0 22 0.99 29.09 22.0 23.2 26.0 28.0 31.0 36.0
.95
37.4
lowest : 20 21 22 23 24, highest: 38 41 43 46 50
---------------------------------------------------------------------------
--------------
AirBags
n missing unique
93 0 3
Driver & Passenger (16, 17%), Driver only (43, 46%)
None (34, 37%)
---------------------------------------------------------------------------
--------------
Man.trans.avail
n missing unique
93 0 2
No (32, 34%), Yes (61, 66%)
```

The univariate summary statistics can also be calculated using the `apply()` function. The univariate statistics gives an impression about the shape of the distribution:

```
> n_cars93<-Cars93[,c(5,7,8)]
> c_cars93<-Cars93[,c(3,9,16)]
> apply(n_cars93,2,mean)
Price MPG.city MPG.highway
19.50968 22.36559 29.08602
```

To understand the shape of a distribution for a variable, we can use skewness and box plot. We can also use the `skewness` function, which is available in `library(e1071)`:

```
> library(e1071)
> apply(n_cars93,2,skewness)
Price MPG.city MPG.highway
1.483982 1.649843 1.190507
```

We can create a custom function for measuring skewness and use that along with the `apply()` function:

```
> skewness<-function(x){
+ m3<-sum((x-mean(x))^3)/length(x)
+ s3<-sqrt(var(x))^3
+ m3/s3 }
> apply(n_cars93,2,skewness)
Price MPG.city MPG.highway
1.483982 1.649843 1.190507
```

Skewness is known as a measure of symmetrical distribution where the value of skewness indicates whether a distribution is positively skewed or negatively. When the value of skewness approaches 0 or is close to zero, it indicates that the distribution is symmetric and the mean, median, and mode are exactly the same. When the value of skewness is less than 0, it indicates the value of the mean is less than the mode, and this is because of availability of extreme values on the negative side of the normal distribution. When the value of skewness is greater than 0, it indicates that the value of the mean is greater than the mode, and this is because of availability of extreme values on the right-hand side of the normal distribution. Since identification of outliers and their removal is very important, the measure of skewness helps in that direction, but that is not the only way to figure out outliers. There are other methods such as boxplot, and other custom outlier detection formulas. If we look at the previous three variables, it implies that there are indications that outlier values may exist on the positive side of the normal curve for price, `MPG.city`, and `MPG.highway` variables because the skewness value is greater than 0. To verify the existence of outliers, we can take a boxplot and print the outliers.

Bivariate analysis

The relationship or association between two variables is known as bivariate analysis. There are three possible ways of looking at the relationship:

- Numeric-to-numeric relationship
- Numeric-to-categorical relationship
- Categorical-to-categorical relationship

To know the bivariate relationship between two numeric variables, typically a scatter plot is used if the two variables happen to be continuous, and a bar plot is used if one variable is categorical and the other is continuous:

```
> library(ggplot2)
> library(gridExtra)
> ggplot(Cars93,
aes(Cars93$Price,Cars93$MPG.city))+geom_point(aes(colour=(Cars93$Type)))+ge
om_smooth()
```

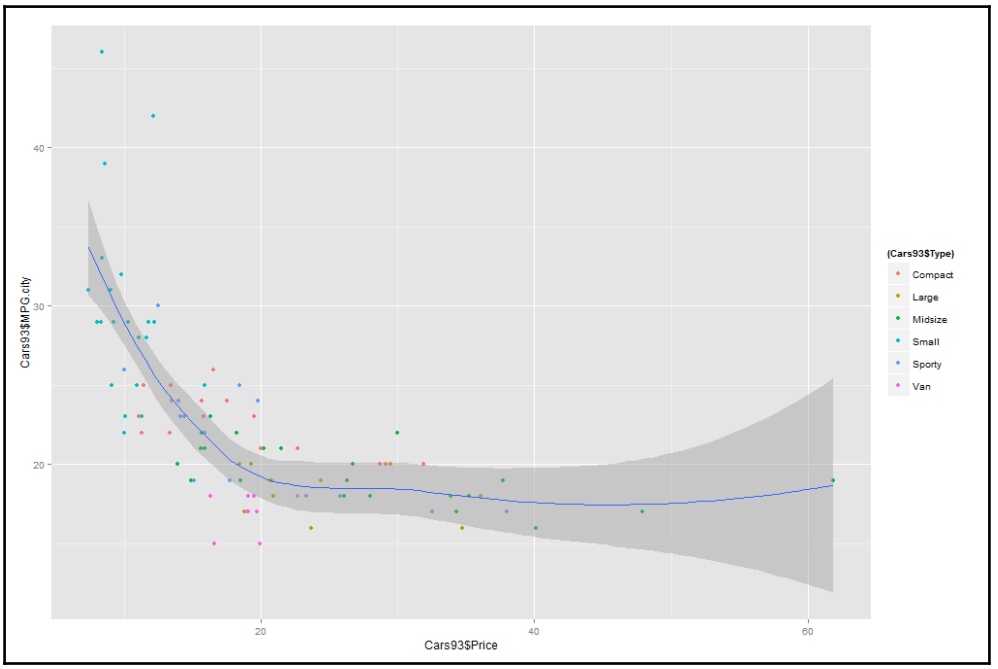


Figure 1: Showing the relationship between price and mileage within a city for different car types

Similarly, the relationship between price and highway mileage can be represented using a scatter plot as well:

```
> library(ggplot2)
> library(gridExtra)
> ggplot(Cars93,
aes(Cars93$Price,Cars93$MPG.highway))+geom_point(aes(colour=(Cars93$Type)))
+geom_smooth()
```

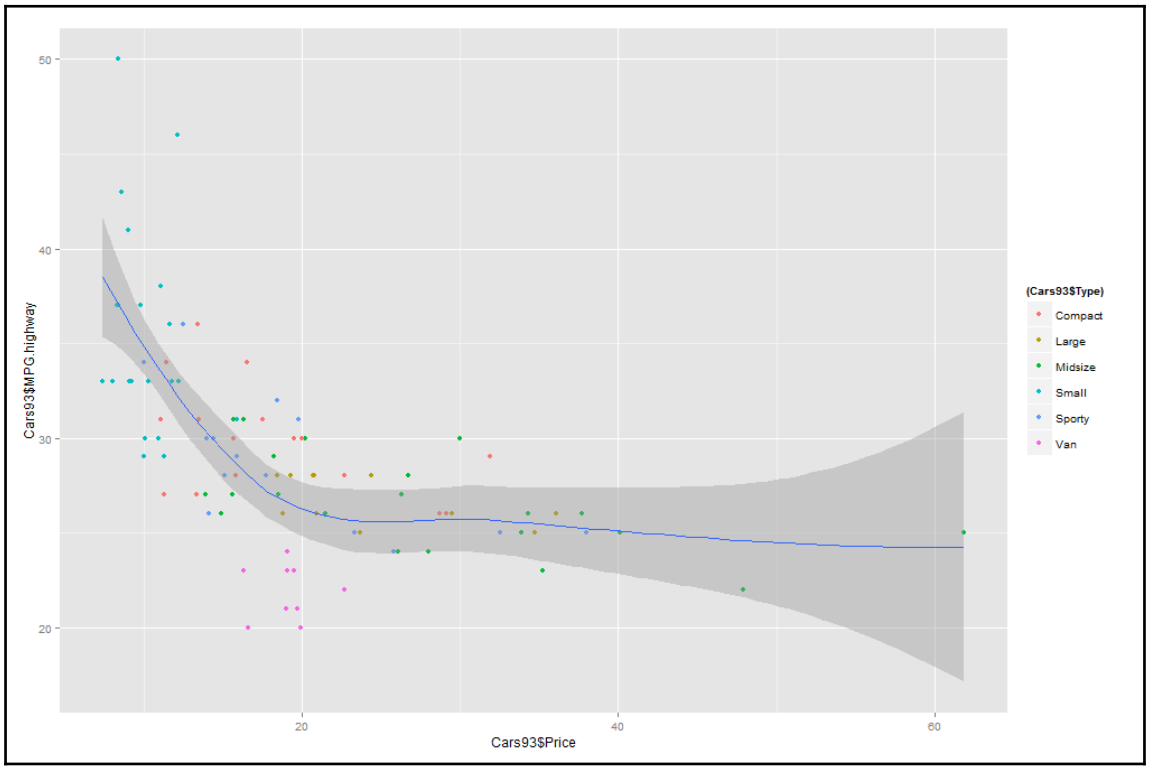


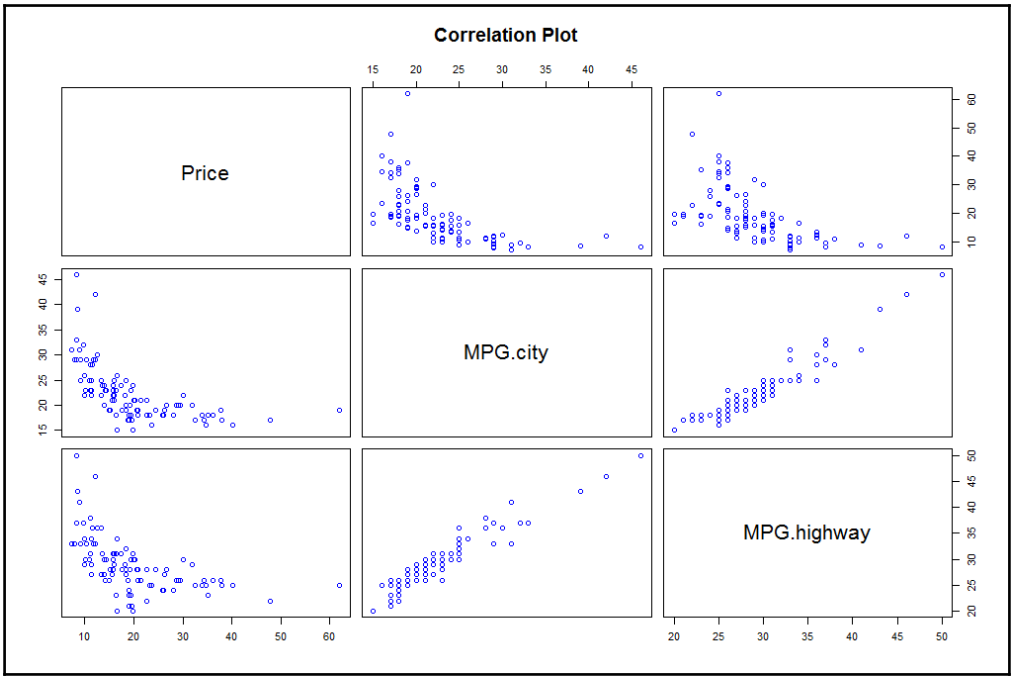
Figure 2: Relationship between price and mileage on highways

The numeric-categorical and two categorical relationships are explained in `Chapter 3`, *Visualize Diamond Dataset*, in detail.

Multivariate analysis

The multivariate relationship is a statistical way of looking at multiple dependent and independent variables and their relationships. In this chapter, we will briefly talk about multivariate relationships between more than two variables, but we will discuss the details of multivariate analysis in our subsequent chapters. Multivariate relationships between various variables can be known by using the correlation method as well as cross tabulation:

```
> pairs(n_cars93,main="Correlation Plot", col="blue")
```



Understanding distributions and transformation

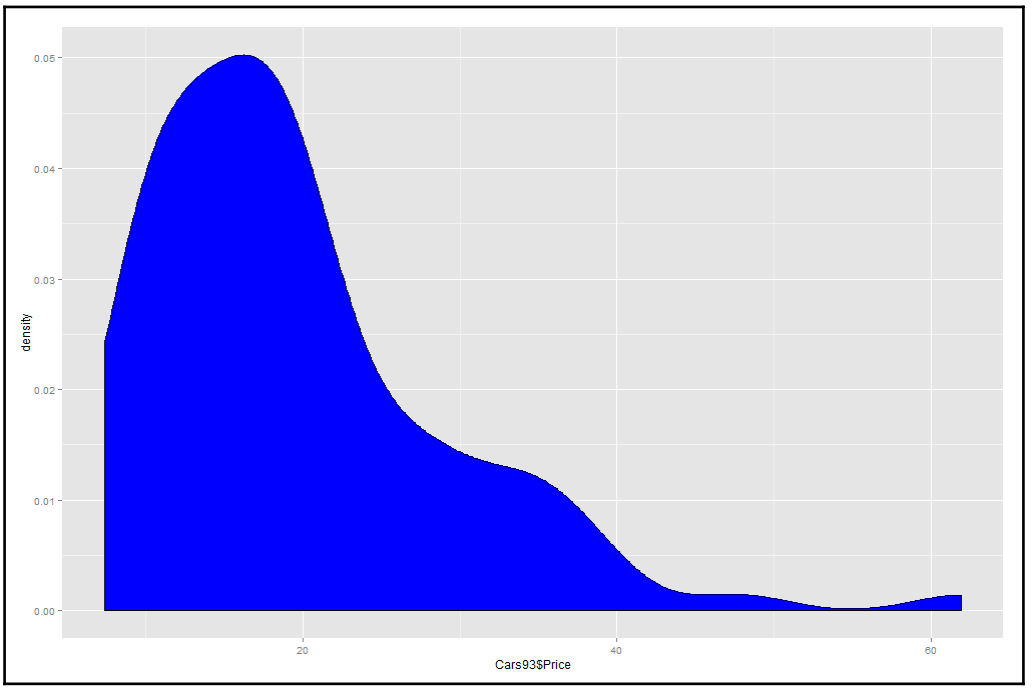
Understanding probability distributions is important in order to have a clear idea about the assumptions of any statistical hypothesis test. For example, in linear regression analysis, the basic assumption is that the error distribution should be normally distributed and the variables' relationship should be linear. Hence, before moving to the stage of model formation, it is important to look at the shape of the distribution and types of transformations that one may look into to make the things right. This is done so that any further statistical techniques can be applied on the variables.

Normal probability distribution

The concept of normal distribution is based on **Central Limit Theorem** (**CLT**), which implies that the population of all possible samples of size n drawn from a population with mean μ and variance $\sigma 2$ approximates a normal distribution with mean μ and $\sigma 2 \backsim n$ when n increases towards infinity. Checking the normality of variables is important to remove outliers so that the prediction process does not get influenced. Presence of outliers not only deviates the predicted values but would also destabilize the predictive model. The following sample code and example show how to check normality graphically and interpret the same.

To test out the normal distribution, we can use the mean, median, and mode for some of the variables:

```
> mean(Cars93$Price)
[1] 19.50968
> median(Cars93$Price)
[1] 17.7
> sd(Cars93$Price)
[1] 9.65943
> var(Cars93$Price)
[1] 93.30458
> skewness(Cars93$Price)
[1] 1.483982
ggplot(data=Cars93, aes(Cars93$Price)) + geom_density(fill="blue")
```



From the preceding image, we can conclude that the `price` variable is positively skewed because of the presence of some outlier values on the right-hand side of the distribution. The mean of the `price` variable is inflated and greater than the mode because the mean is subject to extreme fluctuations.

Now let's try to understand a case where normal distribution can be used to answer any hypothesis.

Suppose the variable mileage per gallon on a highway is normally distributed with a mean of `29.08` and a standard deviation of `5.33`. What is the probability that a new car would provide a mileage of `35`?

```
> pnorm(35,mean(Cars93$MPG.highway),sd(Cars93$MPG.highway),lower.tail = F)
[1] 0.1336708
```

Hence the required probability that a new car would provide a mileage of `35` is 13.36%, since the expected mean is higher than the actual mean; the lower tail is equivalent to false.

Binomial probability distribution

Binomial distribution is known as discrete probability distribution. It describes the outcome of an experiment. Each trial is assumed to have only two outcomes: either success or failure, either yes or no. For instance, in the `Cars93` dataset variable, whether manual transmission is available or not is represented as *yes* or *no*.

Let's take an example to explain where binomial distribution can be used. The probability of a defective car given a specific component is not functioning is 0.1%. You have 93 cars manufactured. What is the probability that at least 1 defective car can be detected from the current lot of `93`:

```
> pbinom(1,93,prob = 0.1)
[1] 0.0006293772
```

So the required probability that a defective car can get identified in a lot of 93 cars is `0.0006`, which is very less, given the condition that the probability of a defective part is `0.10`.

Poisson probability distribution

Poisson distribution is for count data where, given the data and information about an event, you can predict the probability of any number occurring within that limit using the Poisson probability distribution.

Let's take an example. Suppose 200 customers on an average visit a particular e-commerce website portal every minute. Then find the probability of having 250 customers visit the same website in a minute:

```
> ppois(250,200,lower.tail = F)
[1] 0.0002846214
```

Hence, the required probability is `0.0002`, which is very rare. Apart from the aforementioned common probability distributions, which are used very frequently, there are many other distributions that can be used in rare situations.

Interpreting distributions

Calculation of probability distributions and fitting data points specific to various types of distribution and subsequent interpretation help in forming a hypothesis. That hypothesis can be used to estimate the event probability given a set of parameters. Let's take a look at the interpretation of different types of distributions.

Interpreting continuous data

The maximum likelihood estimation of the distributional parameters from any variable in a dataset can be known by fitting a distribution. The density function is available for distributions such as "beta", "cauchy", "chi-squared", "exponential", "f", "gamma", "geometric", "log-normal", "lognormal", "logistic", "negative binomial", "normal", "Poisson", "t", and "weibull". These are recognized, case being ignored. For continuous data, we will be using normal and t distributions:

```
> x<-fitdistr(Cars93$MPG.highway,densfun = "t")
> x$estimate
m s df
28.430527 3.937731 4.237910
> x$sd
m s df
0.5015060 0.5070997 1.9072796
> x$vcov
m s df
m 0.25150831 0.06220734 0.2607635
s 0.06220734 0.25715007 0.6460305
df 0.26076350 0.64603055 3.6377154
> x$loglik
[1] -282.4481
> x$n
[1] 93
```

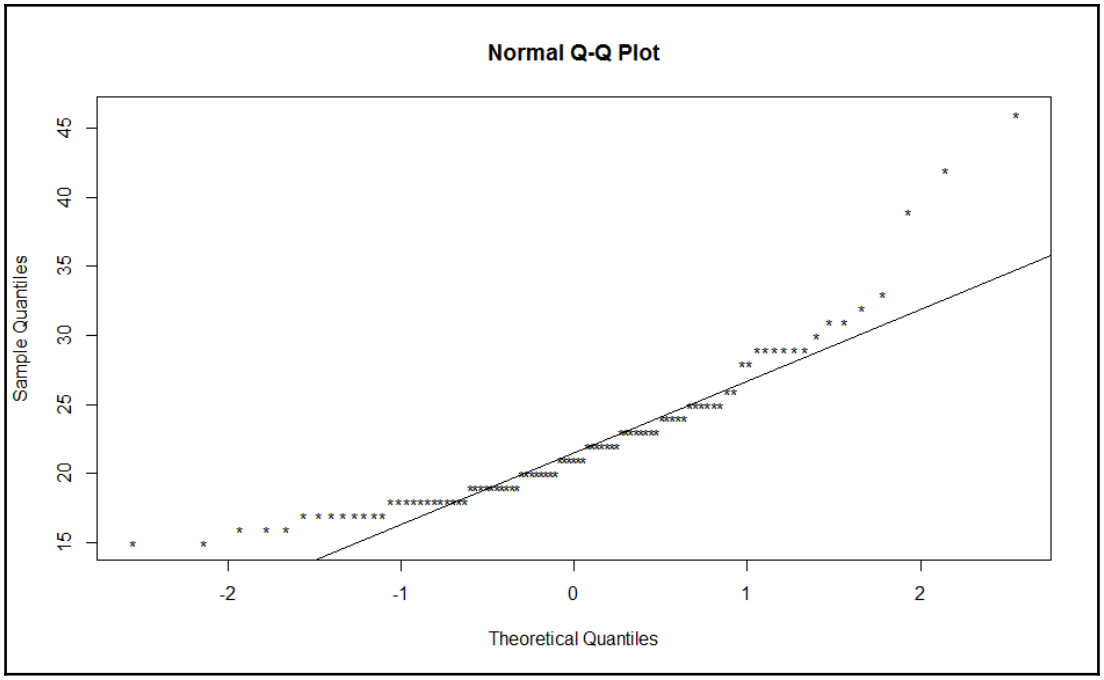
In the preceding code, we have taken the `MPG.highway` variable from the `Cars93` dataset. By fitting a t distribution to the variable, we obtain the parameter estimates, the estimated standard errors, the estimated variance co-variance matrix, the log likelihood value, as well as the total count. Similar activity can be performed by fitting a normal distribution to the continuous variable:

```
> x<-fitdistr(Cars93$MPG.highway,densfun = "normal")
> x$estimate
mean sd
29.086022 5.302983
> x$sd
```

```
mean sd
0.5498938 0.3888336
> x$vcov
mean sd
mean 0.3023831 0.0000000
sd 0.0000000 0.1511916
> x$loglik
[1] -287.1104
> x$n
[1] 93
```

Now we are going to see how to graphically represent the variable normality:

```
> qqnorm(Cars93$MPG.highway)
> qqline(Cars93$MPG.highway)
```



The deviation of data points represented as circles are distanced from the straight line, *t*.

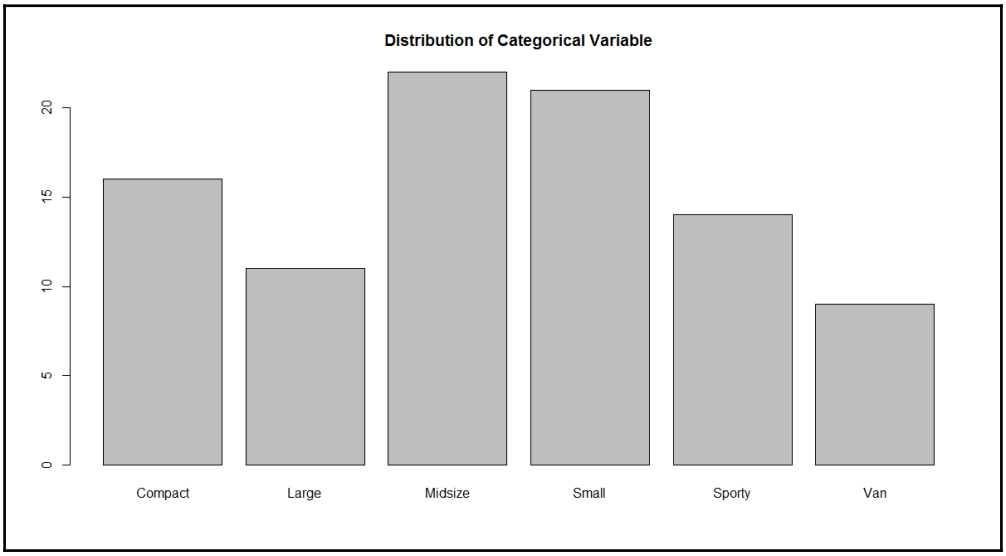
Interpreting discrete data: as all the categories in it:

```
> table(Cars93$Type)
Compact Large Midsize Small Sporty Van
16 11 22 21 14 9
```

```
> freq<-table(Cars93$Type)
> rel.freq<-freq/nrow(Cars93)*100
> options(digits = 2)
> rel.freq
Compact Large Midsize Small Sporty Van
17.2 11.8 23.7 22.6 15.1 9.7
> cbind(freq,rel.freq)
freq rel.freq
Compact 16 17.2
Large 11 11.8
Midsize 22 23.7
Small 21 22.6
Sporty 14 15.1
Van 9 9.7
```

To visualize this graphically, we need to represent it through a bar plot:

```
> barplot(freq, main = "Distribution of Categorical Variable")
```



Variable binning or discretizing continuous data

The continuous variable is the most appropriate step that one needs to take before including the variable in the model. This can be explained by taking one example fuel tank capacity of a car from the `Cars93` dataset. Based on the fuel tank capacity, we can create a categorical variable with high, medium and low, lower medium:

```
> range(Cars93$Fuel.tank.capacity)
[1] 9.2 27.0
> cat
[1] 9.2 13.2 17.2 21.2 25.2
> options(digits = 2)
> t<-cut(Cars93$Fuel.tank.capacity,cat)
> as.data.frame(cbind(table(t)))
V1
(9.2,13.2] 19
(13.2,17.2] 33
(17.2,21.2] 36
(21.2,25.2] 3
```

The range of fuel tank capacity is identified as `9.2` and `27.0`. Then, logically the class difference of 4 is used to arrive at classes. Those classes define how each value from the variable is assigned to each group. The final outcome table indicates that there are 4 groups; the top fuel tank capacity is available on 4 cars only.

Variable binning or discretization not only helps in decision tree construction but is also useful in the case of logistic regression mode and any other form of machine-learning-based models.

Contingency tables, bivariate statistics, and checking for data normality

Contingency tables are frequency tables represented by two or more categorical variables along with the proportion of each class represented as a group. Frequency table is used to represent one categorical variable; however, contingency table is used to represent two categorical variables.

Let's see an example to understand contingency tables, bivariate statistics, and data normality using the `Cars93` dataset:

```
> table(Cars93$Type)
Compact Large Midsize Small Sporty Van
16 11 22 21 14 9
> table(Cars93$AirBags)
Driver & Passenger Driver only None
16 43 34
```

The individual frequency table for two categorical variables `AirBags` and `Type` of the car is represented previously:

```
> contTable<-table(Cars93$Type,Cars93$AirBags)
> contTable
Driver & Passenger Driver only None
Compact 2 9 5
Large 4 7 0
Midsize 7 11 4
Small 0 5 16
Sporty 3 8 3
Van 0 3 6
```

The `conTable` object holds the cross tabulation of two variables. The proportion of each cell in percentage is reflected in the following table. If we need to compute the row percentages or column percentages, then it is required to specify the values in the argument:

```
> prop.table(contTable)
Driver & Passenger Driver only None
Compact 0.022 0.097 0.054
Large 0.043 0.075 0.000
Midsize 0.075 0.118 0.043
Small 0.000 0.054 0.172
Sporty 0.032 0.086 0.032
Van 0.000 0.032 0.065
```

For row percentages, the value needs to be 1, and for column percentages, the value needs to be entered as 2 in the preceding command:

```
> prop.table(contTable,1)
Driver & Passenger Driver only None
Compact 0.12 0.56 0.31
Large 0.36 0.64 0.00
Midsize 0.32 0.50 0.18
Small 0.00 0.24 0.76
Sporty 0.21 0.57 0.21
Van 0.00 0.33 0.67
> prop.table(contTable,2)
```

```
Driver & Passenger Driver only None
Compact 0.125 0.209 0.147
Large 0.250 0.163 0.000
Midsize 0.438 0.256 0.118
Small 0.000 0.116 0.471
Sporty 0.188 0.186 0.088
Van 0.000 0.070 0.176
```

The summary of the contingency table performs a chi-square test of independence between the two categorical variables:

```
> summary(contTable)
Number of cases in table: 93
Number of factors: 2
Test for independence of all factors:
Chisq = 33, df = 10, p-value = 3e-04
Chi-squared approximation may be incorrect
```

The chi-square test of independence for all factors is represented previously. The message that the chi-squared approximation may be incorrect is due to the presence of null or less than 5 values in the cells of the contingency table. As in the preceding case, two random variables, car type and airbags, can be independent if the probability distribution of one variable does not impact the probability distribution of the other variable. The null hypothesis for the chi-square test of independence is that two variables are independent of each other. Since the p-value from the test is less than 0.05, at 5% level of significance we can reject the null hypothesis that the two variables are independent. Hence, the conclusion is that car type and airbags are not independent of each other; they are quite related or dependent.

Instead of two variables, what if we add one more dimension to the contingency table? Let's take `Origin`, and then the table would look as follows:

```
> contTable<-table(Cars93$Type,Cars93$AirBags,Cars93$Origin)
> contTable
, , = USA
Driver & Passenger Driver only None
Compact 1 2 4
Large 4 7 0
Midsize 2 5 3
Small 0 2 5
Sporty 2 5 1
Van 0 2 3
, , = non-USA
Driver & Passenger Driver only None
Compact 1 7 1
Large 0 0 0
```

```
Midsize 5 6 1
Small 0 3 11
Sporty 1 3 2
Van 0 1 3
```

The `summary` command for the test of independence of all factors can be used to test out the null hypothesis:

```
> summary(contTable)
Number of cases in table: 93
Number of factors: 3
Test for independence of all factors:
Chisq = 65, df = 27, p-value = 5e-05
Chi-squared approximation may be incorrect
```

Apart from the graphical methods discussed previously, there are some numerical statistical tests that can be used to know whether a variable is normally distributed or not. There is a library called `norm.test` for performing data normality tests, a list of functions that help in assessing the data normality from this library are listed as follows:

`ajb.norm.test`	**Adjusted Jarque-Bera test for normality**
`frosini.norm.test`	Frosini test for normality
`geary.norm.test`	Geary test for normality
`hegazy1.norm.test`	Hegazy-Green test for normality
`hegazy2.norm.test`	Hegazy-Green test for normality
`jb.norm.test`	Jarque-Bera test for normality
`kurtosis.norm.test`	Kurtosis test for normality
`skewness.norm.test`	Skewness test for normality
`spiegelhalter.norm.test`	Spiegelhalter test for normality
`wb.norm.test`	Weisberg-Bingham test for normality

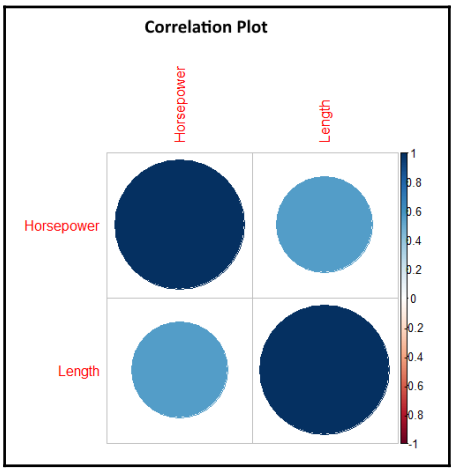
`ad.test`	**Anderson-Darling test for normality**
`cvm.test`	Cramér-von Mises test for normality
`lillie.test`	Lilliefors (Kolmogorov-Smirnov) test for normality
`pearson.test`	Pearson chi-square test for normality
`sf.test`	Shapiro-Francia test for normality

Let's apply the normality test on the `Price` variable from the `Cars93` dataset:

```
> library(nortest)
> ad.test(Cars93$Price) # Anderson-Darling test
Anderson-Darling normality test
data: Cars93$Price
A = 3, p-value = 9e-07
> cvm.test(Cars93$Price) # Cramer-von Mises test
Cramer-von Mises normality test
data: Cars93$Price
W = 0.5, p-value = 6e-06
> lillie.test(Cars93$Price) # Lilliefors (KS) test
Lilliefors (Kolmogorov-Smirnov) normality test
data: Cars93$Price
D = 0.2, p-value = 1e-05
> pearson.test(Cars93$Price) # Pearson chi-square
Pearson chi-square normality test
data: Cars93$Price
P = 30, p-value = 3e-04
> sf.test(Cars93$Price) # Shapiro-Francia test
Shapiro-Francia normality test
data: Cars93$Price
```

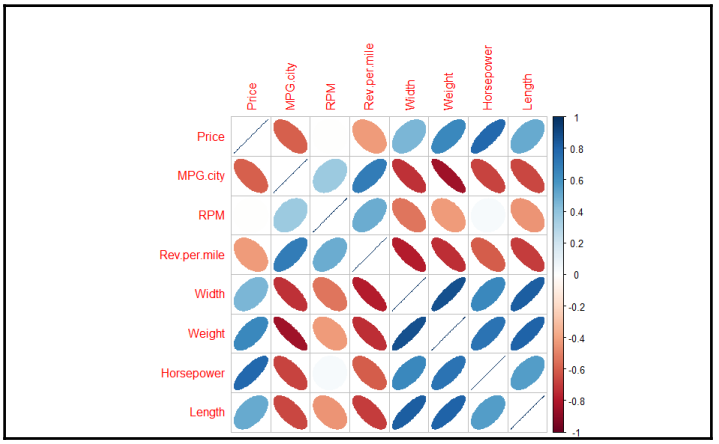
From the previously mentioned tests, it is evident that the `Price` variable is not normally distributed as the p-values from all the statistical tests are less than `0.05`. If we add more dimensions to the bi-variate relationship, it becomes multivariate analysis. Let's try to understand the relationship between horsepower and length of a car from the `Cars93` dataset:

```
> library(corrplot)
> o<-cor(Cars93[,c("Horsepower","Length")])
> corrplot(o,method = "circle",main="Correlation Plot")
```



When we include more variables, it becomes a multivariate relationship. Let's try to plot a multivariate relationship between various variables from the `Cars93` dataset:

```
> library(corrplot)
> t<-
cor(Cars93[,c("Price","MPG.city","RPM","Rev.per.mile","Width","Weight","Hor
sepower","Length")])
> corrplot(t,method = "ellipse")
```



There are various methods that can be passed as an argument to the correlation plot. They are `"circle"`, `"square"`, `"ellipse"`, `"number"`, `"shade"`, `"color"`, and `"pie"`.

Hypothesis testing

The null hypothesis states that nothing has happened, the means are constant, and so on. However, the alternative hypothesis states that something different has happened and the means are different about a population. There are certain steps in performing a hypothesis test:

1. **State the null hypothesis**: A statement about the population is assumed; for example, the average mileage of cars within a city is 40.
2. **State the alternative hypothesis**: If the null hypothesis turns out to be false, then what other possibility is there? For example, if the mileage within the city is not 40, then is it greater than 40 or less than 40? If it is not equal to 40, then it is a non-directional alternative hypothesis.
3. **Calculate the sample test statistic**: The test statistic could be t-test, f-test, z-test, and so on. Select the appropriate test statistic based on the data availability and the hypothesis declared previously.
4. **Decide the confidence limit**: There are three different confidence limits: 90%, 95% and 99% depending on the degree of accuracy related to a specific business problem. It is up to the researcher/analyst to choose the level of confidence interval.
5. **Define the alpha value**: If the confidence level selected is 95%, the alpha value is going to be 5%. Hence deciding the alpha value would help in calculating the p-value for the test.
6. **Decision**: If the p-value selected is less than the alpha level, then there is evidence that the null hypothesis can be rejected; if it is not, then we are going to accept the null hypothesis.

Test of the population mean

Using the hypothesis testing procedure, let's take an example from the `Cars93` dataset to test out the population mean.

One tail test of mean with known variance

Suppose the researcher claims that the average mileage given by all the cars collected in the sample is more than 35. In the sample of 93 cars, it is observed that the mean mileage of all cars is 29. Should you accept or reject the researcher's claim?

The following script explains how you are going to conclude this:

```
Null Hypothesis: mean = 35
Alternative hypothesis= mean > 35> mu<-mean(Cars93$MPG.highway)
> mu
[1] 29
> sigma<-sd(Cars93$MPG.highway)
> sigma
[1] 5.3
> n<-length(Cars93$MPG.highway)
> n
[1] 93
> xbar= 35
> z<-(xbar-mu)/(sigma/sqrt(n))
> z
[1] 11
> #computing the critical value at 5% alpha level
> alpha = .05
> z1 = qnorm(1-alpha)
> z1
[1] 1.6
> ifelse(z > z1,"Reject the Null Hypothesis","Accept the Null Hypothesis")
Null Hypothesis: mean = 35
Alternative hypothesis= mean < 35
Two tail test of mean, with known variance:> mu<-mean(Cars93$MPG.highway)
> mu
[1] 29.09
> sigma<-sd(Cars93$MPG.highway)
> sigma
[1] 5.332
> n<-length(Cars93$MPG.highway)
> n
[1] 93
> xbar= 35
> z<-(xbar-mu)/(sigma/sqrt(n))
> z
[1] 10.7
>
> #computing the critical value at 5% alpha level
> alpha = .05
> z1 = qnorm((1-alpha)/2))
Error: unexpected ')' in "z1 = qnorm((1-alpha)/2))"
> c(-z1,z1)
[1] -1.96 1.96
>
>
> ifelse(z > z1 | z < -z1,"Reject the Null Hypothesis","Accept the Null
Hypothesis")
```

The way we analyzed the one-tail and two-tail test of population mean for the sample data in the case of known variance.

One tail and two tail test of proportions

Using the `Cars93` dataset, suppose 40% of the USA-manufactured cars have an RPM of more than 5000. From the sample data, we found that 17 out of 57 cars have RPM above 5000. What do you interpret from the context?

```
> mileage<-subset(Cars93,Cars93$RPM > 5000)
> table(mileage$Origin)
USA non-USA
17 40
> p1<-17/57
> p0<- 0.4
> n <- length(mileage)
> z <- (p1-p0)/sqrt(p0*(1-p0)/n)
> z
[1] -1.079
> #computing the critical value at 5% alpha level
> alpha = .05
> z1 = qnorm(1-alpha)
> z1
[1] 1.645
> ifelse(z > z1,"Reject the Null Hypothesis","Accept the Null Hypothesis")
[1] "Accept the Null Hypothesis"
```

If the alternative hypothesis is not directional, it is a case of two-tailed test of proportions; nothing from the preceding calculation would change except the critical value calculation. The detailed script is given as follows:

```
> mileage<-subset(Cars93,Cars93$RPM > 5000)
> table(mileage$Origin)
USA non-USA
17 40
> p1<-17/57
> p0<- 0.4
> n <- length(mileage)
> z <- (p1-p0)/sqrt(p0*(1-p0)/n)
> z
[1] -1.079
> #computing the critical value at 5% alpha level
> alpha = .05
> z1 = qnorm(1-alpha/2)
> c(-z1,z1)
[1] -1.96 1.96
```

```
> ifelse(z > z1 | z < -z1,"Reject the Null Hypothesis","Accept the Null
Hypothesis")
[1] "Accept the Null Hypothesis"
```

- **Two sample paired test for continuous data**: The null hypothesis that is being tested in the two-sample paired test would be that there is no impact of a procedure on the subjects, the treatment has no effect on the subjects, and so on. The alternative hypothesis would be there is a statistically significant impact of a procedure, effectiveness of a treatment, or drug on the subjects.

    Though we don't have such a variable in the `Cars93` dataset, we can still assume the paired relationship between minimum prices and maximum prices for different brands of cars.

- **Null hypothesis for the two sample t-test**: There is no difference in the mean prices.
- **Alternative hypothesis**: There is a difference in the mean prices:

```
> t.test(Cars93$Min.Price, Cars93$Max.Price, paired = T)
Paired t-test
data: Cars93$Min.Price and Cars93$Max.Price
t = -9.6, df = 92, p-value = 2e-15
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
-5.765 -3.781
sample estimates:
mean of the differences
-4.773
```

    The p-value is less than 0.05. Hence, it can be concluded that the difference in mean minimum price and maximum price is statistically significant at alpha 95% confidence level.

- **Two sample unpaired test for continuous data**: From the `Cars93` dataset, the mileage on the highway and within the city is assumed to be different. If the difference is statistically significant, can be tested by using the independent samples t-test for comparison of means.
- **Null hypothesis**: There is no difference in the MPG on highway and MPG within city.
- **Alternative hypothesis**: There is a difference in the MPG on highway and MPG within city:

```
Welch Two Sample t-test
data: Cars93$MPG.city and Cars93$MPG.highway
```

```
t = -8.4, df = 180, p-value = 1e-14
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
-8.305 -5.136
sample estimates:
mean of x mean of y
22.37 29.09
```

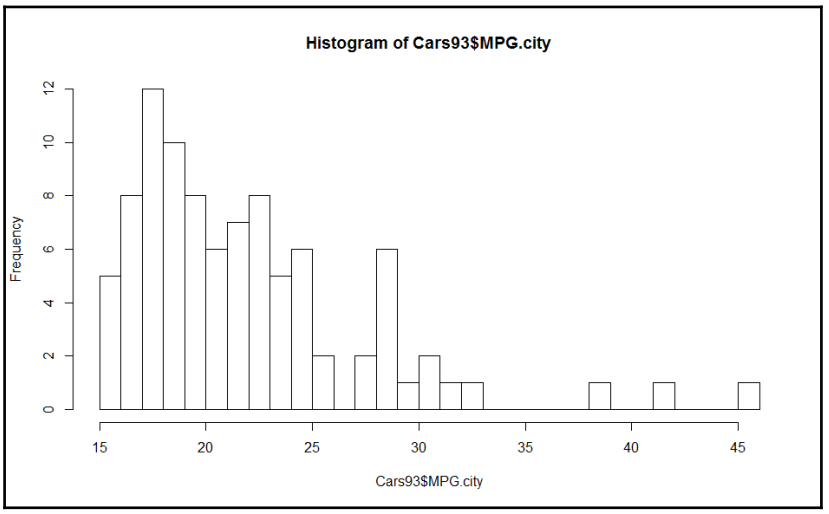
From the two samples t-test, when the two samples are independent, the p-value is less than 0.05; hence, we can reject the null hypothesis that there is no difference in the mean mileage on highway and within city. There is a statistically significant difference in the mean mileage within city and on highway. This can be represented in a slightly different way, by putting a null hypothesis is the mean mileage difference within city is different for manual versus automatic cars:

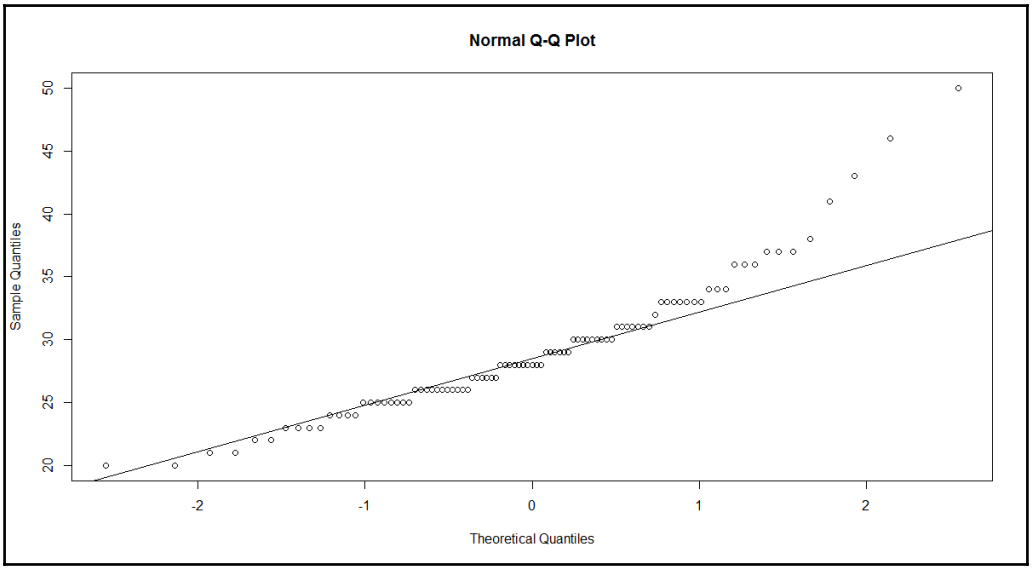
```
, data=Cars93)
Welch Two Sample t-test
data: Cars93$MPG.city by Cars93$Man.trans.avail
t = -6, df = 84, p-value = 4e-08
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
-6.949 -3.504
sample estimates:
mean in group No mean in group Yes
18.94 24.16
```

So, the conclusion from the preceding test is that there is a statistically significant difference in the mean mileage between automatic and manual transmission vehicle types; this is because the p-value is less than 0.05.

Before applying t-test, it is important to check the data normality; normality of a variable can be assessed using the Shapiro test function:

```
> shapiro.test(Cars93$MPG.city)
Shapiro-Wilk normality test
data: Cars93$MPG.city
W = 0.86, p-value = 6e-08
)
> qqline(Cars93$MPG.city)
```





Looking at the QQ plot for the lineage per gallon within city and the histogram, it can be concluded that the variable is not normally distributed. Since the mileage variable is not normally distributed, it is required to apply non-parametric methods such as Wilcoxon signed rank test or Kolmogorov-Smirnov test.

Two sample variance test

To compare the variances of two samples, F-test is used as a statistic:

```
> var.test(Cars93$MPG.highway~Cars93$Man.trans.avail, data=Cars93)
F test to compare two variances
data: Cars93$MPG.highway by Cars93$Man.trans.avail
F = 0.24, num df = 31, denom df = 60, p-value = 5e-05
alternative hypothesis: true ratio of variances is not equal to 1
95 percent confidence interval:
0.1330 0.4617
sample estimates:
ratio of variances
0.2402
```

Since the p-value is less than 0.05, we can reject the null hypothesis that there is no difference in the variance of mileage on a highway for manual and automatic cars. This implies there is a statistically significant difference in the variance of two samples at 95% confidence level.

The variances of the two groups can also be tested using Bartlett test:

```
> bartlett.test(Cars93$MPG.highway~Cars93$Man.trans.avail, data=Cars93)
Bartlett test of homogeneity of variances
data: Cars93$MPG.highway by Cars93$Man.trans.avail
Bartlett's K-squared = 17, df = 1, p-value = 4e-05
```

From the preceding test, it can also be concluded that the null hypothesis of equal variances can be rejected at alpha 0.05 level, and it can be proven that there is a statistically significant difference in the variance of the two samples.

One-way ANOVA:, one-way ANOVA can be used. The variable considered is RPM, and the grouping variable considered is `Cylinders`.

Null hypothesis: There is no difference in the means of RPM across different cylinder types.

Alternative hypothesis: There is a difference in the mean RPM for at least one cylinder type:

```
> aov(Cars93$RPM~Cars93$Cylinders)
Call:
aov(formula = Cars93$RPM ~ Cars93$Cylinders)
Terms:
Cars93$Cylinders Residuals
Sum of Squares 6763791 25996370
Deg. of Freedom 5 87
Residual standard error: 546.6
Estimated effects may be unbalanced
```

```
> summary(aov(Cars93$RPM~Cars93$Cylinders))
Df Sum Sq Mean Sq F value Pr(>F)
Cars93$Cylinders 5 6763791 1352758 4.53 0.001 **
Residuals 87 25996370 298809
---
Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

From the preceding ANOVA, the p-value is less than 0.05; hence, the null hypothesis can be rejected. That means at least for one cylinder type, the mean RPM is statistically significantly different. To identify which cylinder type is different, a post hoc test can be performed on the results of the ANOVA model:

```
> TukeyHSD(aov(Cars93$RPM~Cars93$Cylinders))
Tukey multiple comparisons of means
95% family-wise confidence level
Fit: aov(formula = Cars93$RPM ~ Cars93$Cylinders)
$`Cars93$Cylinders`
diff lwr upr p adj
4-3 -321.8 -1269.23 625.69 0.9201
5-3 -416.7 -1870.88 1037.54 0.9601
6-3 -744.1 -1707.28 219.11 0.2256
8-3 -895.2 -1994.52 204.04 0.1772
rotary-3 733.3 -1106.11 2572.78 0.8535
5-4 -94.9 -1244.08 1054.29 0.9999
6-4 -422.3 -787.90 -56.74 0.0140
8-4 -573.5 -1217.14 70.20 0.1091
rotary-4 1055.1 -554.08 2664.28 0.4027
6-5 -327.4 -1489.61 834.77 0.9629
8-5 -478.6 -1755.82 798.67 0.8834
rotary-5 1150.0 -801.03 3101.03 0.5240
8-6 -151.2 -817.77 515.47 0.9857
rotary-6 1477.4 -141.08 3095.92 0.0941
rotary-8 1628.6 -74.42 3331.57 0.0692
```

Wherever the p-adjusted value is less than 0.05, the mean difference in RPM is statistically significantly different from other groups.

Two way ANOVA with post hoc tests: The factors considered are origin and airbags. The hypothesis that needs to be tested is: is there any impact of both the categorical variables on the RPM variable?

```
> aov(Cars93$RPM~Cars93$Origin + Cars93$AirBags)
Call:
aov(formula = Cars93$RPM ~ Cars93$Origin + Cars93$AirBags)
Terms:
Cars93$Origin Cars93$AirBags Residuals
Sum of Squares 8343880 330799 24085482
```

```
Deg. of Freedom 1 2 89
Residual standard error: 520.2
Estimated effects may be unbalanced
> summary(aov(Cars93$RPM~Cars93$Origin + Cars93$AirBags))
Df Sum Sq Mean Sq F value Pr(>F)
Cars93$Origin 1 8343880 8343880 30.83 2.9e-07 ***
Cars93$AirBags 2 330799 165400 0.61 0.54
Residuals 89 24085482 270623
---
Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
> TukeyHSD(aov(Cars93$RPM~Cars93$Origin + Cars93$AirBags))
Tukey multiple comparisons of means
95% family-wise confidence level
Fit: aov(formula = Cars93$RPM ~ Cars93$Origin + Cars93$AirBags)
$`Cars93$Origin`
diff lwr upr p adj
non-USA-USA 599.4 384.9 813.9 0
$`Cars93$AirBags`
diff lwr upr p adj
Driver only-Driver & Passenger -135.74 -498.8 227.4 0.6474
None-Driver & Passenger -25.68 -401.6 350.2 0.9855
None-Driver only 110.06 -174.5 394.6 0.6280
```

Non-parametric methods

When a training dataset does not conform to any specific probability distribution because of non-adherence to the assumptions of that specific probability distribution, the only option left to analyze the data is via non-parametric methods. Non-parametric methods do not follow any assumption regarding the probability distribution. Using non-parametric methods, one can draw inferences and perform hypothesis testing without adhering to any assumptions. Now let's look at a set of on-parametric tests that can be used when a dataset does not conform to the assumptions of any specific probability distribution.

Wilcoxon signed-rank test

If the assumption of normality is violated, then it is required to apply non-parametric methods in order to answer a question such as: is there any difference in the mean mileage within the city between automatic and manual transmission type cars?

```
> wilcox.test(Cars93$MPG.city~Cars93$Man.trans.avail, correct = F)
Wilcoxon rank sum test
data: Cars93$MPG.city by Cars93$Man.trans.avail
W = 380, p-value = 1e-06
```

```
alternative hypothesis: true location shift is not equal to 0
```

The argument paired can be used if the two samples happen to be matching pairs and the samples do not follow the assumptions of normality:

```
> wilcox.test(Cars93$MPG.city, Cars93$MPG.highway, paired = T)
Wilcoxon signed rank test with continuity correction
data: Cars93$MPG.city and Cars93$MPG.highway
V = 0, p-value <2e-16
alternative hypothesis: true location shift is not equal to 0
```

Mann-Whitney-Wilcoxon test

If two samples are not matched, are independent, and do not follow a normal distribution, then it is required to use Mann-Whitney-Wilcoxon test to test the hypothesis that the mean difference in the two samples are statistically significantly different from each other:

```
> wilcox.test(Cars93$MPG.city~Cars93$Man.trans.avail, data=Cars93)
Wilcoxon rank sum test with continuity correction
data: Cars93$MPG.city by Cars93$Man.trans.avail
W = 380, p-value = 1e-06
alternative hypothesis: true location shift is not equal to 0
```

Kruskal-Wallis test

To compare means of more than two groups, that is, the non-parametric side of ANOVA analysis, we can use the Kruskal-Wallis test. It is also known as a distribution-free statistical test:

```
> kruskal.test(Cars93$MPG.city~Cars93$Cylinders, data= Cars93)
Kruskal-Wallis rank sum test
data: Cars93$MPG.city by Cars93$Cylinders
Kruskal-Wallis chi-squared = 68, df = 5, p-value = 3e-13
```

Summary

Exploratory data analysis is an important activity in almost all types of data mining projects. Understanding the distribution, shape of the distribution, and vital parameters of the distribution, is very important. Preliminary hypothesis testing can be done to know the data better. Not only the distribution and its properties but also the relationship between various variables is important. Hence in this chapter, we looked at bivariate and multivariate relationships between various variables and how to interpret the relationship. Classical statistical tests, such as t-test, F-test, z-test and other non-parametric tests are important to test out the hypothesis. The hypothesis testing itself is also important to draw conclusions and insights from the dataset.

In this chapter we discussed various statistical tests, their syntax, interpretations, and situations where we can apply those tests. After performing exploratory data analysis, in the next chapter we are going to look at various data visualization methods to get a 360-degree view of data. Sometimes, a visual story acts as the simplest possible representation of data. In the next chapter, we are going to use some inbuilt datasets with different libraries to create intuitive visualizations.

3
Visualize Diamond Dataset

Every data mining project is incomplete without proper data visualization. While looking at numbers and statistics it may tell a similar story for the variables we are looking at by different cuts, however, when we visually look at the relationship between variables and factors it shows a different story altogether. Hence data visualization tells you a message, that numbers and statistics fail to do that. From a data mining perspective, data visualization has many advantages, which can be summarized in three important points:

- Data visualization establishes a robust communication between the data and the consumer of the data
- It imprints a long lasting impact as people may fail to remember numbers but they do remember charts and shapes
- When data scales up to higher dimension, representation in numbers does not make sense, but visually it does

In this chapter, the reader will get to know the basics of data visualization along with how to create advanced data visualization using existing libraries in R programming language. Typically, data visualization approach can be addressed in two different ways:

- What do you want to display to the audience? Is it comparison, relationship, or any other functionality?
- How do you want to display the insights? Which one is the most intuitive way of chart or graph to display the insights?

Based on the preceding two points, let's have a look at the data visualization rules and theories behind each visualization, and then we are going to look at the practical aspect of implementing the graphs and charts using R Script.

From a functional point of view, the following are the graphs and charts which a data scientist would like the audience to look at to infer the information:

- **Comparisons between variables**: Basically, when it is needed to represent two or more categories within a variable, then the following charts are used:
    - Bar chart
    - Box plot
    - Bubble chart
    - Histogram
    - Line graph
    - Stacked bar chart
    - Radar chart
    - Pie chart
- **Testing/viewing proportions**: It is used when there is a need to display the proportion of contribution by one category to the overall level:
    - Bubble chart
    - Bubble map
    - Stacked bar chart
    - Word cloud
- **Relationship between variables**: Association between two or more variables can be shown using the following charts:
    - Scatterplot
    - Bar chart
    - Radar chart
    - Line graph
    - Tree diagram
- **Variable hierarchy**: When it is required to display the order in the variables, such as a sequence of variables, then the following charts are used:
    - Tree diagram
    - Tree map
- **Data with locations**: When a dataset contains the geographic location of different cities, countries, and states names, or longitudes and latitudes, then the following charts can be used to display visualization:
    - Bubble map
    - Geo mapping
    - Dot map
    - Flow map

- **Contribution analysis or part-to-whole**: When it is required to display constituents of a variable and contribution of each categorical level towards the overall variable, then the following charts are used:
    - Pie chart
    - Stacked bar chart
    - Donut chart
- **Statistical distribution**: In order to understand the variation in a variable across different dimensions, represented by another categorical variable, the following charts are used:
    - Box plot
    - Bubble chart
    - Histogram
    - Stem and leaf plot
- **Unseen patterns**: For pattern recognition and relative importance of data points on different dimensions of a variable, the following charts are used:
    - Bar chart
    - Box plot
    - Bubble chart
    - Scatterplot
    - Spiral plot
    - Line chart
- **Spread of values or range**: The following charts only give the spread of the data points across different bounds:
    - Span chart
    - Box plot
    - Histogram
- **Textual data representation**: This is a very interesting way of representing the textual data:
    - Word cloud

Keeping in mind the preceding functionalities that people use in displaying insights to the readers, we can see that one graph is referred by much functionality. In other words, one graph can be used in multiple functions to show the insights. These graphs and charts can be displayed by using various open source R packages, such as `ggplot2`, `ggvis`, `rCharts`, `plotly`, and `googleVis` by taking one open source dataset.

In the light of the previously mentioned ten points, the data visualization rules can be created to select the best representation depending on what you want to represent:

- Relationship between two variables can be represented using a scatterplot
- Relationship between more than two variables can be represented using a bubble chart
- For understanding the distribution of a single variable with few sample size, histogram is used
- For understanding the distribution of a single variable with large sample size, density plot is used
- Distribution of two variables can be represented using scatterplot
- For representation of distribution between 3 variables, 3D scatterplot is used
- Any variable with a time stamp, such as day, week, month, year, and so on, can be represented using line chart
- Time should always be on the horizontal axis and the metric to be measured should always be on the vertical axis
- Each graph should have a name and label so that the user does not have to go back to the data to understand what it is

In this chapter, we will primarily focus on the `ggplot2` library and `plotly` library. Of course we will cover a few more interesting libraries to create data visualization. The graphics packages in R can be organized as per the following sequences:

- Plotting
- Graphic applications (such as effect ordering, large datasets and trees, and graphs)
- Graphics systems
- Devices
- Colors
- Interactive graphics
- Development

A detailed explanation of the various libraries supporting the previous functionalities can be found in the link `https://cran.r-project.org/web/views/Graphics.html`. For good visual object, we need more data points so that the density of the graphs can be more. In this context, we will be using two datasets, `diamonds.csv` and `cars93.csv`, to show data visualization.

Data visualization using ggplot2

There are two approaches to go ahead with data visualization, horizontally and vertically. Horizontal drill down means creating different charts and graphs using `ggplot2` and vertical drill down implies creating one graph and adding different components to the graph. First, we will understand how to add and modify different components to a graph, and then we will move horizontally to create different types of charts.

Let's look at the dataset and libraries required to create data visualization:

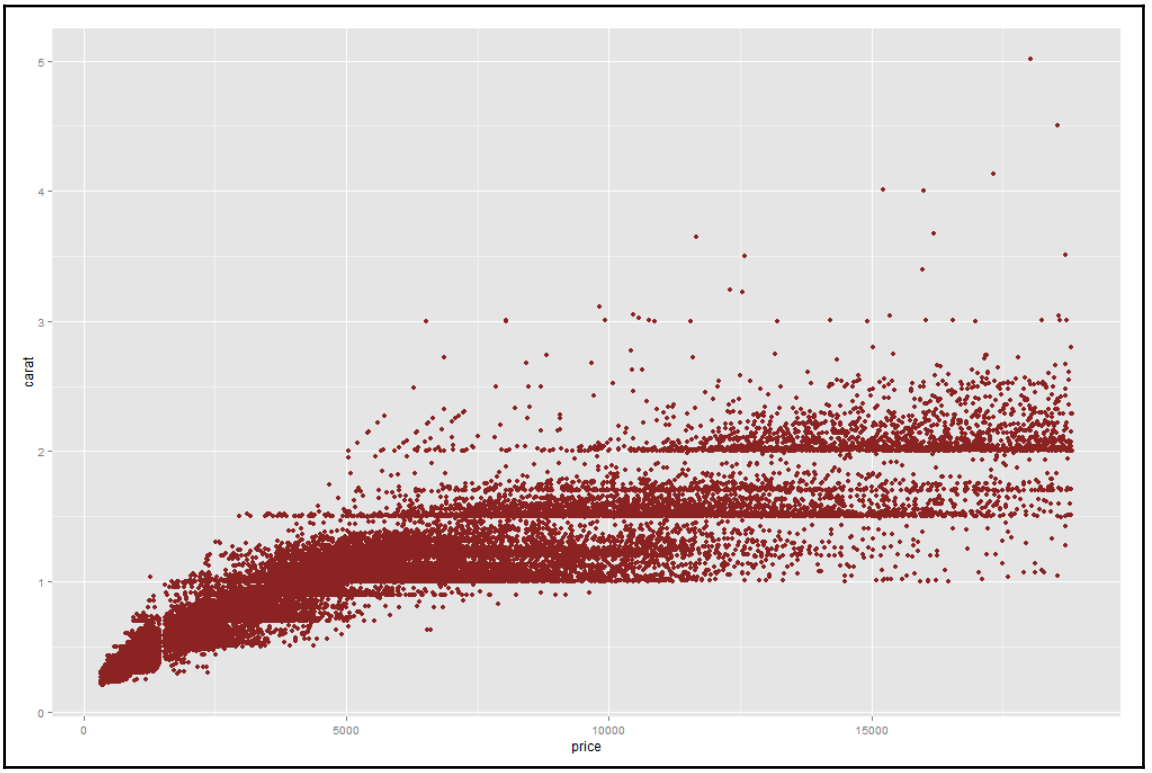
```
> #getting the library
> library(ggplot2);head(diamonds);names(diamonds)
X carat cut color clarity depth table price x y z
1 1 0.23 Ideal E SI2 61.5 55 326 3.95 3.98 2.43
2 2 0.21 Premium E SI1 59.8 61 326 3.89 3.84 2.31
3 3 0.23 Good E VS1 56.9 65 327 4.05 4.07 2.31
4 4 0.29 Premium I VS2 62.4 58 334 4.20 4.23 2.63
5 5 0.31 Good J SI2 63.3 58 335 4.34 4.35 2.75
6 6 0.24 Very Good J VVS2 62.8 57 336 3.94 3.96 2.48
[1] "X" "carat" "cut" "color" "clarity" "depth" "table" "price" "x"
[10] "y" "z"
```

The `ggplot2` library is also known as the grammar of graphics for data visualization. The process to start the graph requires a dataset and two variables, and then different components of a graph can be added to the base graph by using the `+` sign. Let's get into creating a nice visualization using the `diamonds.csv` dataset:

```
> #starting a basic ggplot plot object
> gg<-ggplot(diamonds,aes(price,carat))+geom_point(color="brown4")
> gg
```

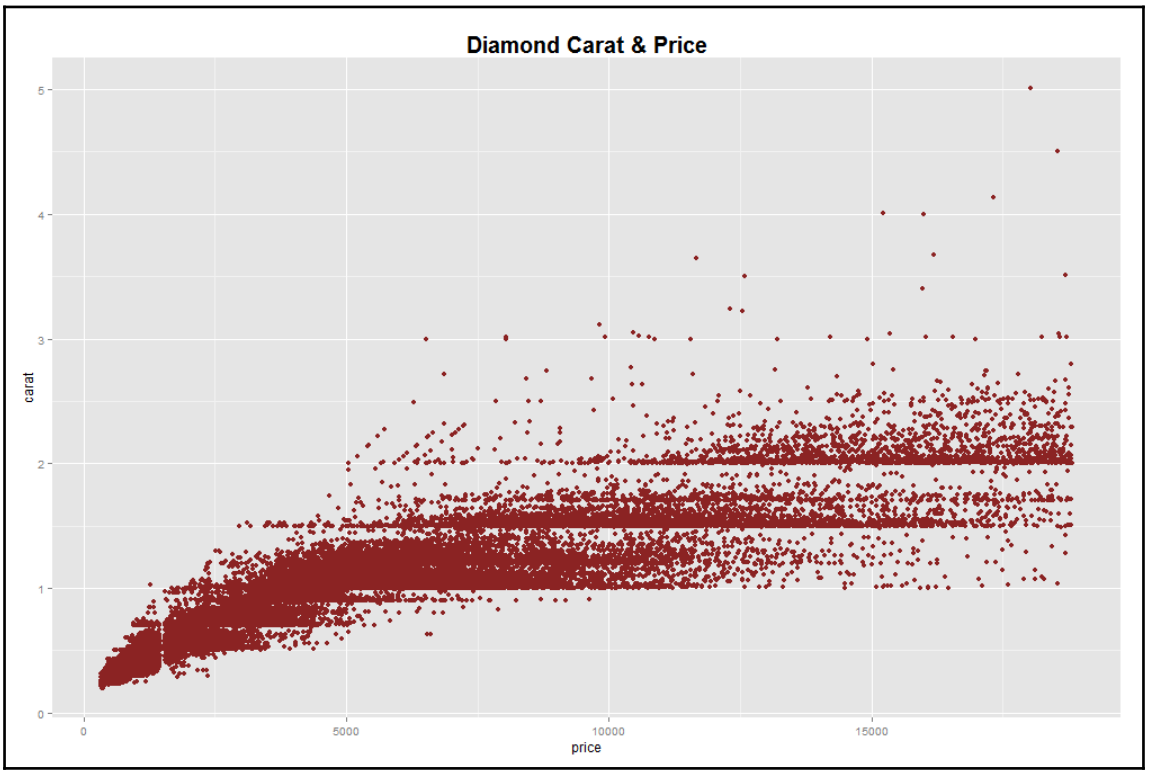
In the preceding script, `diamonds` is the dataset, and `carat` and `price` are the two variables. Using `ggplot` function, the base graph is created and adding point to the `ggplot`, the object is stored in object `gg`.

Now we are going to add various components of a graph to make the `ggplot` graph more interesting:



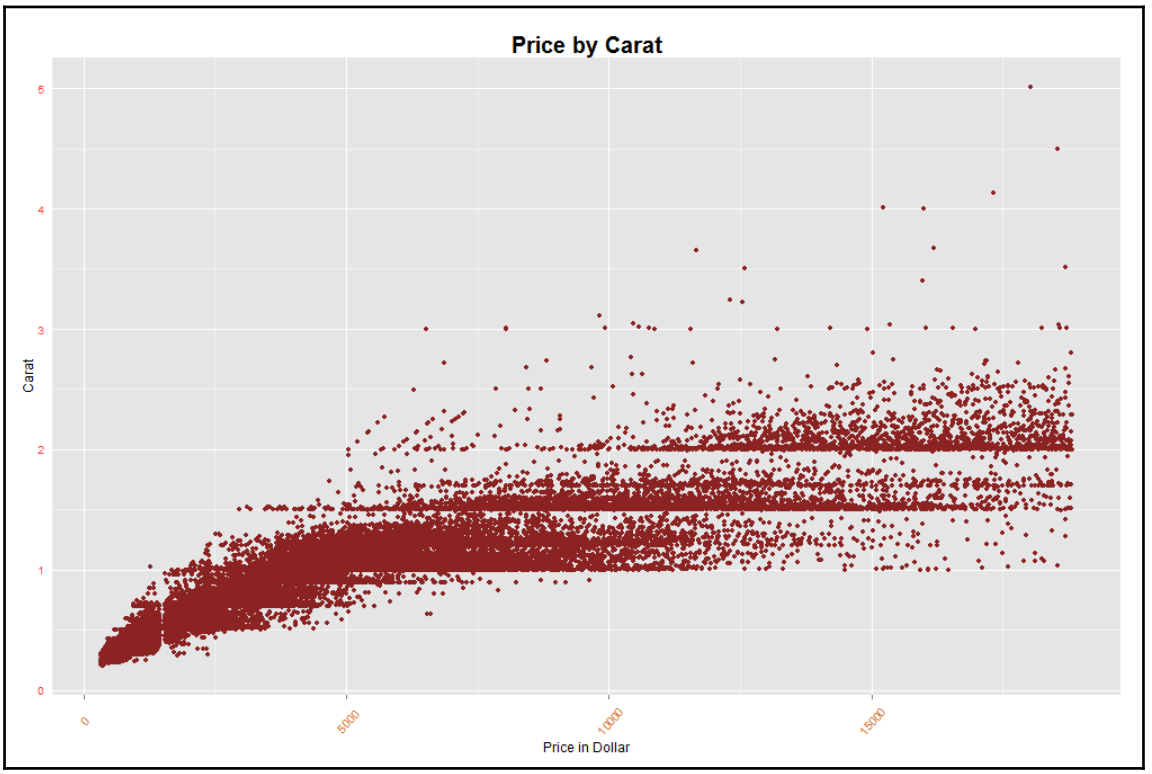
After creating the base plot, it is required to add title and label to the graph. This can be done using two functions, `ggtitle` or `labs`. Then, let's add a theme to the plot for customizing text element:

```
> #adding a title or label to the graph
> gg<-gg+ggtitle("Diamond Carat & Price")
> gg
> gg<-gg+labs("Diamond Carat & Price")
> gg
> #adding theme to the plot
> gg<-gg+theme(plot.title= element_text(size = 20, face = "bold"))
> gg
```



Currently, the graph looks a little congested. To make the graph more intuitive, we need to add labels to the *x* axis and *y* axis, removing ticks and text from any axis to make the graph more clear. Rotating text in any axis is required when the row name or the column name contains text or a large number which is difficult to read in full length:

```
> #adding labels to the graph
> gg<-gg+labs(x="Price in Dollar", y="Carat", )
> gg
> #removing text and ticks from an axis
> gg<-gg+theme(axis.ticks.y=element_blank(),axis.text.y=element_blank())
> gg
> gg<-gg + theme(axis.text.x=element_text(angle=50, size=10, vjust=0.5))
> gg
> gg<-gg + theme(axis.text.x=element_text(color = "chocolate", vjust=0.45),
+ axis.text.y=element_text(color = "brown1", vjust=0.45))
> gg
```



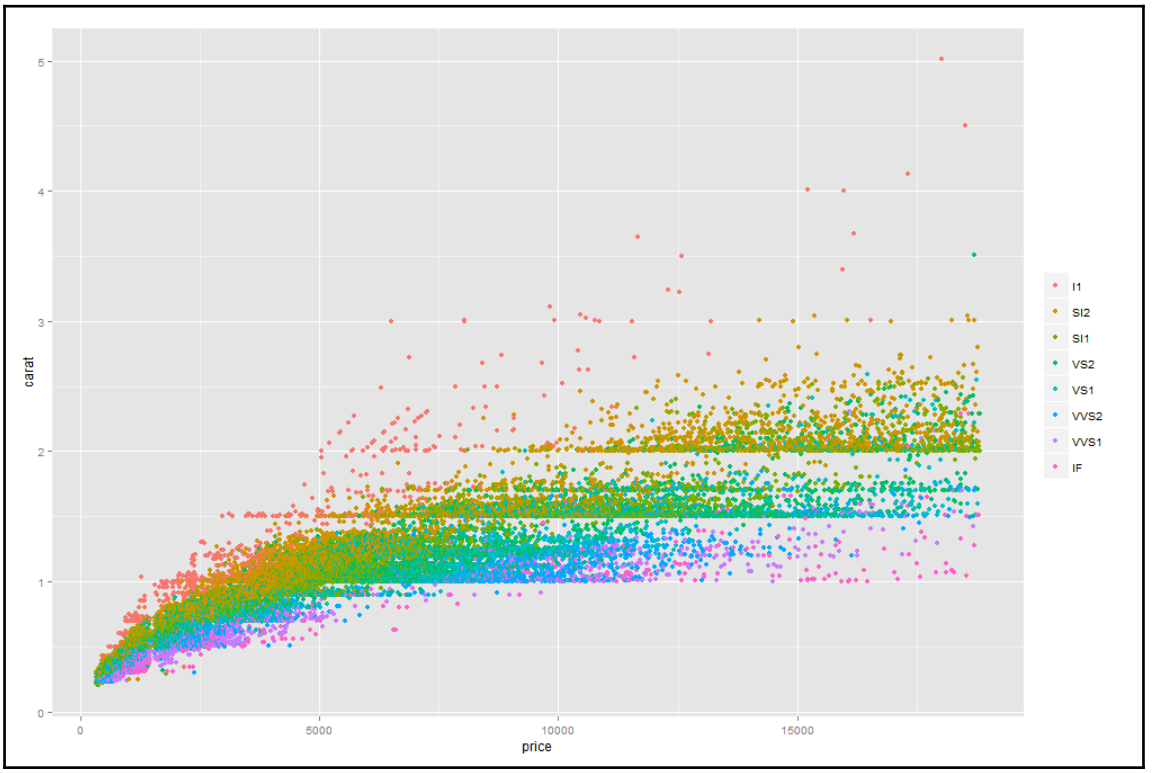
In order to focus on any specific portion of the plot, the *x* axis limit and *y* axis limit can be changed as follows. It also shows the number of rows removed while executing the limit on both the axes:

```
> #setting limits to both axis
> gg<-gg + ylim(0,0.8)+xlim(250,1500)
> gg
Warning message:
Removed 33937 rows containing missing values (geom_point).
```

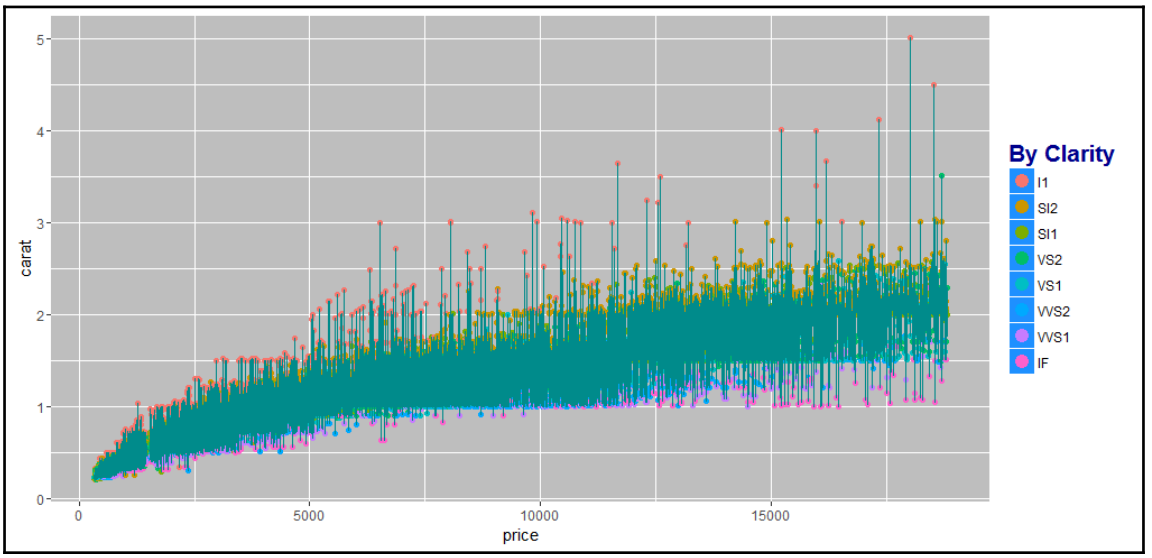
If both *x* axis and *y* axis represent continuous data, any third variable as a factor can be introduced to the `ggplot` object to set legends and look at the data how it is distributed across the `factor` variable:

```
> #how to set legends in a graph
> gg<-ggplot(diamonds,aes(price,carat,color=factor(cut)))+geom_point()
> gg
> gg<-ggplot(diamonds,aes(price,carat,color=factor(color)))+geom_point()
> gg
```

```
> gg<-ggplot(diamonds,aes(price,carat,color=factor(clarity)))+geom_point()
> gg
> gg<-gg+theme(legend.title=element_blank())
> gg
```

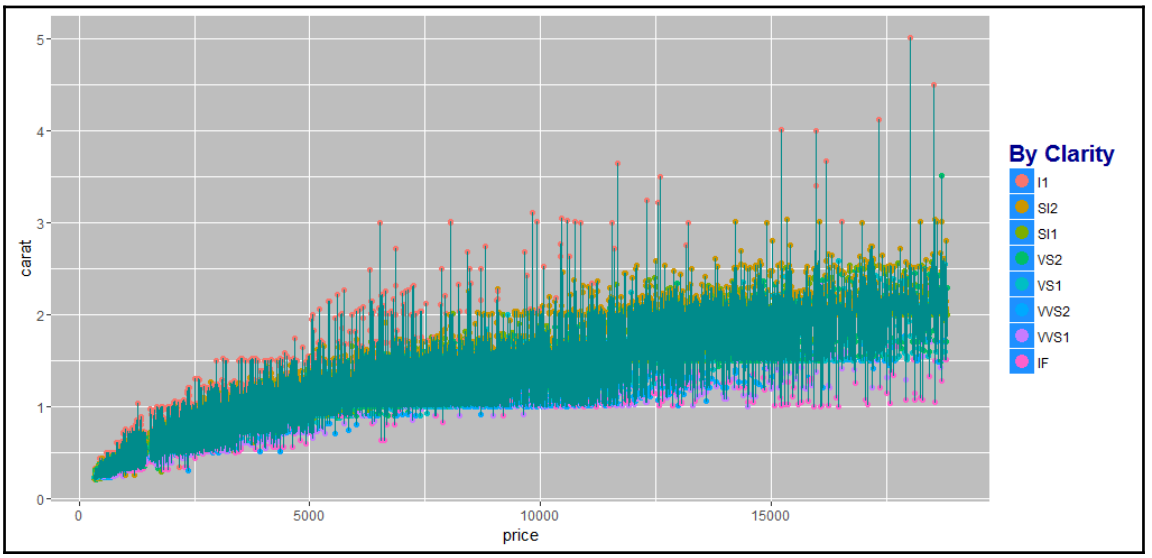



```
> gg<-gg+theme(legend.title = element_text(colour="darkblue", size=16,
+ face="bold"))+scale_color_discrete(name="By Different Grids of Clarity")
> #changing the backgroup boxes in legend
> gg<-gg+theme(legend.key=element_rect(fill='dodgerblue1'))
> gg
> #changing the size of the symbols used in legend
> gg<-gg+guides(colour = guide_legend(override.aes = list(size=4)))
> gg
> #changing the size of the symbols used in legend
> gg<-gg+guides(colour = guide_legend(override.aes = list(size=4)))
> gg
```



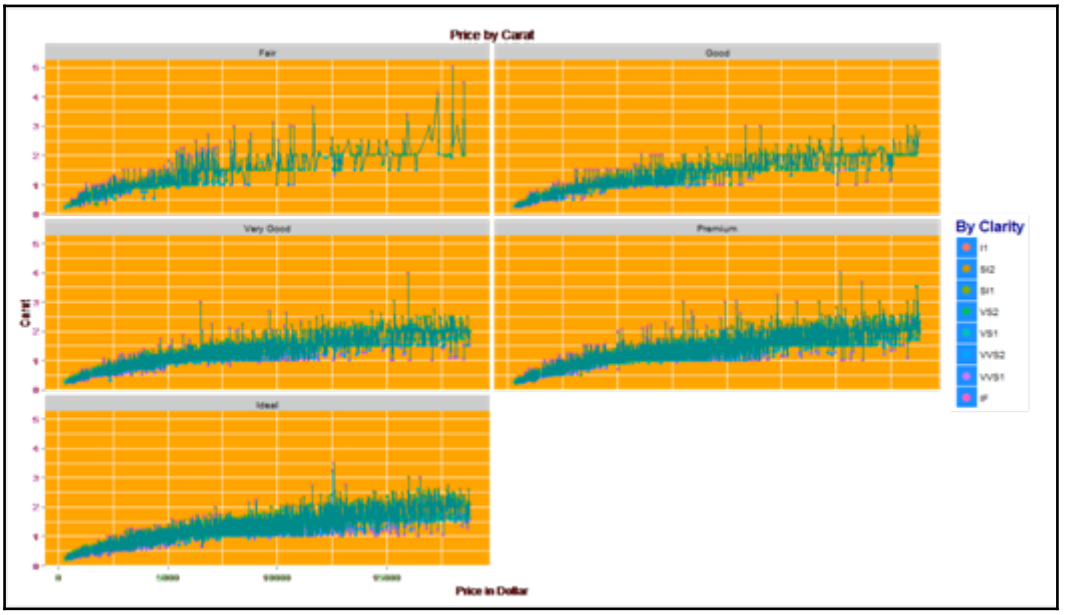
In addition to the previous visualization, it is required to connect the scatterplots and change the background. Add lines to the scatterplot in order to understand the sequence *r* pattern that exists between the variables which are related:

```
> #adding line to the data points
> gg<-gg+geom_line(color="darkcyan")
> gg
> #changing the background of an image
> gg<-gg+theme(panel.background = element_rect(fill = 'chocolate3'))
> gg
> #changing plot background
> gg<-gg+theme(plot.background = element_rect(fill = 'skyblue'))
> gg
```



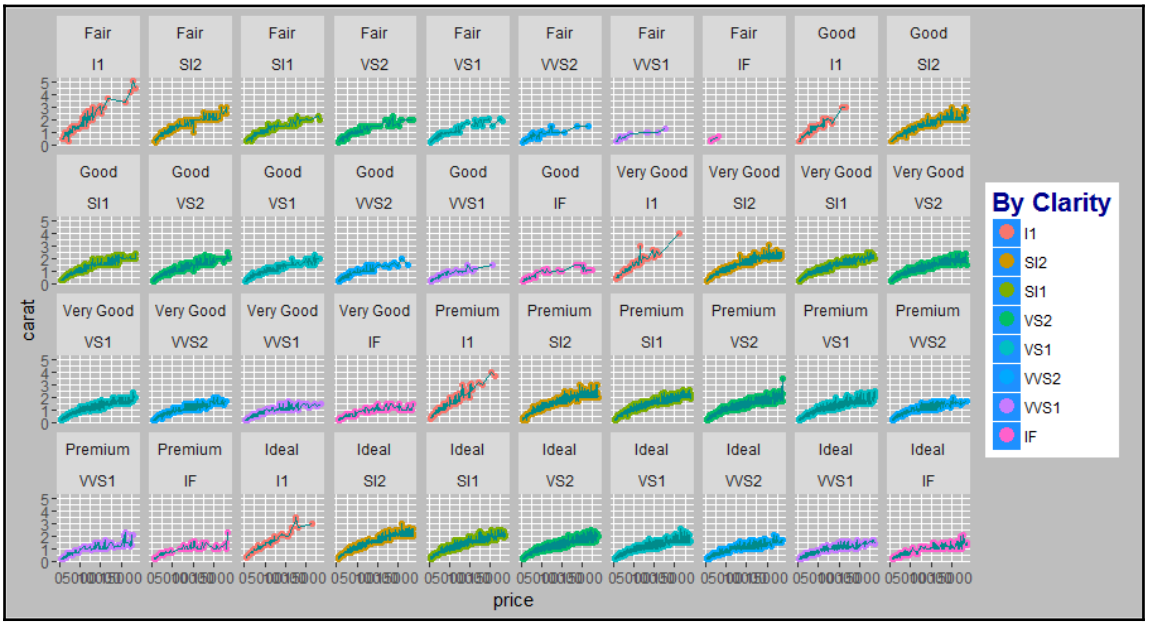
Another important aspect of data visualization is how to display multi-dimensional cuts in a plot. For example, in the `diamonds` datasets, we are currently looking at the relationship between the price of the diamond and the carat it contains. There are three more variables: `cut`, `color`, and `clarity`. It makes sense to understand if the relationship is consistent across those three variables. That means, understanding if we can plot the relationship between the carat and the price of the diamond by different cut, different color, and different clarity categories. Let's look at the distribution of the three categorical variables:

```
> table(diamonds$cut);table(diamonds$clarity);table(diamonds$color)
Fair Good Very Good Premium Ideal
1610 4906 12082 13791 21551
I1 SI2 SI1 VS2 VS1 VVS2 VVS1 IF
741 9194 13065 12258 8171 5066 3655 1790
D E F G H I J
6775 9797 9542 11292 8304 5422 2808
> #adding a multi-variable cut to the graph
> gg<-gg+facet_wrap(~cut, nrow=4)
> gg
```



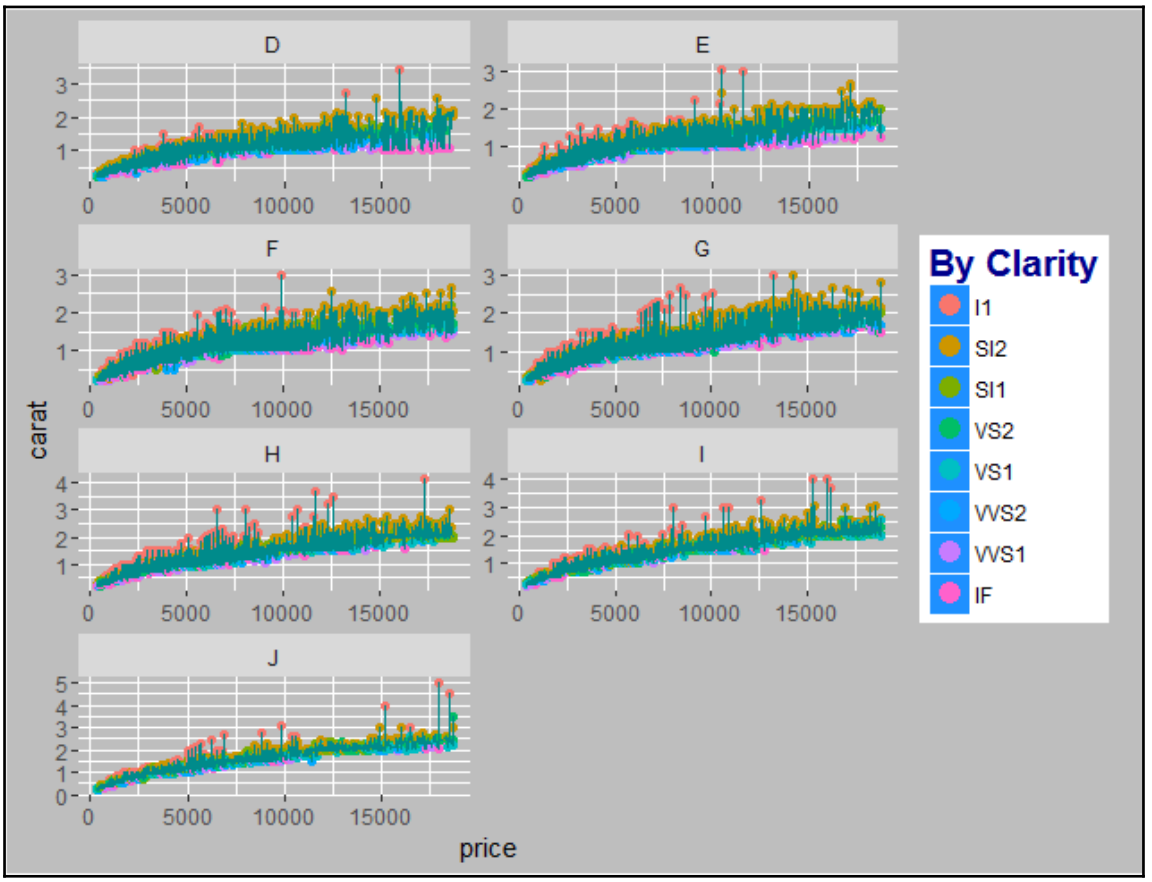
In the preceding graph, the `cut` variable is used to show the relationship between the carat and the price. The number of rows selected is four, to represent the graphs in a clear manner. If we add one more variable to the `cut` variable, that is `clarity`, the graph would be more intuitive and insightful:

```
> #adding two variables as cut to display the relationship
> gg<-gg+facet_wrap(~cut+clarity, nrow=4)
> gg
```



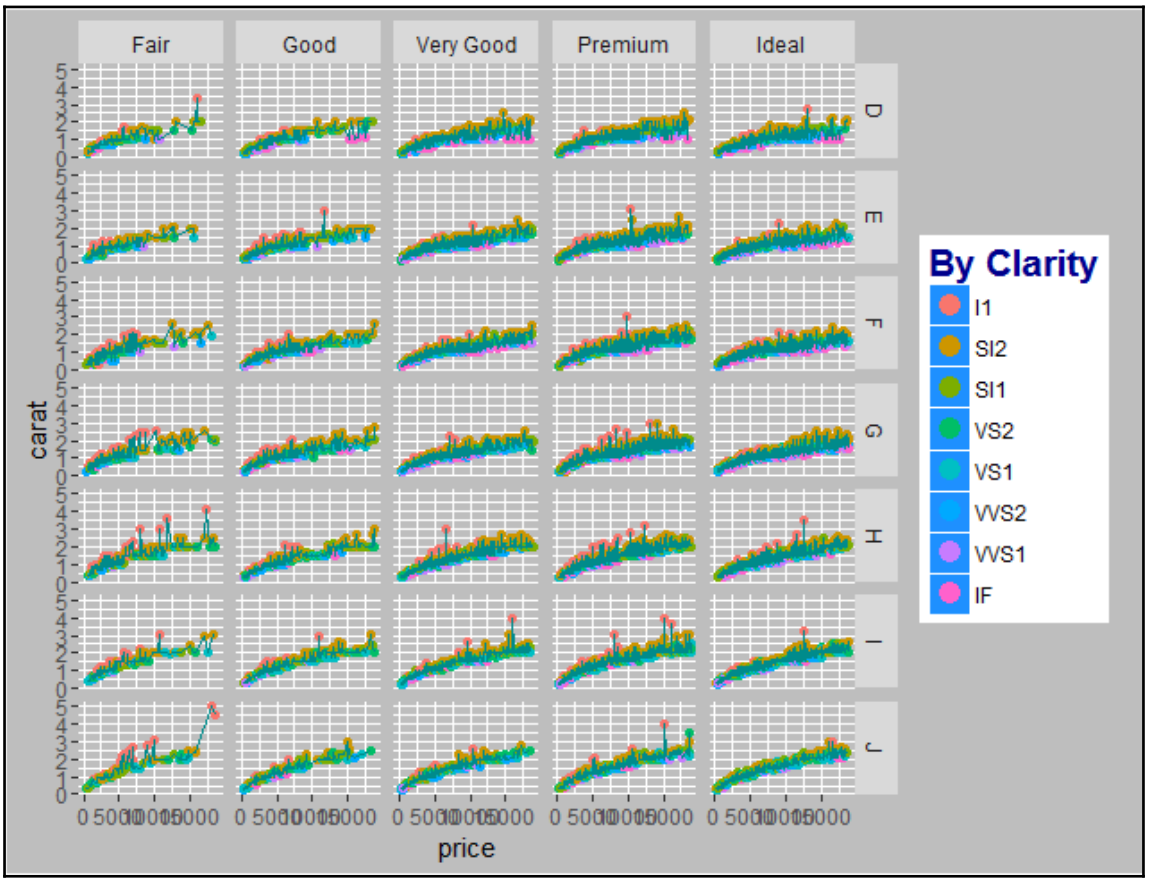
While creating the graphs using a multi-dimensional `cut` variable, it is not necessary that all the graphs would be on the same scale. In automatic mode, the scales become a standard for all the plots, hence, sometimes certain plots get compressed. Thus, it is required to make the graphs scale free in order to rearrange the scales based on observed values:

```
> #scale free graphs in multi-panels
> gg<-gg+facet_wrap(~color, ncol=2, scales="free")
> gg
```



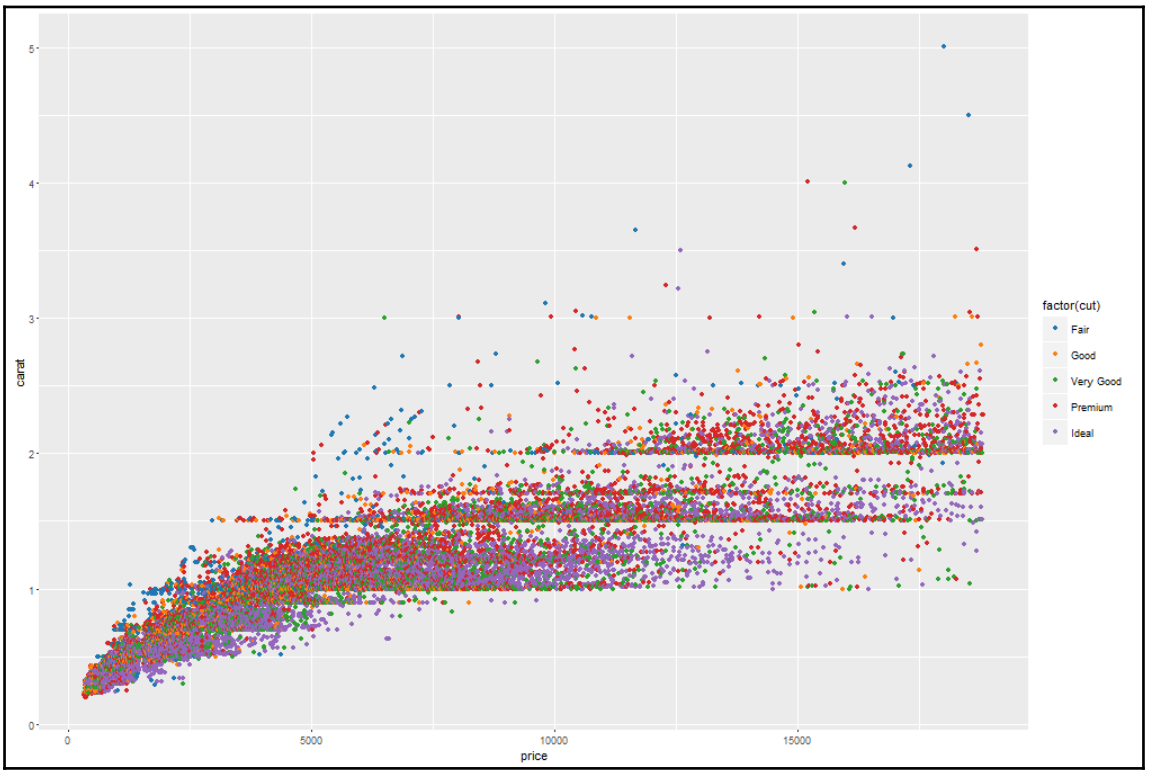
Using the `facet_grid()` option, we can display the bi-variate relationship between two categorical variables using the `ggplot2` library:

```
> #bi-variate plotting using ggplot2
> gg<-gg+facet_grid(color~cut)
> gg
```



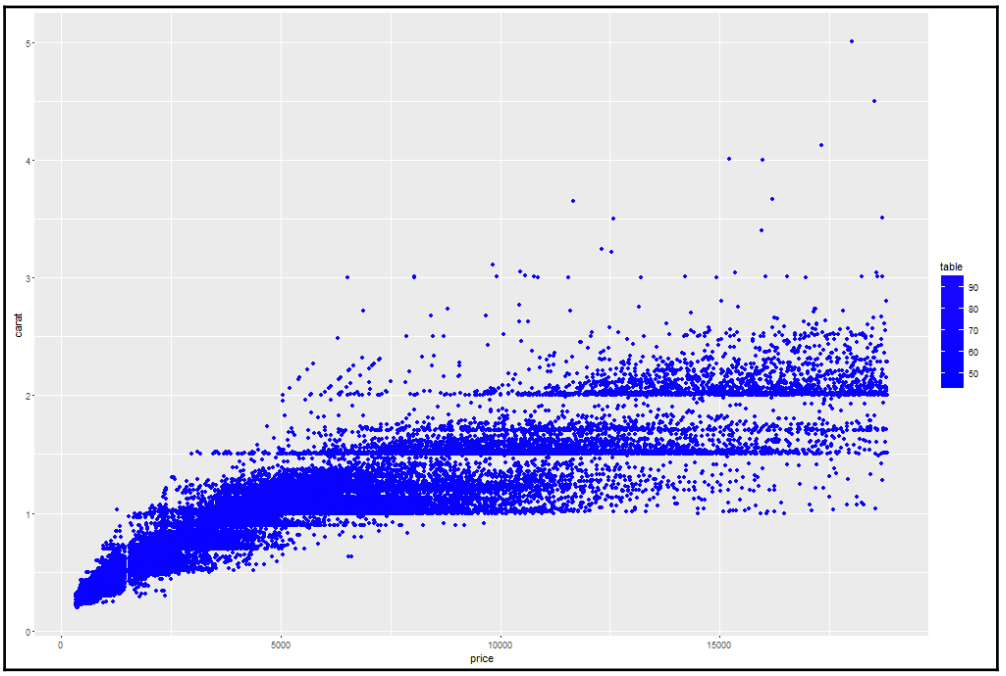
There are certain external graphical themes which can be imported to the `ggplot2` function for visualization, such as `library(ggthemes)`. Tableau, which is a tool known for data visualization, its color, and themes, can also be used along with ggplots:

```
> #changing discrete category colors
> ggplot(diamonds, aes(price, carat, color=factor(cut)))+
+ geom_point() +
+ scale_color_brewer(palette="Set1")
> #Using tableau colors
> library(ggthemes)
> ggplot(diamonds, aes(price, carat, color=factor(cut)))+
+ geom_point() +
+ scale_color_tableau()
```



Plots created can be slightly modified using the `color` gradient and plotting a distribution on the graph itself:

```
> #using color gradient
> ggplot(diamonds, aes(price, carat))+
+ geom_point() +
+ scale_color_gradient(low = "blue", high = "red")
> #plotting a distribution on a graph
> mid<-mean(diamonds$price)
> ggplot(diamonds, aes(price, carat, color=depth))+geom_point()+
+ scale_color_gradient2(midpoint=mid,
+ low="blue", mid="white", high="red" )
```

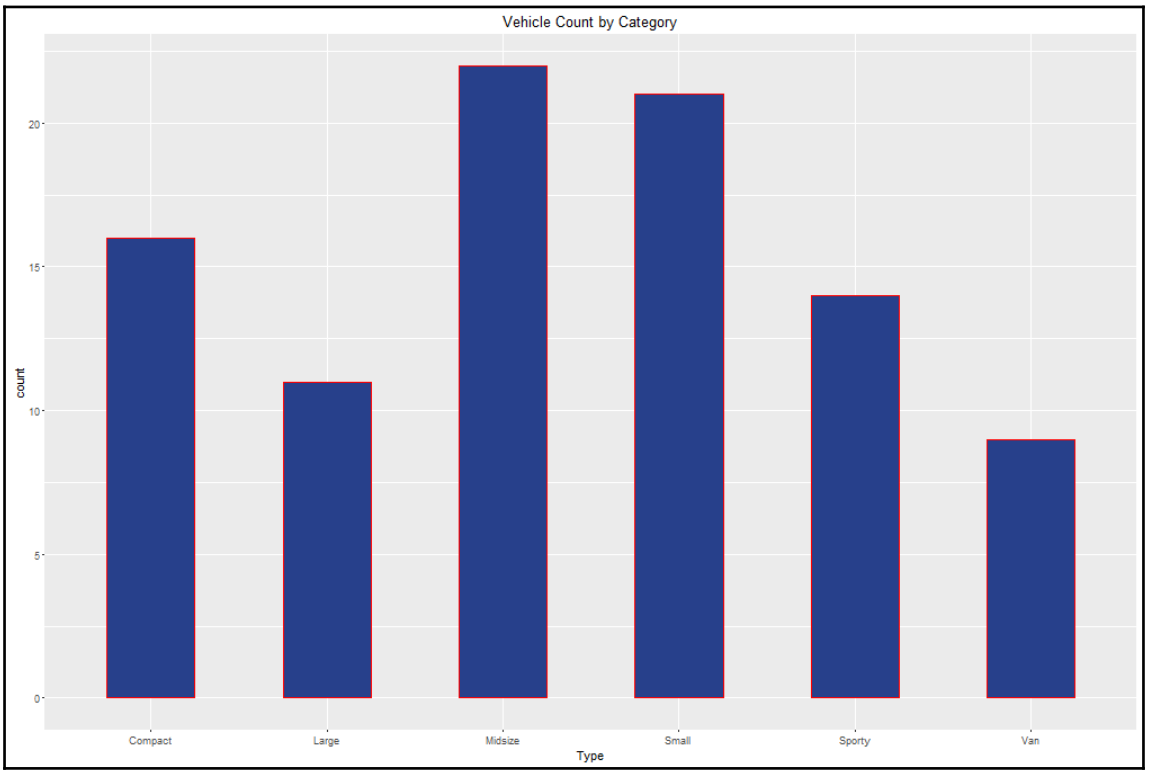


Having discussed in depth about the components in a graph building process, now let's try to understand how to create different charts and graphs using `ggplot2`. `qplot()` is a basic plotting function in `ggplot2`, which is a wrapper for creating different types of plots. There are two options for a user, either go plotting with the `qplot()` or `ggplot()` function. To create different graphs, we are going to use the `Cars93.csv` dataset.

Bar chart

Bar charts are preferred as a method of visualization for the categorical variables, also used to represent the count or percentage of each group. The horizontal axis represents the categories and the vertical axis either represents the count or the percentage:

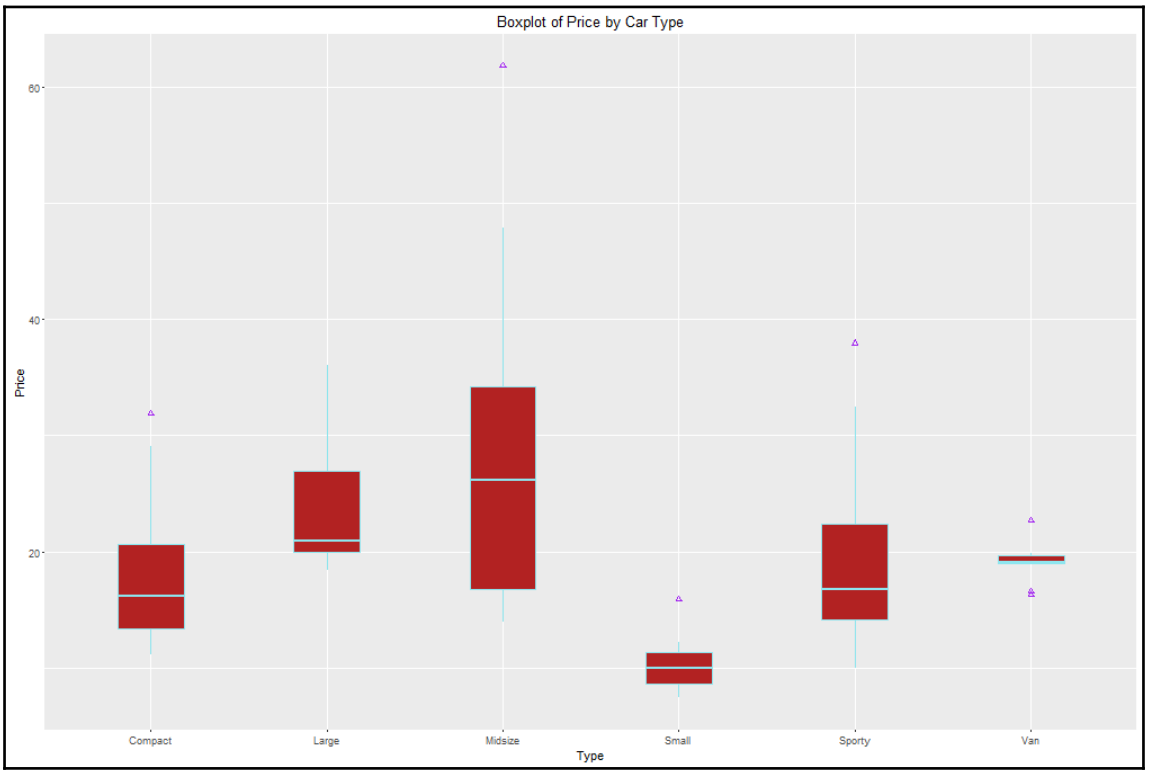
```
> #creating bar chart
> barplot <- ggplot(Cars93,aes(Type))+
+ geom_bar(width = 0.5,fill="royalblue4",color="red")+
+ ggtitle("Vehicle Count by Category")
> barplot
```



Boxplot

It is not only easy to interpret boxplots using the `ggplot` package, but also easy to customize the plot. One can easily recognize the outliers imposed on each of the corresponding boxplots:

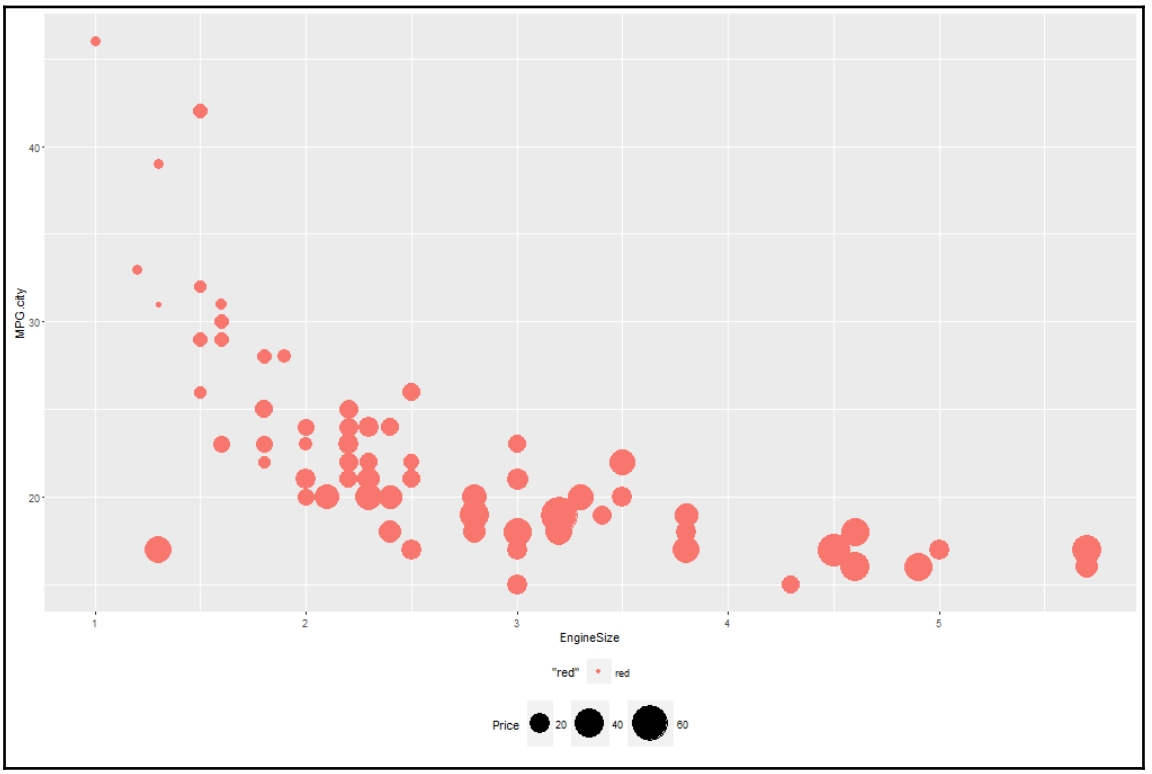
```
> #creating boxplot
> boxplot <- ggplot(Cars93,aes(Type,Price))+
+ geom_boxplot(width = 0.5,fill="firebrick",color="cadetblue2",
+ outlier.colour = "purple",outlier.shape = 2)+
+ ggtitle("Boxplot of Price by Car Type")
> boxplot
```



Bubble chart

The bubble chart belongs to the family of the scatterplot. It is preferred when it is required to represent three quantitative variables. Two quantitative variables are represented on two axis and one quantitative variable is used to represent the size of each bubble in a bubble chart:

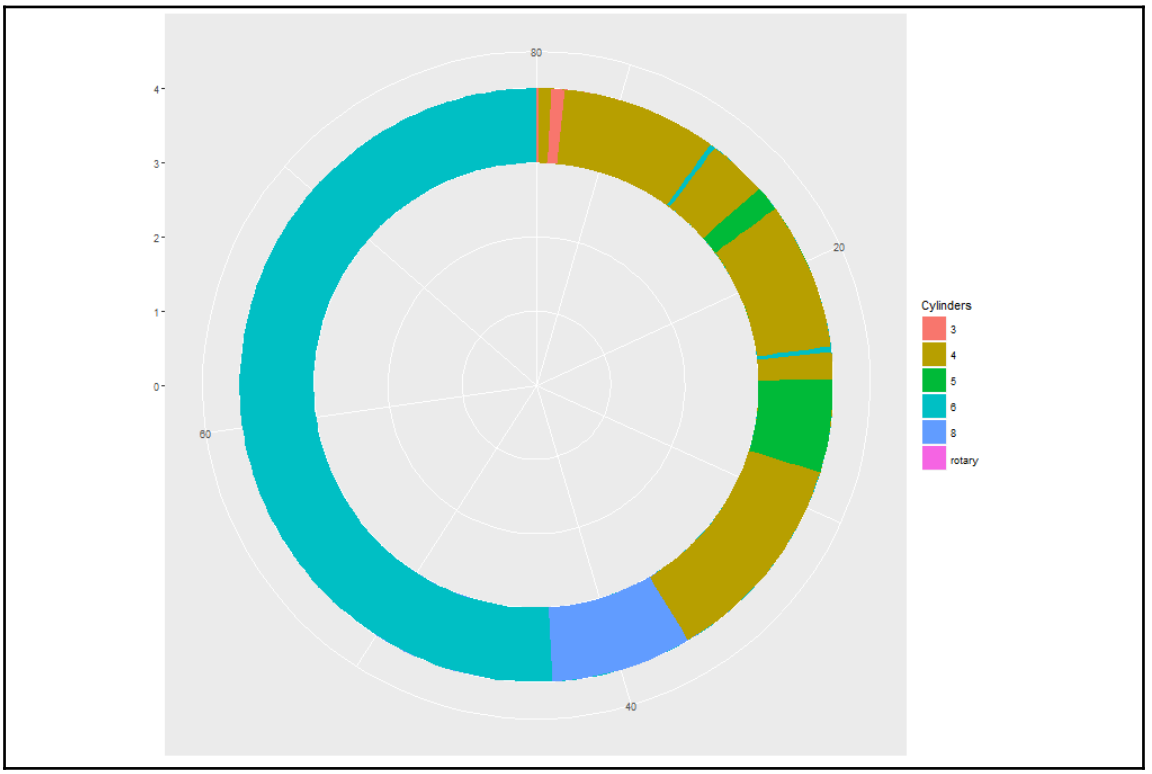
```
> #creatting Bubble chart
> bubble<-ggplot(Cars93, aes(x=EngineSize, y=MPG.city)) +
+ geom_point(aes(size=Price,color="red")) +
+ scale_size_continuous(range=c(2,15)) +
+ theme(legend.position = "bottom")
> bubble
```



Donut chart

Donut chart is used in place of pie chart when the number of categories exceeds five:

```
> #creating Donut charts
> ggplot(Cars93) + geom_rect(aes(fill=Cylinders, ymax=Max.Price,
+ ymin=Min.Price, xmax=4, xmin=3)) +
+ coord_polar(theta="y") + xlim(c(0, 4))
```

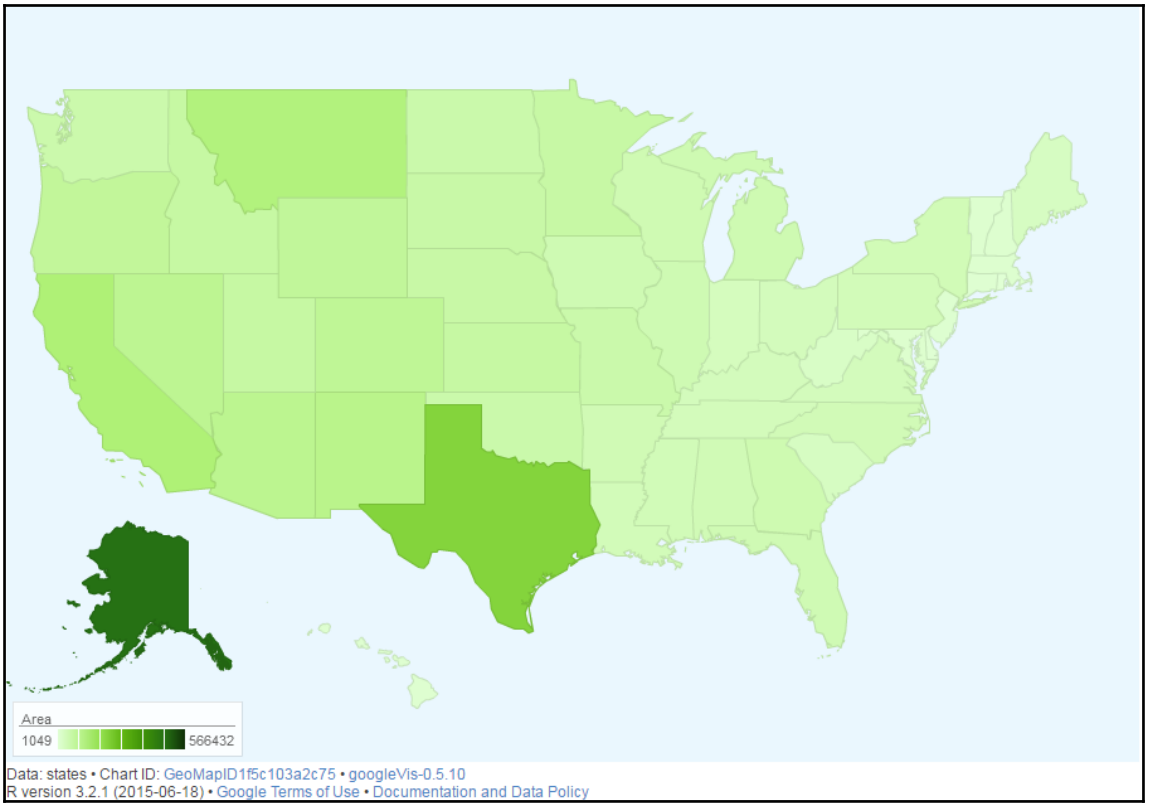


Geo mapping

Any dataset that has a city name and state name or a country name can be plotted on a geographical map using google visualization library in R. Using another open source dataset, `state.x77` inbuilt in R, we can show how a geo map looks. The google visualization library, using Google maps API, tries to plot the geographic locations on the plot along with the enterprise data. It publishes the output in a browser which can be stored back as an image to use it further:

```
> library(googleVis)
> head(state.x77)
Population Income Illiteracy Life Exp Murder HS Grad Frost
Alabama 3615 3624 2.1 69.05 15.1 41.3 20
Alaska 365 6315 1.5 69.31 11.3 66.7 152
Arizona 2212 4530 1.8 70.55 7.8 58.1 15
Arkansas 2110 3378 1.9 70.66 10.1 39.9 65
California 21198 5114 1.1 71.71 10.3 62.6 20
```

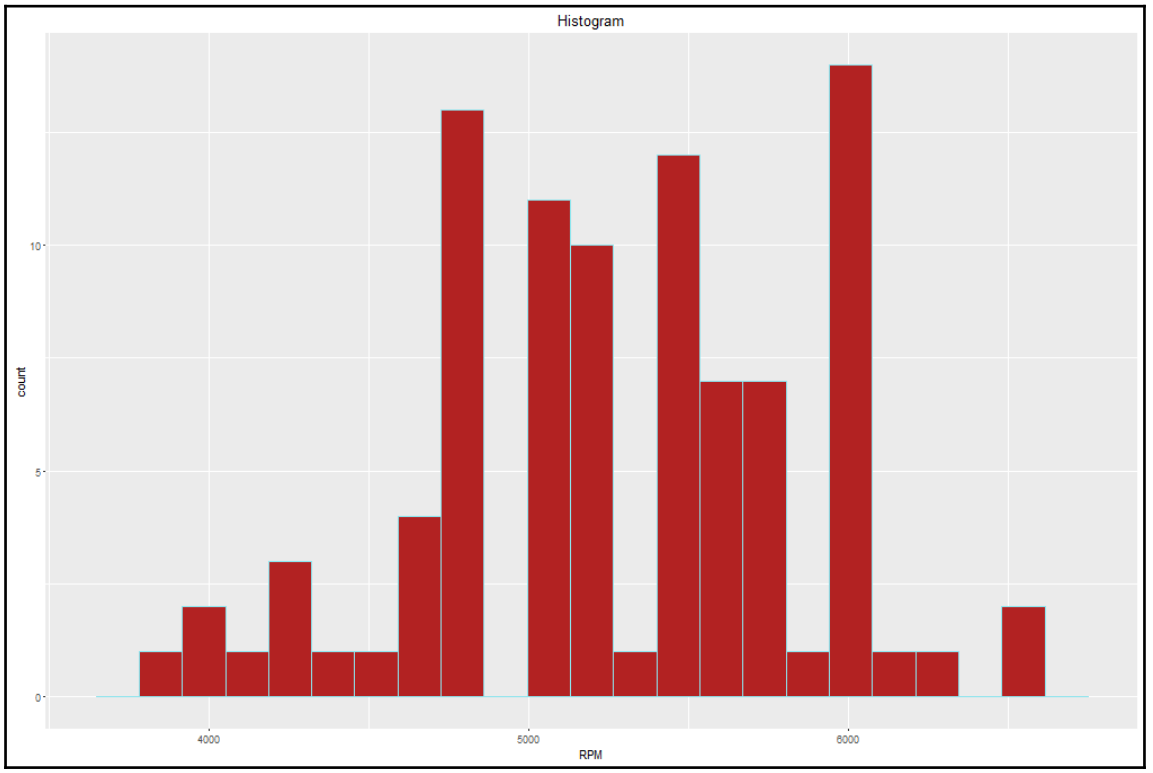
```
Colorado 2541 4884 0.7 72.06 6.8 63.9 166
Area
Alabama 50708
Alaska 566432
Arizona 113417
Arkansas 51945
California 156361
Colorado 103766
> states <- data.frame(state.name, state.x77)
> gmap <- gvisGeoMap(states, "state.name", "Area",
+ options=list(region="US", dataMode="regions",
+ width=900, height=600))
> plot(gmap)
```



Histogram

This is probably the easiest plot that every data mining professional must be doing. The following code explains how a histogram can be created using the `ggplot` library:

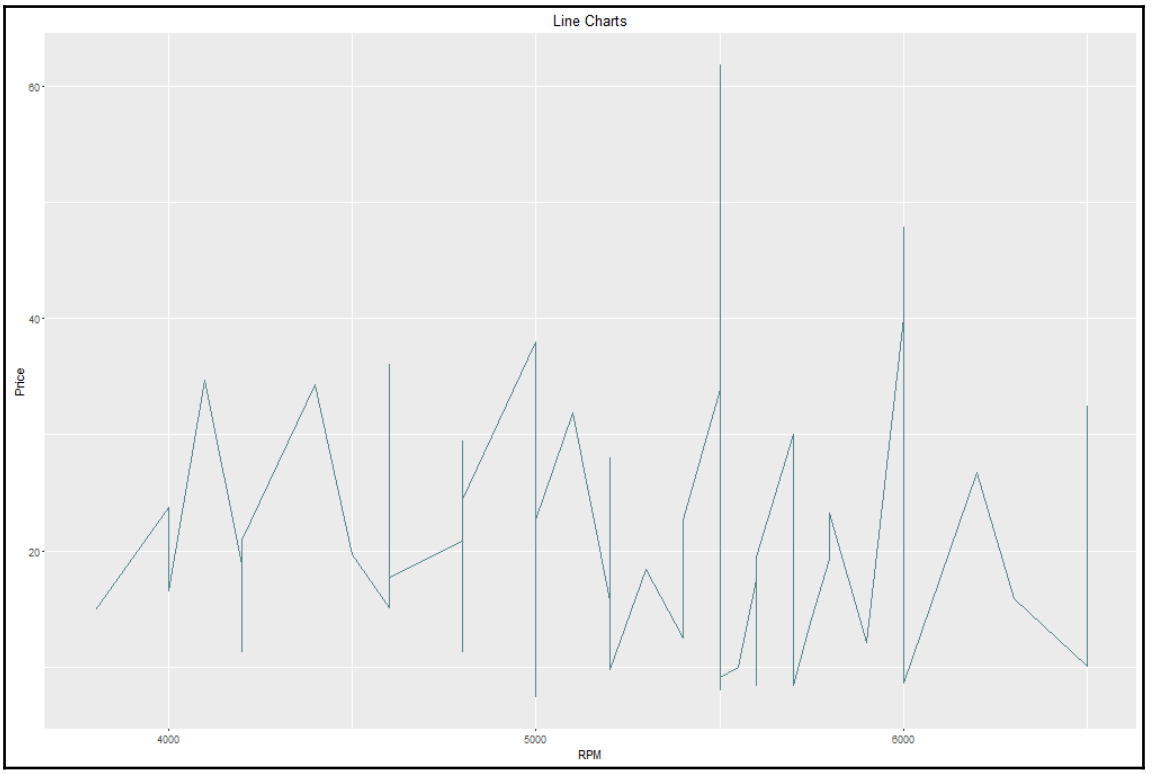
```
> #creating histograms
> histog <- ggplot(Cars93,aes(RPM))+
+ geom_histogram(width = 0.5,fill="firebrick",color="cadetblue2",
+ bins = 20)+
+ ggtitle("Histogram")
> histog
```



Line chart

Line chart is not a preferred chart while showing raw data. However, it is important while showing some variations across different categories relating to some metric. Though it is not a preferred chart, but it depends on the practitioner how he/she wants to display and tell a story to the reader:

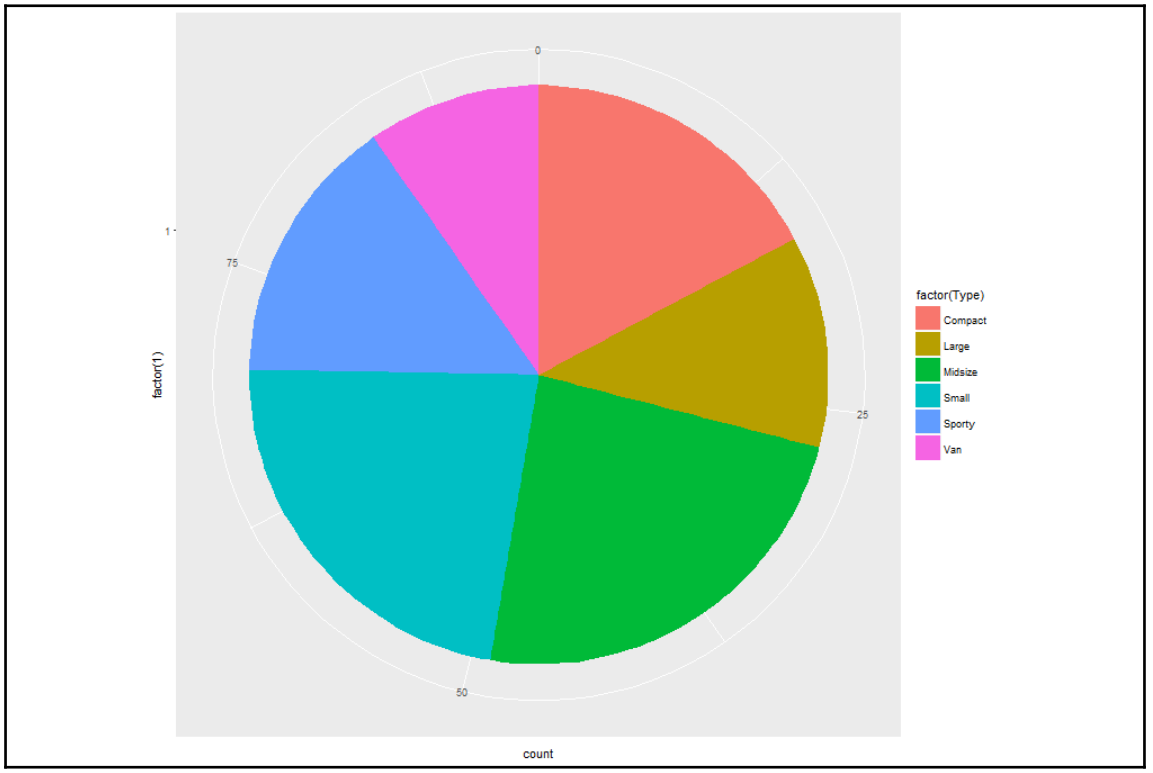
```
> #creating line charts
> linechart <- ggplot(Cars93,aes(RPM,Price))+
+ geom_line(color="cadetblue4")+
+ ggtitle("Line Charts")
>
> linechart
```



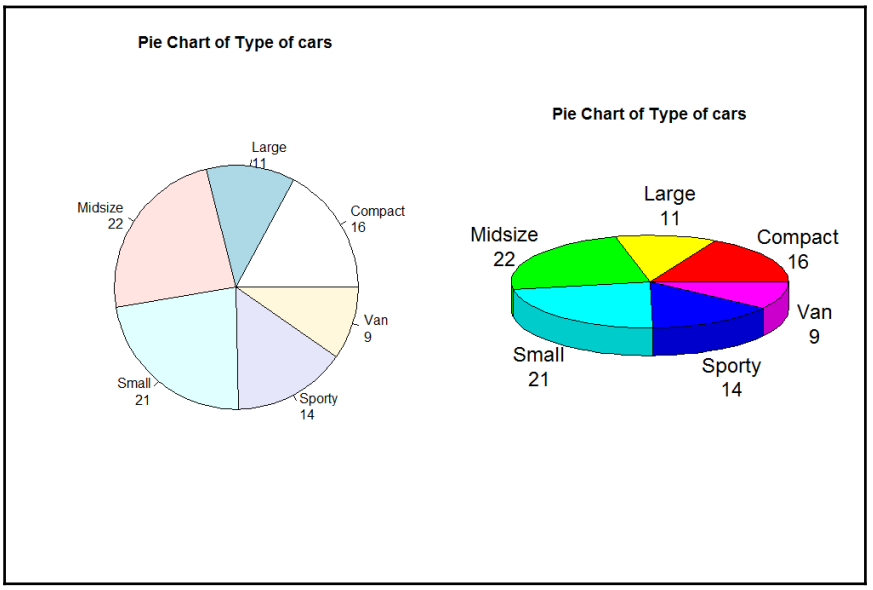
Pie chart

A pie chart is a representation of categorical variables when the label for each categorical variable is less than 10. If it exceeds 10, then it is suggested to look at a histogram or barplot for comparison. Using the `ggplot` library, the pie chart can be created. The script is as follows:

```
> #creating pie charts
> pp <- ggplot(Cars93, aes(x = factor(1), fill = factor(Type))) +
+ geom_bar(width = 1)
> pp + coord_polar(theta = "y")
```



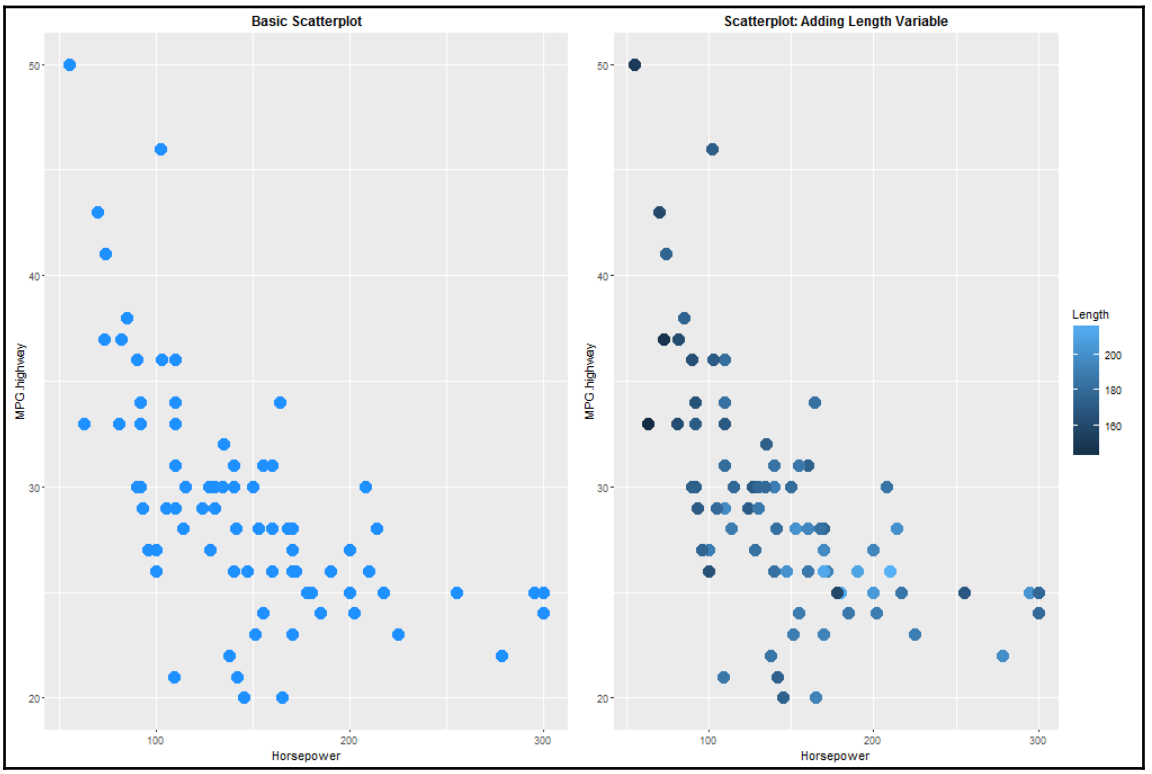

```
> # 3D Pie Chart from data frame
> library(plotrix)
> t <- table(Cars93$Type);par(mfrow=c(1,2))
> pct <- paste(names(t), "\n", t, sep="")
> pie(t, labels = pct, main="Pie Chart of Type of cars")
> pie3D(t,labels=pct,main="Pie Chart of Type of cars")
```



Scatterplot

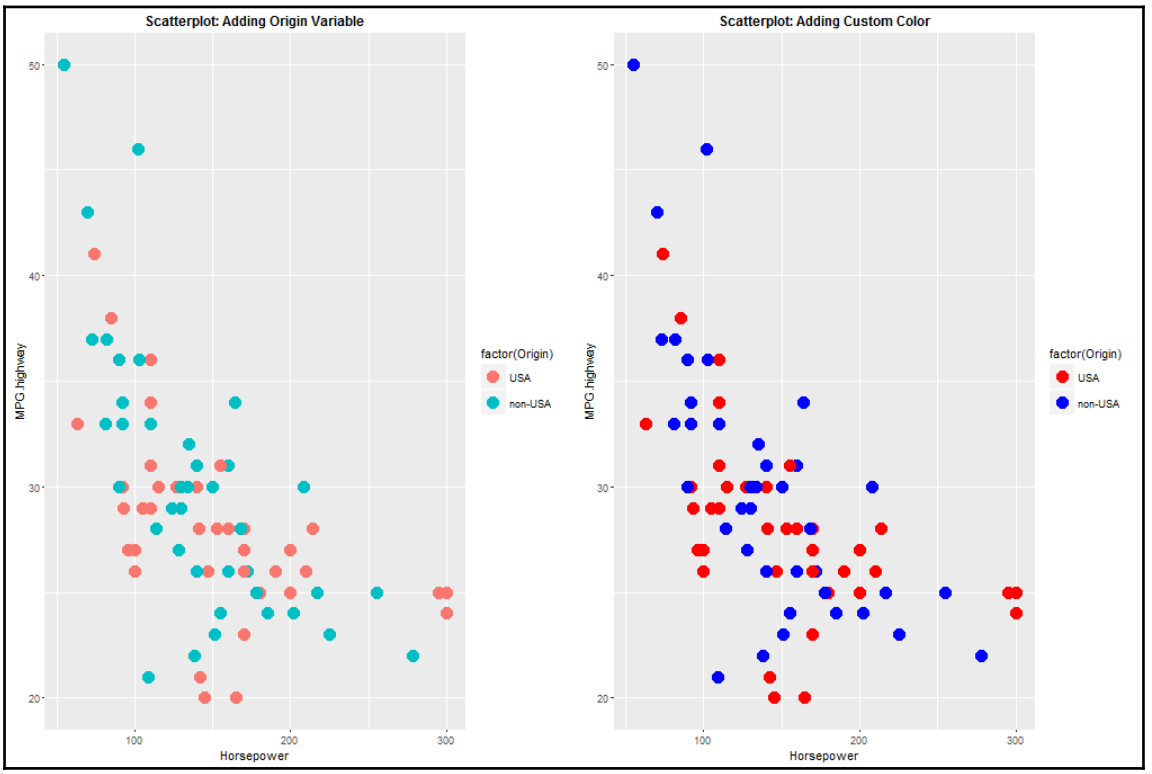
Scatterplot is a very important plot to understand the bivariate relationship that exists in data. It also shows the pattern in which the data is stored over a period of time. It is also important to show the data in a proper way while showing it in scatterplots. The following example shows how a bivariate relationship can be displayed along with some third dimension dictating the visualization in a bivariate relationship. The third dimension could be a continuous variable or a categorical variable. Using the `gridExtra()` library, additional graphing window can be created where two or more plots can be represented side by side, with some relationship:

```
> library(gridExtra)
> sp <- ggplot(Cars93,aes(Horsepower,MPG.highway))+
+ geom_point(color="dodgerblue",size=5)+ggtitle("Basic Scatterplot")+
+ theme(plot.title= element_text(size = 12, face = "bold"))
> sp
> #adding a cantinuous variable Length to scale thee scatterplot points
> sp2<-sp+geom_point(aes(color=Length), size=5)+
+ ggtitle("Scatterplot: Adding Length Variable")+
+ theme(plot.title= element_text(size = 12, face = "bold"))
> sp2
>
> grid.arrange(sp,sp2,nrow=1)
```



In the second graph in the preceding plot, the length variable, which is continuous, is dictating the relationship between the horsepower and the highway mileage per gallon. The light blue colored dots indicate lengthy cars while the darker dots indicate smaller cars. Instead of a continuous variable, if we use a factor variable to scale the relationship between the two variables, we will be able to see a plot like given in first graph in the following plot:

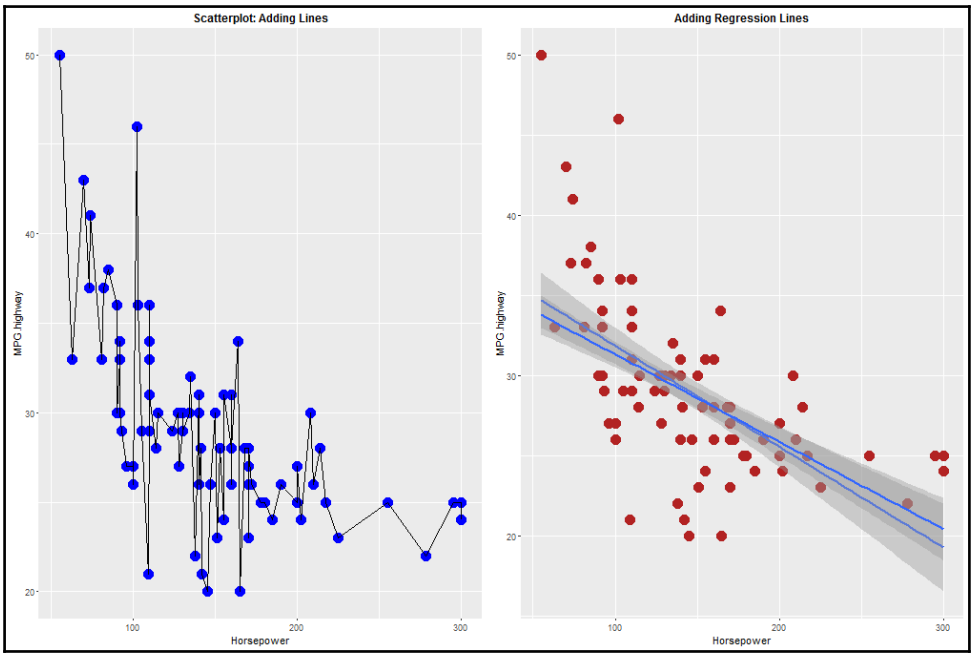
```
> #adding a factor variable Origin to scale the scatterplot points
> sp3<-sp+geom_point(aes(color=factor(Origin)),size=5)+
+ ggtitle("Scatterplot: Adding Origin Variable")+
+ theme(plot.title= element_text(size = 12, face = "bold"))
> sp3
> #adding custom color to the scatterplot
> sp4<-sp+geom_point(aes(color=factor(Origin)),size=5)+
+ scale_color_manual(values = c("red","blue"))+
+ ggtitle("Scatterplot: Adding Custom Color")+
+ theme(plot.title= element_text(size = 12, face = "bold"))
> sp4
> grid.arrange(sp3,sp4,nrow=1)
```



To display the cause and effect relationship, one needs to display a trendline or a regression line on a scatterplot. Using the `ggplot2` library, different regression lines can be plotted, such as linear, non linear, generalized linear, and so on. When the number of observations in a dataset is less than 1000, the loess regression method is applied by default. However, when it is more than 1000, the generalized additive model is applied. The trend lines are displayed next. The first plot indicates a line graph connecting all the points, and the second plot indicates the robust linear model:

```
> sp5<-sp+geom_point(color="blue",size=5)+geom_line()+
+ ggtitle("Scatterplot: Adding Lines")+
+ theme(plot.title= element_text(size = 12, face = "bold"))
> sp5
> #adding regression lines to the scatterplot
> sp6<-sp+geom_point(color="firebrick",size=5)+
+ geom_smooth(method = "lm",se =T)+
+ geom_smooth(method = "rlm",se =T)+
+ ggtitle("Adding Regression Lines")+
+ theme(plot.title= element_text(size = 12, face = "bold"))
```

```
> sp6
> grid.arrange(sp5,sp6,nrow=1)
```

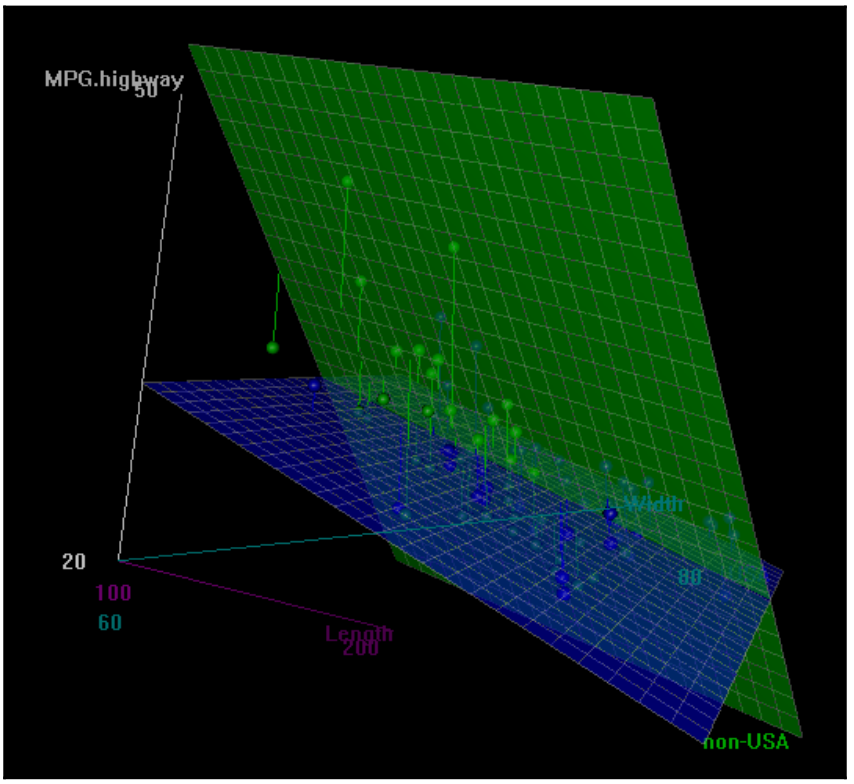


Adding the generalized regression model and loess as a non-linear regression model, we can modify the scatterplots as follows:

```
> sp7<-sp+geom_point(color="firebrick",size=5)+
+ geom_smooth(method = "auto",se =T)+
+ geom_smooth(method = "glm",se =T)+
+ ggtitle("Adding Regression Lines")+
+ theme(plot.title= element_text(size = 20, face = "bold"))
> sp7
> #adding regression lines to the scatterplot
> sp8<-sp+geom_point(color="firebrick",size=5)+
+ geom_smooth(method = "gam",se =T)+
+ ggtitle("Adding Regression Lines")+
+ geom_smooth(method = "loess",se =T)+
+ theme(plot.title= element_text(size = 20, face = "bold"))
> sp8
> grid.arrange(sp7,sp8,nrow=1)
```

3D scatterplot is another addition to the list of scatterplot functions we are looking at. The 3D scatterplot library enables the users to look at the plot and rotate it, to view the data points from different angles. Once executed, the following script would open up a new rgl device window. Just rotate the graph and you would be able to see the data points from different angles:

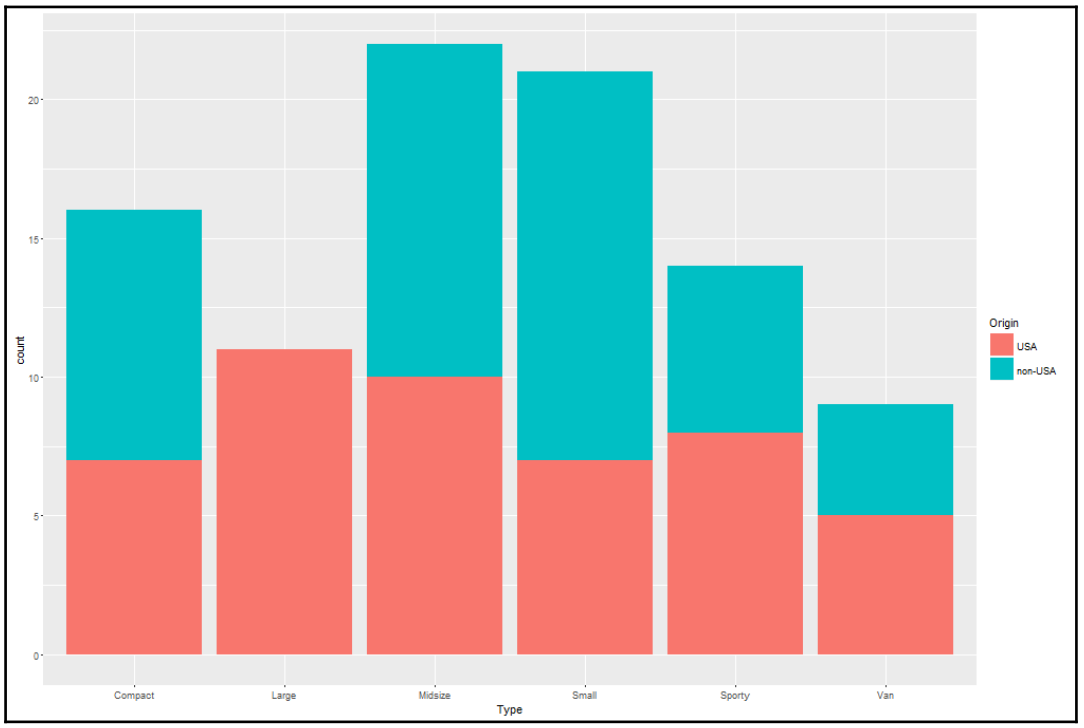
```
> library(scatterplot3d);library(Rcmdr)
> scatter3d(MPG.highway~Length+Width|Origin, data=Cars93,
fit="linear",residuals=TRUE, parallel=FALSE, bg="black", axis.scales=TRUE,
grid=TRUE, ellipsoid=FALSE)
```



Stacked bar chart

Stacked bar charts are just another variant of bar charts, where more than two variables can be plotted with different combinations of color. The following example codes show some variants of stacked bar charts:

```
> qplot(factor(Type), data=Cars93, geom="bar", fill=factor(Origin))
>
> #or
>
> ggplot(Cars93, aes(Type, fill=Origin)) + geom_bar()
```



Stem and leaf plot

A stem and leaf plot is a textual representation of a quantitative variable that segments the values to their most significant numeric digits. For example, the stem and leaf plot for the mileage within the `city` variable from the `Cars93.csv` dataset is represented as follows:

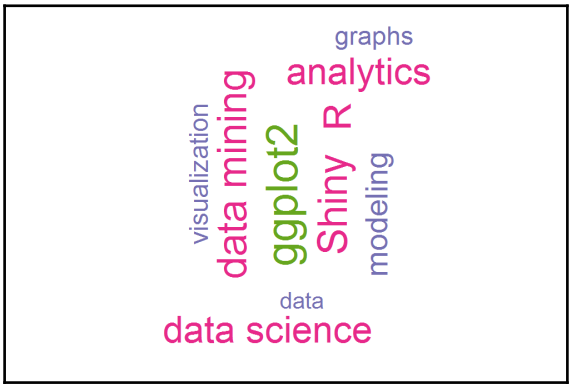
```
> stem(Cars93$MPG.city)
The decimal point is 1 digit(s) to the right of the |
1 | 5566677777777888888888889999999999
2 | 00000000111111222222233333333444444
2 | 5555556688999999
3 | 01123
3 | 9
4 | 2
4 | 6
```

To interpret the results of a stem and leaf plot: if we need to know how many observations are there which are greater than 30, the answer is 8, the digit on the left of pipe indicates items and the numbers on the right indicate units, hence the respective numbers are 30, 31, 31, 32, 33, 39, 42, 46.

Word cloud

Word cloud is a data visualization method which is preferred when it is required to represent a textual data. For example, the representation of a bunch of text files with few words having frequent appearances across those set of documents would summarize the topic of discussion. Hence, word cloud representation is a visual summary of the textual unstructured data. This is mostly used to represent social media posts, such as Twitter tweets, Facebook posts, and so on. There are various pre-processing tasks before arriving to create a word cloud, the final output from a text mining exercise would be a data frame with words and their respective frequencies:

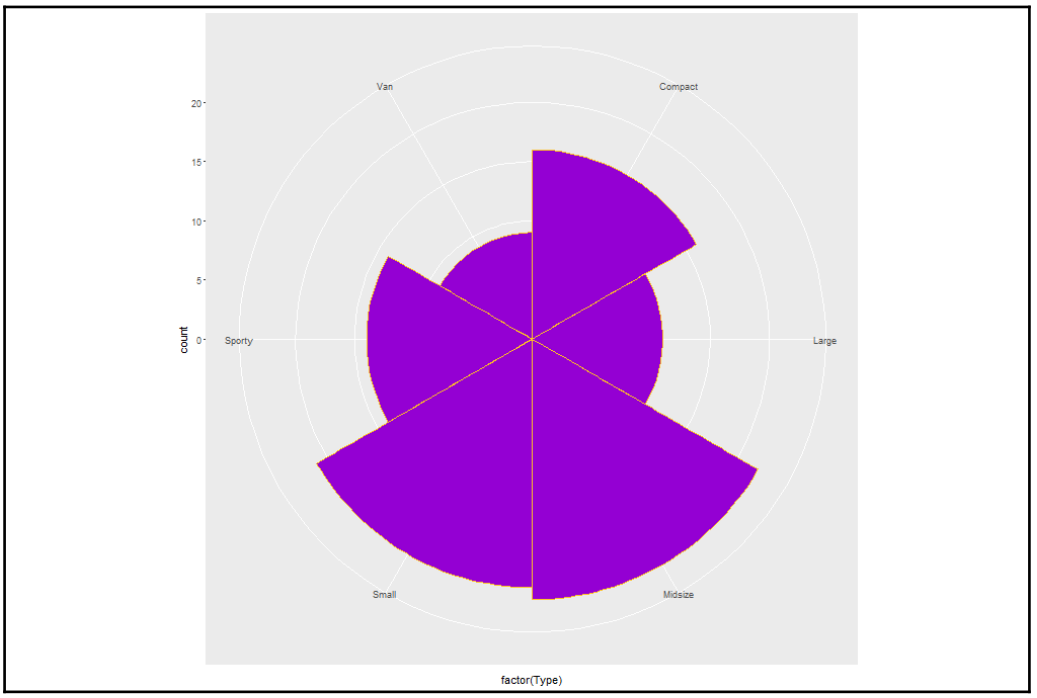
```
#Word cloud representation
library(wordcloud)
words<-c("data","data mining","analytics","statistics","graphs",
"visualization","predictive analytics","modeling","data science",
"R","Python","Shiny","ggplot2","data analytics")
freq<-c(123,234,213,423,142,145,156,176,214,218,213,234,256,324)
d<-data.frame(words,freq)
set.seed(1234)
wordcloud(words = d$words, freq = d$freq, min.freq = 1,c(8,.3),
max.words=200, random.order=F, rot.per=0.35,
colors=brewer.pal(7, "Dark2"))
```



Coxcomb plot

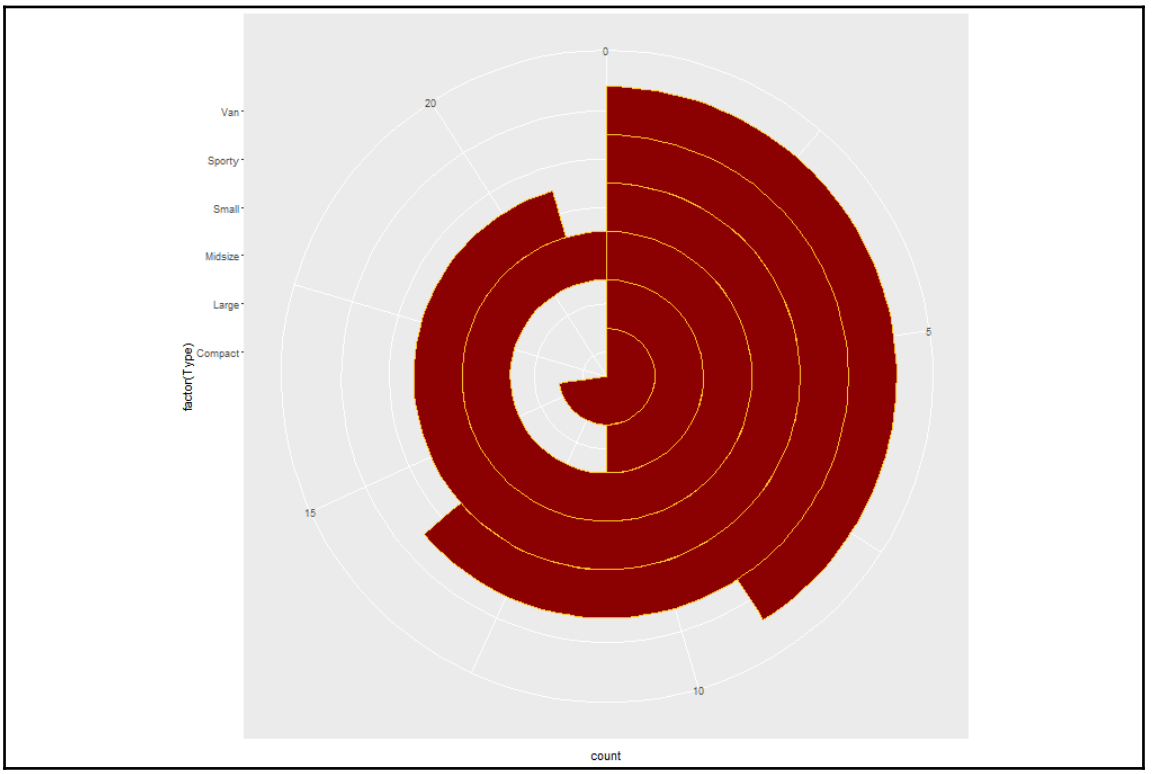
The coxcomb chart, which is also known as polar chart or rose chart, is a combination of pie chart and bar chart. The area of each section is adjusted based on the values of that segment by changing the radius. Anyone can understand the insights represented using coxcomb chart and does not require any technical knowledge:

```
> #coxcomb chart = bar chart + pie chart
> cox<- ggplot(Cars93, aes(x = factor(Type))) +
+ geom_bar(width = 1, colour = "goldenrod1",fill="darkviolet")
> cox + coord_polar()
```



A new variant of coxcomb plot by changing the coordinate polar measure, which is theta:

```
> #coxcomb chart = bar chart + pie chart
> cox<- ggplot(Cars93, aes(x = factor(Type))) +
+ geom_bar(width = 1, colour = "goldenrod1",fill="darkred")
> cox + coord_polar()
> #a second variant of coxcomb plot
> cox + coord_polar(theta = "y")
```



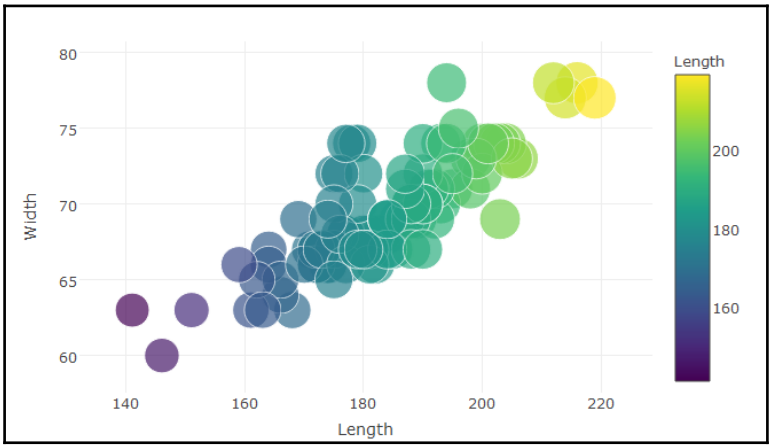
Using plotly

So far, we have looked at various scenarios of creating plots using the `ggplot2` library. In order to take the plotting to a new level, there are many libraries which can be referred to. Out of them, one library is `plotly`, which is designed as an interactive browser-based charting library built on the JavaScript library. Let's look at a few examples on how it works.

Bubble plot

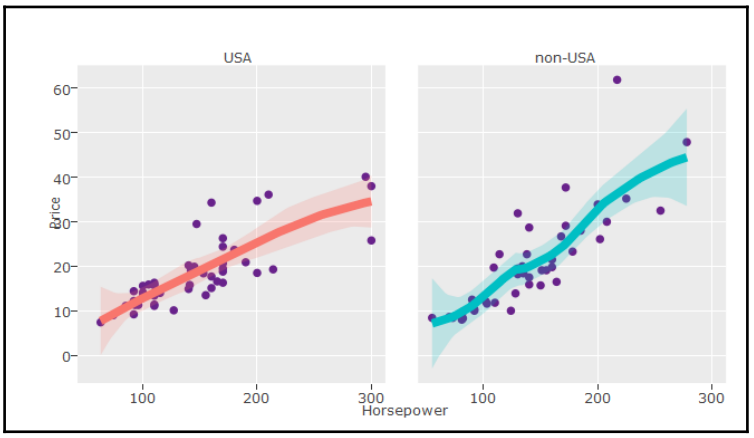
Bubble plot is a nice visualization in which the size of the bubble indicates the weight of each variable as it is present in the dataset. Let's look at the following plot:

```
> #Bubble plot using plotly
> plot_ly(Cars93, x = Length, y = Width, text = paste("Type: ", Type),
+ mode = "markers", color = Length, size = Length)
```



The combination of `ggplot2` with the `plotly` library makes for good visualization, as the features of both the libraries are embedded with the `ggplotly` library:

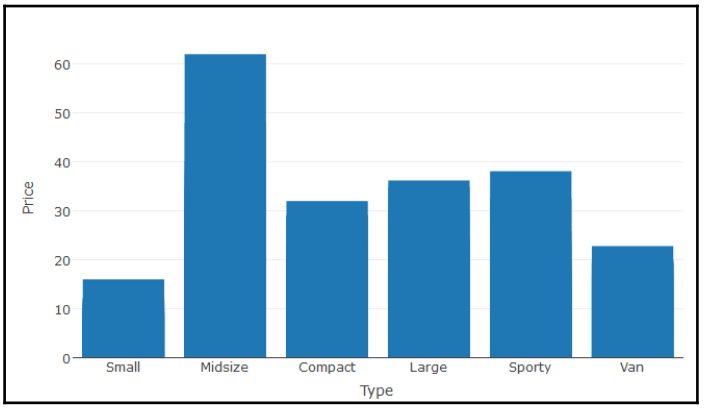
```
> #GGPLOTLY: ggplot plus plotly
> p <- ggplot(data = Cars93, aes(x = Horsepower, y = Price)) +
+ geom_point(aes(text = paste("Type:", Type)), size = 2,
color="darkorchid4") +
+ geom_smooth(aes(colour = Origin, fill = Origin)) + facet_wrap(~ Origin)
>
> (gg <- ggplotly(p))
```



Bar charts using plotly

Bar charts using `plotly` look smarter than regular bar charts available in R, as a base functionality, let's have a look at the graph as follows:

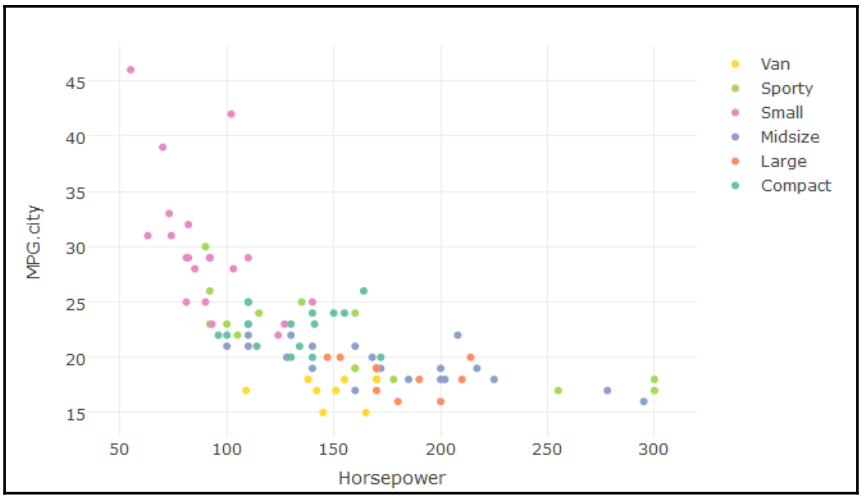
```
> p <- plot_ly(
+ x = Type,
+ y = Price,
+ name = "Price by Type",
+ type = "bar")
> p
```



Scatterplot using plotly

Representation of two continuous variables can be shown using a scatterplot. Let's look at the data represented next:

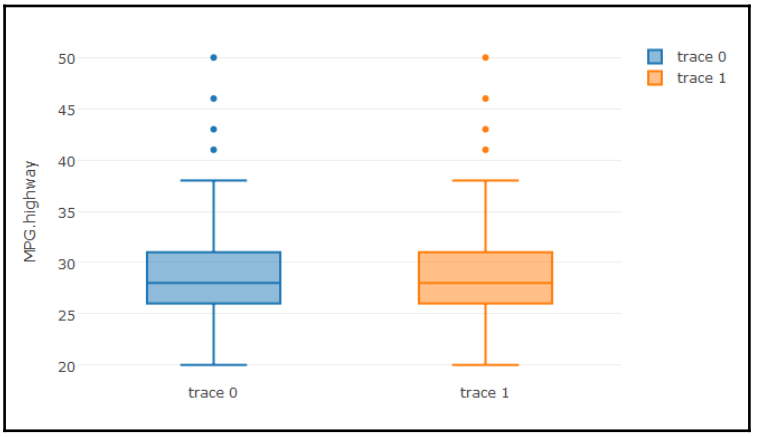
```
> # Simple scatterplot
> library(plotly)
> plot_ly(data = Cars93, x = Horsepower, y = MPG.highway, mode = "markers")
> #Scatter Plot with Qualitative Colorscale
> plot_ly(data = Cars93, x = Horsepower, y = MPG.city, mode = "markers",
+ color = Type)
```



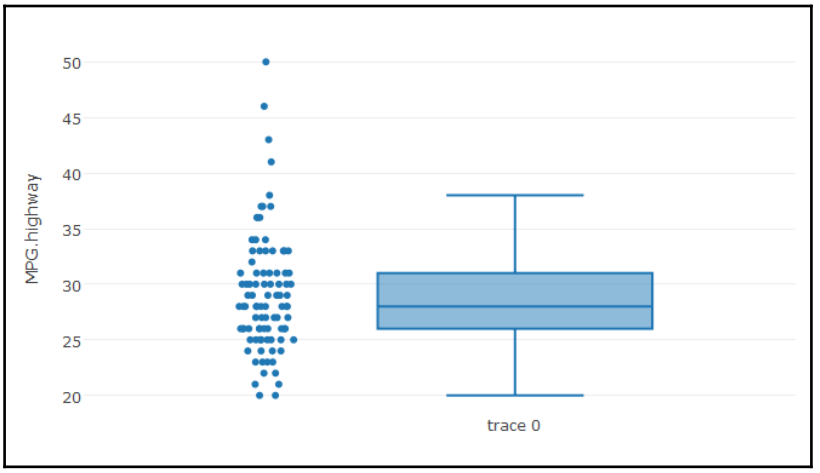
Boxplots using plotly

Following are examples of creating some interesting boxplots using the **plotly** library:

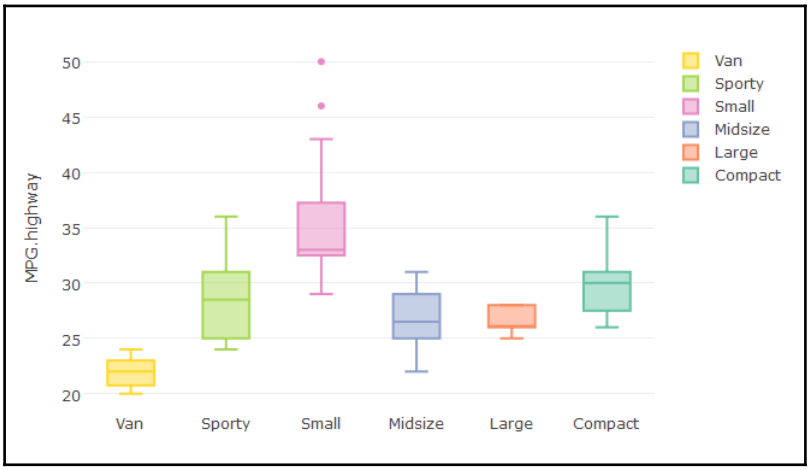
```
> #Box Plots
> library(plotly)
> ### basic boxplot
> plot_ly(y = MPG.highway, type = "box") %>%
+ add_trace(y = MPG.highway)
```



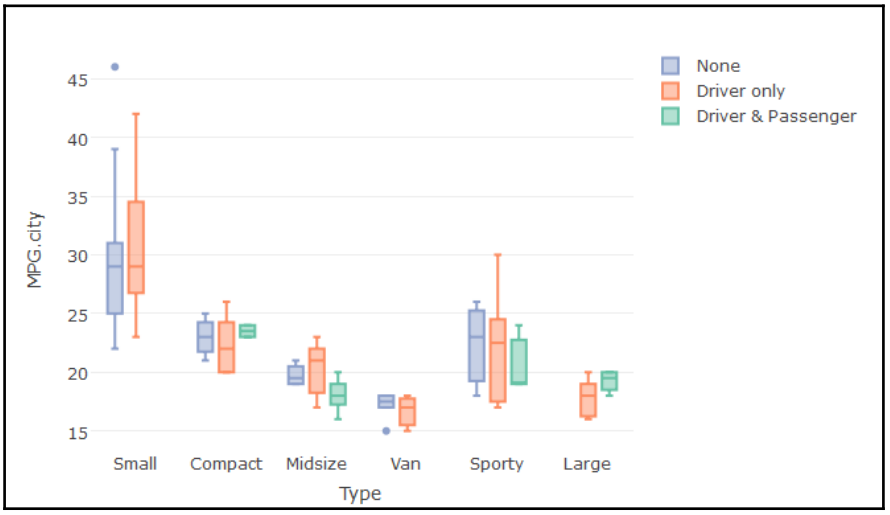

```
> ### adding jittered points
> plot_ly(y = MPG.highway, type = "box", boxpoints = "all", jitter = 0.3,
+ pointpos = -1.8)
```




```
> ### several box plots
> plot_ly(Cars93, y = MPG.highway, color = Type, type = "box")
```



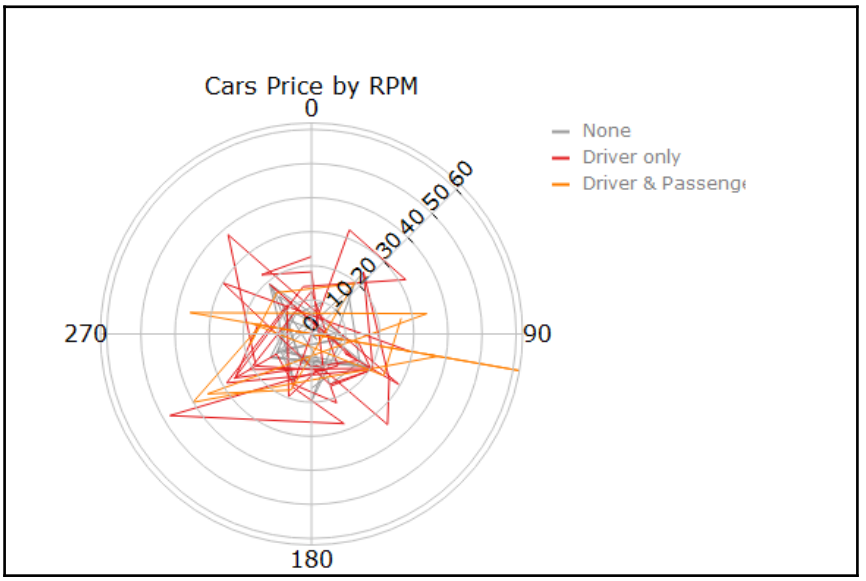

```
> ### grouped box plots
> plot_ly(Cars93, x = Type, y = MPG.city, color = AirBags, type = "box")
%>%
+ layout(boxmode = "group")
```



Polar charts using plotly

The polar chart visualization using `plotly` is more interesting to look at. This is because when you hover your cursor over the chart, the data values become visible and differences in pattern can be recognized. Let's look at the example code as given next:

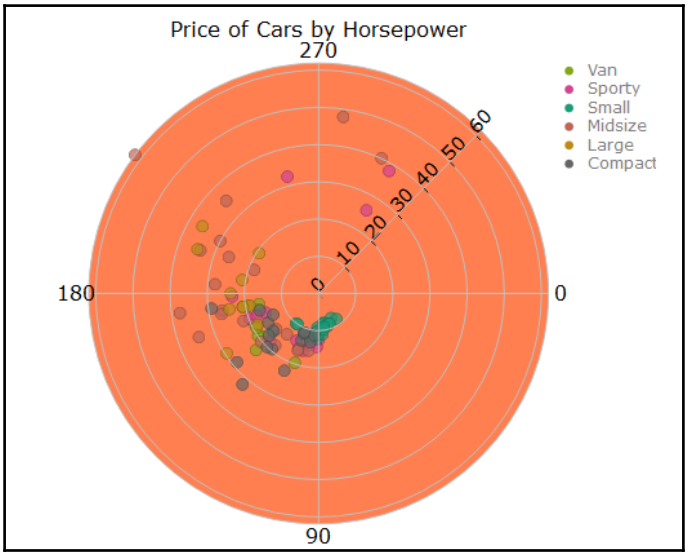
```
> #Polar Charts in R
> library(plotly)
pc <- plot_ly(Cars93, r = Price, t = RPM, color = AirBags,
mode = "lines",colors='Set1')
layout(pc, title = "Cars Price by RPM", orientation = -90,
font='bold')
```



Polar scatterplot using plotly

Using the `plotly` library, there is one additional chart type which a user can create. It's called polar scatterplot. Instead of a two dimensional scatterplot, the points are plotted in a circular fashion:

```
> #Polar Scatter Chart
pc <- plot_ly(Cars93, r = Price, t = Horsepower, color = Type,opacity =
0.7,
mode = "markers",colors = 'Dark2')
layout(pc, title = "Price of Cars by Horsepower",plot_bgcolor =
toRGB("coral"),
font='bold')
```

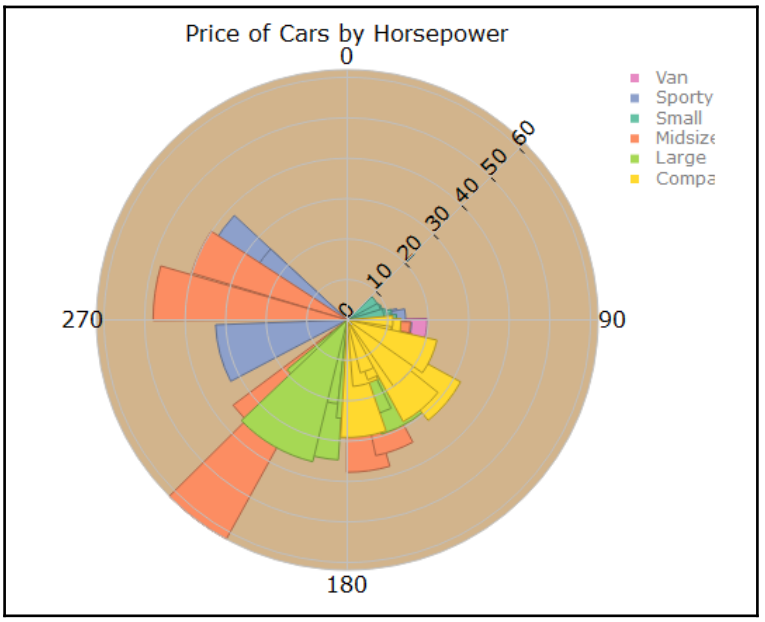


The graph polar scatterplot shows the price of cars by horsepower and the colors indicate the type of cars, the rings indicate proximity or closeness, one midsize car is very distinctly different from other variants as observed from the graph.

Polar area chart

The polar area chart, also known as the radar chart or coxcomb chart as renamed in some other packages, is shown next using a sample code. It shows the relationship between the of cars and horsepower by car type:

```
> #Polar Area Chart
pc <- plot_ly(Cars93, r = Price, t = Horsepower, color = Type, type =
"area")
layout(pc,title = "Price of Cars by Horsepower",orientation = 270,
plot_bgcolor = toRGB("tan"),font='bold')
```



Creating geo mapping

Geo mapping is a type of chart which is used by data mining experts when the dataset contains location information. The geo mapping plots are supported by the `ggmap` library. The location information can be accessed in three different ways:

- By the name of the place, location name, and address
- By the latitude and longitude of the place
- By exact location, lower left longitude, lower left latitude, upper right longitude, and upper left latitude

Once the map location is identified, then by using the `ggmap` function the location can be identified on a map:

```
>library(ggmap)
>gc <- geocode("statue of liberty", source = "google")
>googMap <- get_googlemap(center = as.numeric(gc))
>(bb <- attr(googMap, "bb"))
>bb2bbox(bb)
>gc<-get_map(location = c(lon = gc$lon, lat = gc$lat))
>ggmap(gc)
```

Summary

In this chapter, we looked at the various types of charts. We briefly discussed the syntax to create those charts and we also discussed briefly where to use which type of chart. To create a visual display to explain the insights and message to the audience is a skill. It takes time and becomes perfect by experience. In this chapter, we only looked at the most important data visualization methods used in data mining domain. However, there is overabundance of graphs and charts to be selected to create innovative presentations. So the key take away from this chapter is that the reader now knows the data visualization rules, the types of charts and graphs used in data mining to show the relationship between various variables, and understands the distribution of various variables. At the same time, the reader has got hands on experience by doing side by side practice on two important data visualization libraries: `ggplot2` and `plotly`. In the next chapter, we are going to learn about application of various regression techniques, interpretation of regression results, and visualization of regression results to understand the relationship between various variables.

4
Regression with Automobile Data

One of the primary objectives of data mining projects is to understand the relationship between various variables and establish a cause and effect relationship between the variable of interest and other explanatory variables. In data mining projects, performing predictive analytics not only entails insights on hidden messages in datasets but also helps in making future decisions which might impact the business outcomes. In this chapter, the reader will get to know the basics of predictive analytics using regression methods, including various linear and nonlinear regression methods using R programming. The reader will be able to understand the theoretical background as well as get practical hands-on experience with all the regression methods using R programming language.

In this chapter, we will cover the following topics:

- Regression formulation
- Linear regression
- Logical regression
- Cubic regression
- Stepwise regression
- Penalized regression

Regression introduction

Regression methods help in predicting the future outcomes of a target variable. As an example, here are a few business cases:

- In the sales and marketing domain, how can a business achieve a significant improvement in sales? Can we successfully predict using the if a necessary amendment is made to the drivers of sales?
- In the retail domain, can we predict the number of visitors to a website, so that necessary tech support can be aligned to help operate the site better?
- How can a retail/e-commerce owner predict the number of footfall to their store in a month/week/year?
- In the banking domain, how can a bank predict the number of people applying for home loans, car loans, and personal loans, so that they can maintain their liquid capital to support the demand?
- In the banking domain, prediction of default probability can be estimated using regression methods so that a bank can decide whether to approve a loan/credit card to a customer.
- In the automobile manufacturing domain, the sales unit of vehicles is indirectly proportional to the price of the vehicles and the price of a vehicle is decided by many factors such as usages of different metals/components, and various vehicle features such as RPM, mileage, length, width, and others. How can a manufacturer predict the unit of sales?

There are different methods to perform regression, including both linear as well as non-linear. Regression-based predictive analytics is being used in different industries in a different way. Regression methods support the prediction of a continuous variable, prediction of the probability of a success or failure of a variable, prediction of events based on features, and so on.

Formulation of regression problem

The formulation of a regression problem is very essential in creating a good regression-based predictive model. A typical approach in building a good predictive model follows a process of converting a business problem to a statistical problem, then converting the statistical problem to a statistical solution, and finally converting the statistical solution to a business solution. The following steps are required to build a good predictive model using regression based methods:

- A clear understanding about the background or context is very much required. Sometimes a good predictive model does not make sense for a business from a context point of view. However, a contextually relevant predictive model may not be a good predictive model.
- A clear understanding about the objective is needed: what are you predicting and why are you predicting? Domain expertise is required. Most of the times, non-relevant features get added to a model, having no practical sense, because they show a mere correlation.
- A clear understanding about correlation and regression is important. It's a common phenomenon that people misconstrue a mere correlation or association as regression. "All regression may show causal relationship, but not the contrary".
- Conversion of a business problem to a statistical problem should be done carefully, so that assumptions and business understanding can be taken care of in the model.

Initial exploratory data analysis reveals the relationship between variables so that the variables can be included in a predictive model. The exploratory data analysis involves univariate, bivariate, and multivariate data analysis. Missing value imputation, outlier treatment, and removal of data entry errors are also equally important before proceeding with regression methods.

Case study

We are going to take two datasets, `Cars93_1.csv` and `ArtPiece_2.csv`, to explain various regression methods with a detailed analysis of what regression to use where and under what circumstances. For each regression method, we will take a look at the assumptions, limitations, mathematical formulations, and interpretation of the results.

Linear regression

The linear regression model is used for explaining the relationship between a single dependent variable known as Y and one or more $X's$ independent variables known as input, predictor, or independent variables. The following two equations explain a linear regression scenario:

$$Y = f(X) + \varepsilon \ldots\ldots (1)$$
$$Y = f(X_1, X_2 \ldots\ldots\ldots X_n) + \varepsilon \ldots\ldots (2)$$

Equation 1 shows the formulation of a simple linear regression equation and equation 2 shows the multiple linear equations where many independent variables can be included. The dependent variable must be a continuous variable and the independent variables can be continuous and categorical. We are going to explain the multiple linear regression analysis by taking the `Cars93_1.csv` dataset, where the dependent variable is the price of cars and all the other variables are explanatory variables. The regression analysis includes:

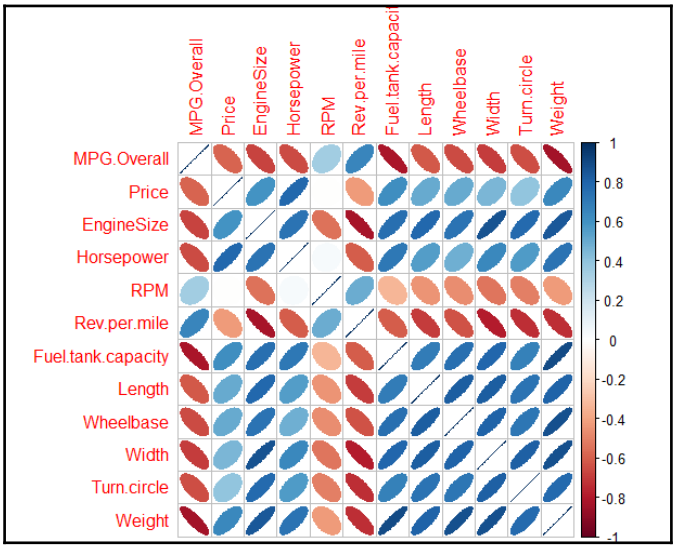
- Predicting the future values of the dependent variable, given the values for independent variables
- Assessment of model fit statistics and comparison of various models
- Interpreting the coefficients to understand the levers of change in the dependent variable
- The relative importance of the independent variables

The primary objective of a regression model is to estimate the beta parameters and minimize the error term epsilon:

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_3 \ldots\ldots \beta_n X_n) + \varepsilon \ldots\ldots (3)$$

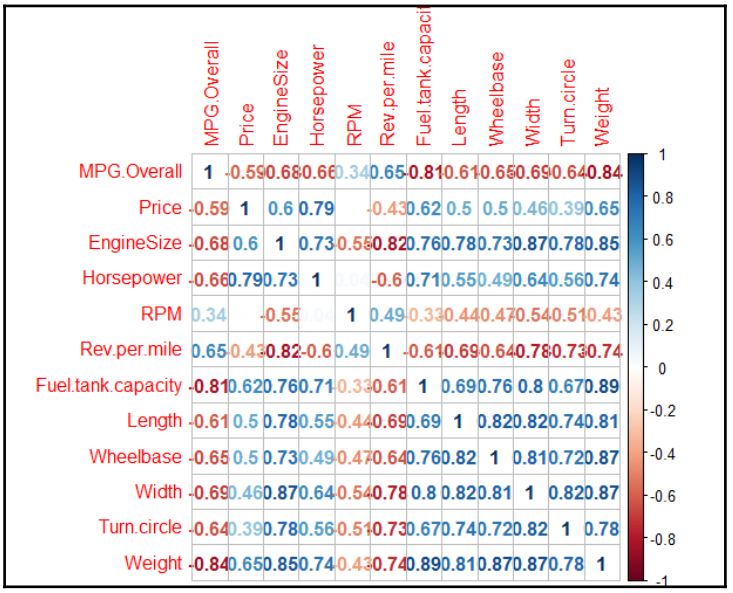
Let's look at the relationship between various variables using a correlation plot:

```
#Scatterplot showing all the variables in the dataset
library(car);attach(Cars93_1)
> #12-"Rear.seat.room",13-"Luggage.room" has NA values need to remove them
> m<-cor(Cars93_1[,-c(12,13)])
> corrplot(m, method = "ellipse")
```



The preceding figure shows the relationship between various variables. From each ellipse we can get to know how the relationship: is it positive or negative, and is it strongly associated, moderately associated, or weakly associated? The correlation plot helps in taking a decision on which variables to keep for further investigation and which ones to remove from further analysis. The dark brown colored ellipse indicates perfect negative correlation and the dark blue colored ellipse indicates the perfect positive correlation. The narrower the ellipse, the higher the degree of correlation.

Given the number of variables in the dataset, it is quite difficult to read the scatterplot. Hence the correlation between all the variables can be represented using a correlation graph, as follows:



Correlation coefficients represented in bold blue color show higher degree of positive correlation and the ones with red color show higher degree of negative correlation.

The assumptions of a linear regression are as follows:

- **Normality**: The error terms from the estimated model should follow a normal distribution
- **Linearity**: The parameters to be estimated should be linear
- **Independence**: The error terms should be independent, hence no-autocorrelation
- **Equal variance**: The error terms should be homoscedastic
- **Multicollinearity**: The correlation between the independent variables should be zero or minimum

While fitting a multiple linear regression model, it is important to take care of the preceding assumptions. Let's look at the summary statistics from the multiple linear regression analysis and interpret the assumptions:

```
> #multiple linear regression model
> fit<-lm(MPG.Overall~.,data=Cars93_1)
> #model summary
> summary(fit)
Call:
lm(formula = MPG.Overall ~ ., data = Cars93_1)
```

```
Residuals:
Min 1Q Median 3Q Max
-5.0320 -1.4162 -0.0538 1.2921 9.8889
Coefficients:
Estimate Std. Error t value Pr(>|t|)
(Intercept) 2.808773 16.112577 0.174 0.86213
Price -0.053419 0.061540 -0.868 0.38842
EngineSize 1.334181 1.321805 1.009 0.31638
Horsepower 0.005006 0.024953 0.201 0.84160
RPM 0.001108 0.001215 0.912 0.36489
Rev.per.mile 0.002806 0.001249 2.247 0.02790 *
Fuel.tank.capacity -0.639270 0.262526 -2.435 0.01752 *
Length -0.059862 0.065583 -0.913 0.36459
Wheelbase 0.330572 0.156614 2.111 0.03847 *
Width 0.233123 0.265710 0.877 0.38338
Turn.circle 0.026695 0.197214 0.135 0.89273
Rear.seat.room -0.031404 0.182166 -0.172 0.86364
Luggage.room 0.206758 0.188448 1.097 0.27644
Weight -0.008001 0.002849 -2.809 0.00648 **
---
Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
Residual standard error: 2.835 on 68 degrees of freedom
(11 observations deleted due to missingness)
Multiple R-squared: 0.7533, Adjusted R-squared: 0.7062
F-statistic: 15.98 on 13 and 68 DF, p-value: 7.201e-16
```

In the previous multiple linear regression model using all the independent variables, we are predicting the mileage per gallon **MPG.Overall** variable. From the model summary, it is observed that few independent variables are statistically significant at 95% confidence level. The coefficient of determination R^2 which is known as the goodness of fit of a regression model is 75.33%. This implies that 75.33% variation in the dependent variable is explained by all the independent variables together. The formula for computing multiple R^2 is given next:

$$R^2 = 1 - \frac{\sum\left(\hat{y}_i - y_i\right)}{\sum\left(y_i - \overline{y}_i\right)^2} \ldots\ldots (4)$$

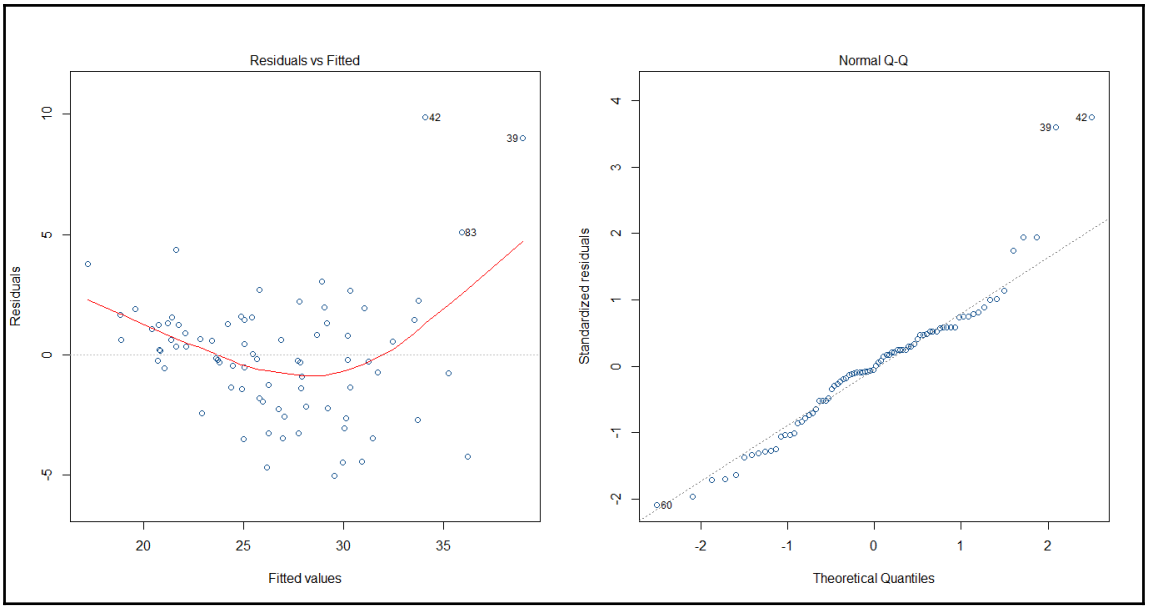
The preceding equation 4 calculated the coefficient of determination or percentage of variance explained by the regression model. The base line score to qualify for a good regression model is at least 80% as R square value; any R square value more than 80% is considered to be very good regression model. Since the R^2 is now less than 80%, we need to perform a few diagnostic tests on the regression results.

The estimated beta coefficients of the model:

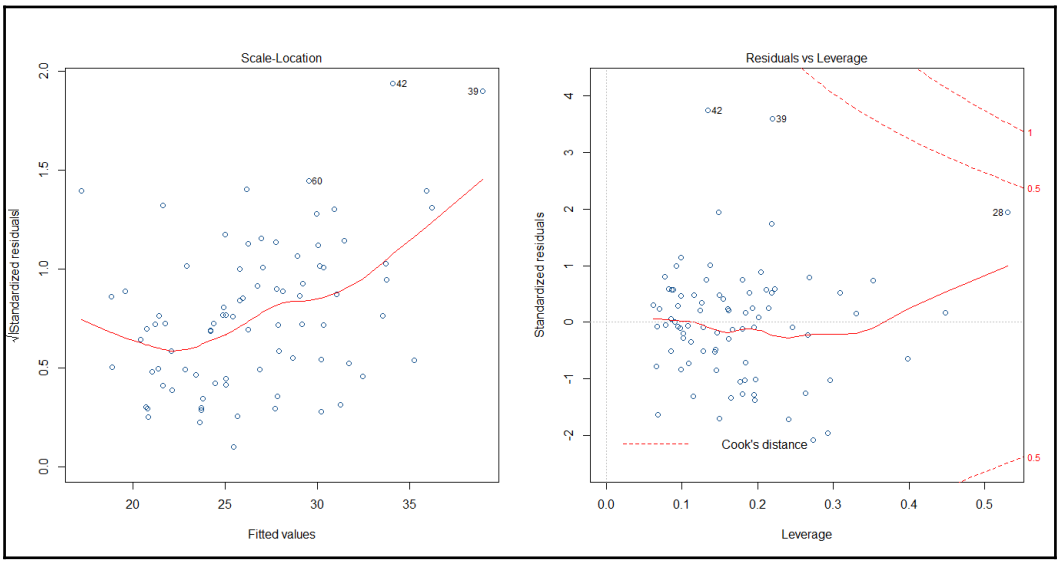
```
> #estimated coefficients
> fit$coefficients
(Intercept) Price EngineSize Horsepower
2.808772930 -0.053419142 1.334180881 0.005005690
RPM Rev.per.mile Fuel.tank.capacity Length
0.001107897 0.002806093 -0.639270186 -0.059861997
Wheelbase Width Turn.circle Rear.seat.room
0.330572119 0.233123382 0.026694571 -0.031404262
Luggage.room Weight
0.206757968 -0.008001444
#residual values
fit$residuals
#fitted values from the model
fit$fitted.values
#what happened to NA
fit$na.action
> #ANOVA table from the model
> summary.aov(fit)
Df Sum Sq Mean Sq F value Pr(>F)
Price 1 885.7 885.7 110.224 7.20e-16 ***
EngineSize 1 369.3 369.3 45.959 3.54e-09 ***
Horsepower 1 37.4 37.4 4.656 0.03449 *
RPM 1 38.8 38.8 4.827 0.03143 *
Rev.per.mile 1 71.3 71.3 8.877 0.00400 **
Fuel.tank.capacity 1 147.0 147.0 18.295 6.05e-05 ***
Length 1 1.6 1.6 0.203 0.65392
Wheelbase 1 35.0 35.0 4.354 0.04066 *
Width 1 9.1 9.1 1.139 0.28969
Turn.circle 1 0.5 0.5 0.060 0.80774
Rear.seat.room 1 0.6 0.6 0.071 0.79032
Luggage.room 1 9.1 9.1 1.129 0.29170
Weight 1 63.4 63.4 7.890 0.00648 **
Residuals 68 546.4 8.0
---
Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
11 observations deleted due to missingness
```

From the preceding **analysis of variance** (**ANOVA**) table, it is observed that the variables `Length`, `Width`, turn circle, rear seat room, and luggage room are not statistically significant at 5% level of alpha. ANOVA shows the source of variation in dependent variable from any independent variable. It is important to apply ANOVA after the regression model to know which independent variable has significant contribution to the dependent variable:

```
> #visualizing the model statistics
> par(mfrow=c(1,2))
> plot(fit, col="dodgerblue4")
```



The residual versus fitted plot shows the randomness in residual values as we move along the fitted line. If the residual values display any pattern against the fitted values, the error term is not probably normal. The residual versus fitted graph indicates that there is no pattern that exists among the residual terms; the residuals are approximately normally distributed. Residual is basically that part of the model which cannot be explained by the model. There are few influential data points, for example, 42nd, 39th, and 83rd as it is shown on the preceding graph, the normal quantile-quantile plot indicates except few influential data points all other standardized residual points follow a normal distribution. The straight line is the zero residual line and the red line is the pattern of residual versus fitted relationship. The scale versus location graph and the residuals versus leverage graph also validate the same observation that the residual term has no trend. The scale versus location plot takes the square root of the standardized residuals and plots it against the fitted values:



The confidence interval of the model coefficients at 95% confidence level can be calculated using the following code and also the prediction interval at 95% confidence level for all the fitted values can be calculated using the code below. The formula to compute the confidence interval is the model coefficient +/- standard error of model coefficient * 1.96:

```
> confint(fit,level=0.95)
2.5 % 97.5 %
(Intercept) -2.934337e+01 34.960919557
Price -1.762194e-01 0.069381145
EngineSize -1.303440e+00 3.971801771
Horsepower -4.478638e-02 0.054797758
RPM -1.315704e-03 0.003531499
Rev.per.mile 3.139667e-04 0.005298219
Fuel.tank.capacity -1.163133e+00 -0.115407347
Length -1.907309e-01 0.071006918
Wheelbase 1.805364e-02 0.643090600
Width -2.970928e-01 0.763339610
Turn.circle -3.668396e-01 0.420228772
Rear.seat.room -3.949108e-01 0.332102243
Luggage.room -1.692837e-01 0.582799642
Weight -1.368565e-02 -0.002317240
> head(predict(fit,interval="predict"))
fit lwr upr
1 31.47382 25.50148 37.44615
2 24.99014 18.80499 31.17528
3 22.09920 16.09776 28.10064
```

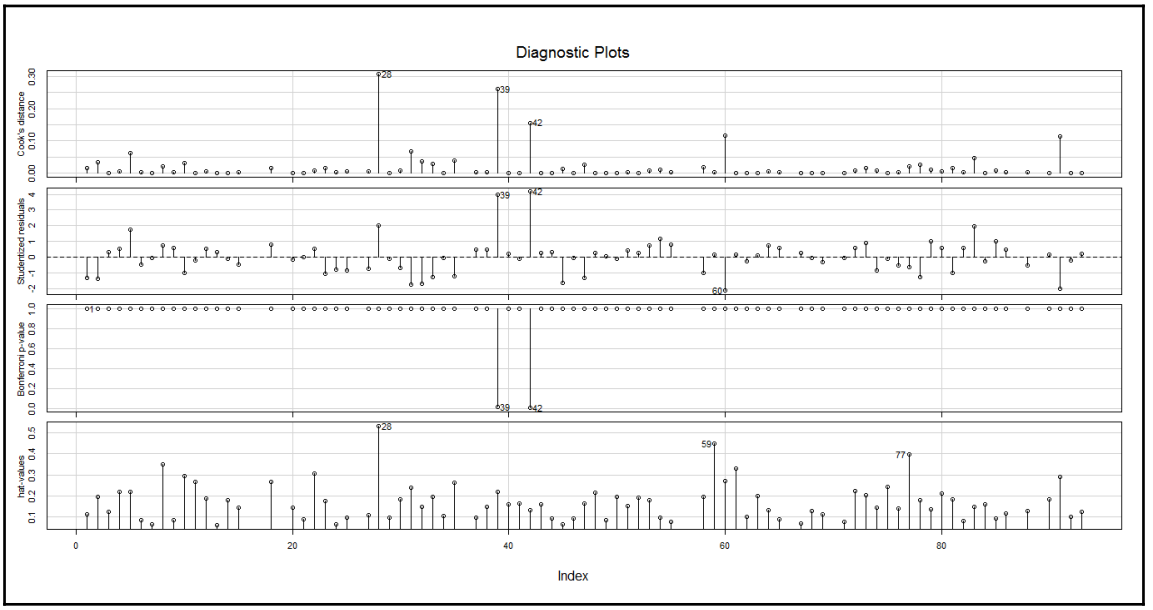
```
4 21.19989 14.95606 27.44371
5 21.62425 15.37929 27.86920
6 27.89137 21.99947 33.78328
```

The existence of influential data points or outlier values may deviate the model result, hence identification and curation of outlier data points in the regression model is very important:

```
# Deletion Diagnostics
influence.measures(fit)
```

The function belongs to the `stats` library, which computes some of the regression diagnostics for linear and generalized models. Any observation labeled with a star (`*`) implies an influential data point, that can be removed to make the model good:

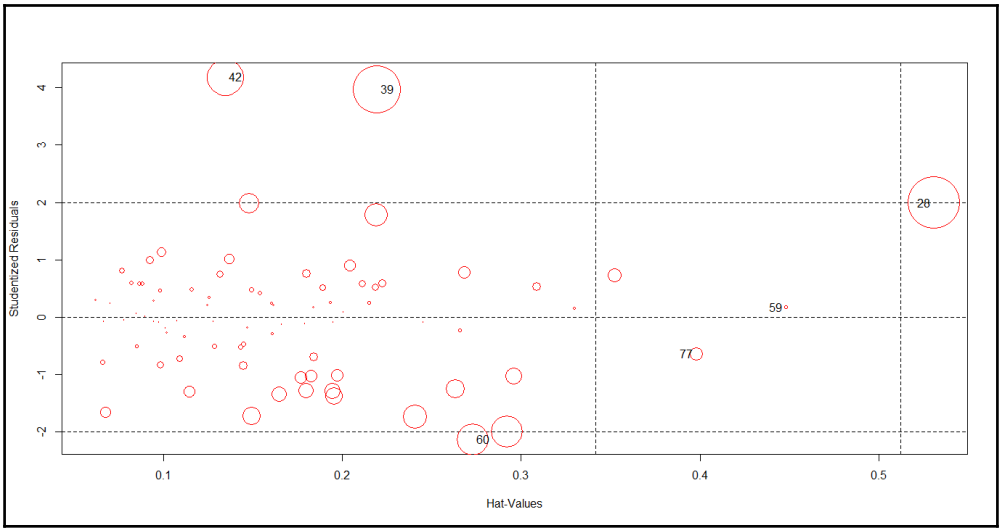
```
# Index Plots of the influence measures
influenceIndexPlot(fit, id.n=3)
```



The influential data points are highlighted by their position in the dataset. In order to understand more about these influential data points, we can write the following command. The size of the red circle based on the cook's distance value indicates the order of influence. Cook's distance is a statistical method to identify data points which have more influence than other data points.

Generally, these are data points that are distant from other points in the data, either for the dependent variable or one or more independent variables:

```
> # A user friendly representation of the above
> influencePlot(fit,id.n=3, col="red")
StudRes Hat CookD
28 1.9902054 0.5308467 0.55386748
39 3.9711522 0.2193583 0.50994280
42 4.1792327 0.1344866 0.39504695
59 0.1676009 0.4481441 0.04065691
60 -2.1358078 0.2730012 0.34097909
77 -0.6448891 0.3980043 0.14074778
```



If we remove these influential data points from our model, we can see some improvement in the model goodness of fit and overall error for the model can be reduced. Removing all the influential points at one go is not a good idea; hence we will take a step-by-step approach to deleting these influential data points from the model and monitor the improvement in the model statistics:

```
> ## Regression after deleting the 28th observation
> fit.1<-lm(MPG.Overall~., data=Cars93_1[-28,])
> summary(fit.1)
Call:
lm(formula = MPG.Overall ~ ., data = Cars93_1[-28, ])
Residuals:
Min 1Q Median 3Q Max
-5.0996 -1.7005 0.4617 1.4478 9.4168
```

```
Coefficients:
Estimate Std. Error t value Pr(>|t|)
(Intercept) -6.314222 16.425437 -0.384 0.70189
Price -0.014859 0.063281 -0.235 0.81507
EngineSize 2.395780 1.399569 1.712 0.09156 .
Horsepower -0.022454 0.028054 -0.800 0.42632
RPM 0.001944 0.001261 1.542 0.12789
Rev.per.mile 0.002829 0.001223 2.314 0.02377 *
Fuel.tank.capacity -0.640970 0.256992 -2.494 0.01510 *
Length -0.065310 0.064259 -1.016 0.31311
Wheelbase 0.407332 0.158089 2.577 0.01219 *
Width 0.204212 0.260513 0.784 0.43587
Turn.circle 0.071081 0.194340 0.366 0.71570
Rear.seat.room -0.004821 0.178824 -0.027 0.97857
Luggage.room 0.156403 0.186201 0.840 0.40391
Weight -0.008597 0.002804 -3.065 0.00313 **
---
Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
Residual standard error: 2.775 on 67 degrees of freedom
(11 observations deleted due to missingness)
Multiple R-squared: 0.7638, Adjusted R-squared: 0.718
F-statistic: 16.67 on 13 and 67 DF, p-value: 3.39e-16
```

Regression output after deleting the most influential data point provides a significant improvement in the R2 value from 75.33% to 76.38%. Let's repeat the same activity and see the model's results:

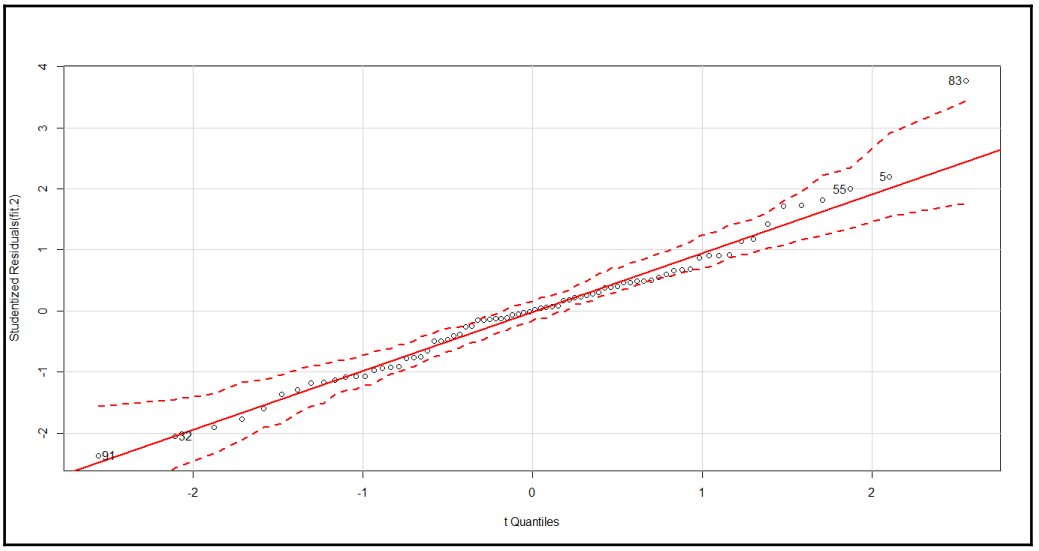
```
> ## Regression after deleting the 28,39,42,59,60,77 observations
> fit.2<-lm(MPG.Overall~., data=Cars93_1[-c(28,42,39,59,60,77),])
> summary(fit.2)
Call:
lm(formula = MPG.Overall ~ ., data = Cars93_1[-c(28, 42, 39,
59, 60, 77), ])
Residuals:
Min 1Q Median 3Q Max
-3.8184 -1.3169 0.0085 0.9407 6.3384
Coefficients:
Estimate Std. Error t value Pr(>|t|)
(Intercept) 21.4720002 13.3375954 1.610 0.11250
Price -0.0459532 0.0589715 -0.779 0.43880
EngineSize 2.4634476 1.0830666 2.275 0.02641 *
Horsepower -0.0313871 0.0219552 -1.430 0.15785
RPM 0.0022055 0.0009752 2.262 0.02724 *
Rev.per.mile 0.0016982 0.0009640 1.762 0.08307 .
Fuel.tank.capacity -0.6566896 0.1978878 -3.318 0.00152 **
Length -0.0097944 0.0613705 -0.160 0.87372
Wheelbase 0.2298491 0.1288280 1.784 0.07929 .
Width -0.0877751 0.2081909 -0.422 0.67477
```

```
Turn.circle 0.0347603 0.1513314 0.230 0.81908
Rear.seat.room -0.2869723 0.1466918 -1.956 0.05494 .
Luggage.room 0.1828483 0.1427936 1.281 0.20514
Weight -0.0044714 0.0021914 -2.040 0.04557 *
---
Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
Residual standard error: 2.073 on 62 degrees of freedom
(11 observations deleted due to missingness)
Multiple R-squared: 0.8065, Adjusted R-squared: 0.7659
F-statistic: 19.88 on 13 and 62 DF, p-value: < 2.2e-16
```

Looking at the preceding output, we can conclude that the removal of outliers or influential data points adds robustness to the R-square value. Now we can conclude that 80.65% of the variation in the dependent variable is being explained by all the independent variables in the model.

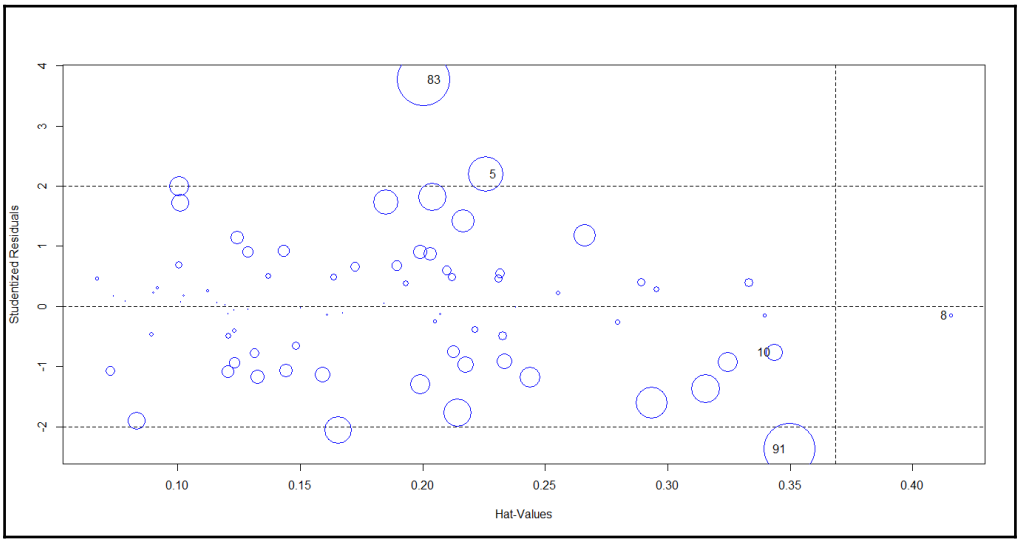
Now, using studentised residuals against t-quantiles in a Q-Q plot, we can identify some more outliers data points, if any, to boost the model accuracy:

```
> # QQ plots of studentized residuals, helps identify outliers
> qqPlot(fit.2, id.n=5)
91 32 55 5 83
1 2 74 75 76
```



From the preceding Q-Q plot, it seems that the data is fairly normally distributed, except few outlier values such as 83rd, 91st, 32nd, 55th, and 5th data points:

```
> ## Diagnostic Plots ###
> influenceIndexPlot(fit.2, id.n=3)
> influencePlot(fit.2, id.n=3, col="blue")
StudRes Hat CookD
5 2.1986298 0.2258133 0.30797166
8 -0.1448259 0.4156467 0.03290507
10 -0.7655338 0.3434515 0.14847534
83 3.7650470 0.2005160 0.45764995
91 -2.3672942 0.3497672 0.44770173
```



The problem of multicollinearity, that is, the correlation between predictor variables, needs to be verified for the final regression model. **Variance Inflation Factor** (**VIF**) is the measure typically used to estimate the multicollinearity. The formula to compute the VIF is $1/(1-R^2)$. Any independent variable having a VIF value of more than 10 indicates multicollinearity; hence such variables need to be removed from the model. Deleting one variable at a time and then again checking the VIF for the model is the best way to do this:

```
> ### Variance Inflation Factors
> vif(fit.2)
Price EngineSize Horsepower RPM
4.799678 20.450596 18.872498 5.788160
Rev.per.mile Fuel.tank.capacity Length Wheelbase
3.736889 5.805824 15.200301 11.850645
```

```
Width Turn.circle Rear.seat.room Luggage.room
10.243223 4.006895 2.566413 2.935853
Weight
24.977015
```

Variable weight has the maximum VIF, hence deleting the variable make sense. After removing the variable, let's rerun the model and calculate the VIF:

```
## Regression after deleting the weight variable
fit.3<-lm(MPG.Overall~ Price+EngineSize+Horsepower+RPM+Rev.per.mile+
Fuel.tank.capacity+Length+Wheelbase+Width+Turn.circle+
Rear.seat.room+Luggage.room, data=Cars93_1[-c(28,42,39,59,60,77),])
summary(fit.3)
> vif(fit.3)
Price EngineSize Horsepower RPM
4.575792 20.337181 15.962349 5.372388
Rev.per.mile Fuel.tank.capacity Length Wheelbase
3.514992 4.863944 14.574352 11.013850
Width Turn.circle Rear.seat.room Luggage.room
10.240036 3.965132 2.561947 2.935690
## Regression after deleting the Enginesize variable
fit.4<-lm(MPG.Overall~ Price+Horsepower+RPM+Rev.per.mile+
Fuel.tank.capacity+Length+Wheelbase+Width+Turn.circle+
Rear.seat.room+Luggage.room, data=Cars93_1[-c(28,42,39,59,60,77),])
summary(fit.4)
vif(fit.4)
## Regression after deleting the Length variable
fit.5<-lm(MPG.Overall~ Price+Horsepower+RPM+Rev.per.mile+
Fuel.tank.capacity+Wheelbase+Width+Turn.circle+
Rear.seat.room+Luggage.room, data=Cars93_1[-c(28,42,39,59,60,77),])
summary(fit.5)
> vif(fit.5)
Price Horsepower RPM Rev.per.mile
4.419799 8.099750 2.595250 3.232048
Fuel.tank.capacity Wheelbase Width Turn.circle
4.679088 8.231261 7.953452 3.357780
Rear.seat.room Luggage.room
2.393630 2.894959
The coefficients from the final regression model:
> coefficients(fit.5)
(Intercept) Price Horsepower RPM
29.029107454 -0.089498236 -0.014248077 0.001071072
Rev.per.mile Fuel.tank.capacity Wheelbase Width
0.001591582 -0.769447316 0.130876817 -0.047053999
Turn.circle Rear.seat.room Luggage.room
-0.072030390 -0.240332275 0.216155256
```

Now we can conclude that there is no multicollinearity in the preceding regression model. We can write the final multiple linear regression equation as follows:

$$MPG.Overall = 29.03 - 0.09 * Price - 0.014 * Horsepower + 0.001 * RPM + 0.001 * Rev.per.mile - 0.769 * Fuel.tank.capacity + 0.131 * Wheelbase + 0.047 * Width$$

The estimated model parameters can be interpreted as, for one unit change in the price variable, the MPG overall is predicted to change by 0.09 unit. Likewise, the estimated model coefficients can be interpreted for all the other independent variables. If we know the values of these independent variables, we can predict the likely value of the dependent variable MPG overall by using the previous equation.

Stepwise regression method for variable selection

In stepwise regression method, a base regression model is formed using the OLS estimation method. Thereafter, a variable is added or subtracted to/from the base model depending upon the **Akaike Information Criteria** (**AIC**) value. The standard rule is that a minimum AIC would guarantee a best fit in comparison to other methods. Taking the `Cars93_1.csv` file, let's create a multiple linear regression model by using the stepwise method. There are three different ways to test out the best model using the step formula:

- Forward method
- Backward method
- Both

In the forward selection method, initially a null model is created and variable is added to see any improvement in the AIC value. The independent variables is added to the null model till there is an improvement in the AIC value. In backward selection method, the complete model is considered to be the base model. An independent variable is deducted from the model and the AIC value is checked. If there is an improvement in AIC, the variable would be removed; the process will go on till the AIC value becomes minimum. In both the methods, alternatively forward and back selection method is used to identify the most relevant model. Let's look at the model result at first iteration:

```
> #base model
> fit<-lm(MPG.Overall~.,data=Cars93_1)
> #stepwise regression
> model<-step(fit,method="both")
```

```
Start: AIC=183.52
MPG.Overall ~ Price + EngineSize + Horsepower + RPM + Rev.per.mile +
Fuel.tank.capacity + Length + Wheelbase + Width + Turn.circle +
Rear.seat.room + Luggage.room + Weight
Df Sum of Sq RSS AIC
- Turn.circle 1 0.147 546.54 181.54
- Rear.seat.room 1 0.239 546.64 181.56
- Horsepower 1 0.323 546.72 181.57
- Price 1 6.055 552.45 182.43
- Width 1 6.185 552.58 182.45
- RPM 1 6.686 553.08 182.52
- Length 1 6.695 553.09 182.52
- EngineSize 1 8.186 554.58 182.74
- Luggage.room 1 9.673 556.07 182.96
<none> 546.40 183.52
- Wheelbase 1 35.799 582.20 186.73
- Rev.per.mile 1 40.565 586.96 187.40
- Fuel.tank.capacity 1 47.646 594.04 188.38
- Weight 1 63.400 609.80 190.53
```

In the beginning, AIC is **183.52**. In the final model, AIC is `175.51` and there are six independent variables, as shown next:

```
Step: AIC=175.51
MPG.Overall ~ EngineSize + RPM + Rev.per.mile + Fuel.tank.capacity +
Wheelbase + Width + Luggage.room + Weight
Df Sum of Sq RSS AIC
- Luggage.room 1 8.976 568.78 174.82
- Width 1 12.654 572.46 175.34
<none> 559.81 175.51
- EngineSize 1 14.022 573.83 175.54
- RPM 1 19.422 579.23 176.31
- Wheelbase 1 28.477 588.28 177.58
- Rev.per.mile 1 37.873 597.68 178.88
- Fuel.tank.capacity 1 52.516 612.32 180.86
- Weight 1 135.462 695.27 191.28
```

In the preceding table, "none" entry denotes the stage which is the final model, where further tuning or variable reduction is not possible. Hence, dropping more variables after that point would not make sense.

Logistic regression

The linear regression model based on the ordinary least square method assumes that the relationship between the dependent variable and the independent variables is linear; however, the logistic regression model assumes the relationship to be logarithmic. There are many real-life scenarios where the variable of interest is categorical in nature, such as buying a product or not, approving a credit card or not, tumor is cancerous or not, and so on. Logistic regression not only predicts a dependent variable class but it predicts the probability of a case belonging to a level in the dependent variable. The independent variables need not be normally distributed and need not have equal variance. Logistic regression belongs to the family of generalized linear regression. If the dependent variable has two levels then logistic regression can be applied, but if it has more than two levels, such as high, medium, and low, then multinomial logistic regression model can be applied. All the independent variables can be continuous, categorical, or nominal.

The logistic regression model can be explained using the following equation:

$$\ln\left[\frac{P(Y)}{1-P(Y)}\right] = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_3 ..... \beta_n X_n) + \varepsilon .....(6)$$

Ln (P(Y)/1-P(Y)) is the log odds of the outcome. The beta coefficients mentioned in the previous equation explain how the log odds of the outcome variable increase or decrease for every one unit increase or decrease in the explanatory variable.

Let's take the `Artpiece.csv` dataset where the dependent variable is a good purchase or not need to be predicted based on the independent variables.

For a logistic regression model, the dependent variable needs to be a binary variable with two levels. If it is not then the first task is to convert the dependent variable to binary. For the independent variables it is important to check the type of the variable as well as the levels. If some independent variables are categorical then binary conversion (creating dummy variables for each category) is required. So here you go with data conversion in necessary format for the logistic regression model:

```
> #data conversion
> Artpiece$IsGood.Purchase<-as.factor(Artpiece$IsGood.Purchase)
> Artpiece$Is.It.Online.Sale<-as.factor(Artpiece$Is.It.Online.Sale)
> #removing NA, Missing values from the data
> Artpiece<-na.omit(Artpiece)
> ## 75% of the sample size
> smp_size <- floor(0.75 * nrow(Artpiece))
```

```
> ## set the seed to make your partition reproductible
> set.seed(123)
> train_ind <- sample(seq_len(nrow(Artpiece)), size = smp_size)
> train <- Artpiece[train_ind, ]
> test <- Artpiece[-train_ind, ]
```

The `train` dataset is used for building the logistic regression model and the `test` dataset would be used for testing the model:

```
> #Logistic regression Model
> model1<-glm(IsGood.Purchase ~.,family=binomial(logit),data=train)
> #Model results/ components
> summary(model1)
Call:
glm(formula = IsGood.Purchase ~ ., family = binomial(logit),
data = train)
Deviance Residuals:
Min 1Q Median 3Q Max
-2.1596 -0.4849 -0.4096 -0.3401 3.8760
Coefficients:
Estimate Std. Error z value Pr(>|z|)
(Intercept) -6.734e-01 1.250e+00 -0.539 0.58995
Critic.Ratings 1.372e-01 1.119e-02 12.267 < 2e-16 ***
Acq.Cost -7.998e-06 1.867e-06 -4.284 1.83e-05 ***
CurrentAuctionAveragePrice -1.460e-05 1.360e-06 -10.731 < 2e-16 ***
Brush.Size1 -1.378e+00 1.246e+00 -1.107 0.26840
Brush.Size2 -1.684e+00 1.246e+00 -1.352 0.17647
Brush.Size3 -1.128e+00 1.252e+00 -0.902 0.36730
Brush.SizeNULL 1.599e+00 1.246e+00 1.283 0.19946
CollectorsAverageprice -2.420e-05 4.677e-06 -5.175 2.28e-07 ***
Is.It.Online.Sale1 -1.710e-01 9.621e-02 -1.777 0.07552 .
Min.Guarantee.Cost 9.830e-06 3.329e-06 2.953 0.00314 **
---
Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
(Dispersion parameter for binomial family taken to be 1)
Null deviance: 40601 on 54500 degrees of freedom
Residual deviance: 34872 on 54490 degrees of freedom
AIC: 34894
Number of Fisher Scoring iterations: 5
```

From the previous model result, all the independent variables are statistically significant at 5% level except the `brush` variable, which is a categorical variable. The model automatically creates dummy variables for each of the levels of a categorical variable. It takes the first level from the categorical variable and compares with other level, if the row belongs to that category then 1 else 0. 95% confidence interval for model coefficients and exponentiated model coefficients can be calculated using the following script. The left hand side equation contains natural logarithm to remove that we have to make exponentiation of the right

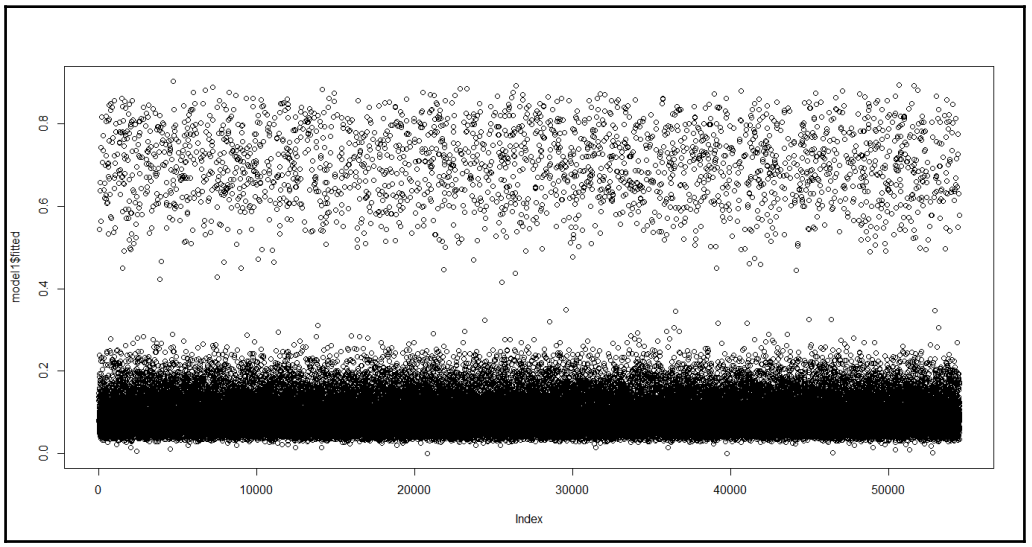
hand side expression which is the model coefficient:

```
> # 95% CI for exponentiated coefficients
> exp(confint(model1))
Waiting for profiling to be done...
2.5 % 97.5 %
(Intercept) 0.02301865 5.5660594
Critic.Ratings 1.12226719 1.1725835
Acq.Cost 0.99998833 0.9999957
CurrentAuctionAveragePrice 0.99998274 0.9999881
Brush.Size1 0.02326462 5.5558138
Brush.Size2 0.01714345 4.0948557
Brush.Size3 0.02953703 7.1850301
Brush.SizeNULL 0.45631813 109.1655370
CollectorsAverageprice 0.99996666 0.9999850
Is.It.Online.Sale1 0.69545357 1.0142229
Min.Guarantee.Cost 1.00000327 1.0000163
> confint(model1)
2.5 % 97.5 %
(Intercept) -3.771450e+00 1.716687e+00
Critic.Ratings 1.153509e-01 1.592094e-01
Acq.Cost -1.166650e-05 -4.348562e-06
CurrentAuctionAveragePrice -1.726348e-05 -1.193180e-05
Brush.Size1 -3.760822e+00 1.714845e+00
Brush.Size2 -4.066139e+00 1.409731e+00
Brush.Size3 -3.522110e+00 1.972000e+00
Brush.SizeNULL -7.845651e-01 4.692865e+00
CollectorsAverageprice -3.333576e-05 -1.500118e-05
Is.It.Online.Sale1 -3.631910e-01 1.412267e-02
Min.Guarantee.Cost 3.268054e-06 1.631710e-05
```

ANOVA can be performed on the results of the logistic regression model using chi-square test statistics. From the ANOVA which shows analysis of deviance in case of a logistic regression here, we can conclude that the deviance of model change is statistically significant when we include more independent variables in the model. The independent variables are added sequentially to the model and at every iteration the deviance of the error term is compared with the previous iteration to see if the addition or deletion of any variable is useful in reducing the error for the model:

```
> #ANOVA
> anova(model1,test="Chisq")
Analysis of Deviance Table
Model: binomial, link: logit
Response: IsGood.Purchase
Terms added sequentially (first to last)
Df Deviance Resid. Df Resid. Dev Pr(>Chi)
NULL 54500 40601
```

```
Critic.Ratings 1 354.6 54499 40246 < 2.2e-16 ***
Acq.Cost 1 477.8 54498 39768 < 2.2e-16 ***
CurrentAuctionAveragePrice 1 166.4 54497 39602 < 2.2e-16 ***
Brush.Size 4 4690.9 54493 34911 < 2.2e-16 ***
CollectorsAverageprice 1 27.3 54492 34884 1.773e-07 ***
Is.It.Online.Sale 1 3.3 54491 34880 0.067877 .
Min.Guarantee.Cost 1 8.6 54490 34872 0.003417 **
---
Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
>
> #Plotting the model
> plot(model1$fitted)
```



The preceding plot displays the fitted versus actual values (index) in a regression model. The predicted probability from the model can be extracted using the `link` option and the class prediction can be extracted using the `response` option, using the `predict` function:

```
> #Predicted Probability
> test$goodP<-predict(model1,newdata=test,type="response")
> test$goodL<-predict(model1,newdata=test,type="link")
```

Automatic logistic regression model selection can be done in three different ways using the stepwise regression method via AIC criteria:

- **Backward selection method**: All the independent variables are part of the initial model. Using Wald chi-square test of significance, variables with lowest value of chi-square or highest p-value such that it exceeds 0.05 (given that 95% confidence interval is used in the model) are removed from the model. The backward selection process of variable removal stops when it is not possible to find a variable that fits the Wald test criteria.
- **Forward selection method**: Model starts with a null model. A variable is entered into the model based on the lowest p-value or highest chi-square test statistic value. The process stops when there are no such cases found.
- **Both**: The third approach is to apply both backward selection and forward selection:

```
    > #auto detection of model
    > fit_step<-stepAIC(model1,method="both")
    Start: AIC=34893.93
    IsGood.Purchase ~ Critic.Ratings + Acq.Cost +
CurrentAuctionAveragePrice +
    Brush.Size + CollectorsAverageprice + Is.It.Online.Sale +
    Min.Guarantee.Cost
    Df Deviance AIC
    <none> 34872 34894
    - Is.It.Online.Sale 1 34875 34895
    - Min.Guarantee.Cost 1 34880 34900
    - Acq.Cost 1 34890 34910
    - CollectorsAverageprice 1 34898 34918
    - CurrentAuctionAveragePrice 1 34988 35008
    - Critic.Ratings 1 35025 35045
    - Brush.Size 4 39554 39568
```

The method selected for variable selection is both forward and backward. For example, in the first step forward selection was used, then at the same time backward section was also applied to check if the said variable could be removed from the model or not.

While applying the stepwise model selection procedure, it is very important to keep in mind the limitations of the stepwise method. Sometimes model overfitting happens, sometimes when you have a large number of predictors and mostly the predictors are highly correlated, then stepwise regression provides higher variability and low accuracy. So to address this issue, we can apply penalized regression method, which we are going to discuss in the next section:

```
> summary(fit_step)
Call:
glm(formula = IsGood.Purchase ~ Critic.Ratings + Acq.Cost +
CurrentAuctionAveragePrice +
Brush.Size + CollectorsAverageprice + Is.It.Online.Sale +
```

```
Min.Guarantee.Cost, family = binomial(logit), data = train)
Deviance Residuals:
Min 1Q Median 3Q Max
-2.1596 -0.4849 -0.4096 -0.3401 3.8760
Coefficients:
Estimate Std. Error z value Pr(>|z|)
(Intercept) -6.734e-01 1.250e+00 -0.539 0.58995
Critic.Ratings 1.372e-01 1.119e-02 12.267 < 2e-16 ***
Acq.Cost -7.998e-06 1.867e-06 -4.284 1.83e-05 ***
CurrentAuctionAveragePrice -1.460e-05 1.360e-06 -10.731 < 2e-16 ***
Brush.Size1 -1.378e+00 1.246e+00 -1.107 0.26840
Brush.Size2 -1.684e+00 1.246e+00 -1.352 0.17647
Brush.Size3 -1.128e+00 1.252e+00 -0.902 0.36730
Brush.SizeNULL 1.599e+00 1.246e+00 1.283 0.19946
CollectorsAverageprice -2.420e-05 4.677e-06 -5.175 2.28e-07 ***
Is.It.Online.Sale1 -1.710e-01 9.621e-02 -1.777 0.07552 .
Min.Guarantee.Cost 9.830e-06 3.329e-06 2.953 0.00314 **
---
Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
(Dispersion parameter for binomial family taken to be 1)
Null deviance: 40601 on 54500 degrees of freedom
Residual deviance: 34872 on 54490 degrees of freedom
AIC: 34894
Number of Fisher Scoring iterations: 5
```

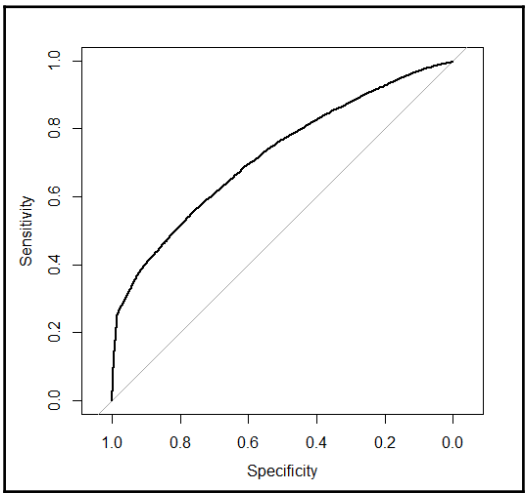
To check the multicollinearity among the independent variables, the VIF measure is used. The VIF computed for the step AIC-based final model is as follows:

```
> library(MASS);library(plyr);library(car)
>
> vif(fit_step)
GVIF Df GVIF^(1/(2*Df))
Critic.Ratings 1.204411 1 1.097457
Acq.Cost 2.582229 1 1.606932
CurrentAuctionAveragePrice 2.527691 1 1.589871
Brush.Size 1.094967 4 1.011405
CollectorsAverageprice 1.084788 1 1.041532
Is.It.Online.Sale 1.007080 1 1.003534
Min.Guarantee.Cost 1.163249 1 1.078540
```

The probability score for the `training` dataset is computed using the response option and the probability score can be used to display the results of classification on an AUC curve, also known as **Receiver Operating Characteristic** (**ROC**) curve:

```
> train$prob=predict(fit_step,type=c("response"))
> library(pROC)
> g <- roc(IsGood.Purchase ~ prob, data = train)
> g
```

```
Call:
roc.formula(formula = IsGood.Purchase ~ prob, data = train)
Data: prob in 47808 controls (IsGood.Purchase 0) < 6693 cases
(IsGood.Purchase 1).
Area under the curve: 0.7201
> plot(g)
```



The curve derived from the model shows the area under the curve. The horizontal axis shows the false positive rate and the vertical axis shows the true positive rate. If the area under the curve is more than 70%, then the model is considered to be a good model as per industry standard:

```
> #Label the prediction result above the certain threshold as Yes and No
> #Change the threshold value and check which give the better result.
> train$prob <- ifelse(prob > 0.5, "Yes", "No")
>
> #print the confusion matrix between the predicted and actual response on
testdata.
> t<-table(train$prob,train$IsGood.Purchase)
>
> #accuracy
> prop.table(t)
0 1
No 0.86407589 0.09229188
Yes 0.01311903 0.03051320
```

The threshold for classification table is 0.50 (50%) as probability. If someone wants to change the percentage of correct classified objects, then he/she can do so by putting a filter on the probability score. The following code is used to create a confusion matrix using a different probability value. From the previous classification table, it can be concluded that 90% people correctly classified.

Cubic regression

Cubic regression is another form of regression where the parameters in a linear regression model are increased up to one or two levels of polynomial calculation. Using the `Cars93_1.csv` dataset, let's understand the cubic regression:

```
> fit.6<-lm(MPG.Overall~ I(Price)^3+I(Horsepower)^3+I(RPM)^3+
+ Wheelbase+Width+Turn.circle, data=Cars93_1[-c(28,42,39,59,60,77),])
> summary(fit.6)
Call:
lm(formula = MPG.Overall ~ I(Price)^3 + I(Horsepower)^3 + I(RPM)^3 +
Wheelbase + Width + Turn.circle, data = Cars93_1[-c(28, 42,
39, 59, 60, 77), ])
Residuals:
Min 1Q Median 3Q Max
-5.279 -1.901 -0.006 1.590 8.433
Coefficients:
Estimate Std. Error t value Pr(>|t|)
(Intercept) 57.078025 12.349300 4.622 1.44e-05 ***
I(Price) -0.108436 0.065659 -1.652 0.1026
I(Horsepower) -0.024621 0.015102 -1.630 0.1070
I(RPM) 0.001122 0.000727 1.543 0.1268
Wheelbase -0.201836 0.079948 -2.525 0.0136 *
Width -0.104108 0.198396 -0.525 0.6012
Turn.circle -0.095739 0.158298 -0.605 0.5470
---
Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
Residual standard error: 2.609 on 80 degrees of freedom
Multiple R-squared: 0.6974, Adjusted R-squared: 0.6747
F-statistic: 30.73 on 6 and 80 DF, p-value: < 2.2e-16
> vif(fit.6)
I(Price) I(Horsepower) I(RPM) Wheelbase Width Turn.circle
4.121923 7.048971 2.418494 3.701812 7.054405 3.284228
> coefficients(fit.6)
(Intercept) I(Price) I(Horsepower) I(RPM) Wheelbase Width
57.07802478 -0.10843594 -0.02462074 0.00112168 -0.20183606 -0.10410833
Turn.circle
-0.09573848
```

Keeping all other parameters constant, one unit change in the independent variable would bring a multiplicative change in the dependent variable and the multiplication factor is the cube of the beta coefficients.

Penalized regression

The problem of maximum likelihood estimation comes into picture when we include large number of predictors or highly correlated predictors or both in a regression model and its failure to provide higher accuracy in regression problems gives rise to the introduction of penalized regression in data mining. The properties of maximum likelihood cannot be satisfying the regression procedure because of high variability and improper interpretation. To address the issue, most relevant subset selection came into picture. However, the subset selection procedure has some demerits. To again solve the problem, a new method can be introduced, which is popularly known as penalized maximum likelihood estimation method. There are two different variants of penalized regression that we are going to discuss here:

- **Ridge regression**: The ridge regression is known as *L2* quadratic penalty and the equation can be represented as follows:

$$f(\beta) = \sum_{j=1}^{n} \beta_j^2$$

  In comparison to the maximum likelihood estimates, the *L1* and *L2* method of estimation shrinks the beta coefficients towards zero. In order to avoid the problem of multicollinearity and higher number of predictors or both typically in a scenario when you have less number of observations, the shrinkage methods reduces the overfitted beta coefficient values.

  Taking the `Cars93_1.csv` dataset, we can test out the model and the results from it. The lambda parameter basically strikes the balance between penalty and fit of log likelihood function. The selection of lambda value is very critical for the model; if it is a small value then the model may overfit the data and high variance would become evident, and if the lambda chosen is very large value then it may lead to biased result.

```
> #installing the library
> library(glmnet)
```

```
> #removing the missing values from the dataset
> Cars93_2<-na.omit(Cars93_1)
> #independent variables matrix
> x<-as.matrix(Cars93_2[,-1])
> #dependent variale matrix
> y<-as.matrix(Cars93_2[,1])
> #fitting the regression model
> mod<-glmnet(x,y,family = "gaussian",alpha = 0,lambda = 0.001)
> #summary of the model
> summary(mod)
Length Class Mode
a0 1 -none- numeric
beta 13 dgCMatrix S4
df 1 -none- numeric
dim 2 -none- numeric
lambda 1 -none- numeric
dev.ratio 1 -none- numeric
nulldev 1 -none- numeric
npasses 1 -none- numeric
jerr 1 -none- numeric
offset 1 -none- logical
call 6 -none- call
nobs 1 -none- numeric
#Making predictions
pred<-predict(mod,x,type = "link")
#estimating the error for the model.
mean((y-pred)^2)
> #Making predictions
> pred<-predict(mod,x,type = "link")
> #estimating the error for the model.
> mean((y-pred)^2)
[1] 6.663406
```

Given the preceding background, the regression model was able to show up with 6.66% error, which implies the model is giving 93.34% accuracy. This is not the final iteration. The model needs to be tested couple of times and various subsamples as out of the bag need to be used to conclude the final accuracy of the model.

- **Least Absolute Shrinkage Operator (LASSO)**: The LASSO regression is known as L1 absolute value penalty (Tibshirani, 1997). The equation for LASSO regression can be expressed as follows:

$$f(\beta) = \sum_{j=1}^{n} |\beta|$$

Taking the `Cars93_1.csv` dataset, we can test out the model and the results from it. The lambda parameter basically strikes the balance between penalty and fit of log likelihood function. The selection of the lambda value is very critical for the model; if it is a small value then the model may over fit the data and high variance would become evident, and if the lambda chosen is very large value then it may lead to biased result. Let's try to implement the model and verify the results:

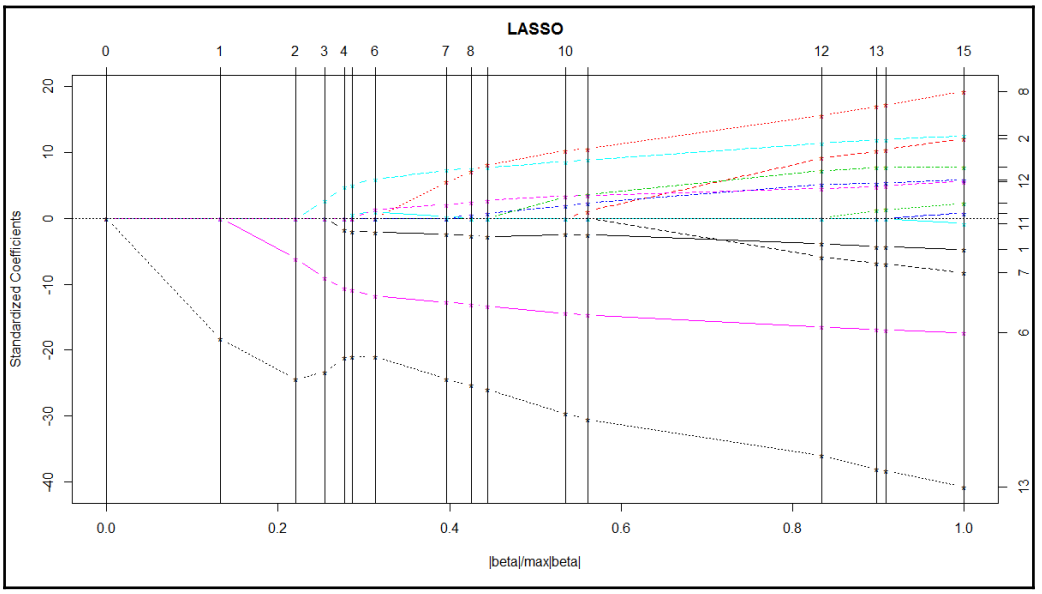
```
> #installing the library
> library(lars)
> #removing the missing values from the dataset
> Cars93_2<-na.omit(Cars93_1)
> #independent variables matrix
> x<-as.matrix(Cars93_2[,-1])
> #dependent variale matrix
> y<-as.matrix(Cars93_2[,1])
> #fitting the LASSO regression model
> model<-lars(x,y,type = "lasso")
> #summary of the model
> summary(model)
LARS/LASSO
Call: lars(x = x, y = y, type = "lasso")
Df Rss Cp
0 1 2215.17 195.6822
1 2 1138.71 63.7148
2 3 786.48 21.8784
3 4 724.26 16.1356
4 5 699.39 15.0400
5 6 692.49 16.1821
6 7 675.16 16.0246
7 8 634.59 12.9762
8 9 623.74 13.6260
9 8 617.24 10.8164
10 9 592.76 9.7695
11 10 587.43 11.1064
12 11 551.46 8.6302
13 12 548.22 10.2275
14 13 547.85 12.1805
15 14 546.40 14.0000
```

The type option within the **Least Angle Regression** (LARS) provides opportunities to apply various variants of the lars model with `"lasso"`, `"lar"`, `"forward.stagewise"`, or `"stepwise"`. Using these we can create various models and compare the results.

From the preceding output of the model we can see the RSS and CP along with degrees of freedom, at each iterations so we need to find out a best model step where the RSS for the model is minimum:

```
> #select best step with a minin error
> best_model<-model$df[which.min(model$RSS)]
> best_model
14
> #Making predictions
> pred<-predict(model,x,s=best_model,type = "fit")$fit
> #estimating the error for the model.
> mean((y-pred)^2)
[1] 6.685669
```

The best model is available at step 14, where the RSS is minimum and hence using that best step we can make predictions. Using the predict function at that step, we can generate predictions and the error for the model chosen is 6.68%. The visualization for the fitted model can be displayed as shown next. The vertical axis shows the standardized coefficients and the horizontal axis shows the model coefficient progression for the dependent variable. The *x* axis shows the ratio of absolute value of beta coefficients upon the max of the absolute value of the beta coefficients. The numerator is the estimated beta coefficient current and the denominator is the beta coefficient for the OLS model. When the shrinkage parameter in LASSO takes a value of zero, the model will correspond to a OLS regression method. As the penalized parameter increases the sum of absolute values of the beta coefficients pulled to zero:



In the preceding graph, the vertical bars indicate when a variable has been pulled to zero. Vertical bar corresponding to 15 shows there are 15 predictor variables need to be reduced to zero, which corresponds to higher penalized parameter lambda value 1.0. When lambda the penalized parameter is very small, the LASSO would approach the OLS regression and as lambda increases you will see fewer variables in your model. Hence, after lambda value of 1.0, there are no variables left in the model.

L1 and *L2* are known as regularized regression methods. *L1* cannot zero out regression coefficients; either you will get all the coefficients or none of them. However, *L2* does parameter shrinkage and variable selection automatically.

Summary

In this chapter, we discussed how to create linear regression, logistic regression, and other nonlinear regression based methods for predicting the outcome of a variable in a business scenario. Regression methods are basically very important in data mining projects, in order to create predictive models to know the future values of the unknown independent variables. We looked at various scenarios where we understood which type of model can be applied where. In the next chapter, we are going to talk about association rules or market basket analysis to understand important patterns hidden in a transactional database.

5
Market Basket Analysis with Groceries Data

Think about a scenario from a retailer or e-commerce store manager when it comes to recommending the right products to customers. Product recommendation is one important domain in data mining practice. Product recommendation can happen in three different ways: by associating customer's behavior with their purchase history, by associating items that are being purchased on each visit, and lastly by looking at the gross sales in each category and then using the retailer's past experience. In this chapter, we will be looking at the second method of product recommendation, popularly known as **Market Basket Analysis** (**MBA**) also known as association rules, which is by associating items purchased at transaction level and finding out the sub-segments of users having similar products and hence, recommending the products.

In this chapter, we will cover the following topics:

- What is MBA
- Where to apply MBA
- Assumptions/prerequisites
- Modeling techniques
- Limitations
- Practical project

Introduction to Market Basket Analysis

Is MBA limited to retail and e-commerce domain only? Now, let's think about the problems where MBA or association rules can be applied to get useful insights:

- In the healthcare sector, particularly medical diagnosis, as an example, having high blood pressure and being diabetic is a common combination. So it can be concluded that people having high blood pressure are more likely to be diabetic and vice versa. Hence, by analyzing the prevalence of medical conditions or diseases, it can be said what other diseases they may likely get in future.
- In retail and e-commerce, promotions and campaigns can be designed once a merchandiser knows the relationship between the purchase patterns of different items. MBA provides insights into the relationship between various items so that product recommendation can be designed.
- In banking and financial services also, MBA can be used. Taking into account the number of products a bank is offering to the customers, such as insurance policies, mutual funds, loans, and credit cards, is there any association between buying insurance policies and mutual funds? If yes, that can be explored using market basket analysis. Before recommending a product, the officer/agent should verify what all products go together so that the upsell and cross-sell of financial products can be designed.
- In the telecommunication domain, analysis of customer usage and selection of product options provide insights into effective designing of promotions and cross-selling of products.
- In unstructured data analysis, particularly in text mining, what words go together in which domain and how the terminologies are related and used by different people can be extracted using association rules.

What is MBA?

Market Basket Analysis is the study of relationships between various products and products that are purchased together or in a series of transactions. In standard data mining literature, the task of market basket analysis is to discover actionable insights in transactional databases. In order to understand MBA or **association rules** (**arules**), it is important to understand three concepts and their importance in deciding rules:

- **Support**: A transaction can contain a single item or a set of items. Consider the item `set x = {bread, butter, jam, curd};` the support of `x` implies the proportion of transactions when all four items are purchased together to the total number of transactions in the database:

Support of X = number of transactions involving X / total number of transactions

- **Confidence**: Confidence always refers to the confidence of a rule. It can be defined as confidence of *x => y*. Hence, *support (x U y)* implies the number of transactions in the database containing both item set `x` and item `y`:

Confidence of X => Y = support of (X U Y) / support of (X)

- **Lift**: Lift can be defined as the proportion of observed support to expected support:

*Lift of a rule X => Y = support of (X U Y) / support of (X) * support of (Y)*

An association rule stands supported in a transactional database if it satisfies the minimum support and minimum confidence criteria.

Let's take an example to understand the concepts.

The following is an example from the `Groceries.csv` dataset from the `arules` library in R:

Transaction ID	**Items**
001	Bread, banana, milk, butter
002	Milk, butter, banana
003	Curd, milk, banana
004	Curd, bread, butter
005	Milk, curd, butter, banana
006	Milk, curd, bread

The support of item set *X* where *X = {milk, banana} = proportion of transactions where milk and banana bought together; 4/6 = 0.67*, hence, the support of the *X* item set is *0.67*.

Let's assume *Y* is curd; the confidence of *X => Y*, which is *{milk, banana} => {curd}*, is the proportion of support *(X U Y) / support(X);* three products–milk, banana, and curd–are purchased together 2 times. Hence the confidence of the *rule = (2/6)/0.67 = 0.5*. This implies that 50% of the transactions containing milk and banana the rule are correct.

Confidence of a rule can be interpreted as a conditional probability of purchasing *Y* such that the item set *X* has already been purchased.

When we create too many rules in a transactional database, we need a measure to rank the rules; lift is a measure to rank the rules. From the preceding example, we can calculate the lift for the rule *X* to *Y*. Lift of *X => Y = support of (X U Y) / support of (X) * support of (Y) = 0.33/0.66*0.66 = 0.33/0.4356 = 0.7575*.

Where to apply MBA?

To understand the relationship between various variables in large databases, it is required to apply market basket analysis or association rules. It is the simplest possible way to understand the association. Though we have explained various industries where the concept of association rule can be applied, the practical implementation depends on the kind of dataset available and the length of data available. For example, if you want to study the relationship between various sensors integrated at a power generation plant to understand which all sensors are activated at a specific temperature level (very high), you may not find enough data points. This is because very high temperature is a rare event and to understand the relationship between various sensors, it is important to capture large data at that temperature level.

Data requirement

Product recommendation rules are generated from the results of the association rule model. For example, what is the customer going to buy next if he/she has already added a Coke, a chips packet, and a candle? To generate rules for product recommendation, we need frequent item sets and hence the retailer can cross-sell products. Therefore the input data format should be transactional, which in real-life projects sometimes happens and sometimes does not. If the data is not in transaction form, then a transformation needs to be done. In R, dataframe is a representation of mixed data. Can we transform the dataframe to transactional data so that we can apply association rules on that transformed dataset? This is because the algorithm requires that the input data format be transactional. Let's have a look at the methods to read transactions from a dataframe.

The `arules` library in R provides a function to read transactions from a dataframe. This function provides two options, single and basket. In single format, each line represents a single item; however, in basket format, each line represents a transaction comprising item levels and separated by a comma, space, or tab. For example:

```
> ## create a basket format
> data <- paste(
+ "Bread, Butter, Jam",
+ "Bread, Butter",
+ "Bread",
```

```
+ "Butter, Banana",
+ sep="\n")
> cat(data)
Bread, Butter, Jam
Bread, Butter
Bread
Butter, Banana
> write(data, file = "basket_format")
>
> ## read data
> library(arules)
> tr <- read.transactions("basket_format", format = "basket", sep=",")
> inspect(tr)
items
1 {Bread,Butter,Jam}
2 {Bread,Butter}
3 {Bread}
4 {Banana,Butter}
```

Now let's look at single format transactional data creation from a dataframe:

```
> ## create single format
> data <- paste(
+ "trans1 Bread",
+ "trans2 Bread",
+ "trans2 Butter",
+ "trans3 Jam",
+ sep ="\n")
> cat(data)
trans1 Bread
trans2 Bread
trans2 Butter
trans3 Jam
> write(data, file = "single_format")
>
> ## read data
> tr <- read.transactions("single_format", format = "single", cols =
c(1,2))
> inspect(tr)
items transactionID
1 {Bread} trans1
2 {Bread,Butter} trans2
3 {Jam} trans3
```

The preceding code explains how a typical piece of transactional data received in a tabular format, just like the spreadsheet format equivalent of R dataframe, can be converted to a transactional data format as required by the `arules` input data format.

Assumptions/prerequisites

Implementation of association rules for performing market basket analysis is based on some assumptions. Here are the assumptions:

- Assume that all data is categorical.
- There should not be any sparse data; the sparsity percentage should be minimum. Sparsity implies a lot of cells in a dataset with no values (with blank cells).
- The number of transactions required to find an interesting relationship between various products is a function of the number of items or products covered in the database. In other words, if you include more products with a less number of transactions, you will get higher sparsity in the dataset and vice versa.
- The higher the number of items you want to include in the analysis, the more number of transactions you will need to validate the rules.

Modeling techniques

There are two algorithms that are very popular for finding frequent item sets that exist in a retail transactional database:

- **Apriori algorithm**: It was developed by Agrawal and Srikant (1994). It considers the breadth first and provides counts of transactions.
- **Eclat algorithm**: This was developed by Zaki et. al. (1997b). It considers depth first; it provides intersection of transactions rather than count of transactions.

In this chapter, we are using the `Groceries.csv` dataset, which will be used in both the algorithms.

Limitations

Though association rules are a top choice among practitioners when it comes to understanding relationships in a large transactional database, they have certain inherent limitations:

- Association rule mining is a probabilistic approach in the sense that it computes the conditional probability of a product being purchased, such that a set of other items has already been purchased/added to the cart by the customer. It only estimates the likelihood of buying the product. The exact accuracy of the rule

may or may not be true.

- Association rule is a conditional probability estimation method using simple count of items as a measure.
- Association rules are not useful practically, even with a high support, confidence, and lift; this is because most of the times, trivial rules may sound genuine. There is no mechanism to filter out the trivial rules from the global set of rules but using a machine learning framework (which is out of the scope of this book).
- The algorithm provides a large set of rules; however, a few of them are important, and there is no automatic method to identify the number of useful rules.

Practical project

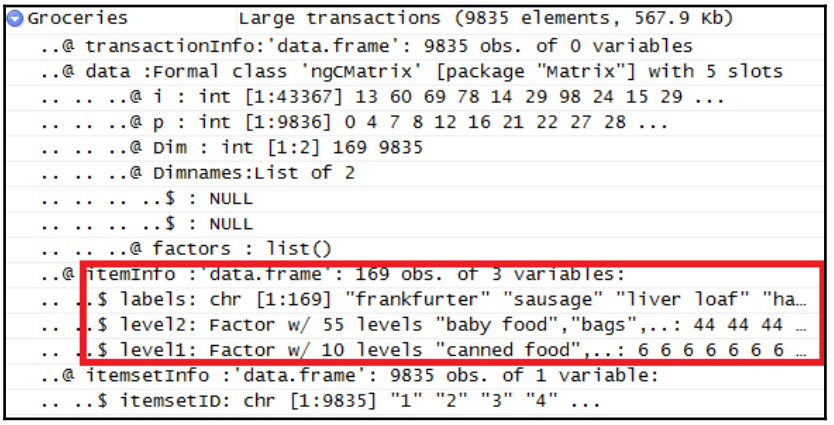
The `Groceries.csv` dataset that we are going to use comprises of 1 month of real-world **point-of-sale** (**POS**) transaction data from a grocery store. The dataset encompasses 9,835 transactions and there are 169 categories. Item sets are defined as a combination of items or products P_i *{i =1…..n}* that customers buy on the same visit. To put it in a simpler way, the item sets are basically the grocery bills that we usually get while shopping from a retail store. The bill number is considered the transaction number and the items mentioned in that bill are considered the market basket. A snapshot of the dataset is given as follows:

brown bread	soda	fruit/vegetable ju	canned beer	newspapers	shopping bags
yogurt	beverages	bottled water	specialty bar		
hamburger meat	other vegetables	rolls/buns	spices	bottled water	hygiene articles
root vegetables	other vegetables	whole milk	beverages	sugar	
pork	berries	other vegetables	whole milk	whipped/sour cre	artif. sweetener
beef	grapes	detergent			

The columns represent items and the rows represent transactions in the sample snapshot just displayed. Let's explore the dataset to understand the features:

```
> # Load the libraries
> library(arules)
> library(arulesViz)
> # Load the data set
> data(Groceries) #directly reading from library
> Groceries<-read.transactions("groceries.csv",sep=",") #reading from local
computer
```

The transactional dataframe contains three columns. The first column represents the name of the product/item, the second column represents the `level2` categorization of the products with 55 levels, and the third variable is the `level3` segment of the product with 10 categories. This screenshot makes it clearer:



The summary of a transactional data provides an idea about the total transactions existing in the database, number of items that are part of the database, sparsity of the data, and also the frequency of the top few items prominent in the transactional database:

```
summary(Groceries)
transactions as itemMatrix in sparse format with
9835 rows (elements/itemsets/transactions) and
169 columns (items) and a density of 0.02609146
most frequent items:
whole milk other vegetables rolls/buns soda yogurt
2513 1903 1809 1715 1372
(Other)
34055
```

There are 9,835 transactions in the dataset; out of 169 items, the most frequent items with their corresponding frequencies are previously represented. In a matrix of 9,835 cross 169, only `0.0261` or 2.61% of the cells are filled with values; the rest are empty. This 2.61% is the sparsity of the dataset.

Element (itemset/transaction) length distribution:

```
sizes
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17
2159 1643 1299 1005 855 645 545 438 350 246 182 117 78 77 55 46 29
18 19 20 21 22 23 24 26 27 28 29 32
```

```
14 14 9 11 4 6 1 1 1 1 3 1
Min. 1st Qu. Median Mean 3rd Qu. Max.
1.000 2.000 3.000 4.409 6.000 32.000
includes extended item information - examples:
labels
1 abrasive cleaner
2 artif. sweetener
3 baby cosmetics
```

From the preceding item set length distribution, it can be interpreted that there is one transaction with 32 products bought together. There are 3 transactions with 29 items purchased together. Likewise, we can interpret the frequency of other item sets in the dataset. This can be better understood using the item frequency plot. To know the first three transactions from the dataset, we can use the following command:

```
> inspect(Groceries[1:3])
items
1 {citrus fruit,semi-finished bread,margarine,ready soups}
2 {tropical fruit,yogurt,coffee}
3 {whole milk}
```

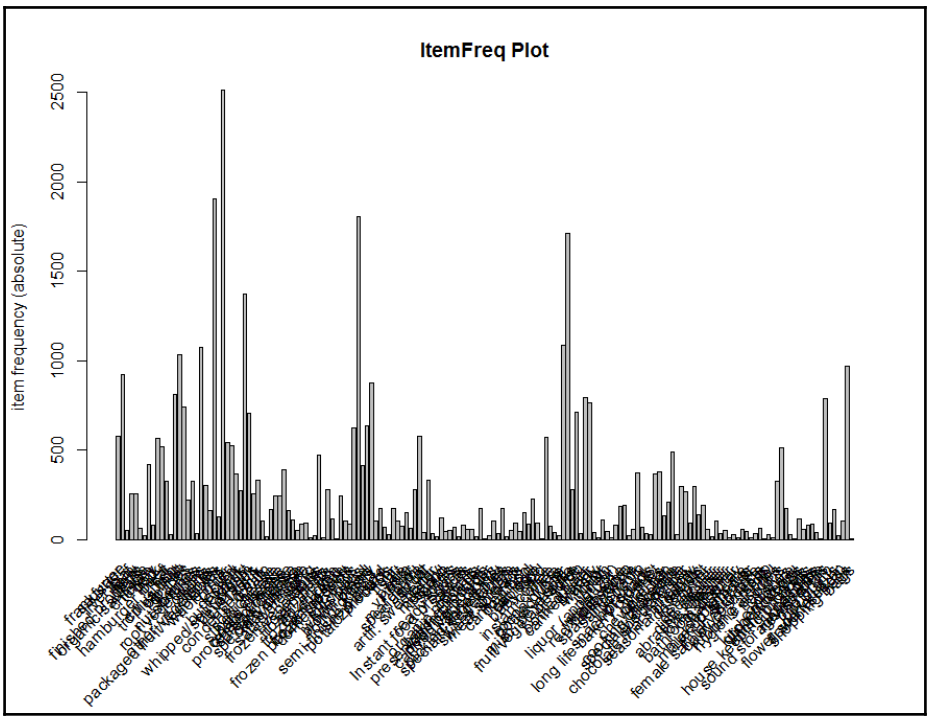
To know the proportion of transactions containing each item in the groceries database, we can use the following command. The first item, frankfurter, is available in 5.89% of the transactions; sausage is available in 9.39% of the transactions; and so on and so forth:

```
> cbind(itemFrequency(Groceries[,1:10])*100)
[,1]
frankfurter 5.8973055
sausage 9.3950178
liver loaf 0.5083884
ham 2.6029487
meat 2.5826131
finished products 0.6507372
organic sausage 0.2236909
chicken 4.2907982
turkey 0.8134215
pork 5.7651246
```

Two important parameters dictate the frequent item sets in a transactional dataset, support and confidence. The higher the support the lesser would be the number of rules in a dataset, and you would probably miss interesting relationships between various variables and vice versa. Sometimes with a higher support, confidence, and lift values also, it is not guaranteed to get useful rules. Hence, in practice, a user can experiment with different levels of support and confidence to arrive at meaningful rules. Even at sufficiently good amounts of minimum support and minimum confidence levels, the rules seem to be quite trivial.

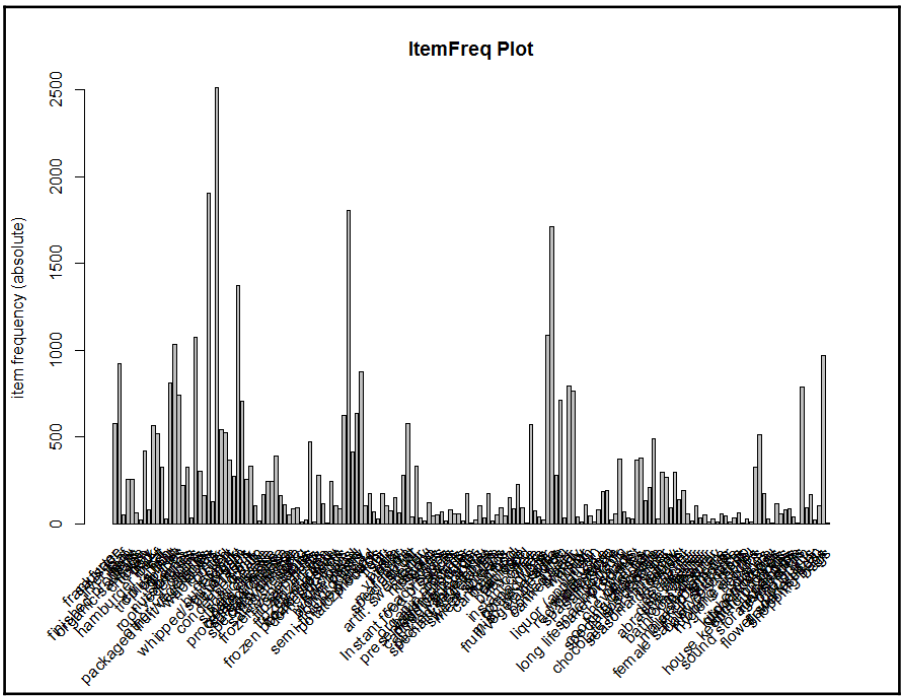
To address the issue of irrelevant rules in frequent item set mining, closed item set mining is considered as a method. An item set *X* is said to be closed if no superset of it has the same support as *X* and *X* is maximal at % support if no superset of *X* has at least % support:

```
> itemFrequencyPlot(Groceries, support=0.01, main="Relative ItemFreq Plot",
+ type="absolute")
```



In the preceding graph, the item frequency in the absolute count is shown with a minimum support of `0.01`. The following graph indicates the top 50 items with relative percentage count, showing what percentage of transactions in the database contain those items:

```
> itemFrequencyPlot(Groceries,topN=50,type="relative",main="Relative Freq
Plot")
```



Apriori algorithm

Apriori algorithm uses a downward closure property, which states that any subsets of a frequent item set and also frequent item sets. The apriori algorithm uses level-wise search for frequent item sets. This algorithm only creates rules with one item in the right-hand side of the equation (RHS), which is known as consequent; left-hand side of the equation (LHS) known as antecedent. This implies that rules with one item in RHS and blank LHS may appear as valid rules; to avoid these blank rules, the minimum length parameter needs to be changed from 1 to 2:

```
> # Get the association rules based on apriori algo
> rules <- apriori(Groceries, parameter = list(supp = 0.01, conf = 0.10))
Parameter specification:
confidence minval smax arem aval originalSupport support minlen maxlen
target ext
0.1 0.1 1 none FALSE TRUE 0.01 1 10 rules FALSE
Algorithmic control:
filter tree heap memopt load sort verbose
0.1 TRUE TRUE FALSE TRUE 2 TRUE
```

```
apriori - find association rules with the apriori algorithm
version 4.21 (2004.05.09) (c) 1996-2004 Christian Borgelt
set item appearances ...[0 item(s)] done [0.00s].
set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
sorting and recoding items ... [88 item(s)] done [0.00s].
creating transaction tree ... done [0.00s].
checking subsets of size 1 2 3 4 done [0.00s].
writing ... [435 rule(s)] done [0.00s].
creating S4 object ... done [0.00s].
```

There are 435 rules with a support value of 1% (proportion of items to be included in rule creation with a minimum of 1%) and confidence level of 10% using apriori algorithm. There are 88 items representing those 435 rules. Using the `summary` command, we can get to know the length of rules and the distribution of rules:

```
> summary(rules)
set of 435 rules
rule length distribution (lhs + rhs):sizes
1 2 3
8 331 96
Min. 1st Qu. Median Mean 3rd Qu. Max.
1.000 2.000 2.000 2.202 2.000 3.000
summary of quality measures:
support confidence lift
Min. :0.01007 Min. :0.1007 Min. :0.7899
1st Qu.:0.01149 1st Qu.:0.1440 1st Qu.:1.3486
Median :0.01454 Median :0.2138 Median :1.6077
Mean :0.02051 Mean :0.2455 Mean :1.6868
3rd Qu.:0.02115 3rd Qu.:0.3251 3rd Qu.:1.9415
Max. :0.25552 Max. :0.5862 Max. :3.3723
mining info:
data ntransactions support confidence
Groceries 9835 0.01 0.1
```

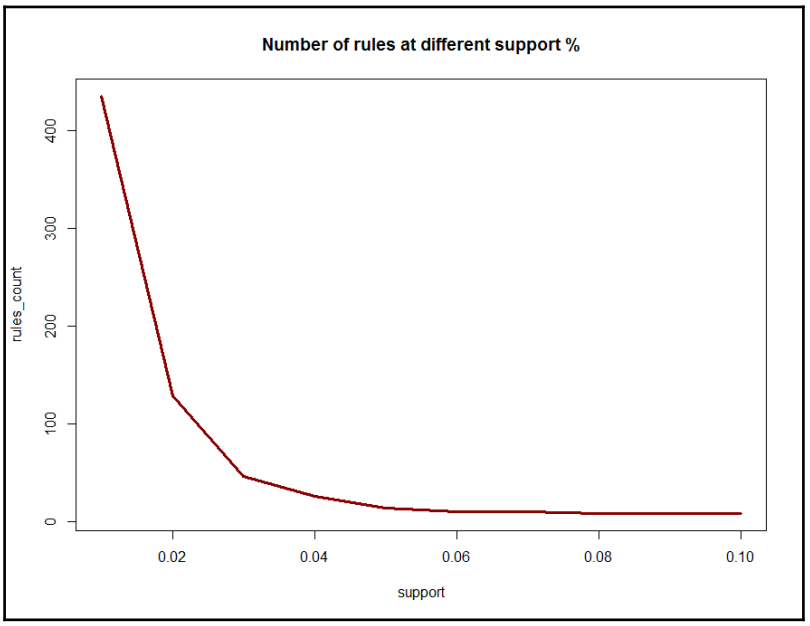
By looking at the summary of the rules, there are 8 rules with 1 item, including LHS and RHS, which are not valid rules. Those are given as follows; to avoid blank rules, the minimum length needs to be two. After using a minimum length of two, the total number of rules decreased to 427:

```
> inspect(rules[1:8])
lhs rhs support confidence lift
1 {} => {bottled water} 0.1105236 0.1105236 1
2 {} => {tropical fruit} 0.1049314 0.1049314 1
3 {} => {root vegetables} 0.1089985 0.1089985 1
4 {} => {soda} 0.1743772 0.1743772 1
5 {} => {yogurt} 0.1395018 0.1395018 1
6 {} => {rolls/buns} 0.1839349 0.1839349 1
```

```
7 {} => {other vegetables} 0.1934926 0.1934926 1
8 {} => {whole milk} 0.2555160 0.2555160 1
> rules <- apriori(Groceries, parameter = list(supp = 0.01, conf = 0.10,
minlen=2))
Parameter specification:
confidence minval smax arem aval originalSupport support minlen maxlen
target ext
0.1 0.1 1 none FALSE TRUE 0.01 2 10 rules FALSE
Algorithmic control:
filter tree heap memopt load sort verbose
0.1 TRUE TRUE FALSE TRUE 2 TRUE
apriori - find association rules with the apriori algorithm
version 4.21 (2004.05.09) (c) 1996-2004 Christian Borgelt
set item appearances ...[0 item(s)] done [0.00s].
set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
sorting and recoding items ... [88 item(s)] done [0.00s].
creating transaction tree ... done [0.00s].
checking subsets of size 1 2 3 4 done [0.00s].
writing ... [427 rule(s)] done [0.00s].
creating S4 object ... done [0.00s].
> summary(rules)
set of 427 rules
rule length distribution (lhs + rhs):sizes
2 3
331 96
```

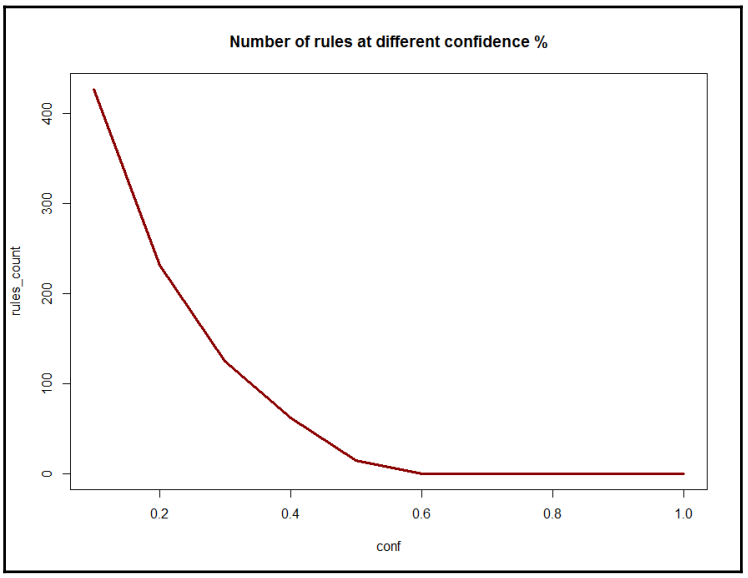
How do we identify what is the right set of rules? After squeezing the confidence level, we can reduce the number of valid rules. Keeping the confidence level at 10% and changing the support level, we can see how the number of rules is changing. If we have too many rules, it is difficult to implement them; if we have small number of rules, it will not correctly represent the hidden relationship between the items. Hence, having right set of valid rules is a trade-off between support and confidence. The following scree plot shows the number of rules against varying levels of support, keeping the confidence level constant at 10%:

```
> support<-seq(0.01,0.1,0.01)
> support
[1] 0.01 0.02 0.03 0.04 0.05 0.06 0.07 0.08 0.09 0.10
> rules_count<-c(435,128,46,26,14, 10, 10,8,8,8)
> rules_count
[1] 435 128 46 26 14 10 10 8 8 8
> plot(support,rules_count,type = "l",main="Number of rules at different
support %",
+ col="darkred",lwd=3)
```



Looking at the support 0.04 and confidence level 10%, the right number of valid rules for the dataset would be 26, based on the scree-plot result. The reverse can happen too to identify relevant rules; keeping the support level constant and varying the confidence level, we can create another scree plot:

```
> conf<-seq(0.10,1.0,0.10)
> conf
[1] 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
> rules_count<-c(427,231,125,62,15,0,0,0,0,0)
> rules_count
[1] 427 231 125 62 15 0 0 0 0 0
> plot(conf,rules_count,type = "l",main="Number of rules at different
confidence %",
+ col="darkred",lwd=3)
```



Looking at the confidence level of `0.5` and confidence level changing by 10%, the right number of valid rules for the groceries dataset would be 15, based on the scree plot result.

Eclat algorithm

Eclat algorithm uses simple intersection operations for homogenous class clustering with a bottom-up approach. The same code can be re-run using the `eclat` function in R and the result can be retrieved. The `eclat` function accepts two arguments, support and maximum length:

```
> rules_ec <- eclat(Groceries, parameter = list(supp = 0.05))
parameter specification:
tidLists support minlen maxlen target ext
FALSE 0.05 1 10 frequent itemsets FALSE
algorithmic control:
sparse sort verbose
7 -2 TRUE
eclat - find frequent item sets with the eclat algorithm
version 2.6 (2004.08.16) (c) 2002-2004 Christian Borgelt
create itemset ...
set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
sorting and recoding items ... [28 item(s)] done [0.00s].
creating sparse bit matrix ... [28 row(s), 9835 column(s)] done [0.00s].
```

```
writing ... [31 set(s)] done [0.00s].
Creating S4 object ... done [0.00s].
> summary(rules_ec)
set of 31 itemsets
most frequent items:
whole milk other vegetables yogurt rolls/buns frankfurter
4 2 2 2 1
(Other)
23
element (itemset/transaction) length distribution:sizes
1 2
28 3
Min. 1st Qu. Median Mean 3rd Qu. Max.
1.000 1.000 1.000 1.097 1.000 2.000
summary of quality measures:
support
Min. :0.05236
1st Qu.:0.05831
Median :0.07565
Mean :0.09212
3rd Qu.:0.10173
Max. :0.25552
includes transaction ID lists: FALSE
mining info:
data ntransactions support
Groceries 9835 0.05
```

Using the eclat algorithm, with a support of 5%, there are 31 rules that explain the relationship between different items. The rule includes items that have a representation in at least 5% of the transactions.

While generating a product recommendation, it is important to recommend those rules that have high confidence level, irrespective of support proportion. Based on confidence, the top 5 rules can be derived as follows:

```
> #sorting out the most relevant rules
> rules<-sort(rules, by="confidence", decreasing=TRUE)
> inspect(rules[1:5])
lhs rhs support confidence lift
36 {other vegetables,yogurt} => {whole milk} 0.02226741 0.5128806 2.007235
10 {butter} => {whole milk} 0.02755465 0.4972477 1.946053
3 {curd} => {whole milk} 0.02613116 0.4904580 1.919481
33 {root vegetables,other vegetables} => {whole milk} 0.02318251 0.4892704
1.914833
34 {root vegetables,whole milk} => {other vegetables} 0.02318251 0.4740125
2.449770
```

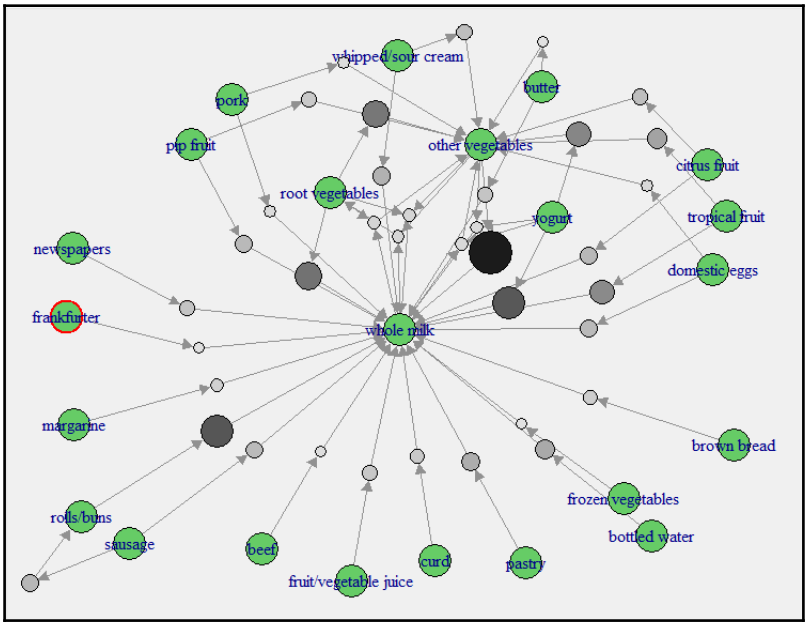
Rules also can be sorted based on lift and support proportion, by changing the argument in the sort function. The top 5 rules based on lift calculation are as follows:

```
> rules<-sort(rules, by="lift", decreasing=TRUE)
> inspect(rules[1:5])
lhs rhs support confidence lift
35 {other vegetables,whole milk} => {root vegetables} 0.02318251 0.3097826
2.842082
34 {root vegetables,whole milk} => {other vegetables} 0.02318251 0.4740125
2.449770
27 {root vegetables} => {other vegetables} 0.04738180 0.4347015 2.246605
15 {whipped/sour cream} => {other vegetables} 0.02887646 0.4028369 2.081924
37 {whole milk,yogurt} => {other vegetables} 0.02226741 0.3974592 2.054131
```

Visualizing association rules

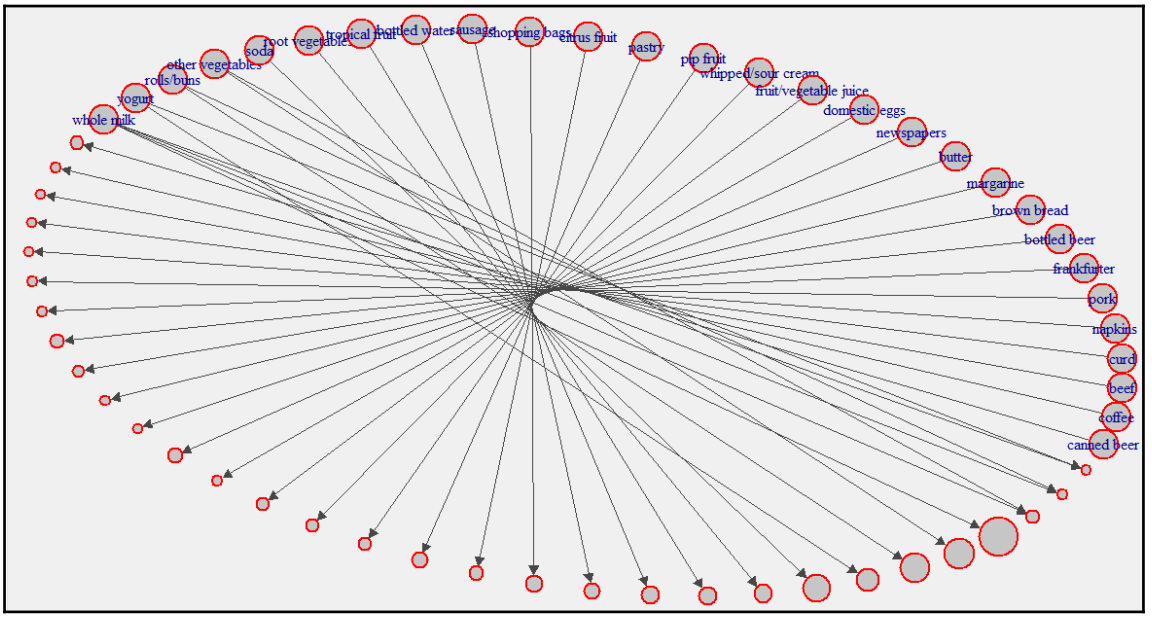
How the items are related and how the rules can be visually represented is as much important as creating the rules:

```
> #visualizign the rules
> plot(rules,method='graph',interactive = T,shading = T)
```



The preceding graph is created using apriori algorithm. In maximum number of rules, at least you would find either whole milk or other vegetables as those two items are well connected by nodes with other items.

Using eclat algorithm for the same dataset, we have created another set of rules; the following graph shows the visualization of the rules:



Implementation of arules

Once a good market basket analysis or arules model is built, the next task is to integrate the model. arules provides a PMML interface, which is a predictive modeling markup language interface, to integrate with other applications. Other applications can be other statistical software such as, SAS, SPSS, and so on; or it can be Java, PHP, and Android-based applications. The PMML interface makes it easier to integrate the model. When it comes to rule implementation, two important questions a retailer would like to get answer are:

- What are customers likely to buy before buying a product?
- What are the customers likely to buy if they have already purchased some product?

Let's take a product `yogurt`, and the retailer would like to recommend this to customers. Which are the rules that can help the retailer? So the top 5 rules are as follows:

```
> rules<-apriori(data=Groceries, parameter=list(supp=0.001,conf = 0.8),
+ appearance = list(default="lhs",rhs="yogurt"),
+ control = list(verbose=F))
> rules<-sort(rules, decreasing=TRUE,by="confidence")
> inspect(rules[1:5])
lhs rhs support confidence
4 {root vegetables,butter,cream cheese } => {yogurt} 0.001016777 0.9090909
10 {tropical fruit,whole milk,butter,sliced cheese} => {yogurt} 0.001016777
0.9090909
11 {other vegetables,curd,whipped/sour cream,cream cheese } => {yogurt}
0.001016777 0.9090909
13 {tropical fruit,other vegetables,butter,white bread} => {yogurt}
0.001016777 0.9090909
2 {sausage,pip fruit,sliced cheese} => {yogurt} 0.001220132 0.8571429
lift
4 6.516698
10 6.516698
11 6.516698
13 6.516698
2 6.144315
> rules<-apriori(data=Groceries, parameter=list(supp=0.001,conf =
0.10,minlen=2),
+ appearance = list(default="rhs",lhs="yogurt"),
+ control = list(verbose=F))
> rules<-sort(rules, decreasing=TRUE,by="confidence")
> inspect(rules[1:5])
lhs rhs support confidence lift
20 {yogurt} => {whole milk} 0.05602440 0.4016035 1.571735
19 {yogurt} => {other vegetables} 0.04341637 0.3112245 1.608457
18 {yogurt} => {rolls/buns} 0.03436706 0.2463557 1.339363
15 {yogurt} => {tropical fruit} 0.02928317 0.2099125 2.000475
17 {yogurt} => {soda} 0.02735130 0.1960641 1.124368
```

Using the `lift` criteria also, product recommendation can be designed to offer to the customers:

```
> # sorting grocery rules by lift
> inspect(sort(rules, by = "lift")[1:5])
lhs rhs support confidence lift
1 {yogurt} => {curd} 0.01728521 0.1239067 2.325615
8 {yogurt} => {whipped/sour cream} 0.02074225 0.1486880 2.074251
15 {yogurt} => {tropical fruit} 0.02928317 0.2099125 2.000475
4 {yogurt} => {butter} 0.01464159 0.1049563 1.894027
11 {yogurt} => {citrus fruit} 0.02165735 0.1552478 1.875752
```

Finding out the subset of rules based on the availability of some item names can be done using the following code:

```
# finding subsets of rules containing any items
itemname_rules <- subset(rules, items %in% "item name")
inspect(itemname_rules[1:5])
> # writing the rules to a CSV file
> write(rules, file = "groceryrules.csv", sep = ",", quote = TRUE,
row.names = FALSE)
>
> # converting the rule set to a data frame
> groceryrules_df <- as(rules, "data.frame")
```

Summary

In this chapter, we discussed how to create association rules, what all factors determine the rules' existence, and how rules explain the underlying relationship between different items or variables. Also we looked at how we can get most compressed rules based on minimum support and confidence value. The objective of the association rules model was not to come up with rules but to implement the rules in business use cases for generating recommendation for cross-selling and upselling products, and designing campaign bundles based on association. The rules would provide necessary guidance for the store managers to place products and merchandising design in a retail setup. Having said this, in our next chapter, we are going to cover various methods of performing clustering for segmentation, which would provide more insights into product recommendation using clustering methods.

6
Clustering with E-commerce Data

In data mining literature, clustering plays an important role in bringing insights from a dataset that is actionable and provides important business directions. In simple language, clustering aims at bringing similar observations such as similar customers, similar patients, similar users, and so on. Clustering techniques are not limited to the domain of retail but can be extended to any domain. Segmentation is no more limited to the retail or e-commerce domain; it is also applicable to all domains and industries.

In this chapter, we are going to learn the following things:

- What is segmentation?
- How can clustering be applied to perform segmentation?
- What are the methods used for clustering?
- A comparative view of the various methods
- A practical project on segmentation

In this chapter, you will know the basics of segmentation using various clustering methods. There are different methods used to perform clustering. The following examples clarify where and how clustering can be used to create segments that can be used to drive business value:

- In the retail/e-commerce industry, it is not possible to understand the behavior of millions of customers to design the campaign, marketing strategy, and sales promotion engagements. These customers actually belong to a finite group of customers displaying similar behavior within a group and dissimilar behavior between groups.

- In the telecommunications industry, the positioning of towers to optimize user experience is driven by segmentation exercise. Also the design of telecom plans, such as custom offers (especially for students, older people, professionals, and so on), based on the usage of services are also made using clustering methods.
- In the healthcare industry, to decide the size of hospital beds for different departments, such as emergency care, acute care, and so on, clustering exercise helps. Also, reporting of various diseases across different states can be used to create a cluster of similar regions or places where a set of procedures can be adopted.
- In governance, law enforcement agencies strengthen their presence in those areas where crime evidence/instance is more. So clustering techniques are used to create groups of cities such as high, medium, and low-crime areas. Based on the data, they can align their forces accordingly.
- In insurance, clustering is used to decide differential premiums by grouping different cities based on parameters such as demographic information, geolocation, and risk zone.

Understanding customer segmentation

In standard data mining practice, customer segmentation is a way of dividing all customers into various subgroups relevant to the business and the business problem. That subgroup creation can be done either by using a subjective approach or by keeping the business objective in mind. In different industries, customer segmentation has different applications; for example, in retail, it is important to know the purchase behavior of customers, and different subgroups displaying unique purchase behavior is relevant to the business.

Why understanding customer segmentation is important

In the retail and e-commerce industry, offers, campaigns, loyalty programs, and discount strategies work based on the purchase behavior of the subgroup of customers. In other industries, sales, marketing strategy, and business plans run keeping in mind the customer's behavior, and the behavior drives the sales. That unique customer behavior one can be understood by performing segmentation on the data.

How to perform customer segmentation?

Having discussed various benefits of performing customer segmentation, it is important to know how to perform customer segmentation. There are two broad methods used to perform customer segmentation:

- Customer segmentation using clustering-based methods
- Customer segmentation using the **recency, frequency, and monetary** (**RFM**) model

In this chapter, we will discuss the clustering-based approach to perform customer segmentation; the second method is not in the scope of this chapter.

Various clustering methods available

All clustering algorithms can be classified into four groups, as follows:

- **Hierarchical clustering**: Using this method, a number of clusters are prepared from the dataset, forming a hierarchy, and the clusters are then grouped using a tree structure so that the entire dataset can be represented as a single tree structure. The advantage of this method is that unlike partitioning-based clustering, we don't have to specify the number of clusters to be created from the dataset.
- **Partitioning clustering**: In this approach, the dataset is divided into a finite group of clusters based on a distance function in such a way that the within-cluster similarity is maximum and between-cluster similarity is minimum. K-means, which is a popular method in data mining practice, falls under this category.
- **Model-based clustering**: Using the data model, a group of clusters is created from the dataset; the methods used in the model-based clustering approach can be divided into two groups:
    - Maximum likelihood estimation
    - Bayesian estimation
- **Other clustering algorithms**: There are other clustering techniques available such as fuzzy clustering, bagged clustering, and so on. Fuzzy clustering uses a fuzzy logic on the data so that the same data point can be a part of more than one cluster. In other words, there is no exclusivity in terms of cluster membership. This method is basically different from the other three forms of clustering, where a data point belongs to a single cluster. Another algorithm that falls under this group is known as **self-organizing maps** (**SOM**).

Before analyzing various methods used for clustering of similar objects or observations from a dataset, it is important to decide on a metric or distance method to measure the similarity or dissimilarity between the observations. Based on certain predefined characteristics known as input features, how one observation is different from or close to another observation can be known by using this metric.

A similarity metric measures the degree of closeness of data points and a distance metric measures how different the data points are. Both similarity and distance metric calculations are part of the clustering process. Let's take a look at various similarity/distance measures:

- **Euclidean distance**: The formula to compute the distance between two data points x and y is as follows:

$$Dist(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

- **Manhattan distance**: This is the formula used to compute the Manhattan distance:

$$Dist(x, y) = \sum_{i=1}^{n} |x_i - y_i|$$

- **Cosine similarity**: The formula used to compute the cosine similarity is as follows:

$$Sim(x, y) = \frac{\sum_{i=1}^{n} x_i y_i}{\sqrt{\sum_{i=1}^{n} x_i^2} * \sqrt{\sum_{i=1}^{n} y_i^2}}$$

Now let's start segmenting the dataset using various clustering methods by taking a dataset from healthcare (`PimaIndiansDiabetes.csv [ref: 1]`) and a dataset from retail/e-commerce (`Wholesalecustomers.csv [ref: 1]`). Before running a clustering exercise, data sufficiency conditions need to be checked.

K-means clustering

K-means clustering is an unsupervised learning algorithm that tries to combine similar objects into a group in such a way that the within-group similarity should be maximum and the between-group object similarity should be minimum. "Object" here implies the data points that are entered into an algorithm. The within-group similarity is computed based on a centrality measure called centroid (arithmetic mean) and a distance function that measures how close the objects within a group are to the center. The "K" in K-means clustering implies the number of clusters that a user might be interested in. The following steps need to be followed for k-means clustering. Since the mean is used as a measure of estimating the centroid, it is not free from the presence of extreme observations or outliers. Hence, it is required to check the presence of outliers in a dataset before running k-means clustering.

1. **Checking the presence of outliers**: Let's use the boxplot method of identifying the presence of any outliers in the `Wholesalecustomers.csv` dataset:

```
> r_df<-read.csv("Wholesalecustomers.csv")
> ####Data Pre-processing
> par(mfrow=c(2,3))
> apply(r_df[,c(-1,-2)],2,boxplot)
```

   The dotted points after the whisker line in each preceding boxplot for all the variables indicate that there are lots of outliers in the dataset. The capping method is mostly used by practitioners to remove the outliers from the dataset. If any data point exceeds the quantiles, 90th percentile, 95th percentile, or 99th percentile for each variable, then those data points need to be capped at the respective percentiles:

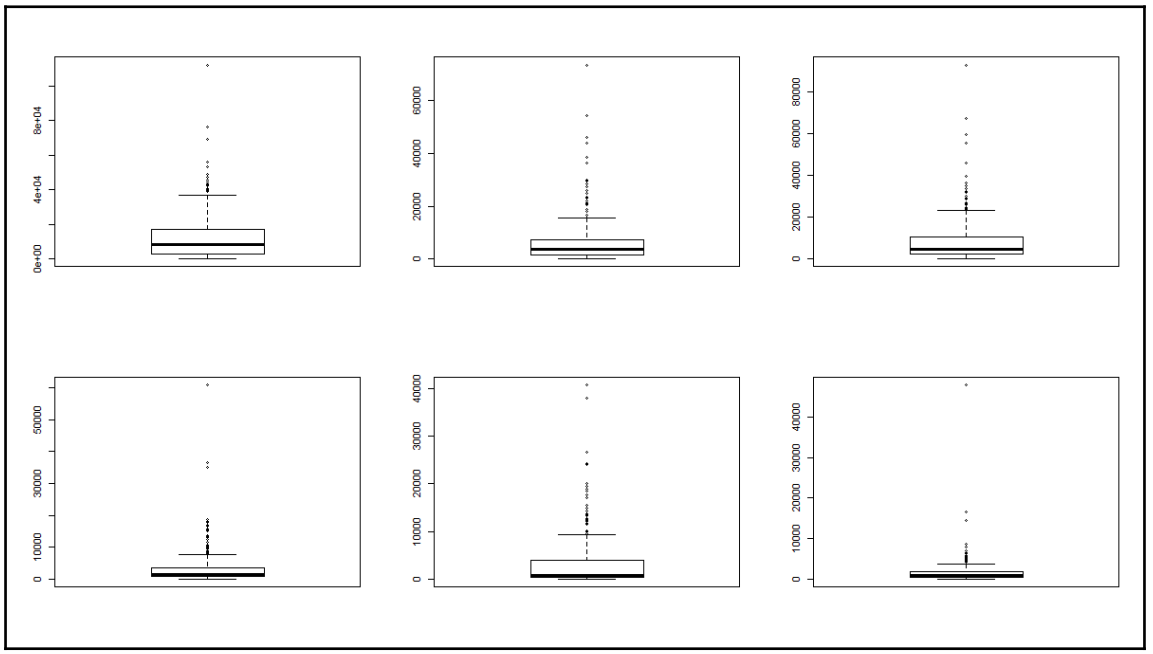
```
> #computing quantiles
> quant<-function(x){quantile(x,probs=c(0.95,0.90,0.99))}
> out1<-sapply(r_df[,c(-1,-2)],quant)
```

Now, this `quant` function computes the three percentiles and the same are applied to the dataset, excluding the first two variables. The following code is used to modify the variables:

```
    > #removing outliers
    > r_df$Fresh<-ifelse(r_df$Fresh>=out1[2,1],out1[2,1],r_df$Fresh)
    > r_df$Milk<-ifelse(r_df$Milk>=out1[2,2],out1[2,2],r_df$Milk)
    > r_df$Grocery<-ifelse(r_df$Grocery>=out1[2,3],out1[2,3],r_df$Grocery)
    > r_df$Frozen<-ifelse(r_df$Frozen>=out1[2,4],out1[2,4],r_df$Frozen)
    > r_df$Detergents_Paper<-
ifelse(r_df$Detergents_Paper>=out1[2,5],out1[2,5],r_df$Detergents_Paper)
    > r_df$Delicassen<-
ifelse(r_df$Delicassen>=out1[2,6],out1[2,6],r_df$Delicassen)
```

Next, let's check the dataset to find out whether the outliers have been removed, using the same boxplot method:

```
> #Checking outliers after removal
> apply(r_df[,c(-1,-2)],2,boxplot)
```



From the preceding graph, we can conclude that all the outliers have been removed by capping the values at 90th percentile value for all the variables.

2. **Scaling the dataset**: Inclusion of only continuous variables in the K-means clustering exercise does not guarantee that all the variables would be on the same scale. For example, the age of a customer ranges from 0-100 and the income of a customer ranges from 0-100,000. Since both units of measurement are different, they might fall into different clusters, which becomes more complex when we have more data points to be represented. In order to compare segments in a clustering process, all the variables need to be standardized on a scale; here we have applied standard normal transformation (Z-transformation) for all the variables:

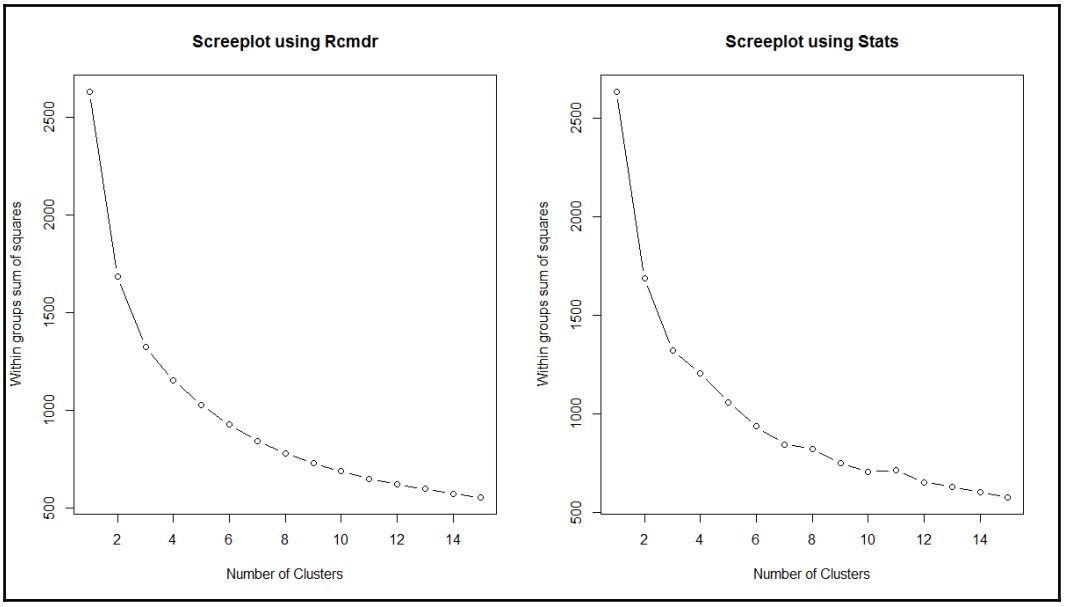
```
> r_df_scaled<-as.matrix(scale(r_df[,c(-1,-2)]))
> head(r_df_scaled)
Fresh Milk Grocery Frozen Detergents_Paper Delicassen
[1,] 0.2351056 1.2829174 0.11321460 -0.96232799 0.1722993 0.1570550
[2,] -0.4151579 1.3234370 0.45658819 -0.30625122 0.4120701 0.6313698
[3,] -0.4967305 1.0597962 0.13425842 -0.03373355 0.4984496 1.8982669
[4,] 0.3041643 -0.9430315 -0.45821928 1.66113143 -0.6670924 0.6443648
[5,] 1.3875506 0.1657331 0.05110966 0.60623797 -0.1751554 1.8982669
[6,] -0.1421677 0.9153464 -0.30338465 -0.77076036 -0.1681831 0.2794239
```

3. **Choose initial cluster seeds**: The selection of initial random seeds decides the number of iterations required for convergence of the model. Typically, initial cluster seeds chosen at random are temporary means of the clusters. The objects closer to the centroid measured by the distance function are assigned the cluster membership. With the addition of a new member into the cluster, again the centroid is computed and each seed value is replaced by the respective cluster centroid. In K-means clustering, this process of adding more objects to the group and hence updating the centroid goes on until the centroids are not moving and the objects are not changing the group membership.
4. **Decide the number of cluster K**: In the R programming language, there are two libraries, `Rcmdr` (R Commander) and `stats`, that support K-means clustering and they use two different approaches. The `Rcmdr` library needs to be loaded to run the algorithm, but stats is already built into base R. To compute the number of clusters required to represent the data using K-means algorithm is decided by the scree plot using the following formula:

```
> library(Rcmdr)
> > sumsq<-NULL
> #Method 1
> par(mfrow=c(1,2))
> for (i in 1:15) sumsq[i] <-
sum(KMeans(r_df_scaled,centers=i,iter.max=500,
num.seeds=50)$withinss)
>
```

```
> plot(1:15,sumsq,type="b", xlab="Number of Clusters",
+ ylab="Within groups sum of squares",main="Screeplot using Rcmdr")
>
> #Method 2
> for (i in 1:15) sumsq[i] <-
sum(kmeans(r_df_scaled,centers=i,iter.max=5000,
      algorithm = "Forgy")$withinss)
>
> plot(1:15,sumsq,type="b", xlab="Number of Clusters",
+ ylab="Within groups sum of squares",main="Screeplot using Stats")
```

The following graph explains the scree plot. Using both approaches, the ideal number of clusters is 4, decided by looking at the "elbow" point on the scree plot. The vertical axis shows the within-group sum of squares dropping as the cluster number increases on the horizontal axis:



Once the number of clusters is decided, the cluster centers and distance can be computed, and the following code explains the functions to use in K-means clustering. The objective here is to classify the objects in such a way that they are as much dissimilar as possible from one group to another group and as much similar as possible within each group. The following function can be used to compute the within-group sum of squares from the cluster centers to the individual objects:

$$Sumsq = \sum_{i=1}^{n} \left(x_i - c_i \right)^2$$

The sum of squares from each of the clusters is computed and then added up to compute the total within-group sum of squares for the dataset. The objective here is to minimize the within-group sum of squares total.

5. **Grouping based on minimum distance**: The K-means syntax and explanation for the arguments are as follows:

```
kmeans(x, centers, iter.max = 10, nstart = 1,
algorithm = c("Hartigan-Wong", "Lloyd", "Forgy",
"MacQueen"), trace=FALSE)
```

x	**Numeric matrix of data, or an object that can be coerced to such a matrix (such as a numeric vector or a data frame with all numeric columns)**
`centers`	Either the number of clusters, say *k*, or a set of initial (distinct) cluster centers. If it's a number, a random set of (distinct) rows in *x* is chosen as the initial centers.
`iter.max`	The maximum number of iterations allowed.
`nstart`	If the center is a number, how many random sets should be chosen?
`algorithm`	Character. This may be abbreviated. Note that `"Lloyd"` and `"Forgy"` are alternative names for one algorithm.

Table 1: The description for the K-means syntax

```
> #Kmeans Clustering
> library(cluster);library(cclust)
> set.seed(121)
> km<-kmeans(r_df_scaled,centers=4,nstart=17,iter.max=50000, algorithm
=
"Forgy",trace = T)
```

The scaled dataset is used to create clusters, with four clusters and 17 initial starting points. Given the 50,000 iterations with the Forgy algorithm, the following results are obtained from the model:

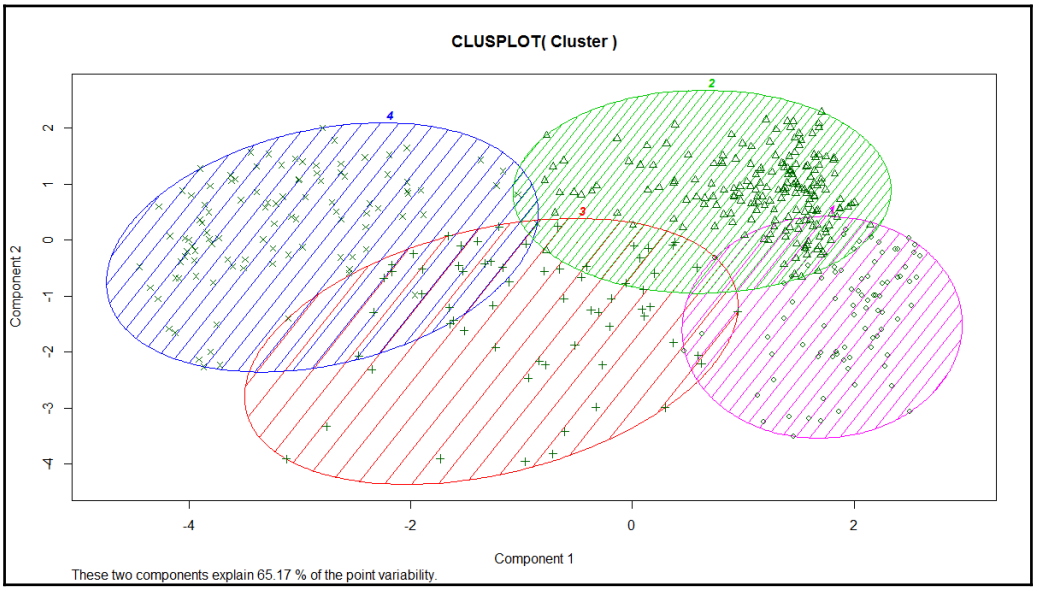
```
#checking results
summary(km)
```

```
Length Class Mode
cluster 440 -none- numeric
centers 24 -none- numeric
totss 1 -none- numeric
withinss 4 -none- numeric
tot.withinss 1 -none- numeric
betweenss 1 -none- numeric
size 4 -none- numeric
iter 1 -none- numeric
ifault 0 -none- NULL
> km$centers
Fresh Milk Grocery Frozen Detergents_Paper Delicassen
1 0.7272001 -0.4741962 -0.5839567 1.54228159 -0.64696856 0.13809763
2 -0.2327058 -0.6491522 -0.6275800 -0.44521306 -0.55388881 -0.57651321
3 0.6880396 0.6607604 0.3596455 0.02121206 -0.03238765 1.33428207
4 -0.6025116 1.1545987 1.3947616 -0.45854741 1.55904516 0.09763056
> km$withinss
[1] 244.3466 324.8674 266.3632 317.5866
```

The following code explains how to attach cluster information with the dataset and the profiling of various clusters with their respective group means:

```
> #attaching cluster information
> Cluster<-cbind(r_df,Membership=km$cluster)
> aggregate(Cluster[,3:8],list(Cluster[,9]),mean)
Group.1 Fresh Milk Grocery Frozen Detergents_Paper Delicassen
1 1 16915.946 2977.868 3486.071 6123.576 558.9524 1320.4940
2 2 8631.625 2312.926 3231.096 1434.122 799.2500 660.5957
3 3 16577.977 7291.414 9001.375 2534.643 2145.5738 2425.0954
4 4 5440.073 9168.309 15051.573 1402.660 6254.0670 1283.1252
```

Now, let's look at visualization of the groups/clusters using a scatter plot:



From the preceding graph, we can see that all the four clusters are overlapping each other to a certain extent; however, the overall objective of a clustering exercise is to come up with non-overlapping groups. There are six variables; hence, six dimensions represented in a two-dimensional space show 65.17% variability explained by two components. This is acceptable because the degree of overlap is less. If the degree of overlap is more, then post clustering, the similar groups can be merged as part of a profiling exercise. Profiling is an exercise performed after K-means clustering to generate insights for business users; hence, while generating insights, similar groups can be merged.

6. **Predicting cluster membership for new data**: Once the K-means clustering model is created, using that model, cluster membership for a new dataset can be created by using the following predict function:

```
> #Predicting new data for KMeans
> predict.kmeans <- function(km, r_df_scaled)
+ {k <- nrow(km$centers)
+ d <- as.matrix(dist(rbind(km$centers, r_df_scaled)))[-(1:k),1:k]
+ n <- nrow(r_df_scaled)
+ out <- apply(d, 1, which.min)
+ return(out)}
```

For the same dataset, we can predict the cluster membership, and the actual cluster membership versus the predicted cluster membership can be represented in the following table:

```
> #predicting cluster membership
> Cluster$Predicted<-predict.kmeans(km,r_df_scaled)
> table(Cluster$Membership,Cluster$Predicted)
1 2 3 4
1 84 0 0 0
2 0 188 0 0
3 0 0 65 0
4 0 0 0 103
```

7. **Implementing K-means in live applications**: For implementation of K-means clustering with any other statistical software and web applications, PMML script can be used. **PMML** stands for **Predictive Model Markup Language**, which is recognized as an industry standard for deploying predictive models:

```
#pmml code
library(pmml);
library(XML);
pmml(km)
```

When we save a model in PMML format, it prepares an XML script for the model where the model parameters are embedded in the script. It is written in such a way that any other external statistical software can read the model.

Hierarchical clustering

There are two different ways of implementing the hierarchical clustering algorithm, but one thing is common; both use a distance measure to estimate the similarity between cluster members:

- **Agglomerative method**: Bottom-up approach
- **Divisive method**: Top-down approach

To perform hierarchical clustering, the input data has to be in a distance matrix form. The method selected has to be from one of `"single"`, `"complete"`, `"average"`, `"mcquitty"`, `"ward.D"`, `"ward.D2"`, `"centroid"`, or `"median"`.

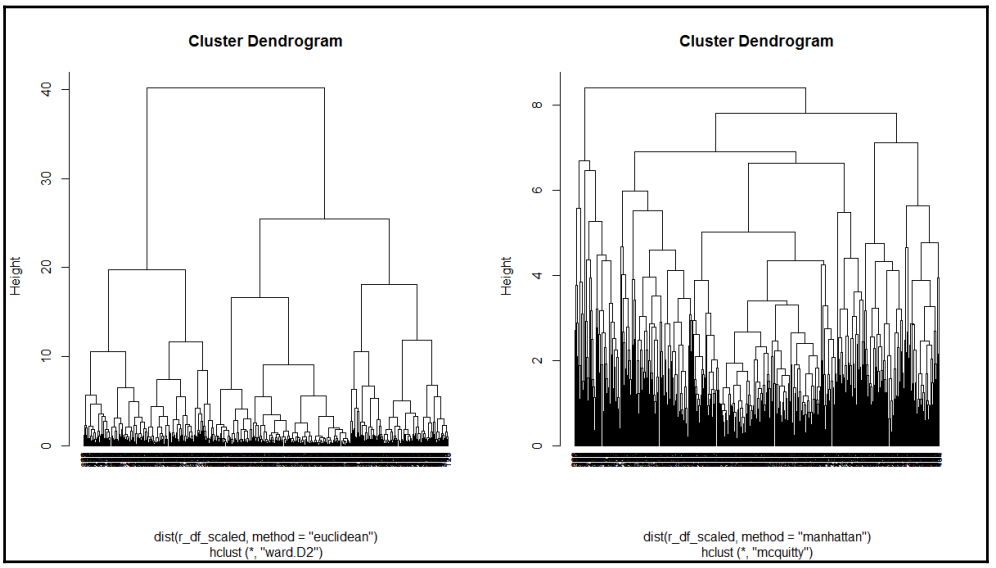
The hierarchical algorithm follows these steps:

1. Start with N singleton clusters (nodes) labeled *−1, . . . , −N*, which represent the initial set of input points.
2. Find a pair of nodes / cluster with minimal distance among all pairwise distances using a distance function.
3. Combine the two nodes into a new node and remove the two old nodes. The new nodes are labeled consecutively *1, 2, . . .*.
4. The distances from the new node to all other nodes are determined by the method parameter.
5. Repeat *N − 1* times from step 2 until there is one big node that contains all original input points.

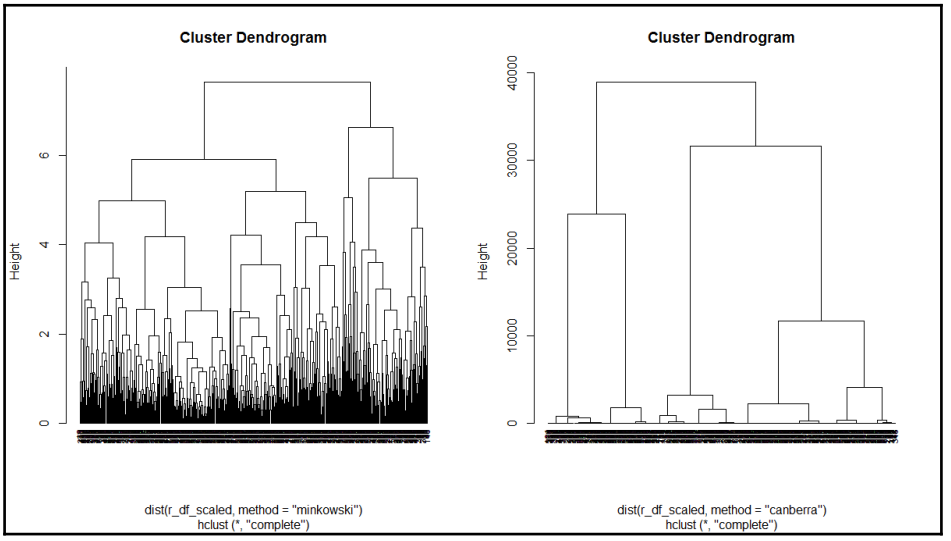
If the method chosen to perform hierarchical clustering is ward, then the distance measure to be used for clustering is Euclidean. The following code shows how to perform agglomerative hierarchical clustering:

```
> hfit<-hclust(dist(r_df_scaled,method = "euclidean"),method="ward.D2")
> par(mfrow=c(1,2))
> plot(hfit,hang=-0.005,cex=0.7)
> hfit<-hclust(dist(r_df_scaled,method = "manhattan"),method="mcquitty")
> plot(hfit,hang=-0.005,cex=0.7)
> hfit<-hclust(dist(r_df_scaled,method = "minkowski"))
> plot(hfit,hang=-0.005,cex=0.7)
> hfit<-hclust(dist(r_df_scaled,method = "canberra"))
> plot(hfit,hang=-0.005,cex=0.7)
```

Here is the first dendrogram graph:



Here is the second dendrogram graph:



Various methods are used to show how to perform agglomerative hierarchical clustering in the preceding two dendrogram graphs. The parameter `method` is one of the strings `single`, `centroid`, `median`, `ward`, and the argument method is one of the strings `single`,

`complete`, `average`, `mcquitty`, `centroid`, `median`, `ward.D`, `ward.D2`. Each method is different. Single linkage uses the smallest minimum pairwise distance to merge two clusters. Complete linkage uses the similarity between the dissimilar objects in two different clusters. Likewise each method has a unique way to compute the similarity between different clusters:

```
> summary(hfit)
Length Class Mode
merge 878 -none- numeric
height 439 -none- numeric
order 440 -none- numeric
labels 0 -none- NULL
method 1 -none- character
call 3 -none- call
dist.method 1 -none- character
```

Divisive hierarchical clustering can be performed using the `diana` function from the library `cluster`. The syntax is shown as follows:

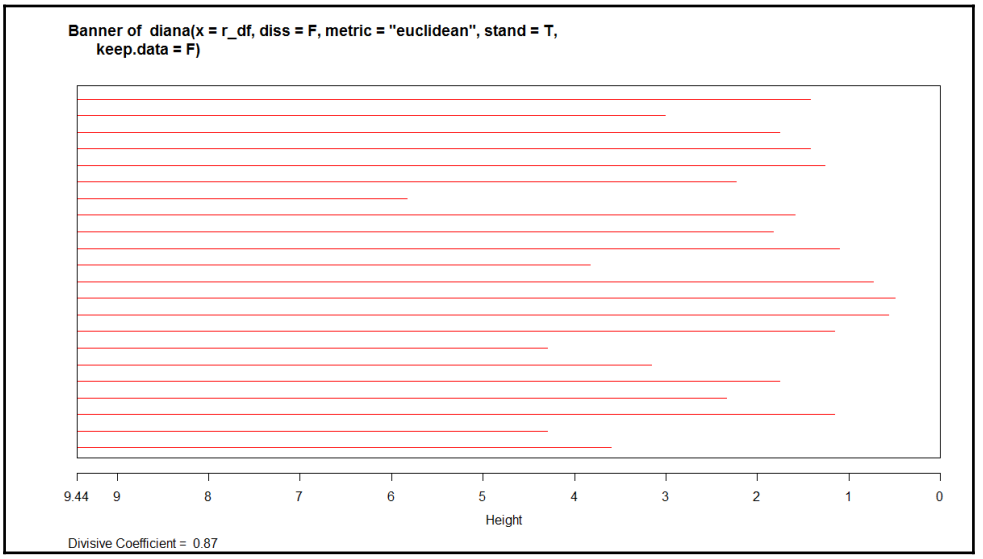
```
diana(x, diss = inherits(x, "dist"), metric = "euclidean", stand = FALSE,
keep.diss = n < 100, keep.data = !diss, trace.lev = 0)
```

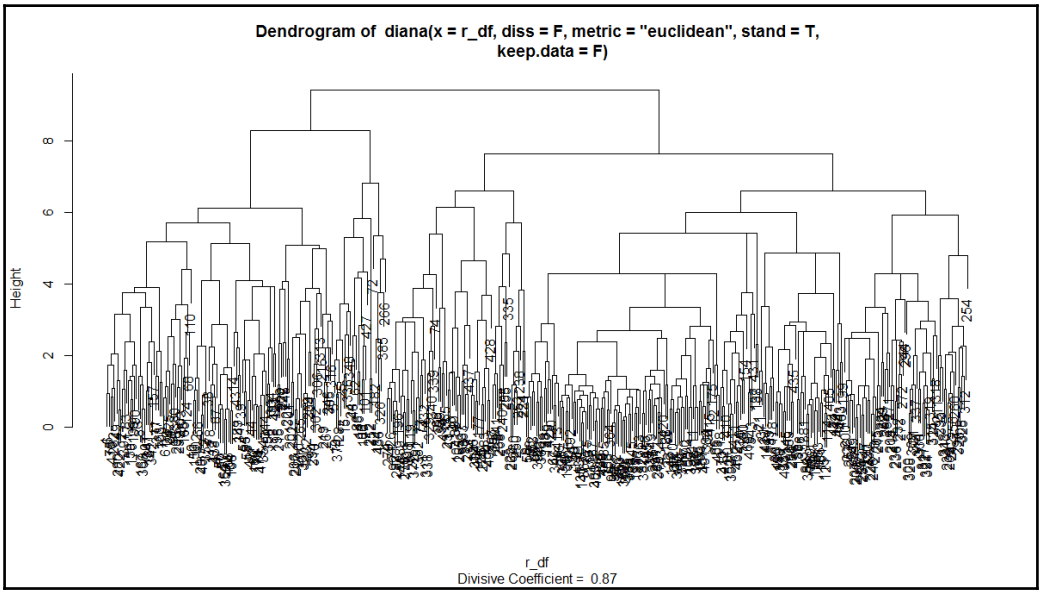
To apply the `diana` function, the input data should be in matrix or dataframe type. The `diana` function accepts both a distance matrix as well as a dataframe as an input. There is a logical flag argument meant to control the input structure to the function. The `metric` argument is a character string specifying the metric to be used to calculate distance; it has two options, `"euclidean"` and `"manhattan"`. Euclidean distances are root sums of squares of differences and manhattan distances are the sums of absolute differences, for which the mathematical formula has already been explained in this chapter. If `x`, the input dataset, is already a dissimilarity matrix, then this `diss` argument will be ignored from execution. The argument `stand` if takes the value as true, this implies that the raw dataset is used and further the distance matrix needs to be created.

The `diana` function constructs a hierarchy of clusters, as the name suggests, starting with one large cluster containing all observations. Clusters are divided until each cluster contains only a single observation. At each hierarchy, the cluster with the largest diameter based on the distance from the cluster center to the outermost data point as measured by the distance function is selected:

```
dfit<-diana(r_df,diss=F,metric = "euclidean",stand=T,keep.data = F)
summary(dfit)
plot(dfit)
```

The `plot` command on divisive clustering provides two graph options, a banner indicating the divisive coefficient and a dendrogram displaying the tree structure:

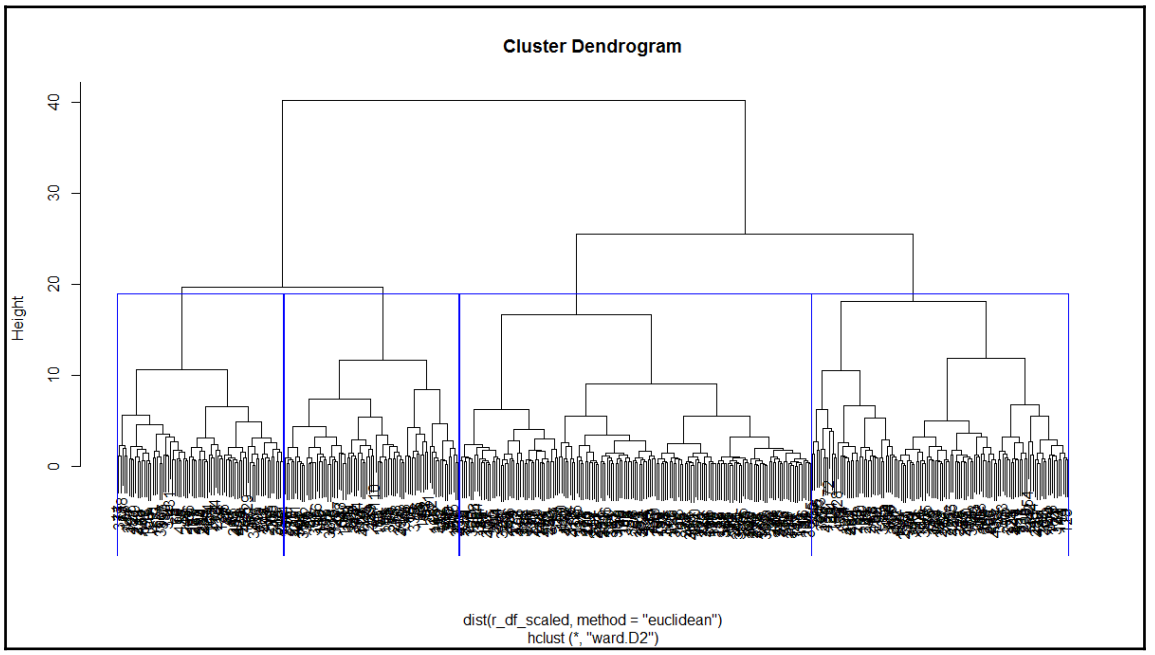




For the hierarchical clustering algorithms, one additional step is to cut the dendrogram tree structure into a finite group of clusters so that the results from various clustering methods can be compared. The following function is used to cut the tree structure:

```
> #cutting the tree into groups
> g_hfit<-cutree(hfit,k=4)
> plot(hfit)
> table(g_hfit)
g_hfit
1  2  3  4
77 163 119 81
> rect.hclust(hfit,k=4,border = "blue")
```

This graph shows the tree structure:



Model-based clustering

There is a library in R known as `mclust` that provides Gaussian finite mixture models fitted viaEM algorithm for model-based clustering, including Bayesian regularization. In this section, we discuss the syntax for `Mclust` and the outcome, and the rest of the results can be analyzed using profiling information. The model-based approach applies a set of data models, the maximum likelihood estimation method, and Bayes criteria to identify the cluster, and the `Mclust` function estimates the parameters through the expectation maximization algorithm.

The `Mclust` package provides various data models based on three parameters: *E*, *V*, and *I*. The model identifiers use the three letters *E*, *V*, and *I* to code geometric characteristics such as volume, shape, and orientation of the cluster. *E* means equal, *I* implies identity matrix in specifying the shape, and *V* implies varying variance. Keeping in mind these three letters, there is a set of models that is by default run using the `mclust` library, and those models can be classified into three groups such as spherical, diagonal and ellipsoidal. The details are given in the following table:

Model code	Description
`EII`	Spherical, equal volume
`VII`	Spherical, unequal volume
`EEI`	Diagonal, equal volume and shape
`VEI`	Diagonal, varying volume, equal shape
`EVI`	Diagonal, equal volume, varying shape
`VVI`	Diagonal, varying volume and shape
`EEE`	Ellipsoidal; equal volume, shape, and orientation
`EEV`	Ellipsoidal, equal volume and shape
`VEV`	Ellipsoidal, equal shape
`VVV`	Ellipsoidal' varying volume, shape, and orientation

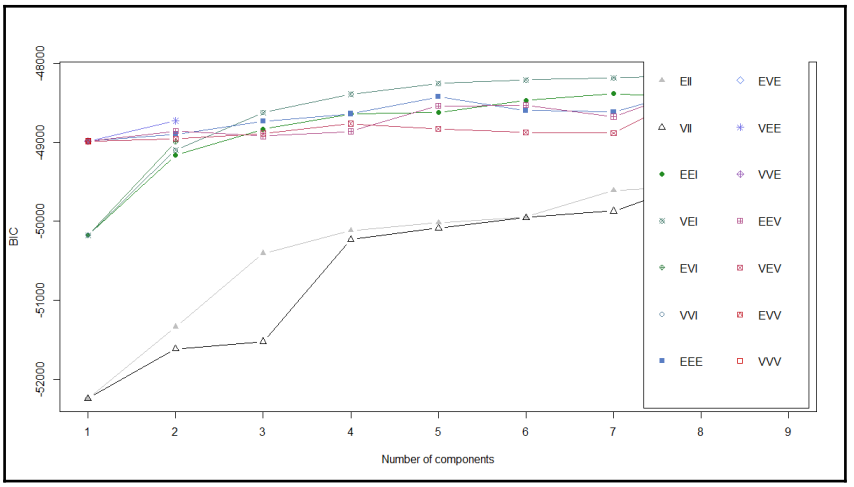
Table 2: Model description using finite mixture approach

Now let's use the `Mclust` method of doing clustering and grouping of observations based on the *E*, *V*, and *I* parameters:

```
> clus <- Mclust(r_df[,-c(1,2)])
> summary(clus)
----------------------------------------------------
```

```
Gaussian finite mixture model fitted by EM algorithm
----------------------------------------------------
Mclust VEI (diagonal, equal shape) model with 9 components:
log.likelihood n df BIC ICL
-23843.48 440 76 -48149.56 -48223.43
Clustering table:
1 2 3 4 5 6 7 8 9
165 21 40 44 15 49 27 41 38
```

From the preceding summary, there are nine clusters created using the VEI model specification and the BIC is lowest for this model. Let's also plot all other model specifications using a plot:



From the preceding graph, it is clear that the VEI model specification is the best. Using the following code, we can print the cluster membership as well as the number of rows in each of the clusters:

```
# The clustering vector:
clus_vec <- clus$classification
clus_vec
clust <- lapply(1:3, function(nc) row.names(r_df)[clus_vec==nc])
clust # printing the clusters
```

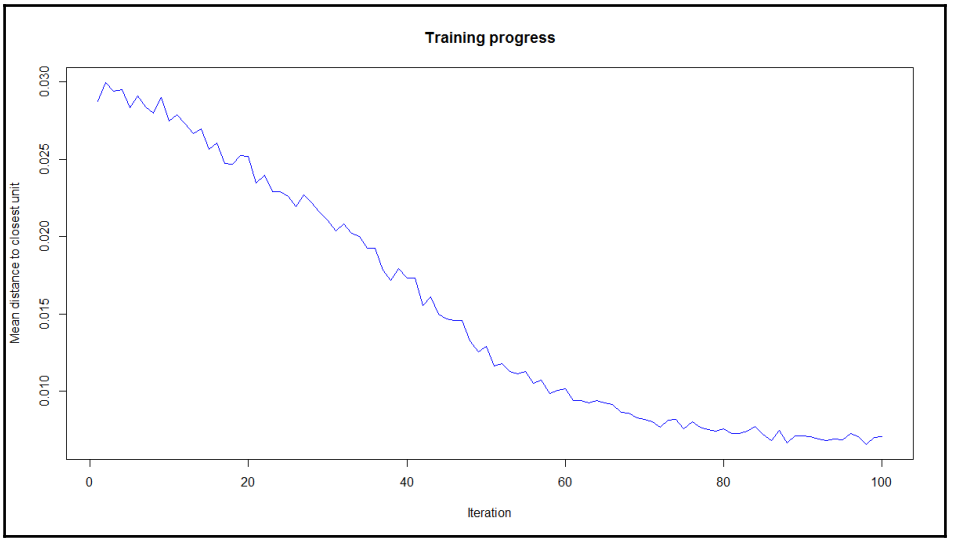
Using the `summary` command with parameter is true, mean, variance and probability for each variable we can compute:

```
summary(clus, parameters = T)
```

Other cluster algorithms

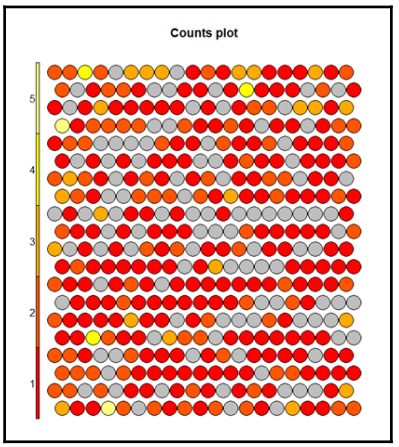
Self-organizing map (**SOM**) is another approach to clustering; it uses the visual method of representing data. SOM is basically used to derive unique nodes. SOM creates a network structure where the data objects would be connected by nodes and edges. Using those nodes, the similarity between different objects can be drawn to create clustering solutions. The `kohonen` library contains the SOM function:

```
> library(kohonen)
> som_grid <- somgrid(xdim = 20, ydim=20, topo="hexagonal")
> som_model <- som(r_df_scaled,
+ grid=som_grid,
+ rlen=100,
+ alpha=c(0.05,0.01),
+ keep.data = TRUE,
+ n.hood="circular")
plot(som_model, type="changes",col="blue")
```



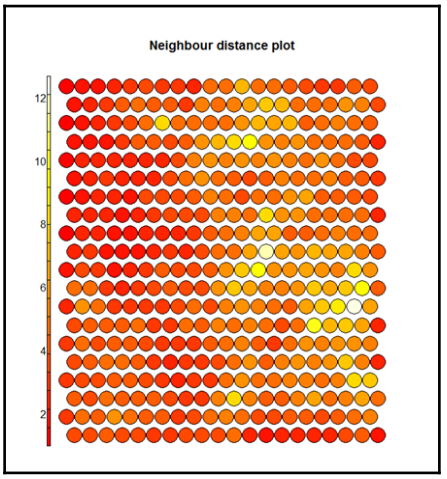
The preceding graph explains how the mean distance to the closest unit changes with increase in the iteration level:

```
> plot(som_model, type="count")
```



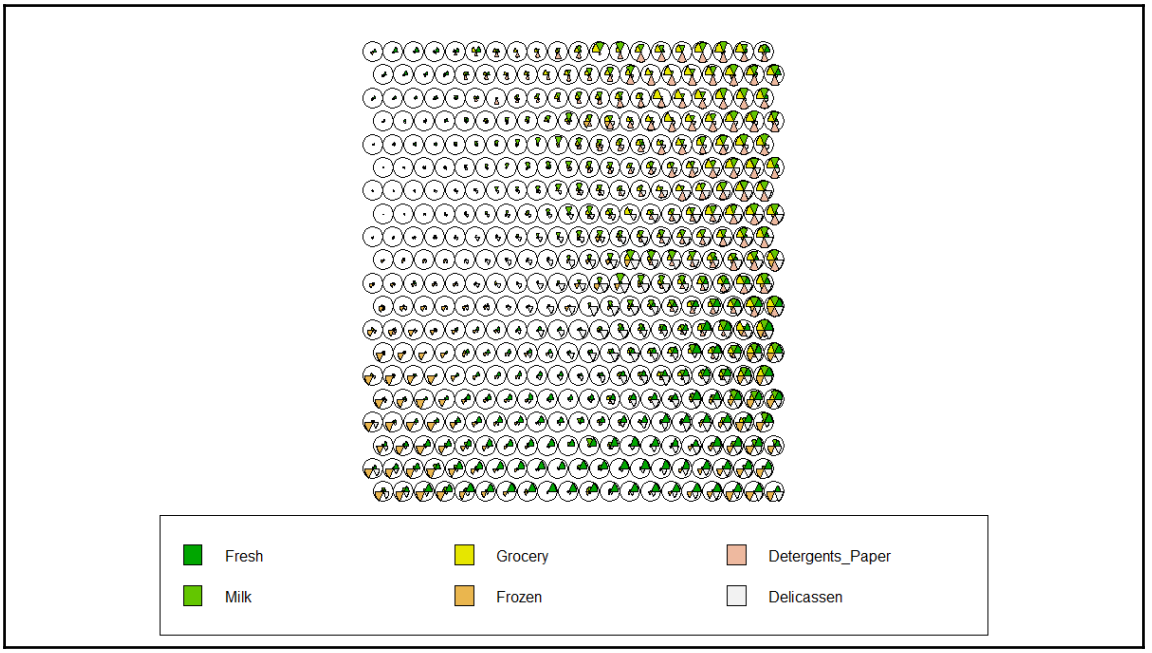
The preceding graph displays the counts plot using the SOM approach. The following graph displays the neighbor distance plot. The map is colored by different variables and the circles with colors denote where those input variables fall in n-dimensional space. When the plot type is counts, it shows the number of objects mapped to individual units. The units in gray color indicate empty. When the type is distance neighbors, it shows the sum of distances to all immediate neighbors. This is shown in the following graph:

```
> plot(som_model, type="dist.neighbours")
```




```
> plot(som_model, type="codes")
```

Finally, the next graph displays the code using all the variables:



In R, the `kohonen` library is used to create self-organizing maps using hierarchical clustering. When the type of graph is codes, as it is shown in the preceding graph, this shows the code book vectors; each color indicates a feature from the input dataset. The preceding results and graphs show how to create a clustering solution and how clustering can be visualized using the SOM plot function. SOM is a data visualization process used to show similar objects in a dataset based on input data vectors. Hence, the conclusion from the results would be how clusters or segments can be created by visual inspection.

Comparing clustering methods

Here are some of the advantages and disadvantages of various clustering methods:

For K-means clustering:

Merits and demerits of K-means clustering	
It is simple to understand and easy to interpret	It is not based on robust methodology. The big problem is initial random points
It is more flexible in comparison to other clustering algorithms	Different random points display different results. Hence, more iteration is required.
Performs well on a large number of input features	Since more iteration is required, it may not be efficient for large datasets from a computational efficiency point of view. It would require more memory to process.

For SOM:

Merits and Demerits of SOM	
Simple to understand visually	Only works on continuous data
New data points can be predicted visually	Difficult to represent when the dimensions increase

References

Lichman, M. (2013). *UCI Machine Learning Repository* (`http://archive.ics.uci.edu/ml`). Irvine, CA: University of California, School of Information and Computer Science.

Summary

In this chapter, we discussed various methods of segmentation for unsupervised data mining problems and their implementations. Clustering is a subjective segmentation method and it requires further investigation such as profiling by cluster and calculating different measures such as mean and median for different variables. If any cluster displays closeness to another cluster and from business point of view they make sense, then they can be combined together. We discussed which model to use where and what the data requirement for each of the models is. In this chapter, we specifically highlighted the importance of data visualization when it comes to representing clustering solutions.

7
Building a Retail Recommendation Engine

In this age of Internet, everything available over the Internet is not useful for everyone. Different companies and entities use different approaches to find out relevant content for their audiences. People started building algorithms to construct a relevance score, based on which recommendations could be built and suggested to the users. In my day to day life, every time I see an image on Google, 3-4 other images are recommended to me by Google, every time I look for some videos on YouTube, 10 more videos are recommended to me, every time I visit Amazon to buy some products, 5-6 products are recommended to me, and, every time I read one blog or article, a few more articles and blogs are recommended to me. This is an evidence of algorithmic forces at play to recommend certain things based on user's preferences or choices, since the user's time is precious and content available over Internet is unlimited. Hence, recommendation engine helps organizations customize their offerings based on the user's preferences so that the user does not have to spend time in exploring what is required.

In this chapter, the reader will learn the following things and their implementation using R programming language:

- What is recommendation and how does that work
- Types and methods for performing recommendation
- Implementation of product recommendation using R

What is recommendation?

Recommendation is a method or technique by which an algorithm detects what the user is looking at. Recommendation can be of products, services, food, videos, audios, images, news articles, and so on. What recommendation is can be made much clearer if we look around and note what we observe over the Internet day in and day out. For example, consider news aggregator websites and how they recommend articles to users. They look at the tags, timing of the article loaded into Internet, number of comments, likes, and shares for that article, and of course geolocation, among other details, and then model the metadata information to arrive at a score. Based on that score, they start recommending articles to new users. Same thing happens when we watch a video on YouTube. Similarly, beer recommendation system also works; the user has to select any beer at random and based on the first product chosen, the algorithm recommends other beers based on similar users' historical purchase information. Same thing happens when we buy products from online e-commerce portals.

Types of product recommendation

Broadly, there are four different types of recommendation systems that exist:

- Content-based recommendation method
- Collaborative filtering-based recommendation method
- Demographic segment-based recommendation method
- Association rule based-recommendation method

In content-based method, the terms, concepts which are also known as keywords, define the relevance. If the matching keywords from any page that the user is reading currently are found in any other content available over Internet, then start calculating the term frequencies and assign a score and based on that score whichever is closer represent that to the user. Basically, in content-based methods, the text is used to find the match between two documents so that the recommendation can be generated.

In collaborative filtering, there are two variants: user-based collaborative filtering and item-based collaborative filtering.

In user-based collaborative filtering method, the user-user similarity is computed by the algorithm and then based on their preferences the recommendation is generated. In item based collaborative filtering, item-item similarity is considered to generate recommendation.

In demographic segment based category, the age, gender, marital status, and location of the users, along with their income, job, or any other available feature, are taken into consideration to create customer segments. Once the segments are created then at a segment level the most popular product is identified and recommended to new users who belong to those respective segments.

Techniques to perform recommendation

To filter out abundant information available online and recommend useful information to the user at first hand is the prime motivation for creating recommendation engine. So how does the collaborative filtering work? Collaborative filtering algorithm generates recommendations based on a subset of users that are most similar to the active user. Each time a recommendation is requested, the algorithm needs to compute the similarity between the active user and all the other users, based on their co-rated items, so as to pick the ones with similar behaviour. Subsequently, the algorithm recommends items to the active user that are highly rated by his/her most similar users.

In order to compute the similarities between users, a variety of similarity measures have been proposed, such as Pearson correlation, cosine vector similarity, Spearman correlation, entropy-based uncertainty measure, and mean square difference. Let's look at the mathematical formula behind the calculation and implementation using R.

The objective of a collaborative filtering algorithm is to suggest new items or to predict the utility of a certain item for a particular user based on the user's previous likings and the opinions of other like-minded users.

Memory-based algorithms utilize the entire user-item database to generate a prediction. These systems employ statistical techniques to find a set of users, known as neighbours, who have a history of agreeing with the target user (that is, they either rate the same items similarly or they tend to buy similar sets of items). Once a neighbourhood of users is formed, these systems use different algorithms to combine the preferences of neighbours to produce a prediction or top-N recommendation for the active user. The techniques, also known as nearest-neighbour or user-based collaborative filtering, are very popular and widely used in practice.

Model-based collaborative filtering algorithms provide item recommendation by first developing a model of user ratings. Algorithms in this category take a probabilistic approach and envision the collaborative filtering process as computing the expected value of a user prediction, given his/her ratings on other items. The model building process is performed by different machine learning algorithms, such as Bayesian network, clustering, and rule-based approaches.

One critical step in the item-based collaborative filtering algorithm is to compute the similarity between items and then to select the most similar items. The basic idea in similarity computation between two items *i* and *j* is to first isolate the users who have rated both of these items and then to apply a similarity computation technique to determine the similarity *S(i,j)*.

There are a number of different ways to compute the similarity between items. Here we present three such methods. These are cosine-based similarity, correlation-based similarity, and adjusted-cosine similarity:

- **Cosine-based similarity**: Two items are thought of as two vectors in the *m* dimensional user-space. The similarity between them is measured by computing the cosine of the angle between these two vectors:

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}$$

Preceding is the formula for computing cosine similarity between two vectors. Let's take a numerical example to compute the cosine similarity:

Cos (d1, d2) = 0.44

d1	**d2**	**d1*d2**	**||d1||**	**||d2||**	**||d1||*||d2||**
6	1	6	36	1	36
4	0	0	16	0	0
5	3	15	25	9	225
1	2	2	1	4	4
0	5	0	0	25	0
2	0	0	4	0	0
4	6	24	16	36	576
0	5	0	0	25	0
3	0	0	9	0	0
6	1	6	36	1	36

		53	143	101	877
			11.95826	10.04988	

Table 5: Results of cosine similarity

So cosine similarity between `d1` and `d2` is 44%.

Cos (d1, d2) = 0.441008707

- **Correlation-based similarity**: Similarity between two items *i* and *j* is measured by computing the Pearson-r correlation `corr`. To make the correlation computation accurate, we must first isolate the co-rated cases.
- **Adjusted cosine similarity**: One fundamental difference between the similarity computation in user-based CF and item-based CF is that in case of user-based CF the similarity is computed along the rows of the matrix, but in case of the item-based CF the similarity is computed along the columns, that is, each pair in the co-rated set corresponds to a different user. Computing similarity using basic cosine measure in item-based case has one important drawback – the difference in rating scale between different users is not taken into account. The adjusted cosine similarity offsets this drawback by subtracting the corresponding user average from each co-rated pair. The example is given in *Table 1*, where the co-rated set belongs to different users, users are in different rows of the table.

Assumptions

The dataset that we are going to use for this chapter contains 100 jokes and 5000 users, which is a built-in dataset in the library recommenderlab. The dataset contains real ratings expressed by users in a scale of -10 to +10, with -10 being the worst and +10 being the best joke. We will implement various recommendation algorithms using this dataset. Using this dataset, the objective is to recommend jokes to new users based on their past preferences.

What method to apply when

Though there are four different types of recommendation methods, now which one to apply when. If the products or items are bought in a batch, then it is preferred by practitioners to apply association rules, which is also known as market basket analysis. In a retail or e-commerce domain, the items are generally purchased in a lot. Hence, when a user adds a certain product to his/her cart, other products can be recommended to him/her based on the aggregate basket component as reflected by majority of the buyers.

If the ratings or reviews are explicitly given for a set of items or products, it makes sense to apply user based collaborative filtering. If some of the ratings for few items are missing still the data can be imputed, once the missing ratings predicted, the user similarity can be computed and hence recommendation can be generated. For user-based collaborative filtering, the data would look as follows:

User	Item1	Item2	Item3	Item4	Item5	Item6
user1	-7.82	8.79	-9.66	-8.16	-7.52	-8.5
user2	4.08	-0.29	6.36	4.37	-2.38	-9.66
user3					9.03	9.27
user4		8.35			1.8	8.16
user5	8.5	4.61	-4.17	-5.39	1.36	1.6
user6	-6.17	-3.54	0.44	-8.5	-7.09	-4.32
user7					8.59	-9.85
user8	6.84	3.16	9.17	-6.21	-8.16	-1.7
user9	-3.79	-3.54	-9.42	-6.89	-8.74	-0.29
user10	3.01	5.15	5.15	3.01	6.41	5.15

Table 1: User-based item sample dataset

If the binary matrix is given as an input, where the levels represent whether the product is bought or not, then it is recommended to apply item-based collaborative filtering. The sample dataset is given next:

User	Item1	Item2	Item3	Item4	Item5	Item6
user1	1	1	1	1	1	0
user2	1	1	0	0	1	
user3	0	0		1	0	0
user4	0	1	1		1	1
user5	1	0		1	0	
user6	0	0	1			0
user7		1	0	0	1	1
user8	1	0	1	1	0	

user9	0	1	0	0		0
user10	1	0	1	1	1	1

Table 2: Item-based collaborative filtering sample dataset

If the product description details and the item description are given and the user search query is collected, then the similarity can be measured using content-based collaborative filtering method:

Title	**Search query**
Celestron LAND AND SKY 50TT Telescope	good telescope
(S5) 5-Port Mini Fast Ethernet Switch S5	mini switch
(TE-S16)Tenda 10/100 Mbps 16 Ports Et...	ethernet ports
(TE-TEH2400M) 24-Port 10/100 Switch	ethernet ports
(TE-W300A) Wireless N300 PoE Access P...	ethernet ports
(TE-W311M) Tenda N150 Wireless Adapte...	wireless adapter
(TE-W311M) Tenda N150 Wireless Adapte...	wireless adapter
(TE-W311MI) Wireless N150 Pico USB Ad...	wireless adapter
101 Lighting 12 Watt Led Bulb - Pack Of 2	led bulb
101 Lighting 7 Watt Led Bulb - Pack Of 2	led bulb

Table 3: Content-based collaborative filtering sample dataset

Limitations of collaborative filtering

User-based collaborative filtering systems have been very successful in past, but their widespread use has revealed some real challenges, such as:

- **Sparsity**: In practice, many commercial recommender systems are used to evaluate large item sets (for example, Amazon.com recommends books and CDNow.com recommends music albums). In these systems, even active users may have purchased well under 1% of the items 1%). Accordingly, a recommender system based on nearest neighbor algorithms may be unable to make any item recommendations for a particular user. As a result, the accuracy of recommendations may be poor.

- **Scalability**: Nearest neighbor algorithms require computation that grows with both the number of users and the number of items. With millions of users and items, a typical web-based recommender system running existing algorithms will suffer serious scalability problems.

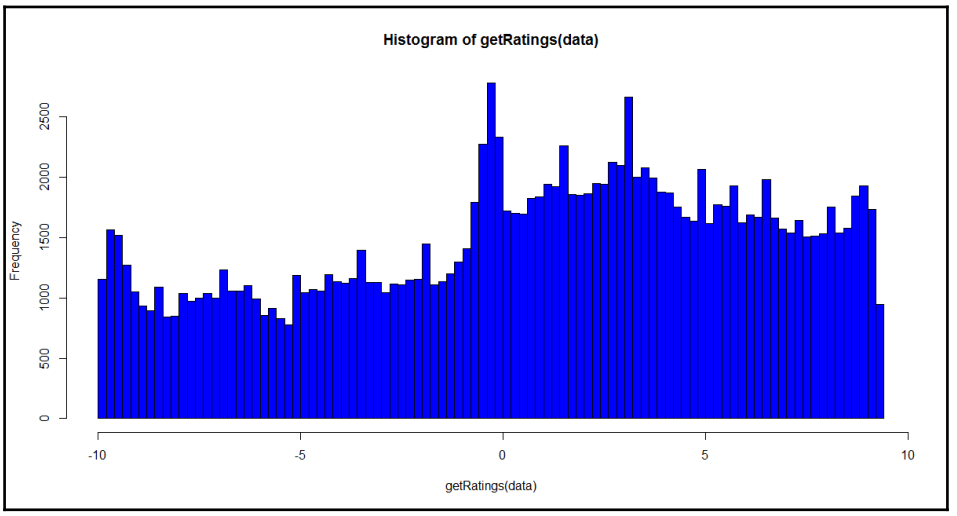
Practical project

The dataset contains a sample of 5000 users from the anonymous ratings data from the Jester Online Joke Recommender System collected between April 1999 and May 2003 (Goldberg, Roeder, Gupta, and Perkins 2001). The dataset contains ratings for 100 jokes on a scale from -10 to 10. All users in the data set have rated 36 or more jokes. Let's load the `recommenderlab` library and the `Jester5K` dataset:

```
> library("recommenderlab")
> data(Jester5k)
> Jester5k@data@Dimnames[2]
[[1]]
[1] "j1" "j2" "j3" "j4" "j5" "j6" "j7" "j8" "j9"
[10] "j10" "j11" "j12" "j13" "j14" "j15" "j16" "j17" "j18"
[19] "j19" "j20" "j21" "j22" "j23" "j24" "j25" "j26" "j27"
[28] "j28" "j29" "j30" "j31" "j32" "j33" "j34" "j35" "j36"
[37] "j37" "j38" "j39" "j40" "j41" "j42" "j43" "j44" "j45"
[46] "j46" "j47" "j48" "j49" "j50" "j51" "j52" "j53" "j54"
[55] "j55" "j56" "j57" "j58" "j59" "j60" "j61" "j62" "j63"
[64] "j64" "j65" "j66" "j67" "j68" "j69" "j70" "j71" "j72"
[73] "j73" "j74" "j75" "j76" "j77" "j78" "j79" "j80" "j81"
[82] "j82" "j83" "j84" "j85" "j86" "j87" "j88" "j89" "j90"
[91] "j91" "j92" "j93" "j94" "j95" "j96" "j97" "j98" "j99"
[100] "j100"
```

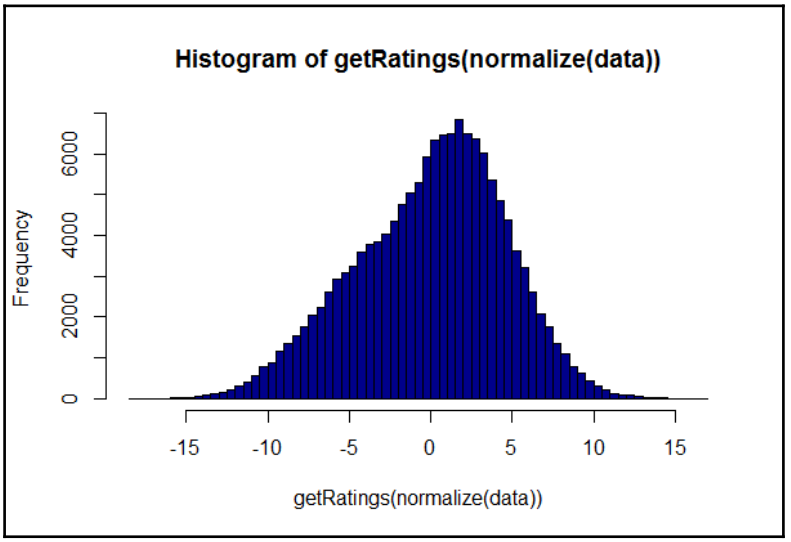
The following image shows the distribution of real ratings given by 2000 users:

```
> data<-sample(Jester5k,2000)
> hist(getRatings(data),breaks=100,col="blue")
```



The input dataset contains the individual ratings; normalization function reduces the individual rating bias by centering the row, which is a standard z-score transformation. Subtracting each element from the mean and then dividing by standard deviation. The following graph shows normalized ratings for the preceding dataset:

```
> hist(getRatings(normalize(data)),breaks=100,col="blue4")
```



To create a recommender system:

A recommendation engine is created using the `recommender()` function. A new recommendation algorithm can be added by the user using the `recommenderRegistry$get_entries()` function:

```
> recommenderRegistry$get_entries(dataType = "realRatingMatrix")
$IBCF_realRatingMatrix
Recommender method: IBCF
Description: Recommender based on item-based collaborative filtering (real
data).
Parameters:
k method normalize normalize_sim_matrix alpha na_as_zero minRating
1 30 Cosine center FALSE 0.5 FALSE NA
$POPULAR_realRatingMatrix
Recommender method: POPULAR
Description: Recommender based on item popularity (real data).
Parameters: None
$RANDOM_realRatingMatrix
Recommender method: RANDOM
Description: Produce random recommendations (real ratings).
Parameters: None
$SVD_realRatingMatrix
Recommender method: SVD
Description: Recommender based on SVD approximation with column-mean
imputation (real data).
Parameters:
k maxiter normalize minRating
1 10 100 center NA
$SVDF_realRatingMatrix
Recommender method: SVDF
Description: Recommender based on Funk SVD with gradient descend (real
data).
Parameters:
k gamma lambda min_epochs max_epochs min_improvement normalize
1 10 0.015 0.001 50 200 1e-06 center
minRating verbose
1 NA FALSE
$UBCF_realRatingMatrix
Recommender method: UBCF
Description: Recommender based on user-based collaborative filtering (real
data).
Parameters:
method nn sample normalize minRating
1 cosine 25 FALSE center NA
```

The preceding `Registry` command helps in identifying the methods available in `recommenderlab`, parameters for the model.

There are six different methods for implementing recommender system: popular, item-based, user based, PCA, random, and SVD. Let's start the recommendation engine using popular method:

```
> rc <- Recommender(Jester5k, method = "POPULAR")
> rc
Recommender of type 'POPULAR' for 'realRatingMatrix'
learned using 5000 users.
> names(getModel(rc))
[1] "topN" "ratings"
[3] "minRating" "normalize"
[5] "aggregationRatings" "aggregationPopularity"
[7] "minRating" "verbose"
> getModel(rc)$topN
Recommendations as 'topNList' with n = 100 for 1 users.
```

The objects such as top N, verbose, aggregation popularity, and others can be printed using names of the `getmodel()` command.

```
recom <- predict(rc, Jester5k, n=5)
recom
```

To generate recommendation, we can use the `predict` function against the same dataset and validate the accuracy of the predictive model. Here we are generating top 5 recommended jokes to each of the users. The result of the prediction is as follows:

```
> head(as(recom,"list"))
$u2841
[1] "j89" "j72" "j76" "j88" "j83"
$u15547
[1] "j89" "j93" "j76" "j88" "j91"
$u15221
character(0)
$u15573
character(0)
$u21505
[1] "j89" "j72" "j93" "j76" "j88"
$u15994
character(0)
```

For the same `Jester5K` dataset, let's try to implement **item-based collaborative filtering** (**IBCF**):

```
> rc <- Recommender(Jester5k, method = "IBCF")
```

```
> rc
Recommender of type 'IBCF' for 'realRatingMatrix'
learned using 5000 users.
> recom <- predict(rc, Jester5k, n=5)
> recom
Recommendations as 'topNList' with n = 5 for 5000 users.
> head(as(recom,"list"))
$u2841
[1] "j85" "j86" "j74" "j84" "j80"
$u15547
[1] "j91" "j87" "j88" "j89" "j93"
$u15221
character(0)
$u15573
character(0)
$u21505
[1] "j78" "j80" "j73" "j77" "j92"
$u15994
character(0)
```

Principal component analysis (**PCA**) method is not applicable for real rating-based dataset because getting correlation matrix and subsequent eigen vector and eigen value calculation would not be accurate. Hence, we will not show its application. Next we are going to show how the random method works:

```
> rc <- Recommender(Jester5k, method = "RANDOM")
> rc
Recommender of type 'RANDOM' for 'ratingMatrix'
learned using 5000 users.
> recom <- predict(rc, Jester5k, n=5)
> recom
Recommendations as 'topNList' with n = 5 for 5000 users.
> head(as(recom,"list"))
[[1]]
[1] "j90" "j74" "j86" "j78" "j85"
[[2]]
[1] "j87" "j88" "j74" "j92" "j79"
[[3]]
character(0)
[[4]]
character(0)
[[5]]
[1] "j95" "j86" "j93" "j78" "j83"
[[6]]
character(0)
```

In recommendation engine, the SVD approach is used to predict the missing ratings so that recommendation can be generated. Using **singular value decomposition** (**SVD**) method, the following recommendation can be generated:

```
> rc <- Recommender(Jester5k, method = "SVD")
> rc
Recommender of type 'SVD' for 'realRatingMatrix'
learned using 5000 users.
> recom <- predict(rc, Jester5k, n=5)
> recom
Recommendations as 'topNList' with n = 5 for 5000 users.
> head(as(recom,"list"))
$u2841
[1] "j74" "j71" "j84" "j79" "j80"
$u15547
[1] "j89" "j93" "j76" "j81" "j88"
$u15221
character(0)
$u15573
character(0)
$u21505
[1] "j80" "j73" "j100" "j72" "j78"
$u15994
character(0)
```

The result from the user-based collaborative filtering is shown next:

```
> rc <- Recommender(Jester5k, method = "UBCF")
> rc
Recommender of type 'UBCF' for 'realRatingMatrix'
learned using 5000 users.
> recom <- predict(rc, Jester5k, n=5)
> recom
Recommendations as 'topNList' with n = 5 for 5000 users.
> head(as(recom,"list"))
$u2841
[1] "j81" "j78" "j83" "j80" "j73"
$u15547
[1] "j96" "j87" "j89" "j76" "j93"
$u15221
character(0)
$u15573
character(0)
$u21505
[1] "j100" "j81" "j83" "j92" "j96"
$u15994
character(0)
```

Now let's compare the results obtained from all the five different algorithms, except PCA, because PCA requires a binary dataset and does not accept real ratings matrix.

Popular	IBCF	Random method	SVD	UBCF
> head(as(recom,"list"))	> head(as(recom,"list"))	> head(as(recom,"list"))	> head(as(recom,"list"))	> head(as(recom,"list"))
$u2841	$u2841	[[1]]	$u2841	$u2841
[1] "j89" "j72" "j76" "j88" "j83"	[1] "j85" "j86" "j74" "j84" "j80"	[1] "j90" "j74" "j86" "j78" "j85"	[1] "j74" "j71" "j84" "j79" "j80"	[1] "j81" "j78" "j83" "j80" "j73"
$u15547	$u15547	[[2]]	$u15547	$u15547
[1] "j89" "j93" "j76" "j88" "j91"	[1] "j91" "j87" "j88" "j89" "j93"	[1] "j87" "j88" "j74" "j92" "j79"	[1] "j89" "j93" "j76" "j81" "j88"	[1] "j96" "j87" "j89" "j76" "j93"
$u15221	$u15221	[[3]]	$u15221	$u15221
character(0)	character(0)	character(0)	character(0)	character(0)
$u15573	$u15573	[[4]]	$u15573	$u15573
character(0)	character(0)	character(0)	character(0)	character(0)
$u21505	$u21505	[[5]]	$u21505	$u21505
[1] "j89" "j72" "j93" "j76" "j88"	[1] "j78" "j80" "j73" "j77" "j92"	[1] "j95" "j86" "j93" "j78" "j83"	[1] "j80" "j73" "j100" "j72" "j78"	[1] "j100" "j81" "j83" "j92" "j96"
$u15994	$u15994	[[6]]	$u15994	$u15994
character(0)	character(0)	character(0)	character(0)	character(0)

Table 4: Results comparison between different recommendation algorithms

One thing clear from the table is that for users `15573` and `15221`, none of the five methods generate a recommendation. Hence, it is important to look at methods to evaluate the recommendation results. To validate the accuracy of the model let's implement accuracy measures and compare the accuracy of all the models.

For the evaluation of the model results, the dataset is divided into 90 percent for training and 10 percent for testing the algorithm. The definition of good rating is updated as `5`:

```
> e <- evaluationScheme(Jester5k, method="split",
+ train=0.9,given=15, goodRating=5)
> e
Evaluation scheme with 15 items given
Method: 'split' with 1 run(s).
Training set proportion: 0.900
Good ratings: >=5.000000
Data set: 5000 x 100 rating matrix of class 'realRatingMatrix' with 362106
ratings.
```

The following script is used to build the collaborative filtering model, apply it on a new dataset for predicting the ratings and then the prediction accuracy is computed the error matrix is shown as follows:

```
> #User based collaborative filtering
> r1 <- Recommender(getData(e, "train"), "UBCF")
> #Item based collaborative filtering
> r2 <- Recommender(getData(e, "train"), "IBCF")
> #PCA based collaborative filtering
> #r3 <- Recommender(getData(e, "train"), "PCA")
> #POPULAR based collaborative filtering
> r4 <- Recommender(getData(e, "train"), "POPULAR")
> #RANDOM based collaborative filtering
> r5 <- Recommender(getData(e, "train"), "RANDOM")
> #SVD based collaborative filtering
> r6 <- Recommender(getData(e, "train"), "SVD")
> #Predicted Ratings
> p1 <- predict(r1, getData(e, "known"), type="ratings")
> p2 <- predict(r2, getData(e, "known"), type="ratings")
> #p3 <- predict(r3, getData(e, "known"), type="ratings")
> p4 <- predict(r4, getData(e, "known"), type="ratings")
> p5 <- predict(r5, getData(e, "known"), type="ratings")
> p6 <- predict(r6, getData(e, "known"), type="ratings")
> #calculate the error between the prediction and
> #the unknown part of the test data
> error <- rbind(
+ calcPredictionAccuracy(p1, getData(e, "unknown")),
+ calcPredictionAccuracy(p2, getData(e, "unknown")),
+ #calcPredictionAccuracy(p3, getData(e, "unknown")),
+ calcPredictionAccuracy(p4, getData(e, "unknown")),
+ calcPredictionAccuracy(p5, getData(e, "unknown")),
+ calcPredictionAccuracy(p6, getData(e, "unknown"))
+ )
> rownames(error) <- c("UBCF","IBCF","POPULAR","RANDOM","SVD")
> error
RMSE MSE MAE
UBCF 4.485571 20.12034 3.511709
IBCF 4.606355 21.21851 3.466738
POPULAR 4.509973 20.33985 3.548478
RANDOM 7.917373 62.68480 6.464369
SVD 4.653111 21.65144 3.679550
```

From the preceding result, `UBCF` has the lowest error in comparison to the other recommendation methods. Here, to evaluate the results of the predictive model, we are using k-fold cross validation method; `k` is assumed to be taken as `4`:

```
> #Evaluation of a top-N recommender algorithm
```

```
> scheme <- evaluationScheme(Jester5k, method="cross", k=4,
+ given=3,goodRating=5)
> scheme
Evaluation scheme with 3 items given
Method: 'cross-validation' with 4 run(s).
Good ratings: >=5.000000
Data set: 5000 x 100 rating matrix of class 'realRatingMatrix' with 362106
ratings.
```

The results of the models from the evaluation scheme show the runtime versus the prediction time by different cross validation results for different models, the result is shown as follows:

```
> results <- evaluate(scheme, method="POPULAR", n=c(1,3,5,10,15,20))
POPULAR run fold/sample [model time/prediction time]
1 [0.14sec/2.27sec]
2 [0.16sec/2.2sec]
3 [0.14sec/2.24sec]
4 [0.14sec/2.23sec]
> results <- evaluate(scheme, method="IBCF", n=c(1,3,5,10,15,20))
IBCF run fold/sample [model time/prediction time]
1 [0.4sec/0.38sec]
2 [0.41sec/0.37sec]
3 [0.42sec/0.38sec]
4 [0.43sec/0.37sec]
> results <- evaluate(scheme, method="UBCF", n=c(1,3,5,10,15,20))
UBCF run fold/sample [model time/prediction time]
1 [0.13sec/6.31sec]
2 [0.14sec/6.47sec]
3 [0.15sec/6.21sec]
4 [0.13sec/6.18sec]
> results <- evaluate(scheme, method="RANDOM", n=c(1,3,5,10,15,20))
RANDOM run fold/sample [model time/prediction time]
1 [0sec/0.27sec]
2 [0sec/0.26sec]
3 [0sec/0.27sec]
4 [0sec/0.26sec]
> results <- evaluate(scheme, method="SVD", n=c(1,3,5,10,15,20))
SVD run fold/sample [model time/prediction time]
1 [0.36sec/0.36sec]
2 [0.35sec/0.36sec]
3 [0.33sec/0.36sec]
4 [0.36sec/0.36sec]
```

The confusion matrix displays the level of accuracy provided by each of the models, we can estimate the accuracy measures such as precision, recall and TPR, FPR, and so on the result is shown as follows:

```
> getConfusionMatrix(results)[[1]]
TP FP FN TN precision recall TPR FPR
1 0.2736 0.7264 17.2968 78.7032 0.2736000 0.01656597 0.01656597 0.008934588
3 0.8144 2.1856 16.7560 77.2440 0.2714667 0.05212659 0.05212659 0.027200530
5 1.3120 3.6880 16.2584 75.7416 0.2624000 0.08516269 0.08516269 0.046201487
10 2.6056 7.3944 14.9648 72.0352 0.2605600 0.16691259 0.16691259
0.092274243
15 3.7768 11.2232 13.7936 68.2064 0.2517867 0.24036802 0.24036802
0.139945095
20 4.8136 15.1864 12.7568 64.2432 0.2406800 0.30082509 0.30082509
0.189489883
```

Association rules as a method for recommendation engine for building product recommendation in a retail/e-commerce scenario, is used in `Chapter 4`, *Regression with Automobile Data*.

Summary

In this chapter, we discussed various methods of recommending products. We looked at different ways of recommending products to users, based on similarity in their purchase pattern, content, item to item comparison, and so on. As far as the accuracy is concerned, always the user-based collaborative filtering is giving better result in a real rating-based matrix as an input. Similarly, the choice of methods for a specific use case is really difficult, so it is recommended to apply all six different methods and the best one should be selected automatically and the recommendation should also get updated automatically.

8
Dimensionality Reduction

In this chapter, we are going to discuss about various methods to reduce data dimensions in performing analysis. In data mining, traditionally people used to apply **principal component analysis** (**PCA**) as a method to reduce the dimensionality in data. Though now in the age of big data, PCA is still valid, however along with that, many other techniques are being used to reduce dimensions. With the growth of data in volumes and variety, the dimension of data has been continuously on the rise. Dimensionality reduction techniques have many applications in different industries, such as in image processing, speech recognition, recommendation engines, text processing, and so on. The main problem in these application areas is not only high dimensional data but also high sparsity. Sparsity means that many columns in the dataset will have missing or blank values.

In this chapter, we will implement dimensionality reduction techniques such as PCA, **singular value decomposition** (**SVD**), and iterative feature selection method using a practical data set and R programming language.

In this chapter, we are going to discuss:

- Why dimensionality reduction is a business problem and what impact it may have on the predictive models
- What are the different techniques, with their respective positives and negatives, and data requirements and more
- Which technique to apply and in what situations, with little bit of mathematics behind the calculation
- An R programming based implementation and interpretation of results in a project

Why dimensionality reduction?

In various statistical analysis models, especially showing a cause and impact relationship between dependent variable and a set of independent variables, if the number of independent variables increases beyond a manageable stage (for example, 100 plus), then it is quite difficult to interpret each and every variable. For example, in weather forecast, nowadays low cost sensors are being deployed at various places and those sensors provide signals and data are being stored in the database. When 1000 plus sensors provide data, it is important to understand the pattern or at least which all sensors are meaningful in performing the desired task.

Another example, from a business standpoint, is that if more than 30 features are impacting a dependent variable (for example, `Sales`), as a business owner I cannot regulate all 30 factors and cannot form strategies for 30 dimensions. Of course, as a business owner I would be interested in looking at 3-4 dimensions that should explain 80% of the dependent variable in the data. From the preceding two examples, it is pretty clear that dimensionality is still a valid business problem. Apart from business and volume, there are other reasons such as computational cost, storage cost of data, and more, if a set of dimensions are not at all meaningful for the target variable or target function why would I store it in my database.

Dimensionality reduction is useful in big data mining applications in performing both supervised learning and unsupervised learning-based tasks, to:

- Identify the pattern how the variables work together
- Display the relationship in a low dimensional space
- Compress the data for further analysis so that redundant features can be removed
- Avoid overfitting as reduced feature set with reduced degrees of freedom does that
- Running algorithms on a reduced feature set would be much more faster than the base features

Two important aspects of data dimensionality are, first, variables that are highly correlated indicate high redundancy in the data, and the second, most important dimensions always have high variance. While working on dimension reduction, it is important to take note of these aspects and make necessary amendments to the process.

Hence, it is important to reduce the dimensions. Also, it is not a good idea to focus on many variables to control or regulate the target variable in a predictive model. In a multivariate study, the correlation between various independent variables affect the empirical likelihood function and affects the eigen values and eigen vectors through covariance matrix.

Techniques available for dimensionality reduction

The dimensionality reduction techniques can be classified into two major groups: parametric or model-based feature reduction and non-parametric feature reduction. In non-parametric feature reduction technique, the data dimension is reduced first and then the resulting data can be used to create any classification or predictive model, supervised or unsupervised. However, the parametric-based method emphasizes on monitoring the overall performance of a model and its accuracy by changing the features and hence deciding how many features are needed to represent the model. The following are the techniques generally used in data mining literature to reduce data dimensions:

- Non-parametric:
    - PCA method
    - Parametric method
    - Forward feature selection
    - Backward feature selection

We are going to discuss these methods in detail with the help of a dataset using R programming language. Apart from these techniques mentioned earlier, there are few more methods but not that popular, such as removing variables with low variance as they don't add much information to the target variable and also removing variables with many missing values. The latter comes under the missing value treatment method, but can still be used to reduce the features in a high dimensional dataset. There are two methods in R, `prcomp()` and `princomp()`, but they use two slightly different methods; `princomp()` uses eigen vectors and `prcomp()` uses SVD method. Some researchers favor `prcomp()` as a method over `princomp()` method. We are going to use two different methods here.

Which technique to apply where?

The usage of the technique actually depends upon the researcher; what he is looking at from the data. If you are looking at hidden features and want to represent the data in a low dimensional space, PCA is the method one should choose. If you are looking at building a good classification or prediction model, then it is not a good idea to perform PCA first; you should ideally include all the features and remove the redundant ones by any parametric method, such as forward or backward. Selection of techniques vary based on data availability, the problem statement, and the task that someone is planning to perform.

Principal component analysis

PCA is a multivariate statistical data analysis technique applicable for datasets with numeric variables only, which computes linear combinations of the original variables and those principal components are orthogonal to each other in the feature space. PCA is a method that uses eigen values and eigen vectors to compute the principal components, which is a linear combination of original variables. PCA can be used in regression modelling in order to remove the problem of multicollinearity in the dataset and can also be used in clustering exercise in order to understand unique group behaviors or segments existing in the dataset.

PCA assumes that the variables are linearly associated to form principal components and variables with high variance do not necessarily represent the best feature. The principal component analysis is based on a few theorems:

- The inverse of an orthogonal matrix is its transpose
- The original matrix times the transposed version of the original matrix are both symmetric
- A matrix is symmetric if it is an orthogonally diagonalizable matrix
- It uses variance matrix, not the correlation matrix to compute components

Following are the steps to perform principal component analysis:

1. Get a numerical dataset. If you have any categorical variable, remove it from the dataset so that mathematical computation can happen on the remaining variables.
2. In order to make the PCA work properly, data normalization should be done. This is done by deducting the mean of the column from each of the values for all the variables; ensure that the mean of the variables of the transformed data should be equal to 0.

    We applied the following normalization formula:

$$norm(x) = x_i - \overline{x}$$

3. Calculate the covariance matrix on the reduced dataset. Covariance is a measure of two variables. A covariance matrix is a display of all the possible covariance values between all the different dimensions:

$$\operatorname{cov}(x, y) = \frac{\sum_{i=1}^{n} (x_i - \bar{x})(y_i - \bar{y})}{n-1}$$

The sign of covariance is more important than its value; a positive covariance value indicates that both dimensions increase together and vice versa. Similarly, a negative value of covariance indicates that with an increase in one dimension, the other dimension decreases. If the covariance value is zero, it shows that the two dimensions are independent of each other and there is no relationship.

4. Calculate the eigen vectors and eigen values from the covariance matrix. All eigen vectors always come from a square matrix but all square matrix do not always generate an eigen vector. All eigen vectors of a matrix are orthogonal to each other.
5. Create a feature vector by taking eigen values in order from largest to smallest. The eigen vector with highest eigen value is the principal component of the dataset. From the covariance matrix, the eigen vectors are identified and the eigen values are ordered from largest to smallest:

```
Feature Vector = (Eigen1, Eigen2......Eigen 14)
```

6. Create low dimension data using the transposed version of the feature vector and transposed version of the normalized data. In order to get the transposed dataset, we are going to multiply the transposed feature vector and transposed normalized data.

Let's apply these steps in the following section.

Practical project around dimensionality reduction

We are going to apply dimensionality reduction procedure, both model-based approach and principal component-based approach, on the dataset to come up with less number of features so that we can use those features for classification of the customers into defaulters and no-defaulters.

For practical project, we have considered a dataset default of credit card `clients.csv`, which contains 30,000 samples and 24 attributes or dimensions. We are going to apply two different methods of feature reduction: the traditional way and the modern machine learning way.

Attribute description

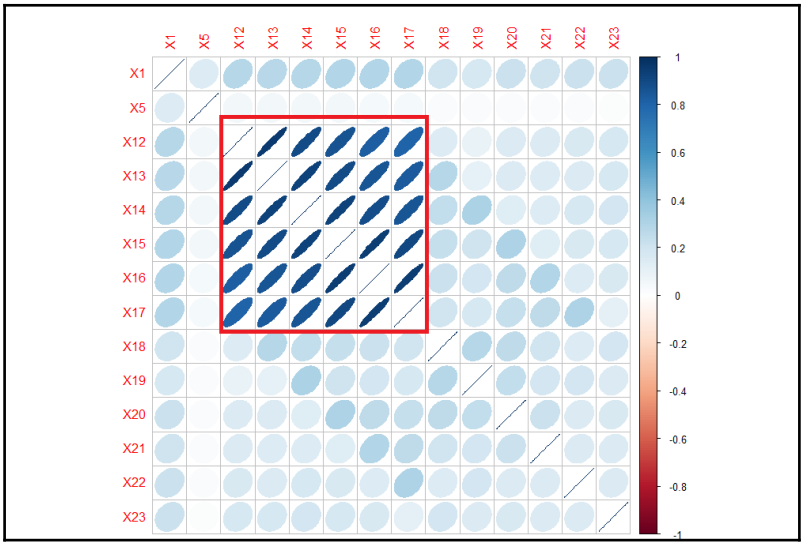
The following are descriptions for the attributes from the dataset:

- `X1`: Amount of the given credit (NT dollar). It includes both the individual consumer credit and his/her family (supplementary) credit.
- `X2`: Gender (`1 = male; 2 = female`).
- `X3`: Education (`1 = graduate school; 2 = university; 3 = high school; 4 = others`).
- `X4`: Marital status (`1 = married; 2 = single; 3 = others`).
- `X5`: Age (`year`).
- `X6 – X11`: History of past payment. We tracked the past monthly payment records (from April to September, 2005) as follows:
    - `X6= the repayment status in September, 2005`
    - `X7 = the repayment status in August, 2005; . . .`
    - `X11 = the repayment status in April, 2005. The measurement scale for the repayment status is: −1 = pay duly; 1 = payment delay for one month; 2 = payment delay for two months; . . . 8 = payment delay for eight months; 9 = payment delay for nine months, and so on.`
- `X12–X17`: Amount of bill statement (NT dollar).
    - `X12 = amount of bill statement in September, 2005;`
    - `X13 = amount of bill statement in August, 2005; . . .`
    - `X17 = amount of bill statement in April, 2005.`
- `X18–X23`: Amount of previous payment (NT dollar).
    - `X18 = amount paid in September, 2005;`
    - `X19 = amount paid in August, 2005; . . .`
    - `X23 = amount paid in April, 2005.`

Let's get the dataset and do the necessary normalization and transformation:

```
> setwd("select the working directory")
> default<-read.csv("default.csv")
corrplot::corrplot(cor(df),method="ellipse")
```

From the preceding correlations matrix, it is clear that there are some strong correlations between different variables from `X12` to `X17`, which possibly can be clubbed together as linear combinations using PCA. The following graph shows the correlations between different variables:



In the previous correlations graph, the strength of the correlation is reflected by the size of the ellipse and the color varying from red to blue. The blue ellipses indicate positive correlation and the white ones show no correlation. Variable `X12` has a high degree of positive correlation with variables `X13` to `X17`, with a correlation value of more than 80%.

Row number 1 contains the variable description and there are some columns which are categorical; we need to remove those from our dataset. Some other variables are shown as categorical and hence need to be converted back to numeric mode:

```
> df<-default[-1,-c(1,3:5,7:12,25)]
> func1<-function(x){
+ as.numeric(x)
+ }
> df<-as.data.frame(apply(df,2,func1))
> normalize-function(x){
```

```
+ (x-mean(x))
+ }
```

Using `func1`, we convert the non-numeric variables into numeric so that we can apply the normalization function (normalize), which is doing data normalization. After applying normalization, the dataset will look like the one as shown next. To apply principal component analysis, either we can use mean normalized dataset as an input or we can use the base dataset with correlations or a covariance matrix as an input. If we are not going to normalize the dataset, the variable with the highest variance will become the first principal component and hence will dominate other relevant principal components:

```
> dn<-as.data.frame(apply(df,2,normalize))
> str(dn)
'data.frame': 30000 obs. of 14 variables:
$ X1 : num -147484 -47484 -77484 -117484 -117484 ...
$ X5 : num -11.49 -9.49 -1.49 1.51 21.51 ...
$ X12: num -47310 -48541 -21984 -4233 -42606 ...
$ X13: num -46077 -47454 -35152 -946 -43509 ...
$ X14: num -46324 -44331 -33454 2278 -11178 ...
$ X15: num -43263 -39991 -28932 -14949 -22323 ...
$ X16: num -40311 -36856 -25363 -11352 -21165 ...
$ X17: num -38872 -35611 -23323 -9325 -19741 ...
$ X18: num -5664 -5664 -4146 -3664 -3664 ...
$ X19: num -5232 -4921 -4421 -3902 30760 ...
$ X20: num -5226 -4226 -4226 -4026 4774 ...
$ X21: num -4826 -3826 -3826 -3726 4174 ...
$ X22: num -4799 -4799 -3799 -3730 -4110 ...
$ X23: num -5216 -3216 -216 -4216 -4537 ...
```

Running principal component analysis:

```
> options(digits = 2)
> pca1<-princomp(df,scores = T, cor = T)
> summary(pca1)
Importance of components:
Comp.1 Comp.2 Comp.3 Comp.4 Comp.5 Comp.6 Comp.7 Comp.8 Comp.9 Comp.10
Comp.11
Standard deviation 2.43 1.31 1.022 0.962 0.940 0.934 0.883 0.852 0.841
0.514 0.2665
Proportion of Variance 0.42 0.12 0.075 0.066 0.063 0.062 0.056 0.052 0.051
0.019 0.0051
Cumulative Proportion 0.42 0.55 0.620 0.686 0.749 0.812 0.867 0.919 0.970
0.989 0.9936
Comp.12 Comp.13 Comp.14
Standard deviation 0.2026 0.1592 0.1525
Proportion of Variance 0.0029 0.0018 0.0017
Cumulative Proportion 0.9965 0.9983 1.0000
```

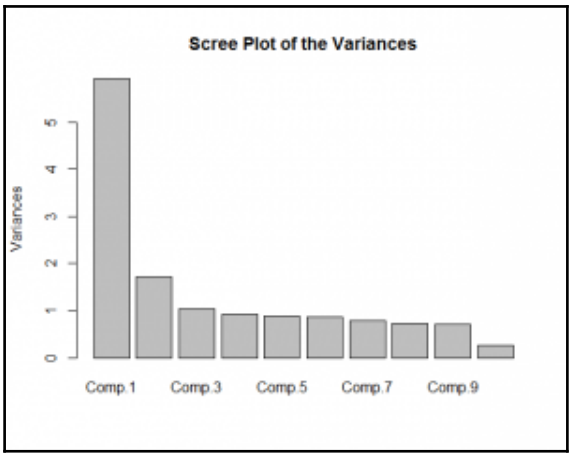
Principal component (**PC**) 1 explains 42% variation in the dataset and principal component 2 explains 12% variation in the dataset. This means that first PC has closeness to 42% of the total data points in the n-dimensional space. From the preceding results, it is clear that the first 8 principal components capture 91.9% variation in the dataset. Rest 8% variation in the dataset is explained by 6 other principal components. Now the question is how many principal components should be chosen. The general rule of thumb is 80-20, if 80% variation in the data can be explained by 20% of the principal components. There are 14 variables; we have to look at how many components explain 80% variation in the dataset cumulatively. Hence, it is recommended, based on the 80:20 Pareto principle, taking 6 principal components which explain 81% variation cumulatively.

In a multivariate dataset, the correlation between the component and the original variables is called the component loading. The loadings of the principal components are as follows:

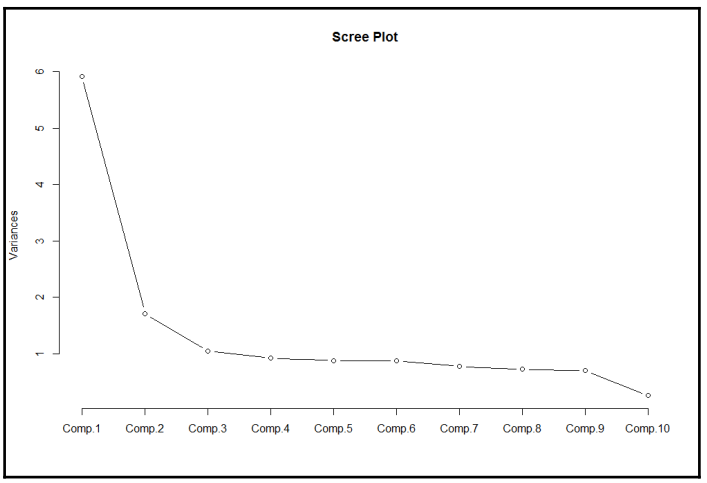
```
> #Loadings of Principal Components
> pca1$loadings
Loadings:
Comp.1 Comp.2 Comp.3 Comp.4 Comp.5 Comp.6 Comp.7 Comp.8 Comp.9 Comp.10
Comp.11 Comp.12 Comp.13 Comp.14
X1 −0.165 0.301 0.379 0.200 0.111 0.822
X5 0.870 −0.338 −0.331
X12 −0.372 −0.191 0.567 0.416 0.433 0.184 −0.316
X13 −0.383 −0.175 0.136 0.387 −0.345 −0.330 0.645
X14 −0.388 −0.127 −0.114 −0.121 0.123 −0.485 −0.496 −0.528
X15 −0.392 −0.120 0.126 −0.205 −0.523 0.490 0.362 0.346
X16 −0.388 −0.106 0.107 −0.420 0.250 −0.718 −0.227
X17 −0.381 0.165 −0.489 0.513 −0.339 0.428
X18 −0.135 0.383 −0.173 −0.362 −0.226 −0.201 0.749
X19 −0.117 0.408 −0.201 −0.346 −0.150 0.407 −0.280 −0.578 0.110 0.147 0.125
X20 −0.128 0.392 −0.122 −0.245 0.239 −0.108 0.785 −0.153 0.145 −0.125
X21 −0.117 0.349 0.579 −0.499 −0.462 0.124 −0.116
X22 −0.114 0.304 0.609 0.193 0.604 0.164 −0.253
X23 −0.106 0.323 0.367 −0.658 −0.411 −0.181 −0.316
Comp.1 Comp.2 Comp.3 Comp.4 Comp.5 Comp.6 Comp.7 Comp.8 Comp.9 Comp.10
Comp.11 Comp.12 Comp.13 Comp.14
SS loadings 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000
1.000 1.000 1.000 1.000
Proportion Var 0.071 0.071 0.071 0.071 0.071 0.071 0.071 0.071 0.071 0.071
0.071 0.071 0.071 0.071
Cumulative Var 0.071 0.143 0.214 0.286 0.357 0.429 0.500 0.571 0.643 0.714
0.786 0.857 0.929 1.000
> pca1
Call:
princomp(x = df, cor = T, scores = T)
Standard deviations:
Comp.1 Comp.2 Comp.3 Comp.4 Comp.5 Comp.6 Comp.7 Comp.8 Comp.9 Comp.10
```

```
2.43 1.31 1.02 0.96 0.94 0.93 0.88 0.85 0.84 0.51
Comp.11 Comp.12 Comp.13 Comp.14
0.27 0.20 0.16 0.15
14 variables and 30000 observations.
```

The following graph indicates the percentage variance explained by the principal components:

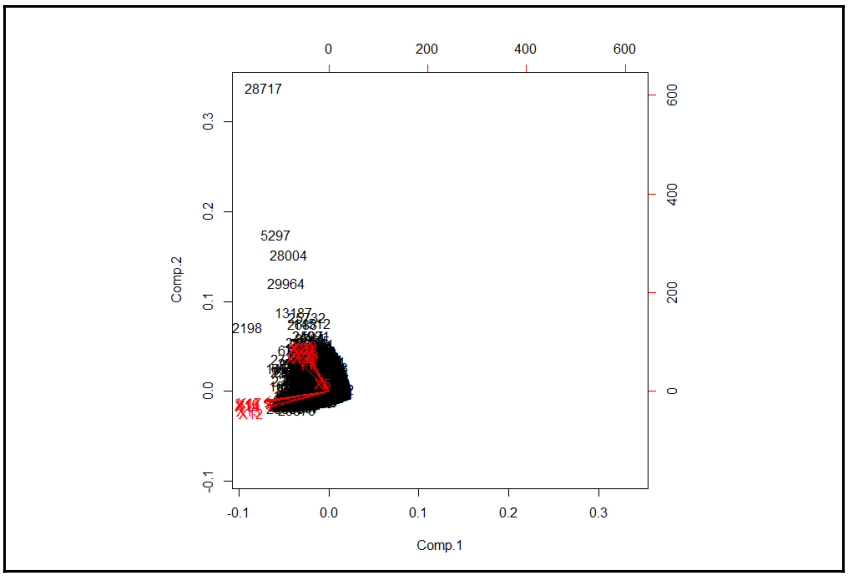


Following scree plot shows how many principal components we should retain for the dataset:



From the preceding scree plot, it is concluded that the first principal component has the highest variance, then second and third respectively. From the graph, three principal components have higher variance than the other principal components.

Following biplot indicates the existence of principal components in a n-dimensional space:



The diagonal elements of the covariance matrix are the variances of the variables; the off-diagonal variables are covariance between different variables:

```
> diag(cov(df))
X1 X5 X12 X13 X14 X15 X16 X17 X18 X19 X20
1.7e+10 8.5e+01 5.4e+09 5.1e+09 4.8e+09 4.1e+09 3.7e+09 3.5e+09 2.7e+08
5.3e+08 3.1e+08
X21 X22 X23
2.5e+08 2.3e+08 3.2e+08
```

The interpretation of scores is little bit tricky because they do not have any meaning until and unless you use them to plot on a straight line as defined by the eigen vector. PC scores are the co-ordinates of each point with respect to the principal axis. Using eigen values you can extract eigen vectors which describe a straight line to explain the PCs. The PCA is a form of multidimensional scaling as it represents data in a lower dimension without losing much information about the variables. The component scores are basically the linear combination of loading times the mean centred scaled dataset.

When we deal with multivariate data, it is really difficult to visualize and build models around it. In order to reduce the features, PCA is used; it reduces the dimensions so that we can plot, visualize, and predict the future, in a lower dimension. The principal components are orthogonal to each other, which means that they are uncorrelated. At the data pre-processing stage, the scaling function decides what type of input data matrix is required. If the mean centring approach is used as transformation then covariance matrix should be used as input data for PCA. If a scaling function does a standard z-score transformation, assuming standard deviation is equal to `1`, then correlation matrix should be used as input data.

Now the question is where to apply which transformation. If the variables are highly skewed then z-score transformation and PCA based on correlation should be used. If the input data is approximately symmetric then mean centring approach and PCA based on covariance should be applied.

Now, let's look at the principal component scores:

```
> #scores of the components
> pca1$scores[1:10,]
Comp.1 Comp.2 Comp.3 Comp.4 Comp.5 Comp.6 Comp.7 Comp.8 Comp.9 Comp.10
Comp.11 Comp.12 Comp.13 Comp.14
[1,] 1.96 -0.54 -1.330 0.1758 -0.0175 0.0029 0.013 -0.057 -0.22 0.020
0.0169 0.0032 -0.0082 0.00985
[2,] 1.74 -0.22 -0.864 0.2806 -0.0486 -0.1177 0.099 -0.075 0.29 -0.073
-0.0055 -0.0122 0.0040 0.00072
[3,] 1.22 -0.28 -0.213 0.0082 -0.1269 -0.0627 -0.014 -0.084 -0.28 -0.016
0.1125 0.0805 0.0413 -0.05711
[4,] 0.54 -0.67 -0.097 -0.2924 -0.0097 0.1086 -0.134 -0.063 -0.60 0.144
0.0017 -0.1381 -0.0183 -0.05794
[5,] 0.85 0.74 1.392 -1.6589 0.3178 0.5846 -0.543 -1.113 -1.24 -0.038
-0.0195 -0.0560 0.0370 -0.01208
[6,] 0.54 -0.71 -0.081 -0.2880 -0.0924 0.1145 -0.194 -0.015 -0.58 0.505
0.0276 -0.1848 -0.0114 -0.14273
[7,] -15.88 -0.96 -1.371 -1.1334 0.3062 0.0641 0.514 0.771 0.99 -2.890
-0.4219 0.4631 0.3738 0.32748
[8,] 1.80 -0.29 -1.182 0.4394 -0.0620 -0.0440 0.076 -0.013 0.26 0.046
0.0484 0.0584 0.0268 -0.04504
[9,] 1.41 -0.21 -0.648 0.1902 -0.0653 -0.0658 0.040 0.122 0.33 -0.028
-0.0818 0.0296 -0.0605 0.00106
[10,] 1.69 -0.18 -0.347 -0.0891 0.5969 -0.2973 -0.453 -0.140 -0.74 -0.137
0.0248 -0.0218 0.0028 0.00423
```

Larger eigen values indicate larger variation in a dimension. The following script shows how to compute eigen values and eigen vectors from the correlation matrix:

```
> eigen(cor(df),TRUE)$values
```

```
[1] 5.919 1.716 1.045 0.925 0.884 0.873 0.780 0.727 0.707 0.264 0.071 0.041
0.025 0.023
head(eigen(cor(df),TRUE)$vectors)
[,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13]
[1,] -0.165 0.301 0.379 0.2004 -0.035 -0.078 0.1114 0.0457 0.822 -0.0291
-0.00617 -0.0157 0.00048
[2,] -0.033 0.072 0.870 -0.3385 0.039 0.071 -0.0788 -0.0276 -0.331 -0.0091
0.00012 0.0013 -0.00015
[3,] -0.372 -0.191 0.034 0.0640 -0.041 -0.044 0.0081 -0.0094 -0.010 0.5667
0.41602 0.4331 0.18368
[4,] -0.383 -0.175 0.002 -0.0075 -0.083 -0.029 -0.0323 0.1357 -0.017 0.3868
0.03836 -0.3452 -0.32953
[5,] -0.388 -0.127 -0.035 -0.0606 -0.114 0.099 -0.1213 -0.0929 0.019 0.1228
-0.48469 -0.4957 0.08662
[6,] -0.392 -0.120 -0.034 -0.0748 -0.029 0.014 0.1264 -0.0392 -0.019
-0.2052 -0.52323 0.4896 0.36210
[,14]
[1,] 0.0033
[2,] 0.0011
[3,] -0.3164
[4,] 0.6452
[5,] -0.5277
[6,] 0.3462
```

The standard deviation of the individual principal components and the mean of all the principal components is as follows:

```
> pca1$sdev
Comp.1 Comp.2 Comp.3 Comp.4 Comp.5 Comp.6 Comp.7 Comp.8 Comp.9 Comp.10
Comp.11 Comp.12 Comp.13 Comp.14
2.43 1.31 1.02 0.96 0.94 0.93 0.88 0.85 0.84 0.51 0.27 0.20 0.16 0.15
> pca1$center
X1 X5 X12 X13 X14 X15 X16 X17 X18 X19 X20 X21 X22 X23
-9.1e-12 -1.8e-15 -6.8e-13 -3.0e-12 4.2e-12 4.1e-12 -1.4e-12 5.1e-13
7.0e-14 4.4e-13 2.9e-13 2.7e-13 -2.8e-13 -3.9e-13
```

Using the normalized data, and with no correlations table as an input, the following result is obtained. There is no difference in the output as we got the `pca2` model in comparison to `pca1`:

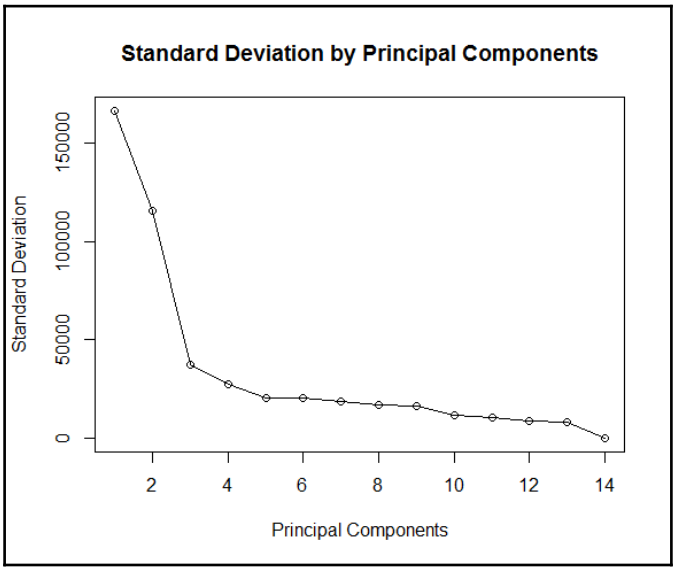
```
> pca2<-princomp(dn)
> summary(pca2)
Importance of components:
Comp.1 Comp.2 Comp.3 Comp.4 Comp.5 Comp.6 Comp.7 Comp.8 Comp.9 Comp.10
Comp.11 Comp.12 Comp.13 Comp.14
Standard deviation 1.7e+05 1.2e+05 3.7e+04 2.8e+04 2.1e+04 2.0e+04 1.9e+04
1.7e+04 1.6e+04 1.2e+04 1.0e+04 8.8e+03 8.2e+03 9.1e+00
Proportion of Variance 6.1e-01 3.0e-01 3.1e-02 1.7e-02 9.4e-03 9.0e-03
```

```
7.5e-03 6.4e-03 5.8e-03 3.0e-03 2.4e-03 1.7e-03 1.5e-03 1.8e-09
Cumulative Proportion 6.1e-01 9.1e-01 9.4e-01 9.5e-01 9.6e-01 9.7e-01
9.8e-01 9.9e-01 9.9e-01 9.9e-01 1.0e+00 1.0e+00 1.0e+00 1.0e+00
```

The next screen plot indicates that three principal components constitute most of the variation in the dataset. The *x* axis shows the principal components and the *y* axis shows the standard deviation (*variance = square of the standard deviation*).

The following script shows the plot being created:

```
> result<-round(summary(pca2)[1]$sdev,0)
> #scree plot
> plot(result, main = "Standard Deviation by Principal Components",
+ xlab="Principal Components",ylab="Standard Deviation",type='o')
```



Using the `prcomp()` function, which uses the SVD method to perform dimensionality reduction can be analysed as follows. `prcomp()` method accepts raw dataset as input and it has a built in argument which requires to make the scaling as true:

```
> pca<-prcomp(df,scale. = T)
> summary(pca)
Importance of components:
PC1 PC2 PC3 PC4 PC5 PC6 PC7 PC8 PC9 PC10
Standard deviation 2.433 1.310 1.0223 0.9617 0.9400 0.9342 0.8829 0.8524
0.8409 0.5142
Proportion of Variance 0.423 0.123 0.0746 0.0661 0.0631 0.0623 0.0557
```

```
0.0519 0.0505 0.0189
Cumulative Proportion 0.423 0.545 0.6200 0.6861 0.7492 0.8115 0.8672 0.9191
0.9696 0.9885
PC11 PC12 PC13 PC14
Standard deviation 0.26648 0.20263 0.15920 0.15245
Proportion of Variance 0.00507 0.00293 0.00181 0.00166
Cumulative Proportion 0.99360 0.99653 0.99834 1.00000
```

From the result we can see that there is little variation in the two methods. The SVD approach also shows that the first 8 components explain 91.91% variation in the dataset. From the documentation of `prcomp` and `princomp`, there is no difference in the type of PCA, but there is a difference in the method used to calculate PCA. The two methods are spectral decomposition and singular decomposition.

In spectral decomposition, as shown in the `princomp` function, the calculation is done using eigen on the correlation or covariance matrix. Using the `prcomp` method, the calculation is done by the SVD method.

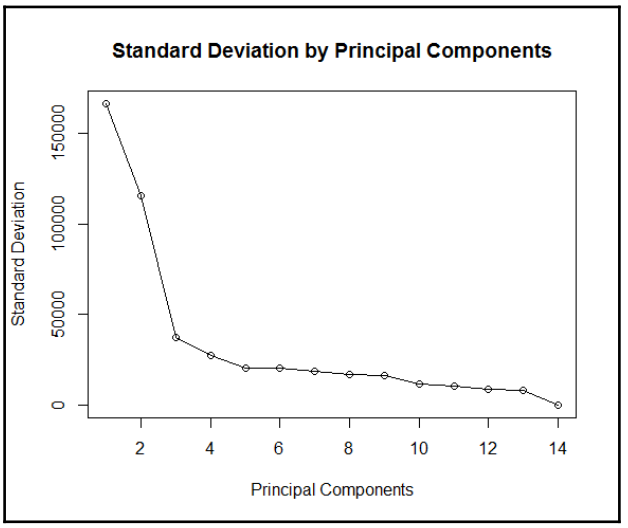
To get the rotation matrix, which is equivalent to the loadings matrix in the `princomp()` method, we can use the following script:

```
> summary(pca)$rotation
PC1 PC2 PC3 PC4 PC5 PC6 PC7 PC8 PC9 PC10 PC11
X1 0.165 0.301 −0.379 0.2004 −0.035 0.078 −0.1114 −0.04567 0.822 −0.0291
0.00617
X5 0.033 0.072 −0.870 −0.3385 0.039 −0.071 0.0788 0.02765 −0.331 −0.0091
−0.00012
X12 0.372 −0.191 −0.034 0.0640 −0.041 0.044 −0.0081 0.00937 −0.010 0.5667
−0.41602
X13 0.383 −0.175 −0.002 −0.0075 −0.083 0.029 0.0323 −0.13573 −0.017 0.3868
−0.03836
X14 0.388 −0.127 0.035 −0.0606 −0.114 −0.099 0.1213 0.09293 0.019 0.1228
0.48469
X15 0.392 −0.120 0.034 −0.0748 −0.029 −0.014 −0.1264 0.03915 −0.019 −0.2052
0.52323
X16 0.388 −0.106 0.034 −0.0396 0.107 0.099 0.0076 0.04964 −0.024 −0.4200
−0.06824
X17 0.381 −0.094 0.018 0.0703 0.165 −0.070 −0.0079 −0.00015 −0.059 −0.4888
−0.51341
X18 0.135 0.383 0.173 −0.3618 −0.226 −0.040 0.2010 −0.74901 −0.020 −0.0566
−0.04763
X19 0.117 0.408 0.201 −0.3464 −0.150 −0.407 0.2796 0.57842 0.110 0.0508
−0.14725
X20 0.128 0.392 0.122 −0.2450 0.239 0.108 −0.7852 0.06884 −0.153 0.1449
−0.00015
X21 0.117 0.349 0.062 0.0946 0.579 0.499 0.4621 0.07712 −0.099 0.1241
0.11581
```

```
X22 0.114 0.304 -0.060 0.6088 0.193 -0.604 -0.0143 -0.16435 -0.253 0.0601
0.09944
X23 0.106 0.323 -0.050 0.3672 -0.658 0.411 -0.0253 0.18089 -0.316 -0.0992
-0.03495
PC12 PC13 PC14
X1 -0.0157 0.00048 -0.0033
X5 0.0013 -0.00015 -0.0011
X12 0.4331 0.18368 0.3164
X13 -0.3452 -0.32953 -0.6452
X14 -0.4957 0.08662 0.5277
X15 0.4896 0.36210 -0.3462
X16 0.2495 -0.71838 0.2267
X17 -0.3386 0.42770 -0.0723
X18 0.0693 0.04488 0.0846
X19 0.0688 -0.03897 -0.1249
X20 -0.1247 -0.02541 0.0631
X21 -0.0010 0.08073 -0.0423
X22 0.0694 -0.09520 0.0085
X23 -0.0277 0.01719 -0.0083
```

The rotated dataset can be retrieved by using argument `x` from the summary result of the `prcomp()` method:

```
> head(summary(pca)$x)
PC1 PC2 PC3 PC4 PC5 PC6 PC7 PC8 PC9 PC10 PC11 PC12
[1,] -1.96 -0.54 1.330 0.1758 -0.0175 -0.0029 -0.013 0.057 -0.22 0.020
-0.0169 0.0032
[2,] -1.74 -0.22 0.864 0.2806 -0.0486 0.1177 -0.099 0.075 0.29 -0.073
0.0055 -0.0122
[3,] -1.22 -0.28 0.213 0.0082 -0.1269 0.0627 0.014 0.084 -0.28 -0.016
-0.1125 0.0805
[4,] -0.54 -0.67 0.097 -0.2924 -0.0097 -0.1086 0.134 0.063 -0.60 0.144
-0.0017 -0.1381
[5,] -0.85 0.74 -1.392 -1.6588 0.3178 -0.5846 0.543 1.113 -1.24 -0.038
0.0195 -0.0560
[6,] -0.54 -0.71 0.081 -0.2880 -0.0924 -0.1145 0.194 0.015 -0.58 0.505
-0.0276 -0.1848
PC13 PC14
[1,] -0.0082 -0.00985
[2,] 0.0040 -0.00072
[3,] 0.0413 0.05711
[4,] -0.0183 0.05794
[5,] 0.0370 0.01208
[6,] -0.0114 0.14273
> biplot(prcomp(df,scale. = T))
```



The orthogonal red lines indicate the principal components that explain the entire dataset.

Having discussed various methods of variable reduction, what is going to be the output; the output should be a new dataset, in which all the principal components will be uncorrelated with each other. This dataset can be used for any task such as classification, regression, or clustering, among others. From the results of the principal component analysis how we get the new dataset; let's have a look at the following script:

```
> #calculating Eigen vectors
> eig<-eigen(cor(df))
> #Compute the new dataset
> eigvec<-t(eig$vectors) #transpose the eigen vectors
> df_scaled<-t(dn) #transpose the adjusted data
> df_new<-eigvec %*% df_scaled
> df_new<-t(df_new)
> colnames(df_new)<-c("PC1","PC2","PC3","PC4",
+ "PC5","PC6","PC7","PC8",
+ "PC9","PC10","PC11","PC12",
+ "PC13","PC14")
> head(df_new)
PC1 PC2 PC3 PC4 PC5 PC6 PC7 PC8 PC9 PC10 PC11 PC12
[1,] 128773 -19470 -49984 -27479 7224 10905 -14647 -4881 -113084 -2291 1655
2337
[2,] 108081 11293 -12670 -7308 3959 2002 -3016 -1662 -32002 -9819 -295 920
[3,] 80050 -7979 -24607 -11803 2385 3411 -6805 -3154 -59324 -1202 7596 7501
[4,] 37080 -39164 -41935 -23539 2095 9931 -15015 -4291 -92461 12488 -11
-7943
```

```
[5,] 77548 356 -50654 -38425 7801 20782 -19707 -28953 -88730 -12624 -9028
-4467
[6,] 34793 -42350 -40802 -22729 -2772 9948 -17731 -2654 -91543 37090 2685
-10960
PC13 PC14
[1,] -862 501
[2,] -238 -97
[3,] 2522 -4342
[4,] -1499 -4224
[5,] 3331 -9412
[6,] -1047 -10120
```

Parametric approach to dimension reduction

To some extent, we touched upon model-based dimension reduction in `Chapter 4`, *Regression with Automobile Data*, *Logistic regression*, where we tried to implement regression modelling and in that process we tried to reduce the data dimension by applying **Akaike Information Criteria** (**AIC**). **Bayesian Information Criteria** (**BIC**) such as AIC can also be used to reduce data dimensions. As far as the model-based method is concerned, there are two approaches:

- **The forward selection method**: In forward selection method, one variable at a time is added to the model and the model goodness of fit statistics and error are computed. If the addition of a new dimension reduces error and increases the model goodness of fit, then that dimension is retained by the model, else that dimension is removed from the model. This is applicable across different supervised based algorithms, such as random forest, logistic regression, neural network, and support vector machine-based implementations. The process of feature selection continues till all the variables get tested.
- **The backward selection method**: In backward selection method, the model starts with all the variables together. Then one variable is deleted from the model and the model goodness of fit statistics and error (any loss function pre-defined) is computed. If the deletion of a new dimension reduces error and increases the model goodness of fit, then that dimension is dropped from the model, else that dimension is kept by the model.

The problem in this chapter we are discussing is a case of supervised learning classification, where the dependent variable is default or no default. The logistic regression method, as we discussed in `Chapter 4`, *Regression with Automobile Data*, *Logistic regression*, uses a step wise dimension reduction procedure to remove unwanted variables from the model. The same exercise can be done on the dataset we discussed in this chapter.

Apart from the standard methods of data dimension reduction, there are some not so important methods available which can be considered, such as missing value estimation method. In a large data set with many dimension sparsity problem will be a common scenario, before applying any formal process of dimensionality reduction, if we can apply the missing value percentage calculation method on the dataset, we can drop many variables. The threshold to drop the variables failing to meet the minimum missing percentage has to be decided by the analyst.

References

Yeh, I. C., & Lien, C. H. (2009). The comparisons of data mining techniques for the predictive accuracy of probability of default of credit card clients. *Expert Systems with Applications*, 36(2), 2473-2480.

Summary

In this chapter, we discussed various methods of performing dimensionality reduction in a sample dataset. Removing redundant features not only improves model accuracy, but also saves computational effort and time. From business user's point of with less number of dimensions, it is more intuitive to build strategies than to focus on large number of features. We discussed which technique to use where and what the data requirement for each of the methods is. The reduction in dimensions also provides meaningful insights into large datasets. In the next chapter, we are going to learn about neural network methods for classification, regression, and time series forecasting.

9
Applying Neural Network to Healthcare Data

Neural-network-based models are gradually becoming the backbone of artificial intelligence and machine learning implementations. The future of data mining will be governed by the usage of artificial neural-network-based advanced modeling techniques. One obvious question: why is neural network gaining so much importance recently though it was invented in 1950s? Borrowed from the computer science domain, a neural network can be defined as a parallel information processing system where the inputs are connected with each other like neurons in the human brain to transmit information so that activities such as face recognition, image recognition, and so on can be performed. In this chapter, we are going to learn about application of neural-network-based methods in various data mining tasks such as classification, regression, time series forecasting, and feature reduction. **Artificial Neural Network** (**ANN**) functions in a way that is similar to the human brain, where billions of neurons link to each other for information processing and insight generation.

In this chapter, you will learn about various types of neural networks, methods, and variants of neural networks with different functions to control the training of artificial neural networks in performing standard data mining tasks such as:

- Prediction of real valued output using regression-based methods
- Prediction of output levels in a classification-based task
- Forecasting future values of a numerical attribute based on historical data
- Compressing features to recognize important ones in order to perform prediction or classification

Introduction to neural networks

The brain's biological network provides the basis for connecting elements in a real-life scenario for information processing and insight generation. It's a hierarchy of neurons connected through layers, where the output of one layer becomes the input for another layer; information passes from one layer to another layer as weights. The weights associated with each neuron contain insights so that the recognition and reasoning become easier for the next level. Artificial neural network is a very popular and effective method that consists of layers associated with weights. The association between different layers is governed by a mathematical equation that passes information from one layer to the other. In fact, a bunch of mathematical equations are at work inside one artificial neural network model. The following graph shows the general architecture for a neural-network-based model:

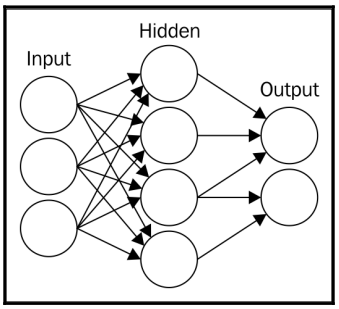


Figure 1

In the preceding graph, there are three layers—**Input**, **Hidden** and **Output** layer—which are the core of any neural network-based architecture. ANNs are a powerful technique used to solve many real-world problems such as classification, regression, and feature selection. ANNs have the ability to learn from new experiences in the form of new input data in order to improve the performance of classification- or regression-based tasks and to adapt themselves to changes in the input environment. Each circle in the preceding figure represents a neuron.

There are different variants of neural networks that are used in multiple different scenarios; we are going to explain a few of them conceptually in this chapter, and also their usage in practical applications:

- **Single hidden layer neural network**: This is the simplest form of neural network, as shown in the preceding figure. In it, there is only one hidden layer.
- **Multiple hidden layer neural networks**: In this form, more than one hidden layer will connect the input data to the output data. The complexity of calculation

increases in this form as it requires more computational power in the system to process information.

- **Feed forward neural networks**: In this form of neural network architecture, the information passed is one-directional from one layer to another layer; there is no iteration from the first level of learning.
- **Back propagation neural networks**: In this form of neural network, there are two important steps. Feed forward works by passing information from the input to the hidden and from the hidden to the output layer; secondly, it calculates the error and propagates it back to the previous layers.

The feed-forward neural network model architecture is shown in the following figure, and backpropagation method is explained in *Figure 3*:

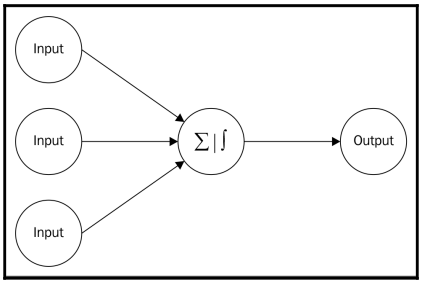


Figure 2

In the following figure, the red-colored arrows indicate information that has not passed through the output layer, and is again fed back to the input layer in terms of errors:

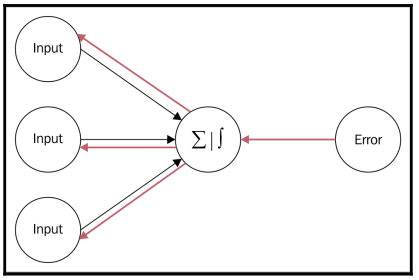


Figure 3

Having displayed the general architecture for different types of neural networks, let's visit the underlying math behind them.

Understanding the math behind the neural network

The neurons present in different layers–input, hidden, and output–are interconnected through a mathematical function called activation function, as displayed in *Figure 1*. There are different variants of the activation function, which are explained as follows. Understanding the activation function will help in implementation of the neural network model for better accuracy:

- **Sigmoid function**: This is frequently used by professionals in data mining and analytics, as it is easier to explain and implement too. The equation is mentioned here:

$$f(x) = \frac{1}{1 + e^{-\beta x}}$$

  The sigmoid function, also known as the logistic function, is mostly used to transform the input data from the input layer to the mapping layer, or the hidden layer.

- **Linear function**: This is one of the simple functions typically used to transfer information from the de-mapping layer to the output layer. The formula is as follows:

$$f(x)=x$$

- **Gaussian function**: Gaussian functions are bell-shaped curves that are applicable for continuous variables, where the objective is to classify the output into multiple classes:

$$f(x) = \frac{1}{\sqrt{2\pi\sigma}} e^{\frac{-(x-\mu)^2}{2\sigma^2}}$$

- **Hyperbolic tangent function**: This is another variant of transformation function; it is used to transform information from the mapping layer to the hidden layer:

$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

- **Log sigmoid transfer function**: The following formula explains the log sigmoid transfer function used in mapping the input layer to the hidden layer:

$$f(x) = \log\left(\frac{1}{1 + e^{-\beta x}}\right)$$

- **Radial basis function**: This is another activation function; it is used to transfer information from the de-mapping layer to the output layer:

$$f(x) = \sum_{j-1}^{m} w_j h_j(x)$$

Different types of transfer functions, as previously discussed, can be interchangeable in neural network architectures. They can be used in different stages such as input to hidden, hidden to output, and so on, to improve the model accuracy.

Neural network implementation in R

R programming for statistical computing provides three different libraries to perform the neural network model for various tasks. These three are `nnet`, `neuralnet`, and `rsnns`. In this chapter, we will use `ArtPiece_1.csv` and two libraries, `nnet` and `neuralnet`, to perform various tasks. The syntax for neural networks in those two libraries can be explained as follows.

The `neuralnet` library depends on two other libraries, `grid` and `mass`; while installing the `neuralnet` library, you have to make sure that these two dependency libraries are installed properly. In fitting a neural network model, the desired level of accuracy in a model defines the required number of hidden layers with the number of neurons in it; the number of hidden layers increases as the level of complexity increases. This library provides an option for training of neural networks using backpropagation, resilient backpropagation with (Riedmiller, 1994) or without weight backtracking (Riedmiller and Braun, 1993), or the modified globally convergent version by Anastasiadis et al. (2005). The package allows flexible settings through custom choice of error and activation functions. Now let's look at the syntax and the components in it that dictate the accuracy of the model:

Formula	**To define the input and output relationship**
`Data`	Dataset with the input and output variables.
`Hidden`	A vector specifying the number of hidden layers in it. For example, (10, 5, 2) means 10 hidden neurons in the first layer, 5 in the second, and 2 in the third layer.
`Stepmax`	Maximum number of steps to be used.
`Rep`	Number of iterations.
`Startweights`	Random weights for the connections.
`Algorithm`	The algorithm to be used to fit a neural network `object.'backprop'`- refers to backpropagation. The `'rprop+'` and `'rprop-'` refer to resilient backpropagation with and without weight backtracking respectively, while `'sag'` and `'slr'` refer to the smallest absolute gradient and smallest learning rate.
`Err.fct`	Two methods: **sum squared error** (SSE) for regression-based prediction and **cross entropy** (**CE**) for classification-based problems.
`Act.fct`	`"logistic"` and `"tanh"` are possible for the logistic function and tangent hyperbolicus.
`Linear.output`	If no activation function is applied to the output layer, it has to be TRUE.
`Likelihood`	If the error function is equal to the negative log-likelihood function, the information criteria AIC and BIC will be calculated.

Table 1: neuralnet syntax description

Table 2 shows the syntax of the `nnet` library, which can also be used to create neural-network-based models:

Formula	**A formula defining the input-output relationship**
`X`	A matrix or data frame containing the input variables.
`Y`	A matrix or data frame containing the output variables.
`Weights`	Weights.
`Size`	Number of units in the hidden layer.
`Data`	Dataset.
`Na.action`	If NAs found what should be taken.
`Entropy`	Switch for entropy (is equal to maximum conditional likelihood) fitting. Default by leastsquares.
`Softmax`	Switch for softmax (log-linear model) and maximum conditional likelihood fitting. `Linout`, `Entropy`, `Softmax`, and `Censored` are mutually exclusive.
`Decay`	Parameter for weight decay.
`Maxit`	Maximum number of iterations.
`Trace`	Switch for tracing optimization.

Table 2: nnet syntax description

Having discussed the syntax in *Table 1* for both the libraries, let's have a look at the dataset to be used for prediction and classification-based tasks:

```
> library(neuralnet)
Loading required package: grid
Loading required package: MASS
Warning message:
package 'neuralnet' was built under R version 3.2.3
```

The structure function for the dataset shows the types of variables and the size of the dataset we are going to use for this chapter:

```
> art<- read.csv("ArtPiece_1.csv")
> str(art)
'data.frame': 72983 obs. of 26 variables:
$ Cid : int 1 2 3 4 5 6 7 8 9 10 ...
$ Art.Auction.House : Factor w/ 3 levels "Artnet","Christie",..: 3 3 3 3 3
3 3 3 3 3 ...
$ IsGood.Purchase : int 0 0 0 0 0 0 0 0 0 0 ...
```

```
$ Critic.Ratings : num 8.9 9.36 7.38 6.56 6.94 ...
$ Buyer.No : int 21973 19638 19638 19638 19638 19638 19638 19638 21973
21973 ...
$ Zip.Code : int 33619 33619 33619 33619 33619 33619 33619 33619 33619
33619 ...
$ Art.Purchase.Date : Factor w/ 517 levels "1/10/2012","1/10/2013",..: 386
386 386 386 386 386 386 386 386 386 ...
$ Year.of.art.piece : Factor w/ 10 levels "01-01-1947","01-01-1948",..: 6 4
5 4 5 4 4 5 7 7 ...
$ Acq.Cost : num 49700 53200 34300 28700 28000 39200 29400 31500 39200
53900 ...
$ Art.Category : Factor w/ 33 levels "Abstract Art Type I",..: 1 13 13 13
30 24 31 30 31 30 ...
$ Art.Piece.Size : Factor w/ 864 levels "10in. X 10in.",..: 212 581 68 837
785 384 485 272 485 794 ...
$ Border.of.art.piece : Factor w/ 133 levels " ","Border 1",..: 2 40 48 48
56 64 74 85 74 97 ...
$ Art.Type : Factor w/ 1063 levels "Type 1","Type 10",..: 1 176 287 398 509
620 731 842 731 953 ...
$ Prominent.Color : Factor w/ 17 levels "Beige","Black",..: 14 16 8 15 15
16 2 16 2 14 ...
$ CurrentAuctionAveragePrice: int 52157 52192 28245 12908 22729 32963 20860
25991 44919 64169 ...
$ Brush : Factor w/ 4 levels "","Camel Hair Brush",..: 2 2 2 2 4 2 2 2 2 2
...
$ Brush.Size : Factor w/ 5 levels "0","1","2","3",..: 2 2 3 2 3 3 3 3 3 2
...
$ Brush.Finesse : Factor w/ 4 levels "Coarse","Fine",..: 2 2 1 2 1 1 1 1 1
2 ...
$ Art.Nationality : Factor w/ 5 levels "American","Asian",..: 3 1 1 1 1 3 3
1 3 1 ...
$ Top.3.artists : Factor w/ 5 levels "MF Hussain","NULL",..: 3 1 1 1 4 3 3
4 3 4 ...
$ CollectorsAverageprice : Factor w/ 13193 levels "#VALUE!","0",..: 11433
11808 7802 4776 7034 8536 7707 7355 9836 483 ...
$ GoodArt.check : Factor w/ 3 levels "NO","NULL","YES": 2 2 2 2 2 2 2 2 2 2
...
$ AuctionHouseGuarantee : Factor w/ 3 levels "GREEN","NULL",..: 2 2 2 2 2 2
2 2 2 2 ...
$ Vnst : Factor w/ 37 levels "AL","AR","AZ",..: 6 6 6 6 6 6 6 6 6 6 ...
$ Is.It.Online.Sale : int 0 0 0 0 0 0 0 0 0 0 ...
$ Min.Guarantee.Cost : int 7791 7371 9723 4410 7140 4158 3731 5775 3374
11431 ...
```

Neural networks for prediction

The use of neural networks for prediction requires the dependent/target/output variable to be numeric, and all the input/independent/feature variables can be of any type. From the `ArtPiece` dataset, we are going to predict what is going to be the current auction average price based on all the parameters available. Before applying a neural-network-based model, it is important to preprocess the data, by excluding the missing values and any transformation if required; hence, let's preprocess the data:

```
library(neuralnet)
art<- read.csv("ArtPiece_1.csv")
str(art)
#data conversion for categorical features
art$Art.Auction.House<-as.factor(art$Art.Auction.House)
art$IsGood.Purchase<-as.factor(art$IsGood.Purchase)
art$Art.Category<-as.factor(art$Art.Category)
art$Prominent.Color<-as.factor(art$Prominent.Color)
art$Brush<-as.factor(art$Brush)
art$Brush.Size<-as.factor(art$Brush.Size)
art$Brush.Finesse<-as.factor(art$Brush.Finesse)
art$Art.Nationality<-as.factor(art$Art.Nationality)
art$Top.3.artists<-as.factor(art$Top.3.artists)
art$GoodArt.check<-as.factor(art$GoodArt.check)
art$AuctionHouseGuarantee<-as.factor(art$AuctionHouseGuarantee)
art$Is.It.Online.Sale<-as.factor(art$Is.It.Online.Sale)
#data conversion for numeric features
art$Critic.Ratings<-as.numeric(art$Critic.Ratings)
art$Acq.Cost<-as.numeric(art$Acq.Cost)
art$CurrentAuctionAveragePrice<-as.numeric(art$CurrentAuctionAveragePrice)
art$CollectorsAverageprice<-as.numeric(art$CollectorsAverageprice)
art$Min.Guarantee.Cost<-as.numeric(art$Min.Guarantee.Cost)
#removing NA, Missing values from the data
fun1<-function(x){
ifelse(x=="#VALUE!",NA,x)
}
art<-as.data.frame(apply(art,2,fun1))
art<-na.omit(art)
#keeping only relevant variables for prediction
art<-art[,c("Art.Auction.House","IsGood.Purchase","Art.Category",
"Prominent.Color","Brush","Brush.Size","Brush.Finesse",
"Art.Nationality","Top.3.artists","GoodArt.check",
"AuctionHouseGuarantee","Is.It.Online.Sale","Critic.Ratings",
"Acq.Cost","CurrentAuctionAveragePrice","CollectorsAverageprice",
"Min.Guarantee.Cost")]
#creating dummy variables for the categorical variables
library(dummy)
art_dummy<-
```

```
dummy(art[,c("Art.Auction.House","IsGood.Purchase","Art.Category",
"Prominent.Color","Brush","Brush.Size","Brush.Finesse",
"Art.Nationality","Top.3.artists","GoodArt.check",
"AuctionHouseGuarantee","Is.It.Online.Sale")],int=F)
art_num<-art[,c("Critic.Ratings",
"Acq.Cost","CurrentAuctionAveragePrice","CollectorsAverageprice",
"Min.Guarantee.Cost")]
art<-cbind(art_num,art_dummy)
## 70% of the sample size
smp_size <- floor(0.70 * nrow(art))
## set the seed to make your partition reproductible
set.seed(123)
train_ind <- sample(seq_len(nrow(art)), size = smp_size)
train <- art[train_ind, ]
test <- art[-train_ind, ]
fun2<-function(x){
as.numeric(x)
}
train<-as.data.frame(apply(train,2,fun2))
test<-as.data.frame(apply(test,2,fun2))
```

In the training dataset, there are 50,867 observations and 17 variables, and in the test dataset, there are 21,801 observations and 17 variables. The current auction average price is the dependent variable for prediction, using only four other numeric variables as features:

```
>fit<- neuralnet(formula = CurrentAuctionAveragePrice ~ Critic.Ratings +
Acq.Cost + CollectorsAverageprice + Min.Guarantee.Cost, data = train,
hidden = 15, err.fct = "sse", linear.output = F)
> fit
Call: neuralnet(formula = CurrentAuctionAveragePrice ~ Critic.Ratings +
Acq.Cost + CollectorsAverageprice + Min.Guarantee.Cost, data = train,
hidden = 15, err.fct = "sse", linear.output = F)
1 repetition was calculated.
Error Reached Threshold Steps
1 54179625353167 0.004727494957 23
```

A summary of the main results of the model is provided by `result.matrix`. A snapshot of the result.matrix is given as follows:

```
> fit$result.matrix
1
error 54179625353167.000000000000
reached.threshold 0.004727494957
steps 23.000000000000
Intercept.to.1layhid1 -0.100084491816
Critic.Ratings.to.1layhid1 0.686332945444
Acq.Cost.to.1layhid1 0.196864454378
CollectorsAverageprice.to.1layhid1 -0.793174429352
```

```
Min.Guarantee.Cost.to.1layhid1 0.528046199494
Intercept.to.1layhid2 0.973616842194
Critic.Ratings.to.1layhid2 0.839826678316
Acq.Cost.to.1layhid2 0.077798897157
CollectorsAverageprice.to.1layhid2 0.988149246218
Min.Guarantee.Cost.to.1layhid2 -0.385031389636
Intercept.to.1layhid3 -0.008367359937
Critic.Ratings.to.1layhid3 -1.409715725621
Acq.Cost.to.1layhid3 -0.384200569485
CollectorsAverageprice.to.1layhid3 -1.019243809714
Min.Guarantee.Cost.to.1layhid3 0.699876747202
Intercept.to.1layhid4 2.085203047278
Critic.Ratings.to.1layhid4 0.406934874266
Acq.Cost.to.1layhid4 1.121189503896
CollectorsAverageprice.to.1layhid4 1.405748076570
Min.Guarantee.Cost.to.1layhid4 -1.043884892202
Intercept.to.1layhid5 0.862634752109
Critic.Ratings.to.1layhid5 0.814364667751
Acq.Cost.to.1layhid5 0.502879862694
```

If the `error` function is equal to the negative log likelihood function, the error refers to the likelihood as it is used to calculate the Akaike Information Criterion (AIC). We can store the covariate and response data in a matrix:

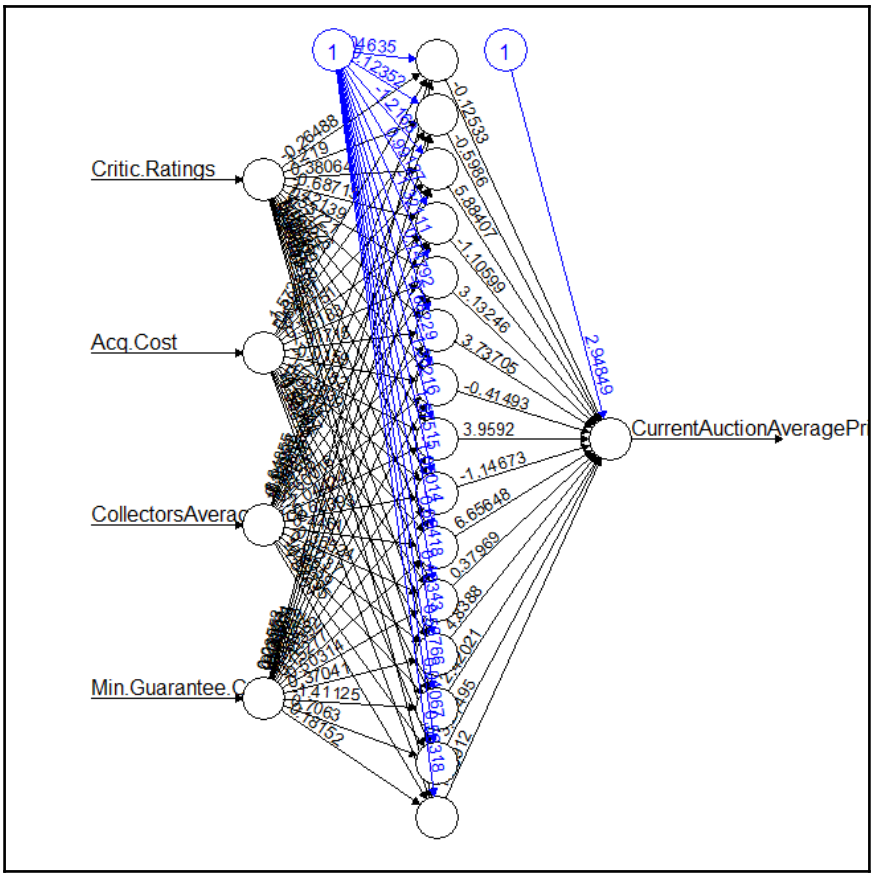
```
> output<-cbind(fit$covariate,fit$result.matrix[[1]])
> head(output)
[,1] [,2] [,3] [,4] [,5]
[1,] 14953 49000 10727 5775 54179625353167
[2,] 35735 38850 9494 12418 54179625353167
[3,] 34751 43750 8738 9611 54179625353167
[4,] 31599 41615 5955 4158 54179625353167
[5,] 10437 34755 8390 4697 54179625353167
[6,] 13177 54670 13024 11921 54179625353167
```

To compare the results of a neural network model, we can use different tuning factors such as changing the algorithm, hidden layer, and learning rate. As an example, only four numeric features were used to generate the prediction; we could have used all the 91 features for prediction of the current auction average price variable. We can also use a different algorithm from the `nnet` library, as follows:

```
> fit<-nnet(CurrentAuctionAveragePrice~Critic.Ratings+Acq.Cost+
+ CollectorsAverageprice+Min.Guarantee.Cost,data=train,
+ size=100)
# weights: 601
initial value 108359809492660.125000
final value 108359250706334.000000
converged
```

```
> fit
a 4-100-1 network with 601 weights
inputs: Critic.Ratings Acq.Cost CollectorsAverageprice Min.Guarantee.Cost
output(s): CurrentAuctionAveragePrice
options were -
```

Both the libraries provide equal results; there is no difference in the model result, but to tune the results further, it is important to look at the model tuning parameters such as learning rate, hidden neurons, and so on. The following graph shows the neural network architecture:



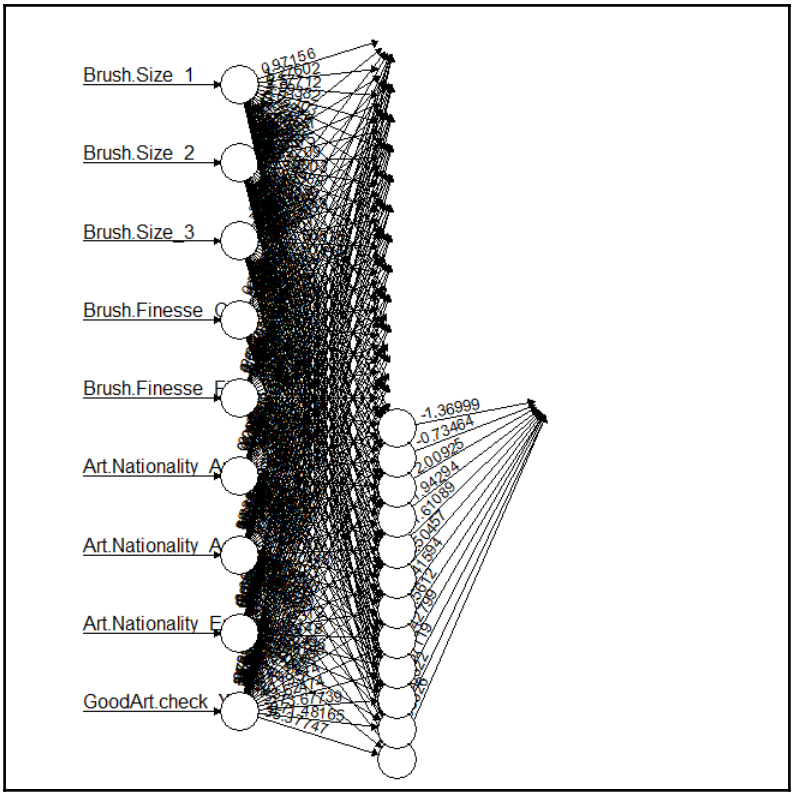
The model for predicting the unseen data points can be implemented using the `compute` function available in the `neuralnet` library, and the `predict` function available in the `nnet` library.

Neural networks for classification

For classification-based projects, the dependent variable can be binary or can have multiple levels, such as credit card fraud detection, classification of customers into different clusters as far as the marketing is concerned, and so on. In the current scenario from the `ArtPiece` dataset, we are trying to predict whether a work of art is a good purchase, or not, by taking a few business-relevant variables. For demo purposes, we have considered only a few features, but other features present in the dataset can be used to generate a better result:

```
> fit<-neuralnet(IsGood.Purchase_1~Brush.Size_1+Brush.Size_2+Brush.Size_3+
+ Brush.Finesse_Coarse+Brush.Finesse_Fine+
+ Art.Nationality_American+Art.Nationality_Asian+
+ Art.Nationality_European+GoodArt.check_YES,data=train[1:2000,],
+ hidden = 25,err.fct = "ce",linear.output = F)
> fit
Call: neuralnet(formula = IsGood.Purchase_1 ~ Brush.Size_1 + Brush.Size_2 +
Brush.Size_3 + Brush.Finesse_Coarse + Brush.Finesse_Fine +
Art.Nationality_American + Art.Nationality_Asian + Art.Nationality_European
+ GoodArt.check_YES, data = train[1:2000, ], hidden = 25, err.fct = "ce",
linear.output = F)
1 repetition was calculated.
Error Reached Threshold Steps
1 666.1522488 0.009864324362 8254
> output<-cbind(fit$covariate,fit$result.matrix[[1]])
> head(output)
[,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
[1,] 1 0 0 0 1 0 0 1 0 666.1522488
[2,] 1 0 0 0 1 1 0 0 0 666.1522488
[3,] 1 0 0 0 1 0 0 1 0 666.1522488
[4,] 0 1 0 1 0 0 0 1 0 666.1522488
[5,] 0 1 0 1 0 1 0 0 0 666.1522488
[6,] 1 0 0 0 1 1 0 0 0 666.1522488
```

The following graph shows the neural network model for classification:



Using another library, `nnet`, for classification-based problems, the following result is derived:

```
> fit.nnet<-
nnet(factor(IsGood.Purchase_1)~Brush.Size_1+Brush.Size_2+Brush.Size_3+
+ Brush.Finesse_Coarse+Brush.Finesse_Fine+
+ Art.Nationality_American+Art.Nationality_Asian+
+ Art.Nationality_European+GoodArt.check_YES,data=train[1:2000,],
+ size=9)
# weights: 100
initial value 872.587818
iter 10 value 684.034783
iter 20 value 667.751170
iter 30 value 667.027963
iter 40 value 666.337669
```

```
iter 50 value 666.156889
iter 60 value 666.138741
iter 70 value 666.137048
iter 80 value 666.136505
final value 666.136439
converged
> fit.nnet
a 9-9-1 network with 100 weights
inputs: Brush.Size_1 Brush.Size_2 Brush.Size_3 Brush.Finesse_Coarse
Brush.Finesse_Fine Art.Nationality_American Art.Nationality_Asian
Art.Nationality_European GoodArt.check_YES
output(s): factor(IsGood.Purchase_1)
options were - entropy fitting
```

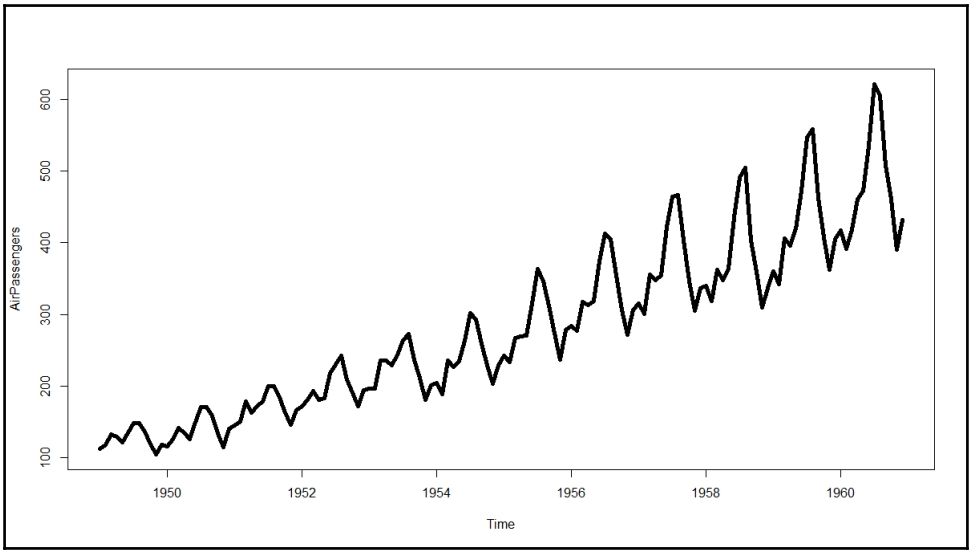
Neural networks for forecasting

Neural networks can also be used to generate forecasts for a time series variable. There is a library called `forecast` in R that deploys feed-forward neural networks with a single hidden layer, and lagged inputs for forecasting univariate time series. For the forecasting example, we have taken an inbuilt dataset available in R called "Air Passengers" to apply the neural network.

The following table reflects the parameter arguments required by the `nnetar` function with the corresponding description of how they are being used in the model:

`X`	**Univariate time series with a time variable**
`p`	Number of non-seasonal lags used as input
`P`	Number of seasonal lags used as input
`Size`	Number of nodes in the hidden layer
`Repeats`	Number of networks to fit with different random settings of weights
`Lambda`	This is known as box-cox transformation parameter
`Xreg`	External regressors used in fitting the model
`Mean`	Point forecasts as mean

The actual time series looks as follows:

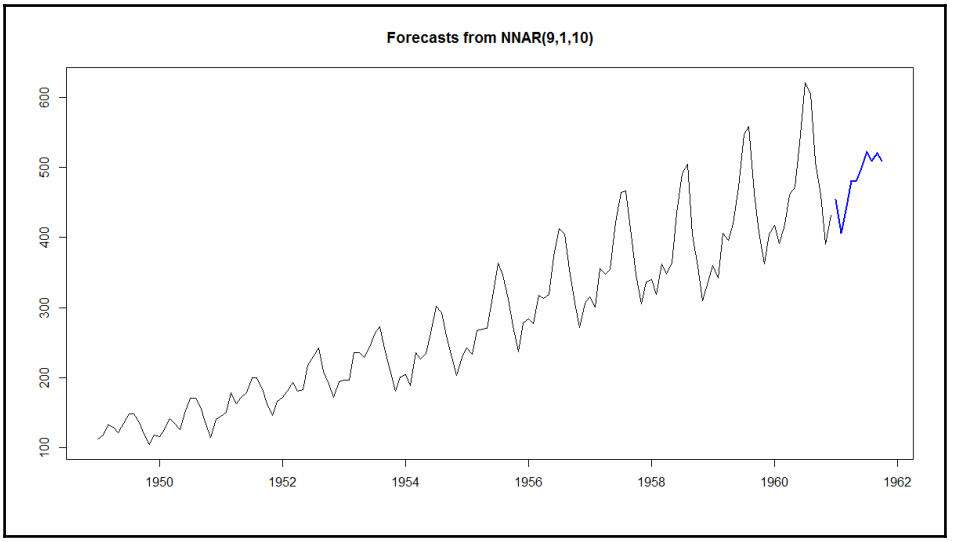



```
> fit<-nnetar(AirPassengers, p=9,P=,size = 10, repeats = 50,lambda = 0)>
plot(forecast(fit,10))
```

A neural-network-based forecasting model generates the following results as an output:

```
> summary(fit)
Length Class Mode
x 144 ts numeric
m 1 -none- numeric
p 1 -none- numeric
P 1 -none- numeric
scale 1 -none- numeric
size 1 -none- numeric
lambda 1 -none- numeric
model 50 nnetarmodels list
fitted 144 ts numeric
residuals 144 ts numeric
lags 10 -none- numeric
series 1 -none- character
method 1 -none- character
call 6 -none- call
```

With a forecast of the next 10 periods, the graph looks as follows:



Merits and demerits of neural networks

The neural network method for performing classification, prediction, and forecasting is still recognized as a black box methodology in different industries. People still provide more importance to logistic regression than neural network because of its complexity in explaining the relationship between the dependent and independent variable.

The limitations of the neural network model can be stated as follows:

- In contrast to decision trees and rule extraction techniques, the knowledge (patterns) "discovered" by neural networks is not represented in a form understandable by humans.
- Knowledge in a trained **neural network** (**NN**) is encoded in its connection weights; hence, NN cannot be used for descriptive data mining (exploration).
- If NN are used for decision making, it is impossible to explain their decisions. Often other techniques have to be combined with NN for explanation.

Here are the merits of neural networks:

- Though it is a bit complex to understand and interpret the results, still it is considered a powerful technique for classification and regression
- It is considered a powerful machine learning technique for automatic predictive modeling
- It captures complex relationships in datasets, which a traditional algorithm such as linear regression or logistic regression fails to understand and interpret

References

Lichman, M. (2013). *UCI Machine Learning Repository* (`http://archive.ics.uci.edu/ml`). Irvine, CA: University of California, School of Information and Computer Science

Summary

In this chapter, we discussed various methods of performing classification, regression, and forecasting using the neural network model. Be it supervised or unsupervised data mining problems, neural-network-based implementations are popular not only among users but also among business stakeholders. We discussed which model to use where and the data requirement for each of the models. In this chapter, we specifically highlighted the importance of a powerful technique with better accuracy always for classification- and regression-based scenarios.

Index

D

E

F

N

P

R