UNIT 14 DATA INTERFACING AND VISUALISATION IN R

Structure **Page Nos.**

14.1 INTRODUCTION

In the previous unit, you have learnt about basic concepts of R programming. This unit explains how to read and analyse data in R from various file types including- CSV, Excel, binary, XML, JSON, etc. It also discusses how to extract and work on data in R from databases and also web data. The unit also explains in detail about data cleaning and pre-processing in R. In the later sections, the unit explores the concept of visualisations in R. Various types of graphs and charts, including - bar charts, box plots, histograms, line graphs and scatterplots, are discussed.

14.2 OBJECTIVES

After going through this Unit, you will be able to:

- explain the various file types and their interface that can be processed for data analysis in R;
- read, write and analyse data in R from different type of files including- CSV, Excel, binary, XML and JSON;
- extract and use data from databases and web for analysis in R;
- explain the steps involved in data cleaning and pre-processing using R;
- Visualise the data using various types of graphs and charts using R and explain their usage.

14.3 READING DATA FROM FILES

In R, you can read data from files outside of the R environment. One may also write data to files that the operating system can store and further access. There is a wide range of file formats, including CSV, Excel, binary, and XML, etc., R can read and write from.

14.3.1 CSV Files

Input as CSV File:

CSV file is a text file in which column values are separated by commas. For example, you can create data with name, programme, phone of students. By copying and pasting this data into Windows Notepad, you can create the CSV file. Using notepad's *Save As* option, save the file as input.csv.

Reading a CSV File:

Function used to read a CSV file: **read.csv()**

```
data <- read.csv("input.csv")
print(data)
```

Figure 14.1: Reading data from a CSV file

Analysing the CSV File:

The read.csv() function returns a data frame as its default output. You can use the following three print functions to:(1) verify if the read data input from CSV file is in frame format or not; (2) find the number of columns in the data; and (3) find the number of rows in the data.

```
print(is.data.frame(data))
print(ncol(data))
print(nrow(data))
```

Figure 14.2: Checking read data

Writing into a CSV File:

The **write.csv()** function of R can be used to generate a CSV file from a data frame. For example, the following function will generate an output.csv file. Please note that this output.csv will be created in the present directory in which you are working.

```
write.csv(output,"output.csv")
```

14.3.2 Excel Files

Microsoft Excel is the most extensively used spreadsheet tool and it uses the.xls or.xlsx file extension to store data. Using various Excel-specific packages, R can read directly from these files. XLConnect, xlsx, and gdata are a few examples of such packages. The xlsx package also allows R to write to an Excel file.

Install xlsx Package

- Command to install 'xlsx' package: **install.packages("xlsx")**
- To Load the library into R workspace: **library("xlsx")**

Reading the Excel File

The **read.xlsx()** function is used to read the input.xlsx file, as illustrated below. In the R environment, the result is saved as a data frame.

```
data <- read.xlsx("input.xlsx", sheetIndex = 1)
print(data)
```

Figure 14.3: Reading data from an Excel file

Writing the Excel File

For writing to a new Excel file, you use the write function, as shown below:

```
write.xlsx(output, "path.filename.xlsx")
```

14.3.3 Binary Files

A binary file is one that solely includes data in the form of bits and bytes. (0's and 1's). When you try to read a binary file, the sequence of bits is translated as bytes or characters, which include numerous other non-printable characters, that are not human readable. Any text editor that tries to read a binary file will display characters like Ø , ð, printable characters and many other characters including beeps.

R has two functions **writeBin()** and **readBin()** to create and read binary files.

Syntax:
writeBin(**object, con**)
readBin(**con, what, n**)

where,

- The connection object **con** is used to read or write a binary file.
- The binary file to be written is the **object.**
- The mode that represents the bytes to be read, such as character, integer, etc is **what**.
- The number of bytes to read from the binary file is given by **n**.

Writing the Binary File(You should read the comments for explanation on each command.)

```
# Read the "mtcars" data frame as a csv file and store only the columns "cyl", "am" and "gear".
write.table(mtcars, file = "mtcars.csv",row.names = FALSE, na = "",col.names = TRUE, sep = ",")

# Store 5 records from the csv file as a new data frame.
new.mtcars <- read.table("mtcars.csv",sep = ",",header = TRUE,nrows = 5)

# Create a connection object to write the binary file using mode "wb".
write.filename = file("/web/com/binmtcars.dat", "wb")

# Write the column names of the data frame to the connection object.
writeBin(colnames(new.mtcars), write.filename)

# Write the records in each of the column to the file.
writeBin(c(new.mtcars$cyl,new.mtcars$am,new.mtcars$gear), write.filename)

# Close the file for writing so that it can be read by other program.
close(write.filename)
```

Figure14.4: An example of Writing data to a Binary file

Reading the Binary File(You should read the comments for explanation on each command.)

```
# Create a connection object to read the file in binary mode using "rb".
read.filename <- file("/web/com/binmtcars.dat", "rb")

# First read the column names. n = 3 as we have 3 columns.
column.names <- readBin(read.filename, character(),  n = 3)

# Next read the column values. n = 18 as we have 3 column names and 15 values.
read.filename <- file("/web/com/binmtcars.dat", "rb")
bindata <- readBin(read.filename, integer(),  n = 18)

# Print the data.
print(bindata)

# Read the values from 4th byte to 8th byte which represents "cyl".
cyldata = bindata[4:8]
print(cyldata)

# Read the values form 9th byte to 13th byte which represents "am".
amdata = bindata[9:13]
print(amdata)

# Read the values form 9th byte to 13th byte which represents "gear".
geardata = bindata[14:18]
print(geardata)

# Combine all the read values to a dat frame.
finaldata = cbind(cyldata, amdata, geardata)
colnames(finaldata) = column.names
print(finaldata)
```

Figure14.5: An example of Reading data to a Binary file

14.3.4 XML Files

XML is an acronym for “extensible markup language”. It is a file format that allows users to share the file format as well as the data over the internet, intranet and other places, as standard ASCII text. XML uses markup tags that describe the meaning of the data stored in the file. This is similar to the markup tags used in HTML wherein the markup tag describes the structure of the page instead.
The "XML" package in R can be used to read an xml file. The following command can be used to install this package:

install.packages("XML")

Reading XML File
R reads the xml file using the function **xmlParse()**. In R, it is saved as a list.

```
# Load the package required to read XML files.
library("XML")

# Also load the other required package.
library("methods")

# Give the input file name to the function.
result <- xmlParse(file = "input.xml")

# Print the result.
print(result)
```

Figure14.6: An example of reading data from a Binary file

XML to Data Frame

In order to manage the data appropriately in huge files, the data in the xml file can be read as a data frame. The data frame should then be processed for data analysis.

```
# Load the packages required to read XML files.
library("XML")
library("methods")

# Convert the input xml file to a data frame.
xmldataframe <- xmlToDataFrame("input.xml")
print(xmldataframe)
```

Figure14.7: converting the read data to a data frame

14.3.5 JSON Files

The data in a JSON file is stored as text in a human-readable format. JavaScript Object Notation is abbreviated as JSON. The rjson package in R can read JSON files.

Install rjson Package

To install the rjson package, type the following command in the R console: **install.packages("rjson")**

Read the JSON File

R reads the JSON file using the function **fromJSON()**. In R, it is saved as a list.

```
# Load the package required to read JSON files.
library("rjson")

# Give the input file name to the function.
result <- fromJSON(file = "input.json")

# Print the result.
print(result)
```

Figure14.8: An example of reading data from JSON file

Convert JSON to a Data Frame

Using the **as.data.frame()** function, you can turn the retrieved data above into a R data frame for further study.

```
# Load the package required to read JSON files.
library("rjson")

# Give the input file name to the function.
result <- fromJSON(file = "input.json")

# Convert JSON file to a data frame.
json_data_frame <- as.data.frame(result)
print(json_data_frame)
```

Figure14.9: Converting read data to data frame

14.3.6 Databases

Data is stored in a normalised way in relational database systems. As a result, you will require quite advanced and complex SQL queries to perform statistical computing. However, R can readily connect to various relational databases, such as MySQL, Oracle, and SQL Server, and retrieve records as a data frame. Once the data is in the R environment, it becomes a standard R data set that can be modified and analysed with all of R's sophisticated packages and functions.

RMySQL Package

R contains a built-in package called "RMySQL" that allows you to connect to a MySql database natively. The following command will install this package in the R environment.

install.packages("RMySQL")

Connecting R to MySQL

```
# Create a connection object to MySQL database.
# We will connect to the sampel database named "sakila" that comes with MySql installation.
mysqlconnection = dbConnect(MySQL(), user = 'root', password = '', dbname = 'admin',
                            host = 'localhost')

# List the tables available in this database.
dbListTables(mysqlconnection)
```

Figure14.10: Connecting to MySQL database

Querying the Tables

Using the MySQL function **dbSendQuery()**, you can query the database tables . The query is run in MySQL, and the results are returned with the **R fetch()** function. Finally, it is saved in R as a data frame.

```
# Query the "actor" tables to get all the rows.
result = dbSendQuery(mysqlconnection, "select * from actor")

# Store the result in a R data frame object. n = 5 is used to fetch first 5 rows.
data.frame = fetch(result, n = 5)
print(data.fame)
```

Figure 14.11: Querying the MySQL table

Updating Rows in the Tables

```
dbSendQuery(mysqlconnection, "update mtcars set disp = 168.5 where hp = 110")
```

Figure 14.12: Updating rows in MySQL table

Inserting Data into the Tables

```
dbSendQuery(mysqlconnection,
            "insert into mtcars(row_names, mpg, cyl, disp, hp, drat, wt, qsec, vs, am, gear, carb)
    values('New Mazda RX4 Wag', 21, 7, 168.5, 112, 3.9, 2.875, 17.05, 0, 1,5, 6)"
)
```

Figure 14.13: Inserting data in a MySQL table

Creating Tables in MySQL

The function **dbWriteTable()** in MySQL can be used to create tables. It takes a data frame as input and overwrites the table if it already exists.

```
# Create the connection object to the database where we want to create the table.
mysqlconnection = dbConnect(MySQL(), user = 'root', password = '', dbname = 'admin',
                            host = 'localhost')

# Use the R data frame "mtcars" to create the table in MySql.
# All the rows of mtcars are taken inot MySql.
dbWriteTable(mysqlconnection, "mtcars", mtcars[, ], overwrite = TRUE)
```

Figure 14.14: Creating a table in MySQL

Dropping Tables in MySQL

```
dbSendQuery(mysqlconnection, 'drop table if exists mtcars')
```

Figure 14.15: Dropping a table in MySQL

14.3.7 Web Data

Many websites make data available for users to consume. The World Health Organization (WHO), for example, provides reports on health and medical information in CSV, txt, and XML formats. You can programmatically extract certain data from such websites using R applications. "RCurl," "XML," and "stringr" are some R packages that are used to scrape data from the web. They are used to connect to URLs, detect required file links, and download the files to the local environment.

Install R Packages

For processing the URLs and links to the files, the following packages are necessary.

install.packages("RCurl")
install.packages("XML")
install.packages("stringr")
install.packages("plyr")

14.4 DATA CLEANING AND PRE-PROCESSING

Data cleaning is the process of identifying, correcting and removing incorrect raw data. The clean data is then fed to the models to build the logical conclusions.
If the data is poorly prepped, unreliable results can destroy the assumptions and insights.
Packages like tidy verse can make complex data manipulations easier.
The following is the checklist of cleaning and preparing data which is mainly

considered among the best practices.

- **Familiarization with the dataset**: to get good domain knowledge so that one is aware which variable represents what.
- **Check for structural errors:** You may check for mislabelled variables, faulty Data types, non-unique (duplicated values) and string inconsistencies or typing errors.
- **Check for data irregularities:** You may check for the invalid values and outliers.
- **Decide on how to deal missing values:** Either delete the observations if they are not providing any meaningful insights to our data or imputing the data with some logical values like mean or median based on the observations.

Check your Progress 1

1. What is the package used to use JSON Files in R?

   …….

……..

2. What are wb and rb mode while dealing with binary files?

   …….

   ……..

3. Mention any 2 checklist points used for cleaning/ preparing data?

   …….

   ……..

14.5 VISUALIZATION IN R

In the previous section, we have discussed about obtaining input from different types of data. This section explains various types of graphs that can be drawn using R. It may please be noted that only selected types of graphs have been presented here.

14.5.1 Bar Charts

A bar chart depicts data as rectangular bars whose length is proportionate to the variable's value. The function barplot() in R is used to make bar charts. In a bar chart, R can create both vertical and horizontal bars. Each of the bars in a bar chart can be coloured differently.

Syntax:
barplot(H,xlab,ylab,main, names.arg,col)

where,

- In a bar chart, H is a vector or matrix containing numeric values.
- The x axis label as **xlab**.
- The y axis label is **ylab**.
- The title of the bar chart is **main**.
- **names.arg** is a list of names that appear beneath each bar.
- **col** is used to color the graph's bars.

```
# Create the data for the chart
H <- c(7,22,14,5,40)

# Give the chart file a name
png(file = "barchart.png")

# Plot the bar chart
barplot(H)

# Save the file
dev.off()
```

Figure 14.16: Creating a Bar chart

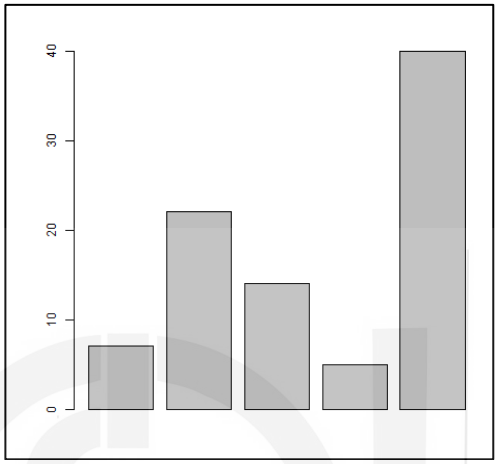


Figure 14.17: A Bar Chart of data of Figure 14.16

Bar Chart Labels, Title and Colors

More parameters can be added to the bar chart to increase its capabilities. The **title** is added using the **main** parameter. Colors are added to the bars using the **col** parameter. To express the meaning of each bar, **args.name** is a vector with the same number of values as the input vector.

```
# Create the data for the chart
H <- c(7,30,12,5,40)
M <- c("Aug","Sep","Oct","Nov","Dec")

# Give the chart file a name
png(file = "barchart_months.png")

# Plot the bar chart
barplot(H,names.arg=M,xlab="Month",ylab="Sample",col="yellow",
        main="Sample chart",border="red")

# Save the file
dev.off()
```

Figure 14.18: Function for plotting Bar chart with labels and colours

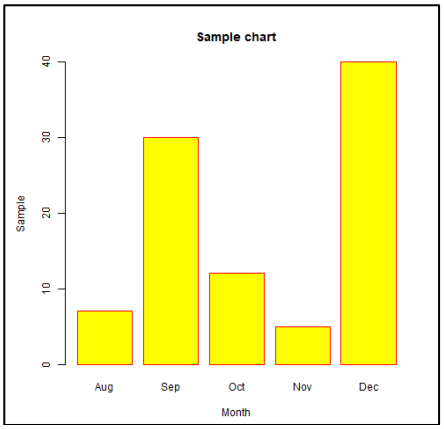


Figure 14.19: A Bar Chart of with labels and colors

14.5.2 Box Plots

In order to determine how evenly the data is distributed in a dataset, you can use boxplot, which is very effective tool. The dataset is split using quartiles. This graph depicts the data set's minimum, first quartile, median, third quartile and maximum. Drawing boxplots for each data set allows you to compare the distribution of data across data sets.

The **boxplot()** function in R is used to make boxplots.

Syntax:

boxplot(x, data, notch, varwidth, names, main)

The parameters of the functions are as follows:

- Parameter **x** either can specify a formula or it can specify a vector.
- Parameter **data** is used to specify the data frame that contains the data required to be plotted.
- Parameter **notch** represents a logical value. In case, you want to draw a notch in the box plot, you may set its value to TRUE.
- Parameter **varwidth** is also logical. It can be set to TRUE, if you want to make the box's width proportional to the sample size.
- Parameter **names** can be used to specify the group labels that will be printed beneath each boxplot.
- main is used to give the graph a **title**.

Creating the Boxplot

```
input <- mtcars[,c('mpg','cyl')]
print(head(input))
# Give the chart file a name.
png(file = "boxplot.png")

# Plot the chart.
boxplot(mpg ~ cyl, data = mtcars, xlab = "Number of Cylinders",
        ylab = "Miles Per Gallon", main = "Mileage Data")

# Save the file.
dev.off()
```

Figure 14.20: Coding for Box plot

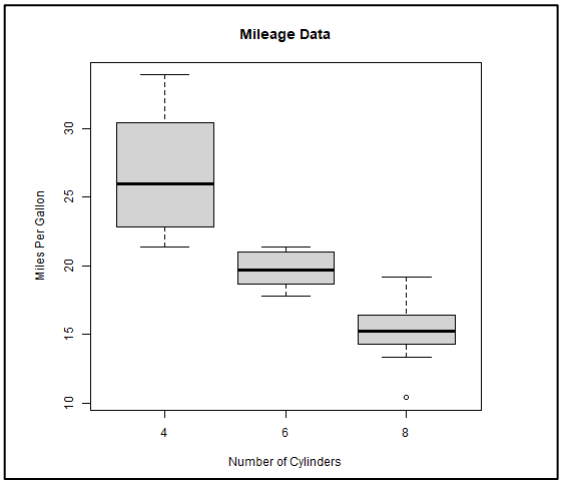


Figure 14.21: A Box plot of Figure 14.20

14.5.3 Histograms

The frequency of values of a variable bucketed into ranges is represented by a histogram. The difference between a histogram and a bar chart is that a histogram groups the numbers into continuous ranges. The height of each bar in a histogram represents the number of items present in that range.

The **hist()** function in R is used to produce a histogram. This function accepts a vector as an input and plots histograms using additional parameters.

Syntax:
hist(v,main,xlab,xlim,ylim,breaks,col,border)

where,

- The parameter **v** is a vector that contains the numeric values for which histogram is to be drawn.
- The title of the chart is shown by the **main**.
- The colour of the bars is controlled by **col**.
- Each bar's border colour is controlled by the **border** parameter.
- The **xlab** command is used to describe the x-axis.
- The x-axis range is specified using the **xlim** parameter.
- The y-axis range is specified with the **ylim** parameter.
- The term "**breaks**" refers to the breadth of each bar.

```
# Create data for the graph.
v <-  c(9,12,20,10,35,25,12,40,30,35,19)

# Give the chart file a name.
png(file = "histogram.png")

# Create the histogram.
hist(v,xlab = "Weight",col = "blue",border = "yellow")

# Save the file.
dev.off()
```

Figure 14.22: Creating a Histogram

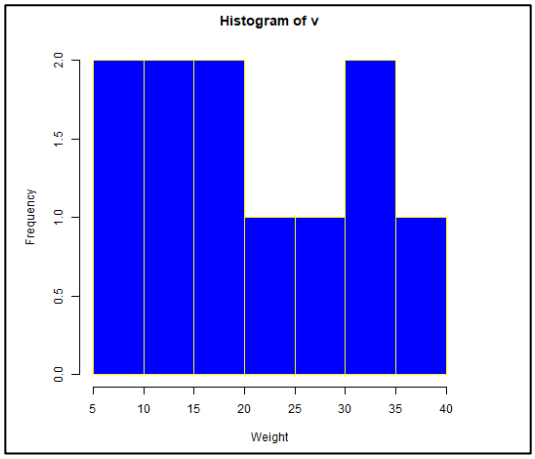


Figure 14.23: Histogram of data used in Figure 14.22

14.5.3 Line Graphs

A graph that uses line segments to connect a set of points is known as the line graph. These points are sorted according to the value of one of their coordinates (typically the x-coordinate). Line charts are commonly used to identify data trends.
The line graph was created using R's **plot()** function.

Syntax:
plot(v,type,col,xlab,ylab)

where,

- The numeric values are stored in **v**, which is a vector.
- **type** takes values, "**p**","**l**","**o**". The value "p" is used to draw only points, "l" is used to draw only lines, and "o" is used to draw both points and lines.
- **xlab** specifies the label for the x axis.
- **ylab** specifies the label for the x axis..
- **Main** is used to specify the title of chart .
- **col** is used to specify the color of the points and/or the lines.

```
# Create the data for the chart.
v <- c(7,30,12,5,40)

# Give the chart file a name.
png(file = "line_chart.jpg")

# Plot the bar chart.
plot(v,type = "o")

# Save the file.
dev.off()
```

Figure 14.24: Function to draw a line chart

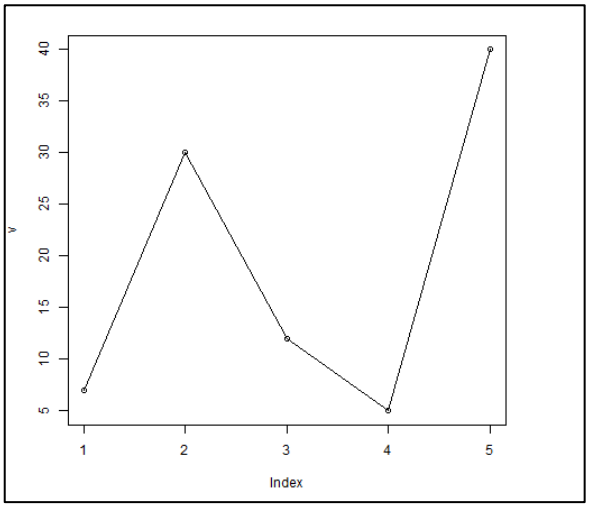


Figure 14.25: A Line Chart for data of Figure 14.24

Multiple Lines in Line Chart & Axis Details

```
# Create the data for the chart.
v <- c(7,30,12,5,40)
t <- c(14,7,6,19,3)

# Give the chart file a name.
png(file = "line_chart_2_lines.jpg")

# Plot the bar chart.
plot(v,type = "o",col = "blue", xlab = "Month", ylab = "Sample",
     main = "Rain fall chart")

lines(t, type = "o", col = "red")

# Save the file.
dev.off()
```

Figure 14.25: Function for Line Chart with multiple lines

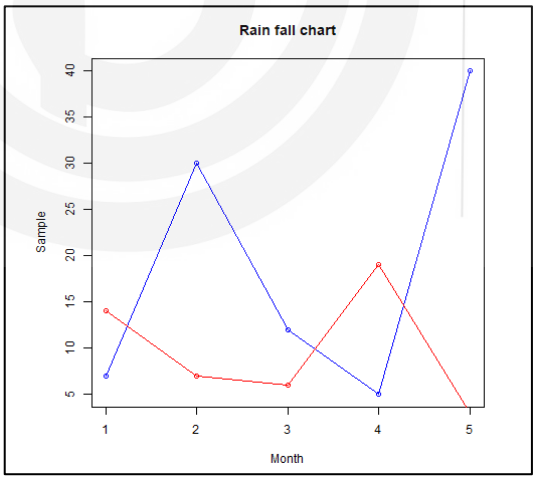


Figure 14.27: A Line Chart with multiple lines for data of Figure 14.26

14.5.4 Scatterplots

Scatterplots are diagrams that display a large number of points shown in the Cartesian plane. The values of two variables are represented by each point.
One variable is chosen on the horizontal axis, while another is chosen on the vertical axis. To create a simple scatterplot, use the plot() method.

Syntax:

plot(x, y, main, xlab, ylab, xlim, ylim, axes)

The parameters of the plot functions are as follows:

- parameter **x** is the data values for x-axis.
- parameter **y** is the data values for y-axis
- parameter **main** is used for title of the grpah
- parameters **xlab** and **ylab** are used to specify the Labels for x-axis and y-axis respectively.
- Parameters **xlim** and **ylim** used define the limits of values of x and y respectively.
- **axes** specifies if the plot should include both axes.

```
input <- mtcars[,c('wt','mpg')]
print(head(input))

# Get the input values.
input <- mtcars[,c('wt','mpg')]

# Give the chart file a name.
png(file = "scatterplot.png")

# Plot the chart for cars with weight between 2.5 to 5 and mileage between 15 and 30.
plot(x = input$wt,y = input$mpg,
     xlab = "Weight",
     ylab = "Milage",
     xlim = c(2.5,5),
     ylim = c(15,30),
     main = "Weight vs Milage"
)

# Save the file.
dev.off()
```

Figure 14.28: Plot function to draw Scatter Plot

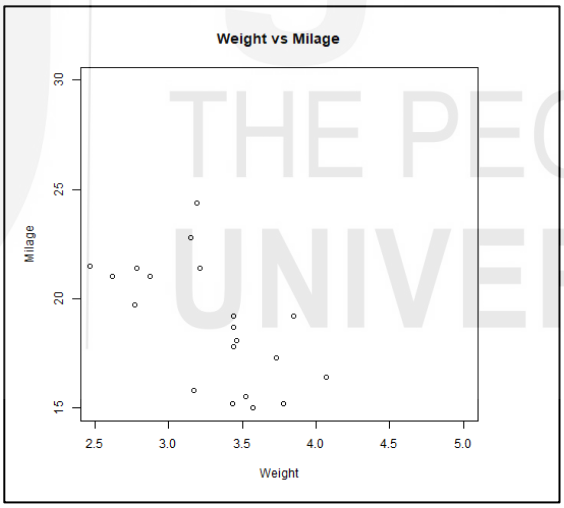


Figure 14.29: Scatter plot for the data of Figure 14.28

Scatterplot Matrices

The scatterplot matrix is used when there are more than two variables and you want to identify the correlation between one variable and the others. To make scatterplot matrices, we use **pairs()** function.

Syntax:

pairs(formula, data)

where,

- The **formula** represents a set of variables that are utilised in pairs.
- The data set from which the variables will be derived is referred to as **data**.

```
# Give the chart file a name.
png(file = "scatterplot_matrices.png")

# Plot the matrices between 4 variables giving 12 plots.

# One variable with 3 others and total 4 variables.

pairs(~wt+mpg+disp+cyl,data = mtcars,
      main = "Scatterplot Matrix")

# Save the file.
dev.off()
```

Figure 14.30: Function for Scatterplot matrix

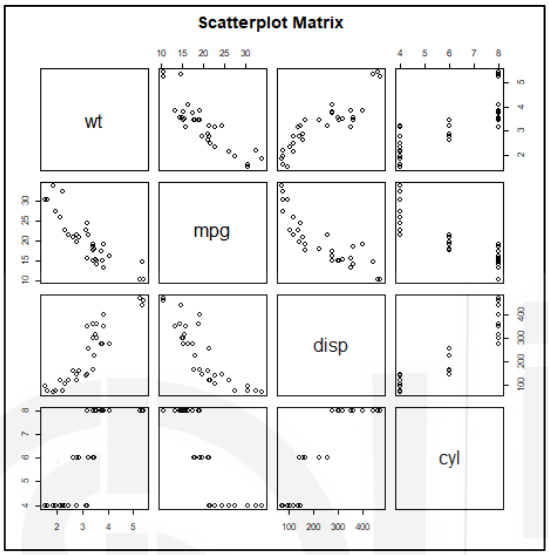


Figure 14.31: A Scatterplot matrix

Check your Progress 2

1. What is scatter plot?
2. When you will use histogram and when you will use bar chart in R?
3. What type of chart you consider when trying to demonstrate "relationship" between variables/parameters?

14.6 Summary

In this unit you have gone though various file types that can be processed for data analysis in R and further discussed their interfaces. R can read and write a variety of file types outside the R environment, including CSV, Excel, binary, XML and JSON. Further, R can readily connect to various relational databases, such as MySQL, Oracle, and SQL Server, and retrieve records as a data frame that can be modified and analysed with all of R's sophisticated packages and functions. The data can also be programmatically extracted from websites using R applications. "RCurl," "XML," and "stringr" are some R packages that are used to scrape data from the web. The unit also explains the concept of data cleaning and pre-processing which is the process of identifying, correcting and removing incorrect raw data, familiarization with the dataset, checking data for structural errors and data irregularities and deciding on how to deal with missing values are the steps involved in cleaning and preparing data which is mainly considered among the best practices. The unit finally explores the concept of

visualisations in R. There are various types of graphs and charts including- bar charts, box plots, histograms, line graphs and scatterplots that can be used to visualise the data effectively. The unit explained the usage and syntax for each of the illustration with graphics.

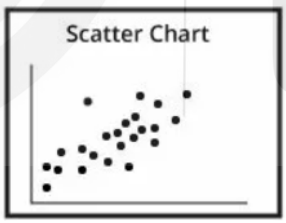
14.7 Answers

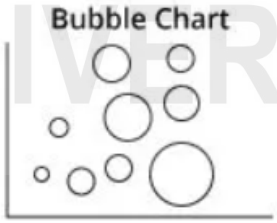
Check your progress 1

1. Install.packages("**rjson"**)
   library(**rjson**)
2. rb mode opens the file in the binary format for reading and wb mode opens the file in the binary format for writing.
3. The checklist points used for cleaning/ preparing data:
   Check for data irregularities: You may check for the invalid values and outliers.
   Decide on how to deal missing values: Either delete the observations if they are not providing any meaningful insights to our data or imputing the data with some logical values like mean or median based on the observations.

Check your progress 2

1. A scatter plot is a chart used to plot a correlation between two or more variables at the same time
2. We use a histogram to plot the distribution of a continuous variable, while we can use a bar chart to plot the distribution of a categorical variable.
3. When you are trying to show "relationship" between two variables, you will use a scatter plot or chart. When you are trying to show "relationship" between three variables, you will have to use a bubble chart.





14.8 REFERENCES AND FURTHER READINGS

1. De Vries, A., & Meys, J. (2015). *R for Dummies*. John Wiley & Sons.
2. Peng, R. D. (2016). *R programming for data science* (pp. 86-181). Victoria, BC, Canada: Leanpub.
3. Schmuller, J. (2017). *Statistical Analysis with R For Dummies*. John Wiley & Sons.
4. Field, A., Miles, J., & Field, Z. (2012). *Discovering statistics using R*. Sage publications.
5. Lander, J. P. (2014). *R for everyone: Advanced analytics and graphics*. Pearson Education.
6. Lantz, B. (2019). *Machine learning with R: expert techniques for predictive modeling*. Packt publishing ltd.
7. Heumann, C., & Schomaker, M. (2016). *Introduction to statistics and data analysis*. Springer International Publishing Switzerland.
8. Davies, T. M. (2016). *The book of R: a first course in programming and statistics*. No Starch Press.
9. https://www.tutorialspoint.com/r/index.html