DATA ANALYTICS THROUGH R

Become Data Analyst in 30 days

Kamakshaiah Musunuru
kamakshaiah.m@gmail.com

Contents

Preface

R is *lingua franca* of statistics. I started writing this book for one simple reason that I got to teach R for business analytics students. I never went any academy nor to a Guru for learning R. My first encounter with analytics software is R. I taught "business statistics" several times during my stint as academic. I always used Excel to teach statistics in my classes. However, over the period of time I got bored using Excel in my classes. R turned as my choice to make a changeover.

I turned into data analyst due to my being struck accidentally across open source software when I was in Ethiopia. I got to use GNU/Linux due to a very simple reason that as I got to find out a way to keep my lappy away from virus infection in the university. I crashed Windows a couple of times when I was working in office. I came to know that GNU/Linux is a wonderful solution to handle viruses. My first experience of GNU/Linux, perhaps, is *Jaunty Jackalope*, I suppose. This must be Ubuntu 9.04, an LTS version, if I am not wrong. However, my involvement in GNU/Linux became a serious affair though *Karmic Koala*. I still remember few of my colleagues used to visit my home for OS installation in those days.

I came across R in Ubuntu Software Applications. I notice this little yet uppercase R in *package manager* while I was searching for statistical software tools in Ubuntu. R changed not only my understanding of Statistics but the very way of learning the same. I addicted to R so much so that for every pretty little calculations I used to refer R manuals. I must be the first academic to introduce R in south Indian business management curriculum, yet remained unknown. I used to write reply mails, in my early years of R practice, denouncing commercial tools whenever I used to get invitations related to the usage of same in academcs. At first I was never understood here in academic fraternity until certain time. After a couple years I started seeing the response in neighborhood universities and institutes. Today, R is one of the best tools for statistical learning and practice in India.

I learned R by self study. I mean, through very informal personal learning. All my learning is from online resources. However, I could produce close to three hundred data analysts through my formal classes by the end of 2019. I am writing all this not to show myself as a valiant learner, but to demonstrate how can a novice and a naive enthusiast like me can learn programming tools like R. Today, I am offering a couple of courses to teach Python, Hadoop and IoT, that is all by passion. So, I would like to give confidence to the readers that you don't need any formal learning in computer science or information technology to learn data science and adopt it as profession. However, you need tons and tons of patience and passion.

This book is a 30 day primer for beginners in data science and analytics. It has 5 chapters split in 30 sections. Each section represents a unique concept of data analytics. This particular scheme is very much helpful for practitioners and academics to acquire knowledge of R in very little short time such as 30 days, nearly in a month. So, even if the reader completes a lesson a day, he or she could transform into a data analyst in 30 days.

Happy reading $\cdots$

Dr. M. Kamakshaiah
Author

About the Author

Dr. M. Kamakshaiah is a open source software evangelist and enterprise architect. He has 20 years of experience in teaching QT and IT practices related to business management. He has travelled to a couple of countries on teaching and research assignments. He has been teaching data science and analytics using open source software tools like R, Python for survey data analytics and Hadoop for big data analytics. He is also expert of real time data analytics using AVR uC and ARM processor. He is a full stack R, Python developer. Creating web applications for teaching and learning practices is his hobby. All his applications are free and open source; please visit https://github.com/Kamakshaiah for his software applications.

Chapter 1

Introduction to R

In real open source, you have the right to control your own destiny.

- Linus Torvald

1.1 Open Source Software

Contents:

1. Unix
2. Linux

1.1.1 Unix

I like this quote very much. Today developments in computing world can be attributable to two individuals namely (1) Linus Torvald and (2) Richard M. Stallman. If these guys had not thought about software differently, we could have lost lot of developments in this world.

I should mention Unix before I do that about Linux. Unix is a family of multitasking, multiuser computer operating systems that derive from the original AT&T Unix, development starting in the 1970s at the Bell Labs research center by Ken Thompson, Dennis Ritchie, and others. Perhaps, Unix is the first OS characterized by a modular design that is sometimes called the "Unix philosophy". At first Unix was named as *Unics*, Uniplexed Information and Computing Service as pun on *Multics*. Multics was an operating system in 1960s and it was a serious project for a group of organizations such as MIT, Bell Labs, GE. Ken Thompson, Dennis Ritchie in Bell had started working on Unix when Multics

Figure 1.1: Ken Thompson (sitting) and Dennis Ritchie working together at a PDP-11

was wound up officially everywhere. The leisure work done by these two individuals turned as great fortune to the world. The operating system was originally written in assembly language, but in 1973, Version 4 Unix was rewritten in C. Bell Labs produced several versions of Unix that are collectively referred to as "Research Unix". In 1975, the first source license for UNIX was sold to Donald B. Gillies at the University of Illinois Department of Computer Science. During the late 1970s and early 1980s, the influence of Unix in academic circles led to large-scale adoption of Unix (BSD and System V) by commercial startups, including Sequent, HP-UX, Solaris, AIX, and Xenix. In the 1990s, Unix and Unix-like systems grew in popularity as BSD and Linux distributions were developed through collaboration by a worldwide network of programmers. In 2000, Apple released Darwin, also a Unix system, which became the core of the Mac OS X operating system, which was later renamed macOS.

1.1.2 Linux

Linux is a family of open source Unix like operating systems based on the Linux kernel, an operating system kernel first developed by Linus Torvalds. Linux is typically packaged in a Linux distribution.

Distributions include the Linux kernel and supporting system software and libraries, many of which are provided by the GNU Project. Many Linux distributions use the word “Linux” in their name, but the Free Software Foundation uses the name GNU/Linux to emphasize the importance of GNU software. Popular Linux distributions include Debian, Fedora, and Ubuntu. Commercial distributions include Red Hat Enterprise Linux and SUSE Linux Enterprise Server. Desktop Linux distributions include a windowing system such as X11 or Wayland, and a desktop environment such as GNOME or KDE.

Linux was originally developed for personal computers based on the Intel x86 architecture, but has since been ported to more platforms than any other operating system. Linux is the leading operating system on servers and other big iron systems such as mainframe computers, and the only OS used on TOP500 supercomputer.

Linux also runs on embedded systems, i.e. devices whose operating system is typically built into the firmware and is highly tailored to the system. This includes routers, automation controls, televisions, digital video recorders, video game consoles, and smartwatches. Many smartphones and tablet computers run Android and other Linux derivatives. Because of the dominance of Android on smartphones, Linux has the largest installed base of all general-purpose operating systems.

In 1991, while attending the University of Helsinki, Torvalds became curious about operating systems. Frustrated by the licensing of MINIX, which at the time limited it to educational use only, he began to work on his own operating system kernel, which eventually became the Linux kernel.

Torvalds began the development of the Linux kernel on MINIX and applications written for MINIX were also used on Linux. Later, Linux matured and further Linux kernel development took place on Linux systems. GNU applications also replaced all MINIX components, because it was advantageous to use the freely available code from the GNU Project with the fledgling operating system; code licensed under the GNU GPL can be reused in other computer programs as long as they also are released under the same or a compatible license. Torvalds initiated a switch from his original license, which prohibited commercial redistribution, to the GNU GPL. Developers worked to integrate GNU components with the Linux kernel, making a fully functional and free operating system.

Adoption of Linux in production environments, rather than being used only by hobbyists, started to take off first in the mid-1990s in the supercomputing community, where organizations such as NASA started to replace their increasingly expensive machines with clusters of inexpensive commodity computers running Linux. Commercial use began when Dell and IBM, followed by Hewlett-Packard, started offering Linux support to escape Microsoft’s monopoly in the desktop operating system market.

Today, Linux systems are used throughout computing, from embedded systems to virtually all supercomputers, and have secured a place in server installations such as the popular LAMP application stack. Use of Linux distributions in home and enterprise desktops has been growing. Linux distributions have also

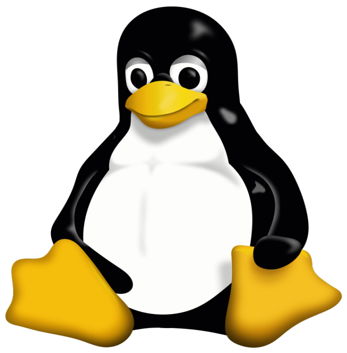


Figure 1.2: Linux Torvald and Linux Mascot

become popular in the netbook market, with many devices shipping with customized Linux distributions installed, and Google releasing their own Chrome OS designed for netbooks.

Linux's greatest success in the consumer market is perhaps the mobile device market, with Android being one of the most dominant operating systems on smartphones and very popular on tablets and, more recently, on wearables. Linux gaming is also on the rise with Valve showing its support for Linux and rolling out its own gaming oriented Linux distribution.

1.2 Installation & Editors

Contents

1. About R
2. Installation
3. Open R
4. Scripts & Commands
5. RStudio

1.2.1 About R

R is a programming language and free software environment for statistical computing and graphics supported by the R Foundation for Statistical Computing. The R language is widely used among statisticians and data miners for developing statistical software and data analysis. Polls, data mining surveys, and

Figure 1.3: John Chambers

studies of scholarly literature databases show substantial increases in popularity; as of September 2019, R ranks 19th in the TIOBE index, a measure of popularity of programming languages. [1]

R is a GNU software, source code for the R software environment is written primarily in C, Fortran, and R itself [2] and is freely available under the GNU General Public License. Pre-compiled binary versions are provided for various operating systems. Although R has a command line interface, there are several graphical user interfaces, such as RStudio, an integrated development environment.

R is an implementation of the S programming language combined with lexical scoping semantics, inspired by Scheme. S was created by John Chambers in 1976, while at Bell Labs. There are some important differences, but much of the code written for S runs unaltered. R was created by Ross Ihaka and Robert Gentleman at the University of Auckland, New Zealand, and is currently developed by the R Development Core Team (of which Chambers is a member). R is named partly after the first names of the first two R authors and partly as a play on the name of S. The project was conceived in 1992, with an initial version released in 1995 and a stable beta version in 2000.

R and its libraries implement a wide variety of statistical and graphical techniques, including linear and nonlinear modelling, classical statistical tests, time-series analysis, classification, clustering, and others. R is easily extensible through functions and extensions, and the R community is noted for its active contributions in terms of packages. Many of R's standard functions are written in R itself, which makes it easy for users to follow the algorithmic choices made. For computationally intensive tasks, C, C++, and Fortran code can be linked and called at run time. Advanced users can write C, C++, Java,

.NET or Python code to manipulate R objects directly. R is highly extensible through the use of user-submitted packages for specific functions or specific areas of study. Due to its S heritage, R has stronger object-oriented programming facilities than most statistical computing languages. Extending R is also eased by its lexical scoping rules. Another strength of R is static graphics, which can produce publication-quality graphs, including mathematical symbols. Dynamic and interactive graphics are available through additional packages.

R Packages

The capabilities of R are extended through user-created packages, which allow specialised statistical techniques, graphical devices, import/export capabilities, reporting tools (Rmarkdown, knitr, Sweave), etc. These packages are developed primarily in R, and sometimes in Java, C, C++, and Fortran. Many of R packages are available in toto together with data sets and vignettes.

A core set of packages is included with the installation of R, with more than 15,000 additional packages (as of September 2018) available at the Comprehensive R Archive Network (CRAN), Bioconductor, Omegahat, GitHub, and other repositories.

Users and developers use these packages as per requirement. There is also a mechanism to install and use packages as by domain. The mechanism is CRAN Task Views also known as `ctv`. CRAN stands for "Comprehensive R Archive Network". CRAN is the official repository that hosts lot of data related to R. "Task Views" page on the CRAN website lists a wide range of tasks (in fields such as Finance, Genetics, High Performance Computing, Machine Learning, Medical Imaging, Social Sciences and Spatial Statistics) to which R has been applied and for which packages are available. R has also been identified by the FDA as suitable for interpreting data from clinical research.

Other places like Crantastic and R-Forge are community based platforms for the collaborative development of R packages, R-related software, and projects. The Bioconductor project provides R packages for the analysis of genomic data. This includes object-oriented data-handling and analysis tools for data from Affymetrix, cDNA microarray, and next-generation high-throughput sequencing methods. There is another novel place for neuroscientists, it is known as `https://www.neuroconductor.org/`. This place is greatly helpful to both beginners and advanced programmers to learn and practice neuroscience.

Editors

The most specialized integrated development environment (IDE) for R is RStudio. Visit `https://rstudio.com/`. RStudio today is not just editor but grown as an ecosystem for R based programming and web development. A similar development interface is R Tools for Visual Studio. Some generic IDEs like

Eclipse, also offer features to work with R. Graphical user interfaces with more of a point-and-click approach include Rattle GUI, R Commander, and RKWard.

Some of the more common editors with varying levels of support for R include Emacs (Emacs Speaks Statistics), Vim (Nvim-R plugin), Neovim (Nvim-R plugin), Kate, LyX, Notepad++, Visual Studio Code, WinEdt, and Tinn-R.

R functionality is accessible from several scripting languages such as Python, Perl, Ruby, $F\#$, and Julia. Interfaces to other, high-level programming languages, like Java and .NET $C\#$ are available as well.

RStudio

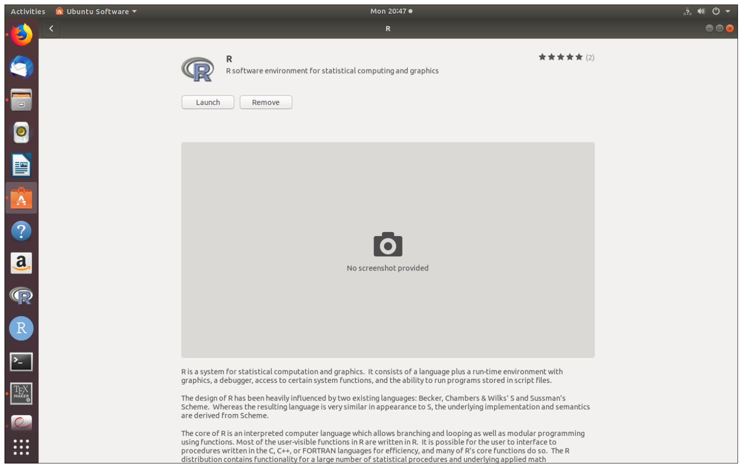
RStudio is an integrated development environment (IDE) for R, a programming language for statistical computing and graphics. The RStudio IDE is developed by RStudio, Inc., a commercial enterprise founded by JJ Allaire, creator of the programming language ColdFusion. RStudio, Inc. has no formal connection to the R Foundation, a not for profit organization located in Vienna Austria, which is responsible for overseeing development of the R environment for statistical computing. RStudio is available in two formats: RStudio Desktop, where the program is run locally as a regular desktop application; and RStudio Server, which allows accessing RStudio using a web browser while it is running on a remote Linux server.

Work on RStudio started around December 2010, and the first public beta version (v0.92) was officially announced in February 2011. Version 1.0 was released on 1 November 2016. Version 1.1 was released on 9 October 2017. RStudio is available with the GNU Affero General Public License version 3. The AGPL v3 is an open source license that guarantees the freedom to share the code. [3] RStudio Desktop and RStudio Server are both available in free and commercial editions. OS support depends on the format/edition of the IDE. Prepackaged distributions of RStudio Desktop are available for Windows, macOS, and Linux. RStudio Server and Server Pro run on Debian, Ubuntu, Red Hat Linux, CentOS, openSUSE and SLES.

RStudio is partly written in the C++ programming language and uses the Qt framework for its graphical user interface. The bigger percentage of the code is written in Java. JavaScript is also amongst the languages used. In April 2019, RStudio released the RStudio Job Launcher, an adjunct to RStudio Server. The launcher provides the ability to start processes within various batch processing systems (e.g. Slurm) and container orchestration platforms (e.g. Kubernetes). This function is only available in RStudio Server Pro (fee-based application).

RStudio and its team have contributed to many R packages. These include:

1. Tidyverse - R packages for data science, including `ggplot2, dplyr, tidyr`, and `purrr`.



R is a system for statistical computation and graphics. It consists of a language plus a run-time environment with graphics, a debugger, access to certain system functions, and the ability to run programs stored in script files.

The design of R has been heavily influenced by two existing languages: Becker, Chambers & Wilks' S and Sussman's Scheme. Whereas the resulting language is very similar in appearance to S, the underlying implementation and semantics are derived from Scheme.

The core of R is an interpreted computer language which allows branching and looping as well as modular programming using functions. Most of the user-visible functions in R are written in R. It is possible for the user to interface to procedures written in the C, C++, or FORTRAN languages for efficiency, and many of R's core functions do so. The R distribution contains functionality for a large number of statistical procedures and underlying applied math

Figure 1.4: Ubuntu Software Center

2. Shiny - An interactive web technology
3. RMarkdown - Insert R code into markdown documents
4. knitr - Dynamic reports combining R, TeX, Markdown & HTML
5. packrat - Package dependency tool
6. devtools - Package development tool

1.2.2 Installation

Installing R is straight in both Windows and Linux. I use Ubuntu [4] If you are in Ubuntu, you just need to open *package manager* or *software center* to install the software. You will find below window for installation.

You can press the button "install", the rest is taken care by Ubuntu OS. There is other alternative way to install R i.e. from Terminal. Press `Ctl+Alt+T` to open Ubuntu or Linux Terminal. [5] Then issue the following command.

```
sudo apt-get install r-base r-base-dev
```

This will take care of entire installation. I mean the base package, dependencies *et cetera*. Since ubuntu uses debian package manager, the activity is very simple. Ubuntu packaging system is very great and gigantic. You find everything for all of your needs. In ubuntu there is also another way to install R but by using *debian package manager* also known as `dpkg`. This method is highly useful for `.deb` packages. Sometimes, we also try to install *.deb* packages, just like `.exe` in Windows. `.deb` packages are ready-made installers available for debian platform. Suppose you have `.deb` package such as the one provided at https://packages.debian.org/sid/all/r-base/download, you need to use `dpkg`.

```
dpkg -i r-base_3.6.1-6_all.deb
```

However, this is not advisable. It is always better to use "Software Center" to install packages, in stead of using `.deb` files. One potential reason is Software Center knows how to fetch and install all recommended dependencies automatically.

I use a couple of other Linux based OSes such as Fedora, SUSE, CentOS. Fedora, CentoS and SUSE belongs to RPM platform. RPM stands for *Redhat Package Management*. RPM is very old yet robust package management systems available for most of the OSes in Linux world. Suppose if you would like to install R in Fedora and CentOS; follow the following command in Terminal. By the way, the short cut kesy to open Terminal in Fedora is `Ctl+Alt+F2`.

```
sudo yum install R r-core
```

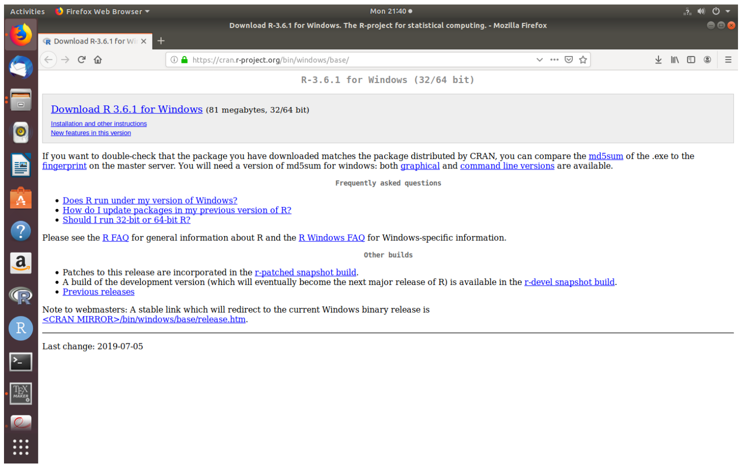
Fedora too has a package manager known as `dnf`. DNF is a software package manager that installs, updates, and removes packages on RPM-based Linux distributions. It automatically computes dependencies and determines the actions required to install packages. DNF also makes it easier to maintain groups of machines, eliminating the need to manually update each one using rpm. Introduced in Fedora 18, it has been the default package manager since Fedora 22. DNF or *Dandified yum* is the next generation version of yum in Fedora. [6] Installing using `dnf` is also straight forward. Use the following statement in Fedora Terminal. [7]

```
dnf install R r-core
```

While coming to SUSE; SUSE is known as most viable commercial Linux OS. There is open source OS known as openSUSE. [8] In SUSE installing software is bit easy and mimicks like Windows. There is a particular mechanism known as "one click installer". R is also available as one-click installer. Visit https://cran.r-project.org/bin/linux/suse/README.html for these installers. You just have to know openSUSE version. Otherwise, suppose if you want to install using Terminal, use the following command in the Terminal.

```
zypper install R-base R-base-devel
```

Zypper is a command line package manager for installing, updating and removing packages as well as for managing repositories. It is especially useful for accomplishing remote software management tasks or managing software from shell scripts.


Activities
Firefox Web Browser
Mon 21:40
Download R-3.6.1 for Windows. The R-project for statistical computing. - Mozilla Firefox
Download R-3.6.1 for Wi
https://cran.r-project.org/bin/windows/base/

R-3.6.1 for Windows (32/64 bit)

Download R 3.6.1 for Windows (81 megabytes, 32/64 bit)
Installation and other instructions
New features in this version

If you want to double-check that the package you have downloaded matches the package distributed by CRAN, you can compare the md5sum of the .exe to the fingerprint on the master server. You will need a version of md5sum for windows: both graphical and command line versions are available.

Frequently asked questions

- Does R run under my version of Windows?
- How do I update packages in my previous version of R?
- Should I run 32-bit or 64-bit R?

Please see the R FAQ for general information about R and the R Windows FAQ for Windows-specific information.

Other builds

- Patches to this release are incorporated in the r-patched snapshot build.
- A build of the development version (which will eventually become the next major release of R) is available in the r-devel snapshot build.
- Previous releases

Note to webmasters: A stable link which will redirect to the current Windows binary release is
<CRAN MIRROR>/bin/windows/base/release.htm.

Last change: 2019-07-05

Figure 1.5: R for Windows.

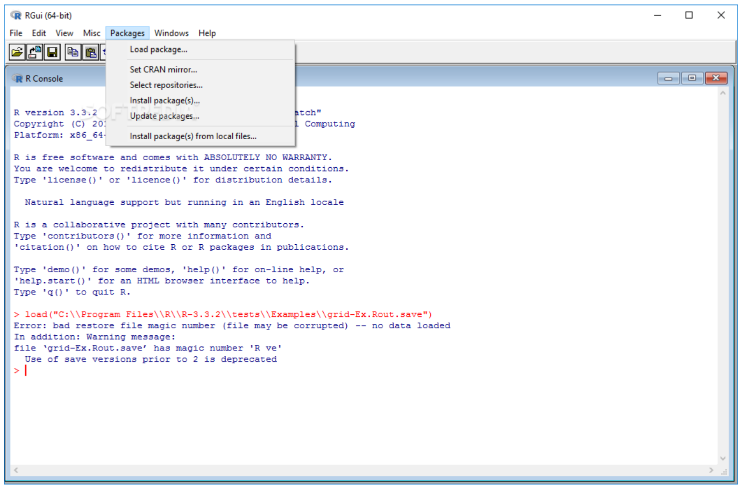
Windows

Now coming to Windows; installing R is more hassle free and straight forward approach. You just have to get `.exe` file. Where do you get it? Just simple! It is available from CRAN. There is more information about CRAN in forthcoming section. Just go to `https://cran.r-project.org/bin/windows/base/` or if you can't remember this url just search for "R download" in default browser in your OS, using google search engine. This will get you the aforementioned url. Download the `.exe` file and install the same just like any other software.

1.2.3 Open R

Go to Windows Start, search for "R" and launch the application. In windows, R has R-GUI. This is really a big blessing for Windows platform. When open R in windows it has this window.

This is just a minimal editor for programming. The windows has few menus along with Terminal/Console as main part of the window. We just have to start typing commands or statements at *R command prompt* which is just a little ">" sign in the Console. Refer to Figure 1.6



```
R version 3.3.2                              atch"
Copyright (C) 20                             l Computing
Platform: x86_64-

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

  Natural language support but running in an English locale

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

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

> load("C:\\Program Files\\R\\R-3.3.2\\tests\\Examples\\grid-Ex.Rout.save")
Error: bad restore file magic number (file may be corrupted) -- no data loaded
In addition: Warning message:
file ‘grid-Ex.Rout.save’ has magic number 'R ve'
  Use of save versions prior to 2 is deprecated
> 
```

Figure 1.6: R Gui for Windows

1.2.4 Scripts & Commands

Windows

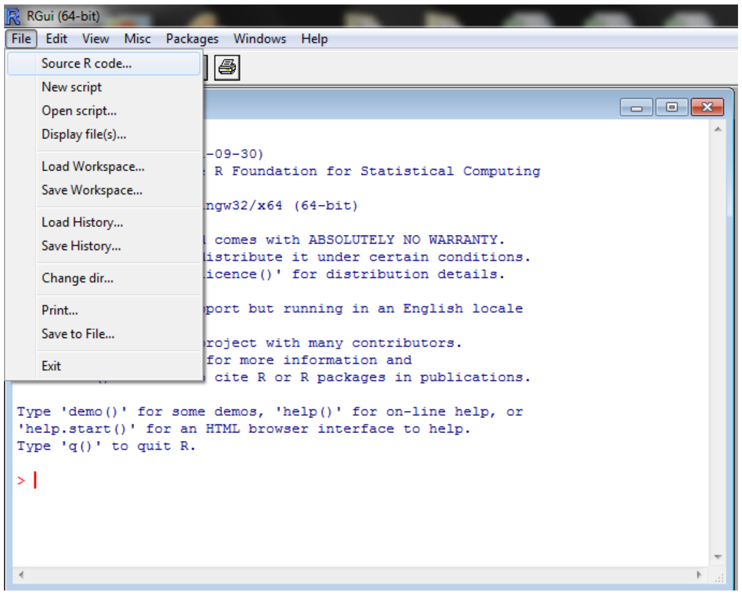
Writing programs or commands in Linux or Windows is same. The only difference is Editor. Go to “File -> New script”. Refer Figure 1.7

Then you will see a new sub-window getting opened in the same main-window, just as shown in the Figure 1.8.

This is a very good practice to write programs in R. The advantage is we can save these scripts for later use. R uses `.R` file format to save, open and use script files.

Scripts in Windows CMD

Whatever may be the programming language and the grammer associated with it, end of the day every program is compiled or interpreted using Command Line Interpreter (CLI). Take for example, C, Java, Python, Perl and many more. All these languages support plain word file to write programs. We don’t need any special editor to execute (compile/interpret) programs. For instance, if you are writing a Java program or Python program, it is just *notepad* that comes handy for work. There is another text editor known as *notepad++*. However, one special care that every programmer needs is to save the file in language specific format. Suppose if we are writing programs in Java, the file format is `.java`. Same fashion, if we are wrting programs using Python, the file format will be



```
-09-30)
 R Foundation for Statistical Computing

ngw32/x64 (64-bit)

 comes with ABSOLUTELY NO WARRANTY.
istribute it under certain conditions.
icence()' for distribution details.

port but running in an English locale

roject with many contributors.
for more information and
 cite R or R packages in publications.

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

> 
```

Figure 1.7: File Menu in RGui in Windows OS

`.py`. Do you notice websites, few websites ends with *.html*, this is none other than a file format. The server tries retrive a file that ends with extension `.html`.

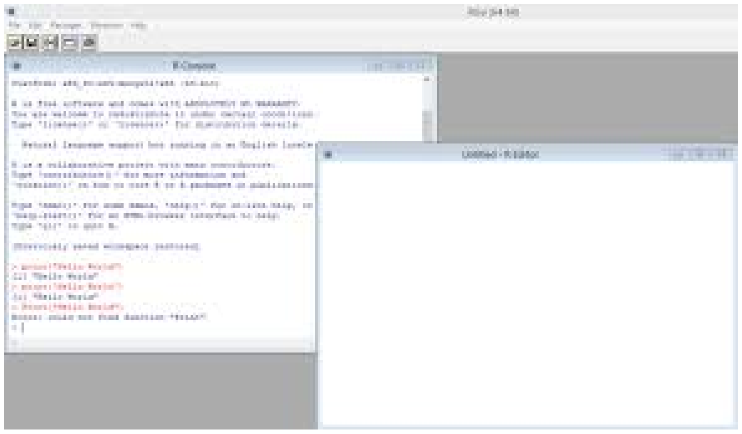
CMD stands for COMMAND in Windows. This is also known as Windows Command Prompt. You need to open CMD using Windows START button. You will find this the left bottom side as 🪟. Write CMD in search. You may find CMD as shown in the Figure 1.9

You can start writing a file using *notepad* text editor. Save the file in your computer at some valid location. Open CMD. Execute the file using the following command.

`Rscript.exe r_program.R`

`Rscript.exe` is the R interpreter which executes R programs. In the above example `r_program.R` is the script name. Notice, the above code works for only R "script".

Windows native executable format is `.bat`. This format supports a language convention known as *batch*. Batch can help you write OS specific programs using this extension and execute them in CMD. `.bat` is treated on par with `.exe` for execution in Windows. Suppose you want to mix Windows specific commands with R and intent to exectute both instructions together, then R offers a unique way, i.e. using `R.exe`. Suppose that you have a file `r_program.bat` in which you have both *batch* and R instructions. You can do as shown below to execute such files.



```
[illegible]

> print("Hello World")
[1] "Hello World"
> print("Hello World")
[1] "Hello World"
> Print("Hello World")
Error: could not find function "Print"
> |
```

Figure 1.8: New Script in RGui in Windows OS

```
r_program.bat
```

The press enter button in your keyboard. That's all. Because, Windows knows that it is batch file and doesn't need any predecessor nor follower to execute the file.

Scripts in Linux Terminal

Linux terminal perhaps is more strong and powerful compared to Windows. This is not a biased statement. In Unix culture every thing is done using Terminal. I remember writing CD/DVDs using Linux Terminal in Ubuntu in my earlier days. Only out of curiosity. Linux people are mostly character users. They use very little mouse but lots of keyboard. This may be due to the reason that they belong to hacking community and hackers are power users who need to code in tons.

In Linux R opens in Terminal as a default mechanism but not in RGui. In Linux culture when any language is opened in Terminal it is refered to REPL. REPL stands for Read-Eval-Print-Loop. Suppose if you go to DASH, refer Figure 1.10, R open in Terminal, refer to Figure 1.11. Anyway, coming Linux executing R scripts is very very simple.

You can start writing statements in the Console. However, R Console doesn't support scripts. We need a word file just like in Windows. There are number of option for writing scripts in Linux, namely, *nano, vim, gedit, touch* and manymore. I always prefer *gedit* for it looks exactly like *notepad*. There are three ways to execute just as it is explained under Windows section.

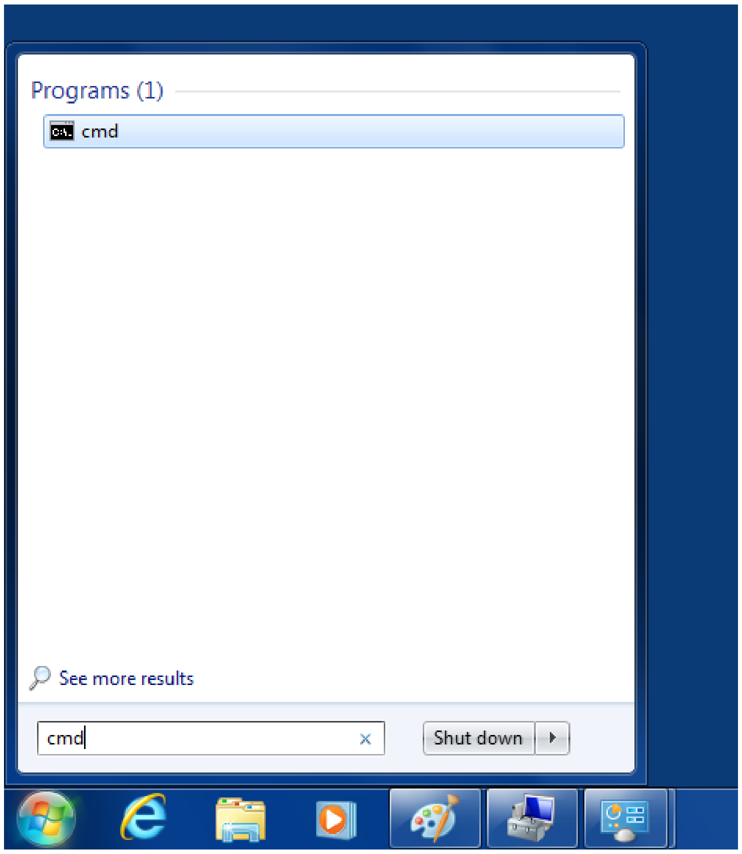
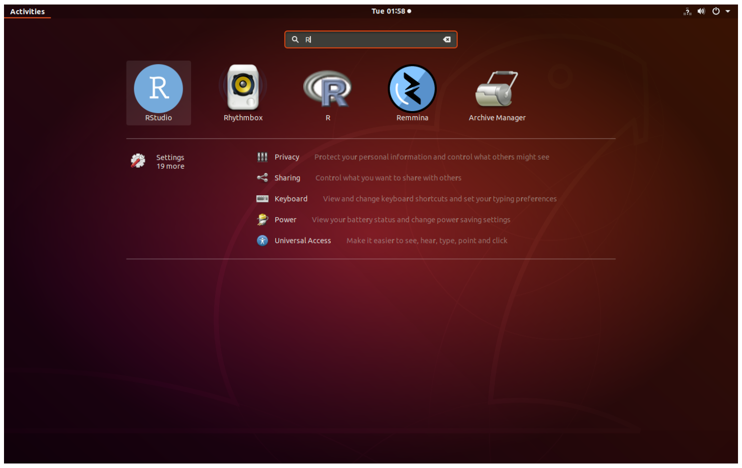
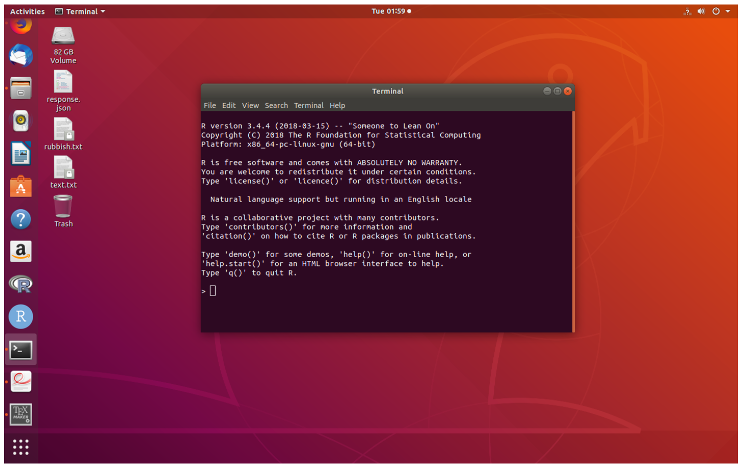


Figure 1.9: CMD Prompt in Windows Start



Privacy Protect your personal information and control what others might see
Sharing Control what you want to share with others
Keyboard View and change keyboard shortcuts and set your typing preferences
Power View your battery status and change power saving settings
Universal Access Make it easier to see, hear, type, point and click

Figure 1.10: Dash in Ubuntu



```
R version 3.4.4 (2018-03-15) -- "Someone to Lean On"
Copyright (C) 2018 The R Foundation for Statistical Computing
Platform: x86_64-pc-linux-gnu (64-bit)

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

  Natural language support but running in an English locale

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

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

>
```

Figure 1.11: R Console in Ubuntu Terminal

Imagine that your program is available in a file named *r_program.R*. Use the following command Suppose you want the output thrown to Trminal.

```
Rscript r_program.R
```

You want the output collected in a file. The you can do as below:

```
R CMD BATCH r_program.R
```

This will create an output file with a name “r_program.Rout”. You need to open using any other application just as

```
cat r_program.Rout
```

The other alternative is to execute the file using R command. Suppose you have a script with a name *r_program.R* and you can execute it with following command.

```
R < r_program.R --no-save
```

You can also use `-save` at the end of the above statement.

The fourth alternative is Linux style. This is called the very famous *dot slash (./)*. Linux has a programming language called BASH. BASH is terminal level programming language which allows users to define functions known as User Defined Functions (UDF). These functions are executed using `./`. This not only for BASH, Linux allows executing any programs by their executable paths. You might be knowing about this, if you had worked with Java or Python. Anyway, you need to add an additional line for path for executable file. Linux creates a path variables and adds to the environment when any application is installed. The common path for these executables is `/user/bin/env`. So there must be `Rscript` available from there for R too. Suppose you have R programs in a file named `r_program.R`. You need to add the following line atop of the program, which means the starting line in the script.

```
#!/usr/bin/env Rscript
```

Now it is possible to execute this file using *dot slash* convention.

```
./r_program
```

Suppose you have certain R function in side this script file such `say_hell()`. Now you can execute the function directly without using file name.

```
./say_hello()
```

Whatever may be the output that will be thrown to Terminal.

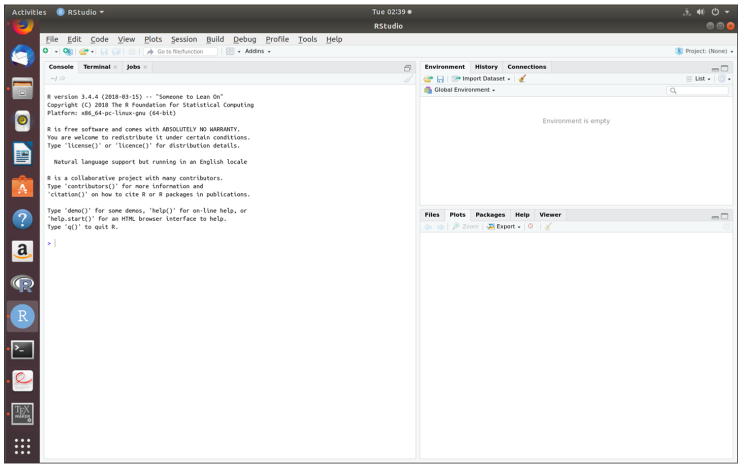
1.2.5 RStudio

We had rather very little introduction to RStudio in previous section i.e. under Day 1 contents. RStudio is not an addin nor a GUI as most of the people confuse, but an Editor or Integrated Development Environment (IDE). What is the Editor/IDE? An Editor/IDE for any programming language is *a software application that provides comprehensive facilities to computer programmers for software development. An IDE normally consists of at least a source code editor, build automation tools, and a debugger.*

RStudio is an integrated development environment (IDE) for R, a programming language for statistical computing and graphics. RStudio today is not just an IDE for R but much more. The `shiny` package developed by RStudio turned R into a web development tool rather than an statistics tool. It is possible to develop web applications using R, just as in Java and Python, but you need Shiny.

Now in this section we will look into installation and usage of the same. Go to `https://rstudio.com/products/rstudio/download/`. Obtain executable file for your platform. RStudio is available for few OSes such as Ubuntu, Debian, Fedora, macOS, OpenSUSE, and Windows. You need to get the file suitable for your platform. If you are in Windows, just donload the `.exe` file to certain valid place (path). Just install as usually as any other software. If you are in Linux just like Ubuntu, you need to download `.deb` file from aforementioned url. Open with software center and click install. Ubuntu does rest of the work for you. You can't find RStudio available from Software Center in Ubuntu. You need to install only from downloaded (.deb) file.

Once after installing RStudio, it just doesn't matter if you are in Windows or Linux or macOS. Any OS can be as best as the other. Because, RStudio has everything that we need to do about R. Figure 1.12 is sample screenshot taken in Ubuntu Linux.



```
R version 3.4.4 (2018-03-15) -- "Someone to Lean On"
Copyright (C) 2018 The R Foundation for Statistical Computing
Platform: x86_64-pc-linux-gnu (64-bit)

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

  Natural language support but running in an English locale

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

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

> 
```

Figure 1.12: RStudio in Ubuntu

Go through all the menus and try to understand each of sub-menu. For instance, you can use File menu to create new scripts, open saved scripts. Edit menu helps in Undo, Redo, Cut, Paste and etc. There is Tools menu which is useful to install packages.

Panes in RStudio

The top-left pane: This pane is useful to write scripts and at times executing and compiling different types of files. This windows or pane is useful to create reports also. I use *R Sweave* which is highly efficient addin in RStudo to create LaTeX related documents. Many times I use this package for reports. At times I do created books in stead of reports and articles. This package needs *TeX* system in your OS.

The top-right pane: This is pane that shows Environment, History and Connections. We will know these things rather more detail in forthcoming sections.

The bottom-left pane: This is rather more important and crux of RStudio. Here you have three other things (1) Console, (2) Terminal and (3) Jobs. Console is R Console. This is none other than R. You can do whatever that can be done in R REPL. The Terminal is the Linux Terminal. Jobs is not very much related to any of our jobs right now. So I leave this for now.

The bottom-right pane: This pane has few very important ingredients such as Files, Plots, Packages, Help and Viewer. We use Plots and Packages quite often from here. Now that we know basics, it is high time to jump into real R.

1.3 Operators & Data Types in R

Contents:

1. Operators
2. Data types

1.3.1 Operators

Assignment operators

R assignment operators. These operators are used to assign values to variables.

Operator	Description
<-, < <-, =	Leftwards assignment
->, -> >	Rightwards assignment

The operators $< -$ and $=$ can be used, almost interchangeably, to assign to variable in the same environment. The $<< -$ operator is used for assigning to variables in the parent environments (more like global assignments). The rightward assignments, although available are rarely used.

Let us try:

```
> x <- 5
> x
[1] 5
> x <<- 6
> x
[1] 6
> x = 9
> x
[1] 9
```

They are all seems to be same at rudimentary level. Believe me there are differences among them. $<< -$ is used while using global variables. To understand global vs. local you need to know a concept known as *scoping*. Usually this parameter is used to create child function inside a parent function. The best example can be:

```
new_counter <- function() {
  i <- 0
  function() {
    # do something useful, then ...
    i <<- i + 1
    i
  }
}

new_counter1 <- function() {
  i <- 0
  function() {
    # do something useful, then ...
    i <- i + 1
    i
  }
}
```

Execute this code using `Ctl+Enter` in RStudio. Now let us verify the code. We will try to create two objects `counter_one()` and `counter_two()` and see how this functions behaves as:

```
> counter <- new_counter()
> counter1 <- new_counter1()
> counter()
[1] 1
> counter()
[1] 2
> counter()
[1] 3
> counter1()
[1] 1
> counter1()
[1] 1
```

`counter()` and `counter1()` are two new objects created using the function `new_counter()`. When we execute `counter()` it gives different counters whereas `counter1()` gives a unique value i.e. 1. This is due to a particular scoping rule known as *functional scoping*. `i` in `new_counter()` is local so everytime the function executes it remembers `i`, whereas the `i` in `new_counter1()` is global. Everytime, the value of `i` set to 1.

Relational Operators

Relational operators are used to compare between values. Here is a list of relational operators available in R.

Suppose imagine that there are two varibles [9] namely a and b. Let us assign two values say, 12, 23. Relational operators are useful greatly if we want to compare these two variables (objects).

```
a = 12; b = 23
```

Operator	Description
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
==	Equal to
!=	Not equal to

```
a<b
```

The output for above script is going to be

```
> a = 12; b = 23
> a<b
[1] TRUE
> a>b
[1] FALSE
> a=b
> a; b
[1] 23
[1] 23
> a = 12; b = 23
> a<=b
[1] TRUE
> a>=b
[1] FALSE
> a!=b
[1] TRUE
```

Arithmetic operators

These operators are used to carry out mathematical operations like addition, subtraction multiplication and division etc. Here is a list of arithmetic operators available in R.

Operator	Description
+	Addition
-	Subtraction
	Multiplication
/	Division
^	Exponent
%%	Modulus (Remainder from division)
%/%	Integer Division

```
> a = 4; b = 8
> b %% a
[1] 0
> a = 3; b = 8
> b %% a
[1] 2
> b %/% a
[1] 2
> a = 2.5; b= 5
> b %/% a
[1] 2
> b %% a
[1] 0
> a = 3; b= 9
> b %/% a
[1] 3
> b %% a
[1] 0
```

Now it might be clear that `%%` gives remainder. `%/%` gives quotient.

Logical Operators

Logical operators are used to carry out Boolean operations like AND, OR etc.

Operator	Description
!	Logical NOT
&	Element-wise logical AND
&&	Logical AND
|	Element-wise logical OR
||	Logical OR

Let us take boolean values such as *TRUE* and *FALSE* for two different objects namely *a* and *b*. [10] Try testing logical operators.

```
a=TRUE; b=TRUE
a & b
a && b
a | b
a || b
```

The output is going to be

```
> a=TRUE; b=TRUE
> a & b
[1] TRUE
> a && b
```

```
[1] TRUE
> a | b
[1] TRUE
> a || b
[1] TRUE
> a=TRUE; b=FALSE
> a & b
[1] FALSE
> a && b
[1] FALSE
> a | b
[1] TRUE
> a || b
[1] TRUE
```

1.3.2 Data types

To make the best of the R language, one needs a strong understanding of the basic data types and data structures and how to operate on them. Data structures are very important to understand because these are the objects you will manipulate on a day-to-day basis in R. Dealing with object conversions is one of the most common sources of frustration for beginners. Everything in R is an object. R has 6 basic data types.

1. character
2. numeric (real or decimal)
3. integer
4. logical
5. complex

Elements of these data types may be combined to form data structures. R has few very valid data structures.

1. atomic vector
2. factors
3. list
4. matrix
5. data frame

R provides quite a few feature functions to examine above types. They are:

1. `class()` - what kind of object is it (high-level)?
2. `typeof()` - what is the object's data type (low-level)?

3. `length()` - how long is it? What about two dimensional objects?
4. `attributes()` - does it have any metadata?

Now let us try and understand few of the above mentioned data types

```
a = "this is a character"
is.character(x)
b = 123
is.numeric(x)
c = 12
is.integer(c)
typeof(c)
d = TRUE
is.logical(d)
is.complex(a+i*b) # throws error
e = complex(a, b)
is.complex(e)
```

You get the following output when you execute the above script.

```
> x = "this is a character"
> is.character(x)
[1] TRUE
> x = 123
> is.numeric(x)
[1] TRUE
> c = 12
> is.integer(c)
[1] FALSE
> typeof(c)
[1] "double"
> d = TRUE
> is.logical(d)
[1] TRUE
> is.complex(a+i*b)
Error: object 'i' not found
> e = complex(a, b)
> is.complex(e)
[1] TRUE
```

The statement `is.complex(a+i*b)` throws error, obviously because the variable `i` is not defined. That is why R has a special function `complex()` to deal with imaginary numbers. The synatax for the same is as follows:

```
complex(length.out = 0, real = numeric(), imaginary = numeric(),
        modulus = 1, argument = 0)
```

Vectors

A vector is the most common and basic data structure in R and is pretty much the workhorse of R. Technically, vectors can be one of two types: (1) atomic vectors and (2) lists. Although the term *vector* most commonly refers to the atomic types not to lists. Although we have a special function `vector()`, we normally don't make much use of it, but we have a couple of other functions to do same. They are `c()`, `:`.

```
x <- c(1:10)
typeof(x)
y <- c("a", "b", 1:10)
typeof(y)
```

The above script gives the following sample output.

```
> x <- c(1:10)
> typeof(x)
[1] "integer"
> c("a", "b", 1:10)
 [1] "a"  "b"  "1"  "2"  "3"  "4"  "5"  "6"  "7"  "8"  "9"  "10"
> y <- c("a", "b", 1:10)
> typeof(y)
[1] "character"
```

The `c` function can also be used to append the existing data vector.

```
> c(x, 11)
 [1]  1  2  3  4  5  6  7  8  9 10 11
```

There is one more way to create number sequences as we did using `:`. That is by using `seq()` function. It is rather more dynamic compared to `:`.

```
> z <- seq(1, 10, 0.5)
> length(z)
[1] 19
```

Factors

Factors are the data objects which are used to categorize the data and store it as levels. They can store both strings and integers. They are useful in the columns which have a limited number of unique values. Like ”Male, ”Female” and True, False etc. They are useful in data analysis for statistical modeling. Factors are created using the `factor()` function by taking a vector as input. Let us try following script.

```
factdat <- c("male", "female", "male", "female",  "
   ↪ female", "male", "female",  "female", "male", "
   ↪ female")
length(factdat)
is.factor(factdat)
summary(factdat)
```

The user object `factdat` is not a *factor*. Why? Because it is still a vector but not a factor.

```
> typeof(factdat)
[1] "character"
> is.vector(factdat)
[1] TRUE
> is.factor(factdat)
[1] FALSE
```

Now let us try this using `factor()`.

```
factdatch <- factor(c("male", "female", "male", "
   ↪ female",  "female", "male", "female",  "female",
   ↪  "male", "female"))
length(factdat)
is.factor(factdat)
summary(factdat)
```

The command `is.factor()` shows that it is “TRUE”.

Lists

Lists are the R objects which contain elements of different types like: numbers, strings, vectors and another list inside it. A list can also contain a matrix or a function as its elements. List is created using `list()` function. Do you remember we created certain objects in one of the earlier section for *a, b, c*. Refer these variables in section 1.3.2. Let us try creating a list using these variables.

```
> a; b; c
[1] TRUE
[1] FALSE
[1] 12
> li <- list(a, b, c)
> li[1]
[[1]]
[1] TRUE
```

```
> li[2]
[[1]]
[1] FALSE

> li[3]
[[1]]
[1] 12
```

The object `li` is user object of the type *list*. The number inside square brackets represents list *index*. We can also accomplish the same thing using the below script.

```
li1 <- list()
li1[1] <- a
li1[2] <- b
li1[3] <- c
> li1
[[1]]
[1] TRUE

[[2]]
[1] FALSE

[[3]]
[1] 12
```

This time we created a user object with a name `li1` and assigned *a, b* and *c* each at a time.

Matrix

A matrix is a collection of data elements arranged in a two-dimensional rectangular layout. Matrix is similar to *vector* but additionally contains the dimension attribute. All attributes of an object can be checked with the `attributes()` function. Dimension can be checked directly with the `dim()` function. Dimension of the matrix can be defined by passing appropriate value for arguments *nrow* and *ncol*. The following script creates a matrix of the order 3×3.

```
> matrix(1:9, nrow = 3, ncol = 3)
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
```

We can see that the matrix is filled column-wise. This can be reversed to row-wise filling by passing TRUE to the argument byrow.

```
> matrix(1:9, nrow=3, byrow=TRUE)
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
```

This time matrix gets values by row in stead of by column. It is possible to name the rows and columns of matrix during creation by passing a 2 element list to the argument dimnames.

```
> x <- matrix(1:9, nrow = 3, dimnames = list(c("X","Y","Z"), c("A","B","C")))
> colnames(x)
[1] "A" "B" "C"
> rownames(x)
[1] "X" "Y" "Z"
> colnames(x) <- c("v1", "v2", "v3")
> rownames(x) <- 1:3
> x
  v1 v2 v3
1  1  4  7
2  2  5  8
3  3  6  9
```

Data Frames

Data frame is a two dimensional data structure in R. It is a special case of a list which has each component of equal length. Each component form the column and contents of the component form the rows.A data frame is a list of vectors which are of equal length. A matrix contains only one type of data. Data frames can be accessed like a matrix by providing index for row and column.

Slice Data Frame

It is possible to slice values of a Data Frame. We select the rows and columns to return into bracket precede by the name of the data frame. A data frame is composed of rows and columns, `df[A, B]`. *A* represents the rows and *B* the columns. We can slice either by specifying the rows and/or columns.

```
> df <- data.frame(airquality)
> df[1:6, ]
  Ozone Solar.R Wind Temp Month Day
1    41     190  7.4   67     5   1
2    36     118  8.0   72     5   2
3    12     149 12.6   74     5   3
4    18     313 11.5   62     5   4
5    NA      NA 14.3   56     5   5
```

```
6    28       NA 14.9   66     5   6
> names(airquality)
[1] "Ozone"   "Solar.R" "Wind"    "Temp"    "Month"   "Day"
> names(df)
[1] "Ozone"   "Solar.R" "Wind"    "Temp"    "Month"   "Day"
> df[1:6, 1:2]
  Ozone Solar.R
1    41     190
2    36     118
3    12     149
4    18     313
5    NA      NA
6    28      NA
```

There is a function called `subset()`, which can be very much helpful while using data frames. You know in `airquality` dataset, there are missing values. Let us use `subset()` to offset all missing data.

```
> head(subset(df, subset = !is.na(Solar.R) ))
  Ozone Solar.R Wind Temp Month Day
1    41     190  7.4   67     5   1
2    36     118  8.0   72     5   2
3    12     149 12.6   74     5   3
4    18     313 11.5   62     5   4
7    23     299  8.6   65     5   7
8    19      99 13.8   59     5   8
```

Data Import

R supports variety of data files such as Excel, Minitab, SPSS, Text, and CSV. For simplicity sake I shall describe CSV file format because CSV is open file format and every software has mechanisms to convert data sets into CSV files.

In CSV file each cell inside such data file is separated by a special character, which usually is a *comma*, although other characters can be used as well. The first row of the data file should contain the column names instead of the actual data.

There is a function known as `read.csv()` in R. The syntax for this function is as follows:

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

We need to pass few parameters such as *file, header, sep, quote, dec, fill, comment.char*. Actually we don't need to care all these parameters if we have the data structured correctly.

```
markdf <- read.csv("/media/ubuntu/C2ACA28AACA27895/
   ↪ Windows/My Writings/Books/R_Book/Datasets/
   ↪ mark_dat.csv")
names(markdf)
typeof(markdf)
```

The sample output is going to be as follows:

```
> names(markdf)
 [1] "gender"         "employment"     "marital.status" "salary"         "age"
 [6] "sat1"           "sat2"           "sat3"           "per1"           "per2"
[11] "per3"           "att1"           "att2"           "att3"
> typeof(markdf)
[1] "list"
```

But is this a data frame? Yes.

```
> is.data.frame(markdf)
[1] TRUE
```

Is this data frame matrix? No.

```
> is.matrix(markdf)
[1] FALSE
```

1.4 About CRAN & Help

1.4.1 CRAN

CRAN is a network of ftp and web servers around the world that store identical, up-to-date, versions of code and documentation for R. The Comprehensive R Archive Network (CRAN) is the main repository for R packages. (If your package concerns computational biology or bioinformatics, you might be interested in Bioconductor, instead.) The main advantage to getting your package on CRAN is that it will be easier for users to install (with `install.packages`). Your package will also be tested daily on multiple systems. There are roughly more than 6000 packages for almost all needs of R users. CRAN is used by all types of individuals such as users, developers. Users use CRAN to download R and those 6000 and odd associated packages, install and use. Developers find CRAN as highly useful for submitting their packages so as to make them available for rest of the community. [11]

CRAN has so many sections to explore. Refer Figure **??**. The very first section is "Mirrors". This section has quite a few ftp mirrors for different locations. People can choose mirrors as by their location. The second section is "What's new?". This section has all latest information related to R. New features and bug fixes of the latest release version of R are documented in the file NEWS (also contained in the R sources). R Core Team makes announcement as and when required to be necessary. These announcements appears under this section. All the announcements are sorted in descending order by year. The third section is “Cran Task Views” also know as CTV in short. CRAN task views aim to provide some guidance which packages on CRAN are relevant for tasks related to a certain topic. They give a brief overview of the included packages and can be automatically installed using the *ctv package*. As on October, 2019 there were 40 task views for various needs of analysts.

Topic	Description
Bayesian	Bayesian Inference
ChemPhys	Chemometrics and Computational Physics
ClinicalTrials	Clinical Trial Design, Monitoring, and Analysis
Cluster	Cluster Analysis & Finite Mixture Models
Databases	Databases with R
DifferentialEquations	Differential Equations
Distributions	Probability Distributions
Econometrics	Econometrics
Environmetrics	Analysis of Ecological and Environmental Data
ExperimentalDesign	Design of Experiments (DoE) & Analysis of Experimental Data
ExtremeValue	Extreme Value Analysis
Finance	Empirical Finance
FunctionalData	Functional Data Analysis
Genetics	Statistical Genetics
Graphics	Graphic Displays & Dynamic Graphics & Graphic Devices & Visualization
HighPerformanceComputing	High-Performance and Parallel Computing with R
Hydrology	Hydrological Data and Modeling
MachineLearning	Machine Learning & Statistical Learning
MedicalImaging	Medical Image Analysis
MetaAnalysis	Meta-Analysis
MissingData	Missing Data
ModelDeployment	Model Deployment with R
Multivariate	Multivariate Statistics
NaturalLanguageProcessing	Natural Language Processing
NumericalMathematics	Numerical Mathematics
OfficialStatistics	Official Statistics & Survey Methodology
Optimization	Optimization and Mathematical Programming
Pharmacokinetics	Analysis of Pharmacokinetic Data
Phylogenetics	Phylogenetics, Especially Comparative Methods
Psychometrics	Psychometric Models and Methods
ReproducibleResearch	Reproducible Research
Robust	Robust Statistical Methods
SocialSciences	Statistics for the Social Sciences
Spatial	Analysis of Spatial Data
SpatioTemporal	Handling and Analyzing Spatio-Temporal Data
Survival	Survival Analysis
TeachingStatistics	Teaching Statistics
TimeSeries	Time Series Analysis
WebTechnologies	Web Technologies and Services
gR	gRaphical Models in R

The next section is *Search.* R user community (without which R would not be what it is today) there are other possibilities to search in R web pages and mail archives. There are number of options to search R information. And most of the data or inforamtion is unorganized. There is tons of information available for R. However, there are few ways for organized search. They are (1) R Site Search: This search will allow to search the contents of the R functions, package vignettes, and task views from online resources. Go to http://finzi.psych.upenn.edu/search.html, (2) R Seek: This is just a wraper to Google search. Visit https://rseek.org/ to know more. The third one in the list is the *Nabble R Forum*. It is an innovative search engine for R messages. Visit https://r.789695.n4.nabble.com/ for more information. R has lot of internal mechanisms to search for information that we will discuss in next section. The other important place is perhaps *R Journal*. The R Journal is the open access, refereed journal of the R project for statistical computing. It features short to medium length articles covering topics that should be of interest to users or developers of R. Visit https://journal.r-project.org/ for more information.

The other important section under CARN are *Packages* and *Manuals*. Coming to *Packages*; there are approximately 6000 and odd number of packages for various needs of statistical analysis. You need to visit CTV for more information. Alternatively visit

```
df.pkgs <- as.data.frame(available.packages(repos = 'https://cloud.r-
project.org/'))
```

This statement needs a package named `curl`. If `curl` is installed properly, the execution creates an object `df.pkgs` in your *workspace*. You may know the number of package by simply executing another statement `dim(df.pkgs)`. This statement brings two numbers first one represents number of rows and second one represents number of columns. [12]

The last section that I would like to highlight is *Manuals*. Manuals are highly useful resources for R users. Users at first may not be knowing anything about usage. Hence, these manuals comes very handy. Manuals for R were created on Debian Linux and may differ from the manuals for Mac or Windows on platform-specific pages, but most parts will be identical for all platforms. The correct version of the manuals for each platform are part of the respective R installations.

There are quite a few things which I am not able to provide through this text. Please visit CRAN repo of R and explore more personally.

1.4.2 Help in R

As I mentioned there are certain functions that helps to know about R. I have to move a bit further or extend this topic beyond *help*. R has a system and every user need to know the very system of R. For instance, when we open R, the R creates an environment for us. This is known as Global Environment. The very first function to execute after installation perhaps is `license()`. This statement gets the following information.

```
This software is distributed under the terms of the GNU General
```

```
Public License, either Version 2, June 1991 or Version 3, June 2007.
The terms of version 2 of the license are in a file called COPYING
which you should have received with
this software and which can be displayed by RShowDoc("COPYING").
Version 3 of the license can be displayed by RShowDoc("GPL-3").

Copies of both versions 2 and 3 of the license can be found
at https://www.R-project.org/Licenses/.

A small number of files (the API header files listed in
R_DOC_DIR/COPYRIGHTS) are distributed under the
LESSER GNU GENERAL PUBLIC LICENSE, version 2.1 or later.
This can be displayed by RShowDoc("LGPL-2.1"),
or obtained at the URI given.
Version 3 of the license can be displayed by RShowDoc("LGPL-3").

'Share and Enjoy.'
```

Frankly we don't need to execute this command because R shows this information in the Console when we open R. However, there is another statement, mind that it is not a function, which gives futher more information is `version`. Following is the sample information.

```
> version
               _
platform       x86_64-pc-linux-gnu
arch           x86_64
os             linux-gnu
system         x86_64, linux-gnu
status
major          3
minor          4.4
year           2018
month          03
day            15
svn rev        74408
language       R
version.string R version 3.4.4 (2018-03-15)
nickname       Someone to Lean On
```

This statement is rather more useful to me to make sense. I can retrieve specific inforamtion as by the parameter. For instance,

```
> ver <- version
> ver$platform
[1] "x86_64-pc-linux-gnu"
> ver$platformos
```

```
NULL
> ver$os
[1] "linux-gnu"
> ver$version.string
[1] "R version 3.4.4 (2018-03-15)"

> cat("I am using R with version", ver$version.string, "with in", ver$os)
I am using R with version R version 3.4.4 (2018-03-15) with in linux-
gnu
```

Notice the last statement! I used a function called `cat()` to make my own custom statements. This is the advantage of programming! Isn't it.

R is programming language and every programming language use *workspace* when we start the program. This particular concept together with *namespace* is highly important for developers. [13] There is guy called *Hadley Wickham*, he is verypopular for his contributions to R community. He did write few very useful yet popular packages for very pretty needs of data analysis. One of his package *ggplot2* is highly popular and also has very efficient methods to plot variety of visuals. He is just like celebrity in R community. He has very nice blog or website. Visit http://hadley.nz/ to know more about him. He had written a book known as *Advanced R*, you can read it at http://adv-r.had.co.nz/. He has provided very nice tutorial on workspace and namespace in R. I may not be able to bring this topic for discussion here, because it is beyond the scope of this book. However, I can help you know in which *Environment* you are in and you can do that by using a command or function `sessionInfo()`. The following is the sample informaton.

```
R version 3.4.4 (2018-03-15)
Platform: x86_64-pc-linux-gnu (64-bit)
Running under: Ubuntu 18.04.3 LTS

Matrix products: default
BLAS: /usr/lib/x86_64-linux-gnu/blas/libblas.so.3.7.1
LAPACK: /usr/lib/x86_64-linux-gnu/lapack/liblapack.so
   ↪ .3.7.1

locale:
 [1] LC_CTYPE=en_IN.UTF-8       LC_NUMERIC=C
    ↪                  LC_TIME=en_IN.UTF-8
 [4] LC_COLLATE=en_IN.UTF-8     LC_MONETARY=en_IN.UTF
    ↪ -8     LC_MESSAGES=en_IN.UTF-8
 [7] LC_PAPER=en_IN.UTF-8       LC_NAME=C
    ↪                     LC_ADDRESS=C
[10] LC_TELEPHONE=C             LC_MEASUREMENT=en_IN.
   ↪ UTF-8 LC_IDENTIFICATION=C
```

```
attached base packages:
[1] stats     graphics  grDevices utils     datasets
   ↪ methods   base

loaded via a namespace (and not attached):
[1] compiler_3.4.4 tools_3.4.4
```

This information represents entire inforamtion about Global Environment. You will find a word *namesapce* at the bottom of the message. If you are global environment then the function `environment()` should be able to get you the following information.

```
> environment()
<environment: R_GlobalEnv>
```

Do you find there is little more detailed information in the above sample message. *attached base packages.* These are the base packages that were installed when we installed *R Base.* These packages are highly useful for basic statistical analysis. If you want to know what are other packages that were installed in your computer. You need to execute `installed.packages()`.

```
ip <- as.data.frame(installed.packages())
names(ip)
head(ip["Package"])
```

This script gives following sample output in the Console.

```
> head(ip["Package"])
              Package
awsMethods awsMethods
dtplyr         dtplyr
ndjson         ndjson
assertthat assertthat
base64enc   base64enc
bindr           bindr
```

Functions

Packages are very special in R. When R is installed it creates a simple system in the computer. This system supports most of the basic needs of statistical analysis. The Package is a collection of several ingredients thy are *functions, data sets, manuals, vignettes* and several other things. Suppose the user wish to do some analysis such as arithmetic mean. How do he or she know. One way is to attend classes or by using Google. R has pretty well defined methods for searching functions. Imagine that I would like to find a function to compute arithmetic mean and many people call it simply *mean* for arithmetic average is a *simple mean*. I might be able to do as below:

```
grep("mean", ls("package:base"))
ls("package:base")[c(grep("mean", ls("package:base")))]
```

This will give you following sample information.

```
> grep("mean", ls("package:base"))
[1] 700 701 702 703 704 705
> ls("package:base")[700]
[1] "mean"
> ls("package:base")[701]
[1] "mean.Date"
> ls("package:base")[c(grep("mean", ls("package:base")
   ↪ ))]
[1] "mean"           "mean.Date"      "mean.default"   "
   ↪ mean.difftime" "mean.POSIXct"   "mean.POSIXlt"
```

The main funtion is `ls()` which is highly sought after function in Linux systems. `ls()` is used for most of the needs while searching for contents in the directory, especially in Linux systems. The function `ls()` is just a wraper for OS function. When we install R in Windows it has MinGW system through which it has access to these Linux based functions. The statement `ls("package:base")` gets all the functions available from package with a name *base*. Do you remember this is one of the package names we have notice while executing `sessionInfo()` above. The statement `grep("mean", ls("package:base"))` retrieves all the functions available from package *base*. However, this function retrieves information as a list of numbers. This means the function we are searching is available in those positions in the list created by this statement. From the above sample output it is clear that the function *mean* is available from the package *base*.

Now the challenge is how do we need to use this function. For that we have another function called *help()*. This function is highly useful to know about any function in R system. Suppose we would like to know about our newly found out function i.e. *mean*. We can do as below.

```
help("mean")
```

Those quotes are very important, because the word is a string. This statement will bring up a manual right side under section “Help” in RStudio. This manual has a content called “example” at the bottom. Suppose if you are interested in knowing only example, then it can be done by using a function `example()`.

```
> example("mean")

mean> x <- c(0:10, 50)
mean> xm <- mean(x)
mean> c(xm, mean(x, trim = 0.10))
[1] 8.75 5.50
>
```

This output really helps in understanding what is all about "mean". In this example it is clear that; the *mean* is a function and it expects an argument this can be a vector of numbers. In the above example the input vector is created by using a function `c()`. Just like:

```
> c(1:10)
 [1]  1  2  3  4  5  6  7  8  9 10
```

Data sets

In R there are quite a few datasets. Most of these datasets are from respective packages. For instance, the dataset *mtcars* is available in *cars* package. The function `data()` can list out all datasets available from one of the base package *datasets*. The package *datasets* is loaded when R open first time. Visit https://vincentarelbundock.github.io/Rdatasets/datasets.html for list of datasets available from package `datasets`. [14] These datasets can be downloadable either in CSV or DOC formats. Try the following command in the Console.

```
data()[[3]][, 3]
```

Gives following sample output.

```
> data()[[3]][1:5, 3]
[1] "AirPassengers"           "BJsales"
   ↪ "BJsales.lead (BJsales)" "BOD"
   ↪                         "CO2"
```

Above sample outpue is First five datasets in the list. For instance, the dataset `AirPassengers` is a very famous one. Excute `help("AirPassengers")` to know more about this dataset. RStudio opens a document file right bottom pane. Use `AirPassengers` to load data into workspace.

```
> AirPassengers
     Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
1949 112 118 132 129 121 135 148 148 136 119 104 118
1950 115 126 141 135 125 149 170 170 158 133 114 140
1951 145 150 178 163 172 178 199 199 184 162 146 166
1952 171 180 193 181 183 218 230 242 209 191 172 194
1953 196 196 236 235 229 243 264 272 237 211 180 201
1954 204 188 235 227 234 264 302 293 259 229 203 229
1955 242 233 267 269 270 315 364 347 312 274 237 278
1956 284 277 317 313 318 374 413 405 355 306 271 306
1957 315 301 356 348 355 422 465 467 404 347 305 336
```

```
1958 340 318 362 348 363 435 491 505 404 359 310 337
1959 360 342 406 396 420 472 548 559 463 407 362 405
1960 417 391 419 461 472 535 622 606 508 461 390 432
> typeof(AirPassengers)
[1] "double"
> attributes(AirPassengers)
$tsp
[1] 1949.000 1960.917   12.000

$class
[1] "ts"
```

Let us try another dataset with a name *BOD*. Use `help("BOD"` to know more about this dataset. This dataset is related to *Biochemical Oxygen Demand Description.* The BOD data frame has 6 rows and 2 columns giving the biochemical oxygen demand versus time in an evaluation of water quality. So this dataset could be a better to start with.

```
> typeof(BOD)
[1] "list"
> names(BOD)
[1] "Time"   "demand"
> head(BOD)
  Time demand
1    1    8.3
2    2   10.3
3    3   19.0
4    4   16.0
5    5   15.6
6    7   19.8
> tail(BOD)
  Time demand
1    1    8.3
2    2   10.3
3    3   19.0
4    4   16.0
5    5   15.6
6    7   19.8
```

Let us try for other dataset *mtcars*. This dataset has information related "Motor Trend Car Road Tests". The data was extracted from the 1974 Motor Trend US magazine, and comprises fuel consumption and 10 aspects of automobile design and performance for 32 automobiles (1973-74 models).

```
> typeof(mtcars)
[1] "list"
> names(mtcars)
 [1] "mpg"  "cyl"  "disp" "hp"   "drat" "wt"   "qsec" "vs"   "am"   "gear" "carb"
```

```
> head(mtcars)
                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
> tail(mtcars)
                mpg cyl  disp  hp drat    wt qsec vs am gear carb
Porsche 914-2  26.0   4 120.3  91 4.43 2.140 16.7  0  1    5    2
Lotus Europa   30.4   4  95.1 113 3.77 1.513 16.9  1  1    5    2
Ford Pantera L 15.8   8 351.0 264 4.22 3.170 14.5  0  1    5    4
Ferrari Dino   19.7   6 145.0 175 3.62 2.770 15.5  0  1    5    6
Maserati Bora  15.0   8 301.0 335 3.54 3.570 14.6  0  1    5    8
Volvo 142E     21.4   4 121.0 109 4.11 2.780 18.6  1  1    4    2
```

This dataset seems to be highly reasonable for analysis. We will try to use this dataset in our forthcoming section. i.e. Visualization in R.

Notes

[1]Please visit https://www.tiobe.com/tiobe-index/ for more information.

[2] Wrathematics, (27 August 2011). “How Much of R Is Written in R”. librestats. http://librestats.com/2011/08/27/how-much-of-r-is-written-in-r/

[3]Visit https://www.gnu.org/licenses/agpl-3.0.en.html for details.

[4]At the time of writing this book I was using Ubuntu 18.04 the code name was “Bionic Beaver”.

[5]You can also use Ubuntu launcher to open Terminal. Press *Windows Super Button*, the button located in the left bottom place at your keyboard. Then search for Terminal. This will show you Terminal Icon. Just click to open Ubuntu or Linux Terminal.

[6]Source: https://fedoraproject.org/wiki/DNF?rd=Dnf

[7]There is more documentation on `dnf` at https://docs.fedoraproject.org/en-US/quick-docs/dnf/

[8]Visit https://www.opensuse.org/ for more details.

[9]In R everything is objective

[10]In R boolena values are capital or upper case letter unline in JS and Python.

[11]Visit https://cran.r-project.org/ for more details on CRAN.

[12]There is really very useful information in R Bloggers with a name “Studying CRAN package names”. This article has methods to retrieve and plot package information from CRAN. Visit https://www.r-bloggers.com/studying-cran-package-names/ for more information.

[13]Visit for more information on workspace and namespace.

[14]Visit https://stat.ethz.ch/R-manual/R-devel/library/datasets/html/00Index.html for a comprehensive view about datasets package.

Chapter 2

Visualization through R

2.1 Graphic devices & Base plots

R follows a concept called *device* to plot graphs. A graphics device is something where you can make a plot appear. Examples include:

1. A window on your computer (screen device)
2. A PDF file (file device)
3. A PNG or JPEG file (file device)
4. A scalable vector graphics (SVG) file (file device)

When you make a plot in R, it has to be "sent" to a specific graphics device. The most common place for a plot to be "sent" is the screen device. On a Mac the screen device is launched with the `quartz()` function, on Windows the screen device is launched with `windows()` function, and on Unix/Linux the screen device is launched with `x11()` function.

When making a plot, you need to consider how the plot will be used to determine what device the plot should be sent to. The list of devices supported by your installation of R is found in `?Devices`. There are also graphics devices that have been created by users and these are aviailable through packages on CRAN. Few devices used in R Base are:

1. pdf - Write PDF graphics commands to a file
2. postscript - Writes PostScript graphics commands to a file
3. xfig - Device for XFIG graphics file format
4. bitmap - bitmap pseudo-device via Ghostscript (if available).

5. pictex - Writes TeX/PicTeX graphics commands to a file (of historical interest only)

How Does a Plot Get Created?

There are two basic approaches to plotting. The first is most common. This involves

1. Call a plotting function like plot, xyplot, or qplot
2. The plot appears on the screen device
3. Annotate the plot if necessary
4. Enjoy

Multiple Open Graphics Devices

It is possible to open multiple graphics devices (screen, file, or both), for example when viewing multiple plots at once. Plotting can only occur on one graphics device at a time, though.

The currently active graphics device can be found by calling `dev.cur()` Every open graphics device is assigned an integer starting with 2 (there is no graphics device 1). You can change the active graphics device with `dev.set(<integer>)` where `<integer>` is the number associated with the graphics device you want to switch to.

The `plot()` function

The core plotting and graphics engine in R is encapsulated in the following packages:

1. `graphics`: contains plotting functions for the "base" graphing systems, including `plot, hist, boxplot` and many others.
2. `grDevices`: contains all the code implementing the various graphics devices, including *X11, PDF, PostScript, PNG*, etc.

The function `plot()` has automatic ploting mechanims. This functions knows as how to make plot based on data type. Suppose you want to plot a dataset say *cars* using `plot()`

```
plot(mtcars)
```

The above statement brings a graph at right bottom pane of RStudio.

The Figure 2.1 is a scatter diagram as by pairs. Each variable is plotted against other variables. The column has names of the dataset. Let us plot scatter

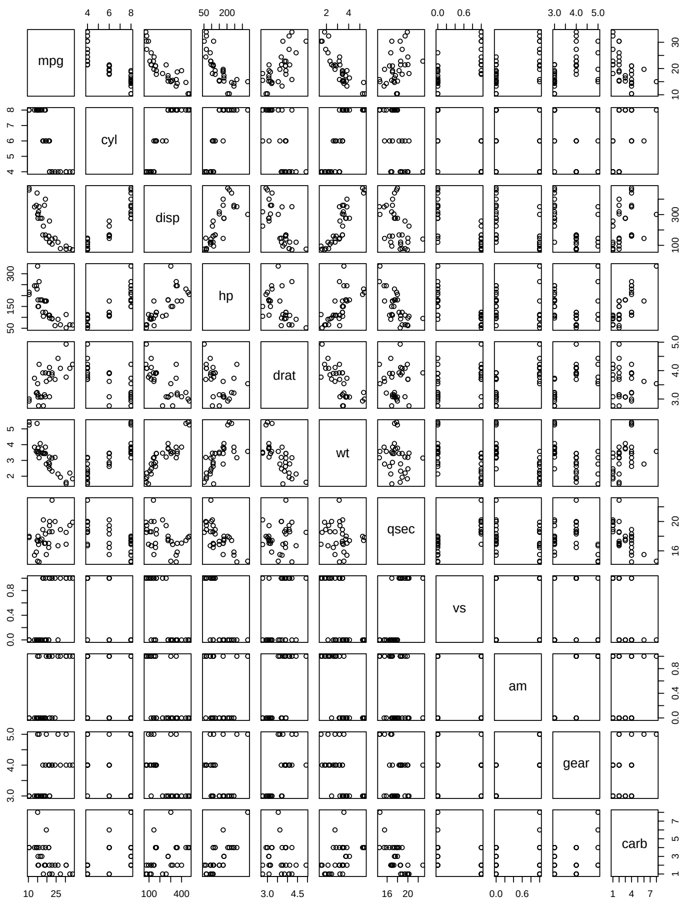


Figure 2.1: `mtcars` plot in RStudio

diagram for two of the variables in *mtcars* namely `mpg` and `hp`. Our assumption is that *mileage (mgp)* depends on *horse power (hp)*.

```
plot(mtcars)
names(mtcars)
with(mtcars, plot(mtcars$mpg, mtcars$hp));
abline(lm(mtcars$mpg ~ mtcars$hp, data=mtcars))
lines(lowess(mtcars$mpg, mtcars$hp), col="blue")
```

The resulting plot will be:

From the Figure 2.2 it is clear that there seems to be inverse relationship between mileage and horse power. Let us try same assumption to difference variables i.e. *mileage (mpg)* and *weight (wt)*.

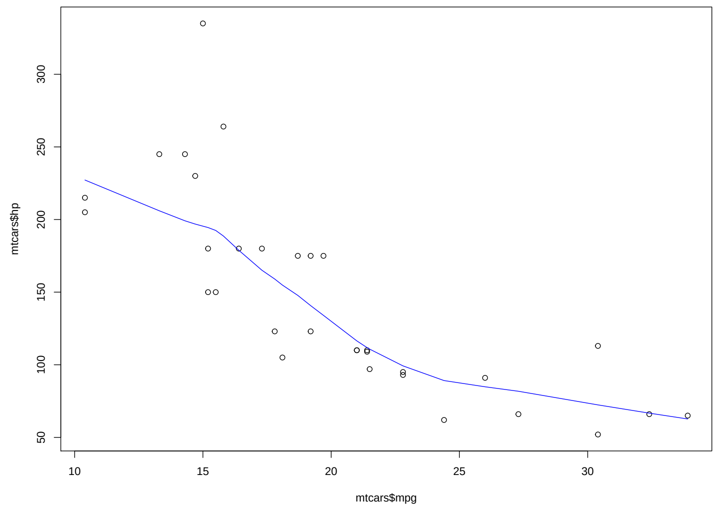


Figure 2.2: `mtcars` plot for *mileage* and *horse power*

```
plot(mtcars$mpg, mtcars$wt);
abline(lm(mtcars$mpg ~ mtcars$wt))
lines(lowess(mtcars$mpg, mtcars$wt), col="blue")
```

There exist inverse relationship even between mileage (mpg) and weight (wt). Of course, it should be. The more the weight of the vehicle and lesser the mileage. This relationship seems to be rather meaningful.

Scatterplot

Scatter plots can be made using **scatter.smooth()** function of *graphics* package. This time let us make two graphs in single canvas.

```
par(mfrow=c(1, 2))
scatter.smooth(mtcars$mpg ~ mtcars$wt)
scatter.smooth(mtcars$mpg ~ mtcars$hp)
```

The statement `par(mfrow=c(1, 2))` is a parameter for graph device. `par()` can be used to set or query graphical parameters. We are instructing device to make two different plots in a single canvas. 1 represents row and 2 represents columns. Each device has its own set of graphical parameters. If the current device is the null device, `par` will open a new device before querying/setting parameters. Parameters are queried by giving one or more character vectors of parameter names to `par`. Several options are available for `par`. You can read by executing `help("par")`

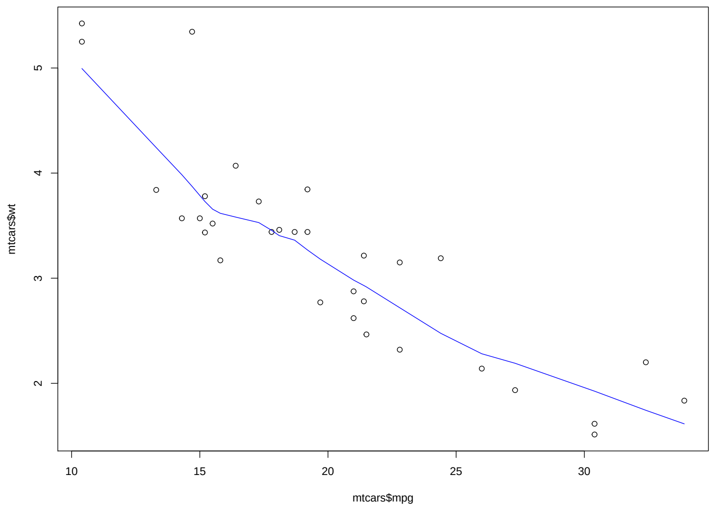


Figure 2.3: `mtcars` plot for *mileage* and *weight*

Let us try another example using `airquality`.

```
with(airquality, plot(Wind, Ozone))
model <- lm(Ozone ~ Wind, airquality)
abline(model, lwd = 2)
```

Generally, the `plot()` function takes two vectors of numbers: one for the x-axis coordinates and one for the y-axis coordinates. However, `plot()` is what's called a generic function in R, which means its behavior can change depending on what kinds of data are passed to the function. We won't go into detail about that behavior for now. The remainder of this chapter will focus on the default behavior of the `plot()` function.

One thing to note here is that although we did not provide labels for the x- and the y-axis, labels were automatically created from the names of the variables (i.e. "Wind" and "Ozone"). This can be useful when you are making plots quickly, but it demands that you have useful descriptive names for the your variables and R objects.

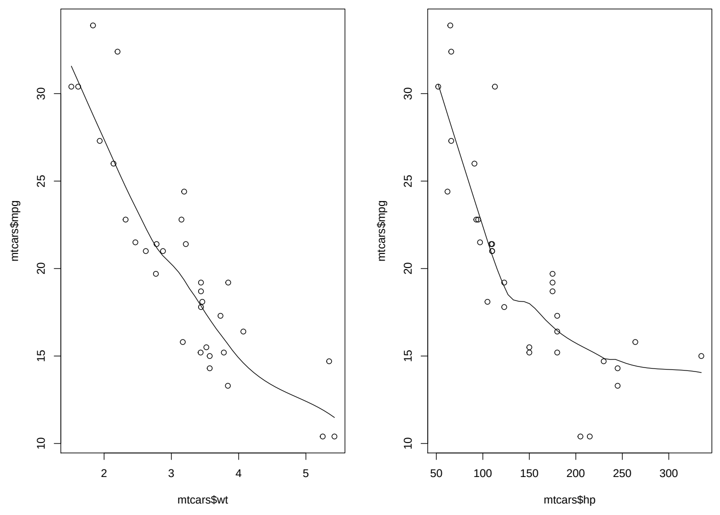


Figure 2.4: `scatter smooth` plot for *mileage* and *weight*

2.2 Base Plots

2.2.1 Histogram

Here is an example of a simple histogram made using the `hist()` function in the graphics package. Run this code and your graphics window is not already open, it should open once you call the `hist()` function.

```
hist(mtcars$mpg)
hist(mtcars$mpg, freq = FALSE); lines(density(mtcars$mpg), col = "red")
```

The data distribution mileage (mpg) seems to be roughly normal.

2.2.2 Box Plots

Boxplots can be made in R using the `boxplot()` function, which takes as its first argument a formula. The formula has form of $y \sim x$. Anytime you see a $\sim$ in R, it's a formula. Here, we are plotting ozone levels in New York by month, and the right hand side of the $\sim$ indicate the month variable. However, we first have to transform the month variable in to a factor before we can pass it to `boxplot()`, or else `boxplot()` will treat the month variable as continuous.

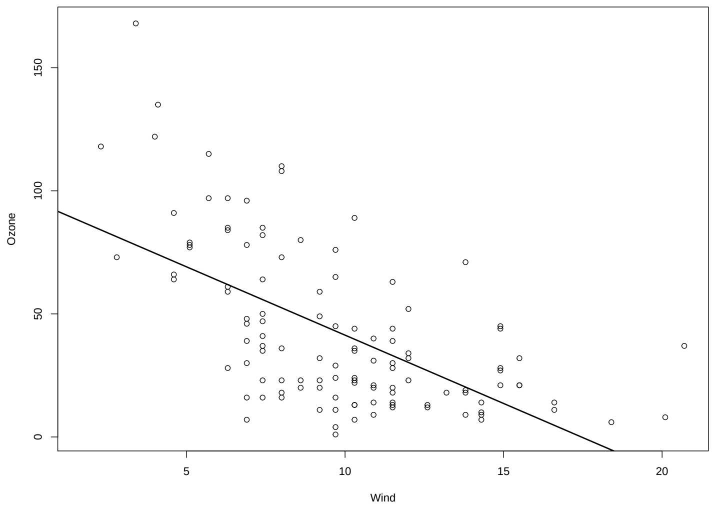


Figure 2.5: `scatter` plot for *airquality* dataset.

```
airquality <- transform(airquality, Month = factor(
    ↪ Month))
boxplot(airquality$Ozone ~ airquality$Month,
    ↪ airquality, xlab = "Month", ylab = "Ozone (ppb)
    ↪ ")
```

Each boxplot shows the median, 25th and 75th percentiles of the data (the "box"), as well as +/− 1.5 times the interquartile range (IQR) of the data (the "whiskers"). Any data points beyond 1.5 times the IQR of the data are indicated separately with circles.

In this case the monthly boxplots show some interesting features. First, the levels of ozone tend to be highest in July and August. Second, the variability of ozone is also highest in July and August. This phenomenon is common with environmental data where the mean and the variance are often related to each other.

From Figure 2.7 it is clear that there are outliers for months 5, 6, 8 and 9. Months 7 and 8 did not have any outliers for Ozone. This means the ozone levels are highly unsteady for all the months excet 7th and 8th months. We need to look into dataset to interpret more accurately if we can look into information. Use `help("airquality"` to understand measurement level for "Ozone".

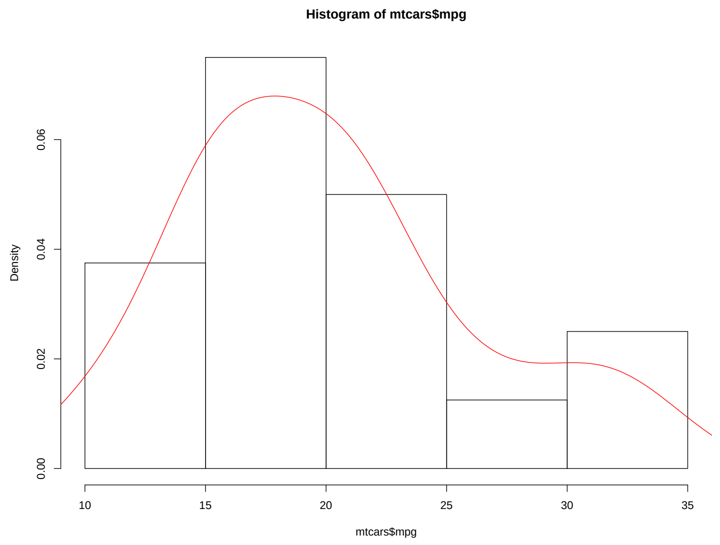


Figure 2.6: histogram for *mileage*

2.2.3 Pie charts

Pie charts are a very bad way of displaying information. The eye is good at judging linear measures and bad at judging relative areas. A bar chart or dot chart is a preferable way of displaying this type of data.

Let us try pie chart using BOD dataset, because this dataset has 6 rows and 2 columns. That makes it very simple for pie chart.

```
pie(BOD$demand,labels = BOD$Time)
```

We can stretch this pie chart further.

```
bodlabels <- round(BOD$demand/sum(BOD$demand)*100, 1)
bodlabels <- paste(bodlabels, "%", "")
pie(BOD$demand, labels = bodlabels)
```

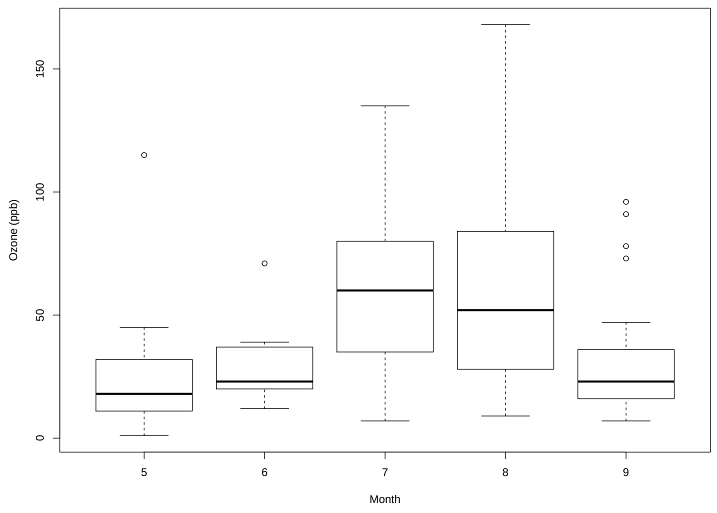


Figure 2.7: Box plot for *Ozone* and *Month* in `airquality` dataset.

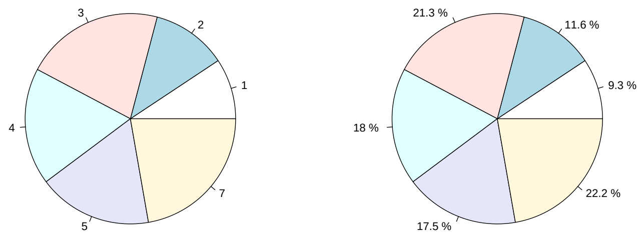


Figure 2.8: Pie chart for `BOD` dataset with labels.

Chapter 3

Univariate Analysis

3.1 A very little introduction about Statistics

"If your experiment needs a statistician, you need a better experiment."

- Ernest Rutherford

Ironically I could not get one positive quote on Statistics. The above quote is partially better among the rest. Whatever, world might think, but to me it is not possible to discuss Statistics in one go. It is continuous learning process. Most of the time people dislike statistics because Statistics is not strictly mathematical in nature. Mathematics is the mother of all sciences. Coming to Statistics, it is not strictly science and there is little pseudoscience in it. Just a joke! May be my being stated as such is due to the way I experience events around me when I am at work. One potential reason for this restlessness might be due to lack of very strict scientific implemetations in Statistics. Science is respected by its operability in action. Science is a fiction as long as it is in theory and untested. At lest to me!

This Chapter deals with few very familiar Statistics. I shall discuss basics such as *univariate, bivariate* and *multivariate* statistics in this text. There is no comprehensive method to classify statistics. Methods are abundant. For instance, as it was mentioned earlier Statistics can be classified as below:

1. Univariate
2. Bivariate
3. Multivariate

and this classification is based on variables. *Univariate* anlaysis deals with only one variable and typical methods might be (1) summaries, (2) descriptives, (3) tests (such as normality tests) and many more. *Bivariate* analysis deals with statistics which attempts to explain relationships or dependencies between two variables. These techniques might include (1) cross tabulations, (2) correlations, (3) regression. *Multivariate* analysis deals with those statistical techniques which analyze data sets with more than two variables. There is another classification based on type of analyses.

1. Descriptive

2. Predictive

3. Prescriptive

This classification consider type of analysis as criteria. *Descriptive statistics* is just like the summary category in the above classification. Descriptive statistics deals with measures that explains location (mean) and scale (dispersion). Briefly descriptive statistics try to cover three types of measures related to *center, dispersion* and *shape*. Coming to *predictive statistics*, it deals with all bivariate methods related to regression (mostly). Many experts mention regression as "hydrozen bomb" in the arsenel of Statistics. Coming to *prescriptive statistics*, it deals with much of Mathematics and less Statistics. Most of the numerical optimization is covered under prescriptive methods. This word is relatively a new term used data analytics community just recently. The third classification I would like to highlight is as follows:

1. Summary

2. Inferential

3. Exploratory

Few texts follows the above classifiction. Again, *summary* statistics is just like as descriptive statistics but not exactly. Summaries deals with aggregations. Aggregation literally means *any process which information is explained by summaries*. Data is processed for few summary statistics such as *sum, maximum, minimum, range* and etc. It is all confusing right! that is why you don't get people who like statistics all the time. While coming to *inferential* statistics, they are pretty much same like *predictive in second classification*. Inference literally means: *"a conclusion reached on the basis of evidence and reasoning."* These statistics most of the time found to be strictly connected with a very high and grand technical word known as *"hypothesis"*. If at all anybody refer statistics to science, that is because of this *hypothesis* testing. There is also lot of criticism on testing hypotheses in the area of Statistics. [15]

Now it is time to discuss about methods. Please be careful and cautious of your brain. Absolutely no guarantee or warranty what so ever. Reader beware!

3.2 Summaries and Descriptices

Summary statistics are used to summarize a set of observations, in order to communicate the largest amount of information as simply as possible. Statisticians commonly try to describe the observations in

1. a measure of location, or central tendency, such as the arithmetic mean
2. a measure of statistical dispersion like the standard mean absolute deviation
3. a measure of the shape of the distribution like skewness or kurtosis
4. if more than one variable is measured, a measure of statistical dependence such as a correlation coefficient

A common collection of order statistics used as summary statistics are the *five number summary*, sometimes extended to a *seven number summary*, and the associated box plot. Entries in an analysis of variance table can also be regarded as summary statistics.

There are two very important functions which helps a lot to compute summaries of data frames. They are (1) with and (2) by. `with` is a function that evaluates an R expression in an environment constructed from data, possibly modifying (a copy of) the original data. The syntax for this function is as follows:

```
with(data, expr, ...)
```

`with` is a generic function that evaluates *expr* in a local environment constructed from data. The other one is `by` and syntax for this function is as follows:

```
by(data, INDICES, FUN, ..., simplify = TRUE)
```

`by` require *indices* to execute *FUN*. We shall use these two functions to obtain summary statistics. Let us take data set we used in Chapter 1 named `markdf`. We might be able to obtain summaries as shown below:

```
> with(markdf, mean(markdf$salary))
[1] 55
> with(markdf, {
+   cbind(mean(markdf$salary), sd(markdf$salary))
+   })
     [,1]     [,2]
[1,]   55 26.24421
```

Suppose we wish to know all other measures in detial. We might be able to use a special but highly generic function `summary()`

```
> summary(markdf$gender)
female   male
    17     13
> summary(markdf$age)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
  19.00   25.25   42.00   40.60   55.00   59.00 
> summary(markdf$salary)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
  12.00   35.00   48.50   55.00   80.75   95.00 
```

We are missing one measure out of those seven that is *mode*. The mode is a frequently occuring observation in the given data distribution. As such *Max.* can serve as *Mode* but is wrong. Mode requires classes. We might be able to compute Mode for *gender* in `markdf` dataset as in following manner.

```
> with(markdf, max(table(markdf$gender)))
[1] 17
```

Both *age* and *salary* need to be converted to categorical dat before computing mode. And the way to do it is using *if()* statement. Let us see how to put `if()` statement into action.

```
markdf$salary2 <- with(markdf, ifelse(salary >= 0 &
   ↪ salary < 20, "0 to 20", ifelse(salary >= 20 &
   ↪ salary < 40, "20 to 40", ifelse(salary >= 40 &
   ↪ salary < 60, "40 to 60", ifelse(salary >= 60 &
   ↪ salary < 80, "60 to 80",  "80 to 100")))))
```

And the additional variable `salary2` will appear as below:

```
> head(markdf)
  gender employment        marital.status salary age sat1 sat2 sat3 per1 per2 per3 att1 att2 att3
1   male        top                single     33  51    3    3    3    5    5    5    4    4    4
2 female     middle               nuclear     93  56    2    2    2    3    3    3    3    3    3
3   male        low              combined     47  43    1    3    1    4    4    5    4    4    4
4 female     middle               widowed     35  58    2    1    1    5    5    5    5    5    5
5 female        low widowed with children     69  20    4    4    4    4    1    4    4    1    4
6   male     middle              combined     36  41    3    3    3    1    1    1    4    4    4
  salary1   salary2
1       2  20 to 40
2       5 80 to 100
3       3  40 to 60
4       2  20 to 40
5       4  60 to 80
6       2  20 to 40
```

There is another way to make tables using *quantiles*. There is one function known as `quantile()` in R to take care of all quantiles such as quartiles, deciles and percentiles etc. First we need to make quartiles. Later these quartiles can be used to make a new variable possibley `age1` in same dataset.

```
per0 <- min(markdf$age)
per25 <- quantile(markdf$age, 0.25)
per75 <- quantile(markdf$age, 0.75)
per100 <- max(markdf$age)
```

The above chunk helps us to make four categories such as $min, < 25, < 75, < 100, max$, the same convention as in summaries. What we are missing is median which represents < 50. Once we make these cut-off points, these points can be used to off-set data.

```
markdf$age1[markdf$age > per0 & markdf$age <= per25] =
   ↪  "less than 25"
markdf$age1[markdf$age > per25 & markdf$age <= per75]
   ↪ = "less than 75"
markdf$age1[markdf$age > per75 & markdf$age <= per100]
   ↪  = "less than 100"
markdf$age1 = factor(markdf$age1, levels = c("less
   ↪ than 25", "less than 75", "less than 100" ))
```

The last statement takes care of assigning names to newly created variable `age1`. Now we might be able to check our dataset.

```
> with(markdf, max(table(markdf$age1)))
[1] 18
> table(markdf$age1)

 less than 25  less than 75 less than 100
           12            18             0
```

3.3 Apply functions - `apply`, `tapply`, `sapply`, `lapply`

R has few other methods to try summaries using peculier functions such as `apply`, `tapply`, `sapply`, and `lapply`. These functions are simply referred to *apply* functions.

The `apply()` collection is bundled with R Base package. These function is available immediately after installing R. The `apply()` function can be feed with many functions to perform redundant application on a collection of object (data frame, list, vector, etc.). The purpose of `apply()` is primarily to avoid explicit uses of loop constructs. They can be used for an input list, matrix or array and apply a function. Any function can be passed into `apply()`

Suppose we want to find mean and that need to be done for all variables in the dataset. The only option in R is to make use of `if()` statement. Such things might be palatable for programmers, but most of the R users are statisticians and they are not into much of programming activity. This problem can be simply achieved using `apply()` function without using `if()` statement.

```
names(markdf)
apply(markdf[, 6:14], 2, median)
```

Above chunk calculates *median* for all those variables that are numerical. Please notice we indexed data i.e. only those variables that are ordinal and also study variables.

Next function is `tapply()`. This function is highly useful whenever we wish to make calculations across the table. Suppose we intent to see *mean* salary for gender. Plesae notice *salary* is not categorical whereas *gender* is categorical. Earlier we converted non-categorical variables into categorical to make calculations. Actually we can escape all that work by using `tapply()`.

```
> tapply(markdf$salary, markdf$gender, mean)
  female     male
58.35294 50.61538
```

Next functions are `sapply` and `lapply`. In fact, these two functions does same activity but the difference is only in the way they output results. The `sapply()` output result as in array, whereas `tapply()` output results as in list. Let us try for study variables in the dataset *markdf*.

```
> sapply(markdf[, 6:14], median)
sat1 sat2 sat3 per1 per2 per3 att1 att2 att3
 3.0  3.0  3.0  3.5  2.5  4.0  4.0  3.5  4.0
> lapply(markdf[, 6:14], median)
$sat1
[1] 3

$sat2
[1] 3

$sat3
[1] 3

$per1
[1] 3.5

$per2
[1] 2.5

$per3
[1] 4

$att1
[1] 4

$att2
[1] 3.5

$att3
[1] 4
```

These functions are highly useful for those whose primary goal is to *use* of R for data analysis. Programmers may not like it because, they migh use raw commands just as conditional statements [`if()`] together with loops [`for()`].

3.4 Frequency Tables & Crosstabs

There is one very simple function called as `table()` to make tables in R. We can try making a table *gender* against *salary4*.

```
> tab <- table(markdf$gender, markdf$salary4)
> tab

         less than 25 less than 75 less than 100
  female            5            6             5
  male              3            6             3
```

The above tables is possess just frequencies of *age* as by *salary2*. At times we would expect marginal values.

```
> margin.table(tab, 1)

female   male
    16     12
```

`margin.table()` calculates sum of all salary-wise categories against *gender*. 16 is the total of all the category wise values for salary4. The opposite scenario i.e. sums of *gender* against *salary* can be done by using following statement.

```
> margin.table(tab, 2)

 less than 25  less than 75 less than 100
            8            12             8
```

`margin.table()` is useful for calculating sums. But in case if we wish to calculate percentages we might use `prop.table()`.

```
> prop.table(tab)

         less than 25 less than 75 less than 100
  female    0.1785714    0.2142857     0.1785714
  male      0.1071429    0.2142857     0.1071429
> prop.table(tab, 1)

         less than 25 less than 75 less than 100
  female       0.3125       0.3750        0.3125
  male         0.2500       0.5000        0.2500
> prop.table(tab, 2)

         less than 25 less than 75 less than 100
  female        0.625         0.500          0.625
  male          0.375         0.500          0.375
```

The first table is cell percentages, which means frequencies of the crosstable is devided by grand totals. The second tables shows proportions by row totals. Third table shows proportions by column totals.

There is another function known as `ftable()`. This function gives rather more information adding little decoration to the freequency table. Suppose we are intersted in more than two variables, the `table()` function gives murky output. `ftable()` gives rather attractive tables.

```
> ftable(markdf$gender, markdf$salary4, markdf$employment)
                     low middle top

female less than 25     3      1   1
       less than 75     3      2   1
       less than 100    4      1   0
male   less than 25     1      1   1
       less than 75     2      4   0
       less than 100    0      2   1
```

There is even other function known as `xtabs()` but this function does nothing more compared to `table()`.

3.5 Tests of independence

3.5.1 Two way tables

For 2 way tables you can use *chisq.test()* to test independence of the row and column variable. By default, the *p-value* is calculated from the asymptotic *chisquared* distribution of the test statistic.

Suppose if we assume that salary depends on gender, which means thre are inter-gender level differences to salary. This assumption can serve as null hypothesis for our test. This hypothesis can be tested using chi-squared test. Usually for chi-squared test the null hypothesis is *the variables under study are independent.*

```
> tab

         less than 25 less than 75 less than 100
  female            5            6             5
  male              3            6             3

> chisq.test(tab)

Pearson's Chi-squared test
```

```
data:  tab
X-squared = 0.4375, df = 2, p-value = 0.8035

Warning message:
In chisq.test(tab) : Chi-squared approximation may be incorrect

> fisher.test(tab)

Fisher's Exact Test for Count Data

data:  tab
p-value = 0.7959
alternative hypothesis: two.sided
```

Unfortunately there isn't evidence in support of our hypothesis. The P Value is 0.8035 this is far beyond compared to threshold value 0.05. So, we fail to reject null hypothesis and the assumption is not true. Hence we infer that the *salary* is independent of *gender*. The gender discrimination is not found in the data.

What do we need to do for 3 way tables? There is a function `mantelhaen.test()`. this function is usful to find inter-categorical differences to another categorical variable provided the third dimension has levels of strata.

Suppose if we wish to find *gender* level differences to *salary* and *employment* serves as stratum. We can perform contingency test as shown below.

```
> mantelhaen.test(markdf$gender, markdf$salary4, markdf$employment)

Cochran-Mantel-Haenszel test

data:  markdf$gender and markdf$salary4 and markdf$employment
Cochran-Mantel-Haenszel M^2 = 0.1852, df = 2, p-value = 0.9116
```

`mantelhaen.test()` function is useful to perform a *Cochran-Mantel-Haenszel chi-squared test* of the null hypothesis that two nominal variables are conditionally independent in each stratum, assuming that there is no three-way interaction. This function requires 3 dimensional data, where the last dimension refers to the strata. So, now looking at the P Value it is possible for us to infer that the conditional independence does not exit in data.

Loglinear Models

Mutual Independence

A log-linear model is a mathematical model that takes the form of a function whose logarithm equals a linear combination of the parameters of the model, which makes it possible to apply (possibly multivariate) linear regression. That is, it has the general form. Suppose if we imagine that *employment*, *gender* and *salary* exhibit pair wise dependence (H_0). We may be able to test using `loglm()` from *MASS* librray.

```
> xtab <- xtabs(~ markdf$gender+markdf$salary4+markdf$employment, markdf)
> library(MASS)
> loglm( ~ markdf$gender+markdf$salary4+markdf$employment, xtab)
Call:
loglm(formula = ~markdf$gender + markdf$salary4 + markdf$employment,
    data = xtab)

Statistics:
                      X^2 df  P(> X^2)
Likelihood Ratio 11.815507 12 0.4606081
Pearson           9.774403 12 0.6357447
```

Both *Likelihood Ratio* and *Pearson r* are statistically insignificant (P Value > 0.05). So we fail to reject H_o that *employment, gender* and *salary* are independent as by pairs. So there isn't any evidence in support of *mutual evidence.*

Partial Independence

Partial Independence assumes that A is partially independent of B and C (i.e., A is independent of the composite variable BC). Let us assume that *salary* is independent of *gender* and *marital status.*

```
> llt <- loglin(xtab, list(c(1, 2), c(2, 3), c(3, 1)))
3 iterations: deviation 0.06122421
> llt
$lrt
[1] 5.572882

$pearson
[1] 4.07703

$df
[1] 4

$margin
$margin[[1]]
[1] "markdf$gender"  "markdf$salary4"

$margin[[2]]
[1] "markdf$salary4"    "markdf$employment"
```

```
$margin[[3]]
[1] "markdf$employment" "markdf$gender"


> 1 - pchisq(llt$lrt, llt$df)
[1] 0.233397
```

The P Value is greter than 0.05, so we fail to reject null hypothesis. *gender, employment* and *salary* are independent as by their respective margins.

Conditional independence

Conditional independence is just like A is independent of B, given C. Imagine that A stands for *salary*, B stands for *employment* and C stands for *gender*. Suppose if we assume that *salary* is independent of *gender* that is because of *employment*.

```
> loglm(~ markdf$gender+markdf$salary4+
   ↪ markdf$employment+markdf$salary4*markdf$gender+
   ↪ markdf$employment*markdf$gender, xtab)
Call:
loglm(formula = ~markdf$gender + markdf$salary4 +
   ↪ markdf$employment +
    markdf$salary4 * markdf$gender + markdf$employment
       ↪  * markdf$gender,
    data = xtab)

Statistics:
                      X^2 df  P(> X^2)
Likelihood Ratio 7.146439  8 0.5209187
Pearson          5.028571  8 0.7545181
```

So we don't have evidence agaist the hypothesis that *salary* depends on *gender* and *employment*. This gives us a rough idea that there may not be any interaction at all in our data. Becuase there is no any type of dependence as far as *gender, employment* and *salary* are concerned. However, our suspicion must be tested using *3 way indepence test*.

3 way independence

```
> loglm(~markdf$salary4+markdf$employment+
   ↪ markdf$gender+markdf$salary4*markdf$employment+
   ↪ markdf$salary4*markdf$gender+markdf$employment*
   ↪ markdf$gender, xtab)
Call:
```

```
loglm(formula = ~markdf$salary4 + markdf$employment +
   ↪ markdf$gender +
    markdf$salary4 * markdf$employment +
       ↪ markdf$salary4 * markdf$gender +
    markdf$employment * markdf$gender, data = xtab)

Statistics:
                      X^2 df  P(> X^2)
Likelihood Ratio 5.572922  4 0.2333935
Pearson          4.083264  4 0.3948545
```

So now we got full evidence that all of these variables are totally independent of each other.

3.6 Statistical diagnosis

Parametric statistics is a branch of statistics which assumes that sample data come from a population that can be adequately modeled by a probability distribution that has a ***fixed*** set of parameters. Conversely a non-parametric model differs precisely in that the parameter set (or feature set in machine learning) is not fixed and can increase, or even decrease, if new relevant information is collected. The normal family of distributions all have the same general shape and are parameterized by mean and standard deviation. That means that if the mean and standard deviation are known and if the distribution is normal, the probability of any future observation lying in a given range is known.

The very base for these parametric tests is *probability distributions*. A probability distribution describes how the values of a random variable is distributed. For example, the collection of all possible outcomes of a sequence of coin tossing is known to follow the binomial distribution. Whereas the means of sufficiently large samples of a data population are known to resemble the normal distribution. Since the characteristics of these theoretical distributions are well understood, they can be used to make statistical inferences on the entire data population as a whole. Few important distributions are as follows:

1. Binomial Distribution
2. Poisson Distribution
3. Continuous Uniform Distribution
4. Exponential Distribution
5. Normal Distribution
6. Chi-squared Distribution
7. Student t Distribution
8. F Distribution

Many of the distributions like *binomial, poisson, uniform, exponential* etc. are situational in nature.

3.6.1 Parametric

R has functions to deal with all these distributions. Suppose if we ask a question that is there 50-50 chance for gender if we randomly pick them from 30. The success is 0.5.

```
> dbinom(1, 2, 1/length(levels(markdf$gender)))
[1] 0.5
```

It is 14% likely. Suppose if we ask a question what is the probability of getting a person whose salary is "("less than 40) (which means response 2) out of 30 individuals? This is a binomial distribution. We know that the probability of succes is 0.2.

```
> dbinom(2, 5, 1/length(levels(markdf$salary4)))
[1] 0.3292181
```

The answer is 32 % likely. Coming to *normal distribution.* Suppose if we wish to test the probability of getting a random individual whose salary is greter than 75 thousand or less from a sample of 30 given their mean and standard deviation. We can do as below:

```
> pnorm(75, mean(markdf$salary), sd(markdf$salary))
[1] 0.7769917
```

So it is 77 % likely. Try using minimum and maximum values. You might find that the likeliness increases with minimum salary, and likeliness descreases with maximum salary.

We can also model our satisfaction variable using *exponential* distribution. If we assume that increase or descrease of satisfaction behaves like exponential distribution. Suppose if the mean satisfaction of our survey respondents is known.

```
> pexp(4, 1/mean(markdf$sat1))
[1] 0.7768698
```

It is 77 % likely that a randomly choosen individual might be less than just satisfied given a average satisfaction is less than 3.

T Test

The t-test is any statistical hypothesis test in which the test statistic follows a Student's t-distribution under the null hypothesis.

A t-test is most commonly applied when the test statistic would follow a normal distribution if the value of a scaling term in the test statistic were known. When the scaling term is unknown and is replaced by an estimate based on the data, the test statistics (under certain conditions) follow a Student's t distribution. The t-test can be used, for example, to determine if the means of two sets of data are significantly different from each other.

Suppose if we wish to compute one sample t-test for *salary.* I choose $H_0 = 0$.

```
> t.test(markdf$salary, mu = 0)

One Sample t-test

data:  markdf$salary
t = 11.479, df = 29, p-value = 2.645e-12
alternative hypothesis: true mean is not equal to 0
95 percent confidence interval:
 45.20025 64.79975
sample estimates:
mean of x
       55
```

Oh! The value is significantly different from zero. So we have evidence in support of H_a and there by we reject H_0 that the mean value of salary is significantly different from zero. From the output it is clear that we need at least 45 to 64 mean value to to accept H_0. The CI is the shock absorber for mean value.

Univariate normality

Normality tests are used to determine if a data set is well modeled by a normal distribution and to compute how likely it is for a random variable underlying the data set to be normally distributed. An informal approach to testing normality is to compare a histogram of the sample data to a normal probability curve. The empirical distribution of the data (the histogram) should be bell shaped and resemble the normal distribution. This might be difficult to see if the sample is small.

A graphical tool for assessing normality is the normal probability plot, a quantile quantile plot (QQ plot) of the standardized data against the standard normal distribution. Here the correlation between the sample data and normal quantiles (a measure of the goodness of fit) measures how well the data are modeled by a normal distribution. For normal data the points plotted in the QQ plot should fall approximately on a straight line, indicating high positive correlation. These plots are easy to interpret and also have the benefit that outliers are easily identified.

QQPlot for *age* distribution in *markdf* dataset.

```
hist(markdf$age, freq = FALSE); lines(density(markdf$age), col = "red")
```

Oh! That's really obnoxious. The data distribution is bimodal and non-normal. Let me confirm with *t test*.

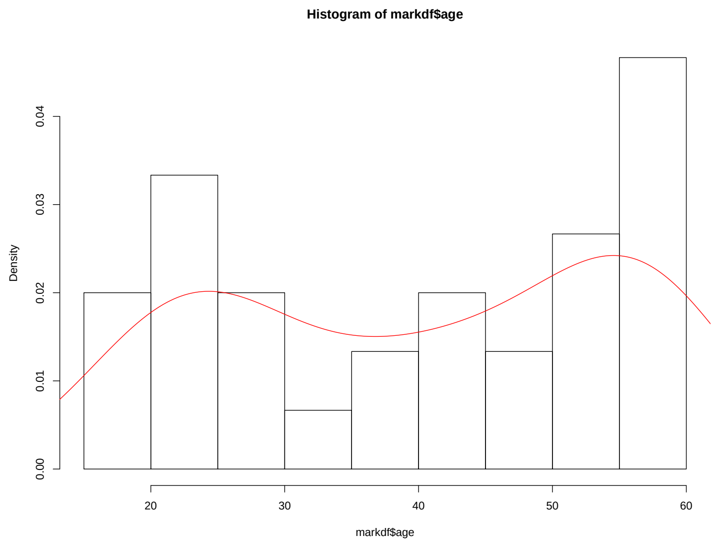


Figure 3.1: Histogram for *age*

```
> t.test(markdf$age, mu =  0)

One Sample t-test

data:  markdf$age
t = 14.975, df = 29, p-value = 3.508e-15
alternative hypothesis: true mean is not equal to 0
95 percent confidence interval:
 35.0549 46.1451
sample estimates:
mean of x
     40.6
```

Yes! the mean is significantly different from zero. And now let us make another visual known as qqline for *age*.

```
qqnorm(markdf$age)
qqline(markdf$age)
```

Look at Figure 3.6.1 it is strictly not normal. This type of analysis is called ***infographics***. However, this might be naive way of coming to conclusion that

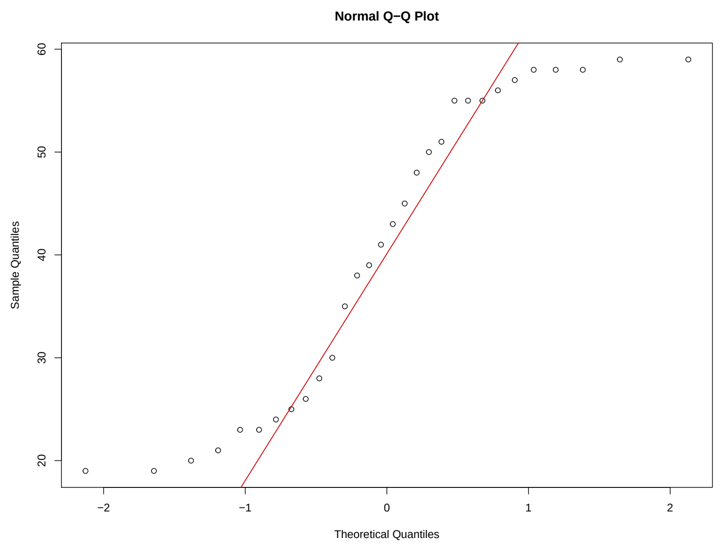


Figure 3.2: Histogram for *age*

the data distribution *age* is non-normal just by looking at visuals. The departure what-so-ever we are perceiving might be due to *sampling error*. This may not be true to in population. If someone ask this type of questions we need to show evidence. And to show evidence we need to perform some statistical test. In R there are quite a few statistical tests for univariate data. Following is the list of very few of such tests:

1. Shapiro Wilk test
2. Anderson Darling test
3. Cramer-von-Mises test
4. Shapiro Francia test
5. Jarque Bera test
6. Kolmogorov Smirnov test
7. Lilliefors test
8. Pearson χ^2 test

Each of which is special in it's own way. Few of them depends on mean and others on different other measures. Cramer, Lilliefors, Cramer etc. are basically variance based tests. There is another test which is not listed in this are *bartlett's test* which tests a particular aspect called *spherecity* and it depends on variance. For this section let us understand two tests namely *Kolmogorov Smirnov Test* and *Shapiro Wilk Test*. These two tests are highly robust and also simple to understand.

Now let us gather evidence in support of earlier understanding related to *age*.

```
> shapiro.test(markdf$age)

Shapiro-Wilk normality test

data:  markdf$age
W = 0.87647, p-value = 0.002341

> ks.test(markdf$age, "pnorm")

One-sample Kolmogorov-Smirnov test

data:  markdf$age
D = 1, p-value < 2.2e-16
alternative hypothesis: two-sided
```

Both tests confirms that the *age* is not normally distributed. We have sufficient evidence to show in support of H_a.

3.6.2 Non-parametric

Nonparametric statistics is the branch of statistics that is not based solely on parametrized families of probability distributions (common examples of parameters are the mean and variance). Nonparametric statistics is based on either being distribution-free or having a specified distribution but with the distribution's parameters unspecified. Nonparametric statistics includes both descriptive statistics and statistical inference.

Non-parametric (or distribution-free) inferential statistical methods are mathematical procedures for statistical hypothesis testing which, unlike parametric statistics, make no assumptions about the probability distributions of the variables being assessed.

Sign test

A Sign Test is used to decide whether a binomial distribution has the equal chance of success and failure. Performs an exact test of a simple null hypothesis about the probability of success in a Bernoulli experiment. The statistical test used for Sign Test is *binomial test*. By definition, the binomial test is an exact test of the statistical significance of deviations from a theoretically expected distribution of observations into two categories. One common use of the binomial test is in the case where the null hypothesis is that two categories are equally likely to occur.

We have different proportions for *gender* in our study data. Suppose if we suspect gender bias that can be tested using `binom.test()` in R. There is also another test known as `prop.test` if there are more number of proportions in the test data. Know more about this test by executing `help("binom.test")` in R Console.

In out data we have one dichotomous variable i.e. *gender*. We might be able to use *binomial test* to know if gender levels are equally likely or not.

```
> length(which(markdf$gender == "male"))
[1] 13
> length(which(markdf$gender == "female"))
[1] 17

> > 17-13/30
[1] 16.56667

> nm <- length(which(markdf$gender == "male"))
> binom.test(nm, 30)

Exact binomial test

data:  nm and 30
number of successes = 13, number of trials = 30, p-value = 0.5847
alternative hypothesis: true probability of success is not equal to 0.5
95 percent confidence interval:
 0.2546075 0.6257265
sample estimates:
probability of success
             0.4333333

> nm <- length(which(markdf$gender == "female"))
> binom.test(nm, 30)

Exact binomial test

data:  nm and 30
number of successes = 17, number of trials = 30, p-value = 0.5847
alternative hypothesis: true probability of success is not equal to 0.5
95 percent confidence interval:
 0.3742735 0.7453925
sample estimates:
probability of success
             0.5666667
```

There is almost 16% of differece between gender levels but that difference is not significant in the data. Same way we can also make test for polytomous data. Such as *employment, marital status, satisfaction* etc. but we need to use *prop.test()* in stead of `binom.test`.

Suppose if we wish to find out level wise differences for employment status we might be able to do as shown below:

```
> nm1 <- length(which(markdf$employment == "top"))
> nm2 <- length(which(markdf$employment == "middle"))
> nm3 <- length(which(markdf$employment == "low"))

> nm1; nm2; nm3
[1] 4
[1] 13
[1] 13

> pttop <- prop.test(nm1, 30)
> ptmid <- prop.test(nm2, 30)
> ptlow <- prop.test(nm3, 30)

> pttop

1-sample proportions test with continuity correction

data:  nm1 out of 30, null probability 0.5
X-squared = 14.7, df = 1, p-value = 0.000126
alternative hypothesis: true p is not equal to 0.5
95 percent confidence interval:
 0.04359708 0.31642383
sample estimates:
        p
0.1333333

> 4/30
[1] 0.1333333

> pttop$p.value; ptmid$p.value; ptlow$p.value
[1] 0.0001260465
[1] 0.5838824
[1] 0.5838824
```

The very first level i.e. *top* level employees are significant in the data. This means the proportion for *top* level employees is statistically significant given 30 employees in the sample. This seems to be true if we notice frequencies for employement levels.

Chapter 4

Bivariate Analysis

The very first technique in bivariate analysis is cross tabulation or contingency tables. Please refer to the topic *two way tables* in Chapter 4. We will try to understand bivariate normality, correlation and regression in this Chapter.

We have quite a few options to do bivariate normality tests. Few of them can be `t test, ks test, wilcoxon test`. T test is useful in testing if the variables under study has come from same population or not. This type of test is known as "test of independence". The other utility is to test if two samples are paired or not. We will use a finance related data set for this section. We will try to analyze capital structure of PSUs from here. We will use two datasets related to Maharatna sector for two PSUs namely *BHEL* and *CIL*. Each of these datasets has 12 columns and 12 rows. Rows prepresents years starting from 2005 to 2019. Columns represents measures and following are measures.

1. Independent variables
    - Total debt
    - Total assets
    - Financial leverage (D/E)
2. Dependent variables
    - Net profit margin (NPM)
    - Return on assets (RoA)
    - Return on equity (RoE)
3. Control variables
    - Size
    - Age

- Growth
- Inflation rate

First we need to import these two data sets. I shall do that using `read.csv()` function.

```
> bhel <- read.csv(file.choose())
> col <- read.csv(file.choose())
> names(bhel)
 [1] "Year"                "Total.Debt"          "Total.Assets"        "Financial
 [5] "X"                   "Net.Profit.Margin"   "Return.on.Assets.ROA." "Return.on
 [9] "X.1"                 "Size"                "Age"                 "Growth"
[13] "Inflation.Rate"
> names(col)
 [1] "Year"                "Total.Debt"          "Total.Assets"        "Financial
 [5] "X"                   "Net.Profit.Margin"   "Return.on.Assets.ROA." "Return.on
 [9] "X.1"                 "Size"                "Age"                 "Growth"
[13] "Inflation.Rate"
```

We have two unwanted data variables in data viz. *X* and *X.1*. These are blank columns in those CSV files. We now have to get rid of those blank columns.

```
bhel <- bhel[, !names(bhel) %in% c("X", "X.1")]
col <- col[, !names(col) %in% c("X", "X.1")]
> names(bhel)
 [1] "Year"                "Total.Debt"          "Total.Assets"        "Financial
 [5] "Net.Profit.Margin"   "Return.on.Assets.ROA." "Return.on.Equity.ROE." "Size"
 [9] "Age"                 "Growth"              "Inflation.Rate"
> names(col)
 [1] "Year"                "Total.Debt"          "Total.Assets"        "Financial
 [5] "Net.Profit.Margin"   "Return.on.Assets.ROA." "Return.on.Equity.ROE." "Size"
 [9] "Age"
```

Now coming back to *t test*; Suppose if we wish to test if financial leverage of BHEL is independent that of COL. The chuck is as follows:

```
> t.test(bhel$Financial.Levergae, col$Financial.Levergae)

Welch Two Sample t-test

data:  bhel$Financial.Levergae and col$Financial.Levergae
t = 10.518, df = 25.276, p-value = 1.011e-10
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 0.2062637 0.3066361
sample estimates:
mean of x mean of y
0.5897662 0.3333162
```

It is clear that they are independent so we may not be able to compare financial leverage of these two firms. The difference between means is highly significant. In the sample output there are means and the standardized difference is nothing but *t value*.

4.1 Bivariate normality

Suppose if we wish to assume that assets and debt are bivariate normal. We might be able to check as shown in the below chunk.

```
> ks.test(bhel$Total.Debt, bhel$Total.Assets)

Two-sample Kolmogorov-Smirnov test

data:  bhel$Total.Debt and bhel$Total.Assets
D = 0.66667, p-value = 0.001837
alternative hypothesis: two-sided
```

These variables does not seems to follow normality. It is better to check through visuals.

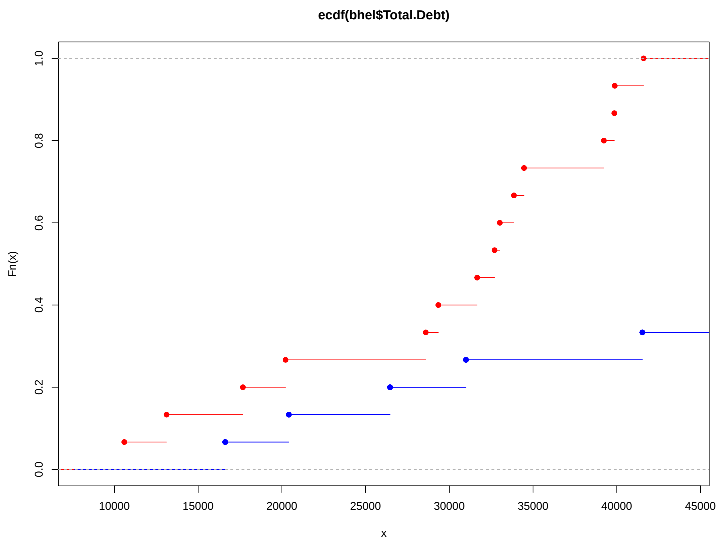


Figure 4.1: Bivariate normality plot for *assets* and *debt* for BHEL

Figure 4.1 shows combined plot for *dots* and *steps*. Though there is relationship but that doesn't seems to be sufficient for comparision.

4.2 Correlation

In the broadest sense correlation is any statistical association, though it commonly refers to the degree to which a pair of variables are linearly related. The most familiar measure of dependence between two quantities is the *Pearson product moment correlation* coefficient, or *Pearson's correlation coefficient*, commonly called simply "the correlation coefficient". It is obtained by dividing the covariance of the two variables by the product of their standard deviations. Karl Pearson developed the coefficient from a similar but slightly different idea by *Francis Galton*.

The correlation coefficient is +1 in the case of a perfect direct (increasing) linear relationship (correlation), 1 in the case of a perfect decreasing (inverse) linear relationship (anticorrelation), and some value in the open interval $(1, 1)$ in all other cases, indicating the degree of linear dependence between the variables. As it approaches zero there is less of a relationship. The closer the coefficient is to either âĹŠ1 or 1, the stronger the correlation between the variables.

In R there is base function known as `cor()`. This works for all types of correlations namely (1) Karl pearson r, (2) Spearman Rho, (3) Kendal's Tau. Use `help()` to know more about this function. The defualt method is *Pearson's r*, *Rho* is useful to compute correlation for ranked data. *Tau* is used to compute correlations if there are *ties* in ranked data.

Coming to our financial data sets; let us find correlation between *Financial Leverage* of BHEL and CIL.

```
> cor(bhel$Financial.Levergae, col$Financial.Levergae)
[1] 0.8156881
> cor(bhel$Financial.Levergae, col$Financial.Levergae, method = "spearman")
[1] 0.6428571
> cor(bhel$Financial.Levergae, col$Financial.Levergae, method = "kendal")
[1] 0.447619
```

Since, the data is non-categorical and numerical, we need to interpret using *Pearson's r*. The r value is 0.81 so the relationship seems to be positive and also strong. Since, the data is not ranked so the other methods are not valid.

Now coming to correlation value. This value represents only the sample but not the population. Our interpretation is only applicable to the data that we collected to this sample. The question is is this relationship is tru in the population. This mean if we take all Maharathna units is it possible for us to do same interpretation? We need to perform correlation significance test to answer this question.

```
> cor.test(bhel$Financial.Levergae, col$Financial.Levergae)

Pearson's product-moment correlation
```

```
data:  bhel$Financial.Levergae and col$Financial.Levergae
t = 5.0839, df = 13, p-value = 0.0002096
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 0.5212119 0.9365970
sample estimates:
      cor
0.8156881
```

`cor.test()` in R does t test for null hypothesis $\rho = 0$. This means the association/relationship in the population is zero. The t value seems to be statistically significant. This means, we can reject null hypothesis and has evidence in support of alternative hypothesis that the same level of relationship can be found in the population too.

4.3 Simple linear regression

Simple linear regression is a linear regression model with a single explanatory variable. That is, it concerns two-dimensional sample points with one independent variable and one dependent variable (conventionally, the x and y coordinates in a Cartesian coordinate system) and finds a linear function (a non-vertical straight line) that, as accurately as possible, predicts the dependent variable values as a function of the independent variables. The adjective simple refers to the fact that the outcome variable is related to a single predictor.

Consider the model function

$$y = \alpha + \beta x \tag{4.1}$$

which describes a line with slope β and y intercept α. In general such a relationship may not hold exactly for the largely unobserved population of values of the independent and dependent variables; we call the unobserved deviations from the above equation the errors. Suppose we observe n data pairs and call them (x_i, y_i), i = 1, ..., n. We can describe the underlying relationship between y_i and x_i involving this error term ε_i by

$$y_i = \alpha + \beta x_i + \varepsilon_i \tag{4.2}$$

This relationship between the true (but unobserved) underlying parameters α and β and the data points is called a linear regression model.

The goal is to find estimated values $\widehat{\alpha}$ and $\widehat{\beta}$ for the parameters α and β which would provide the *best* fit in some sense for the data points. As mentioned in the introduction, in this article the *best* fit will be understood as in the least squares approach: a line that minimizes the sum of squared residuals $\widehat{\varepsilon}_i$

(differences between actual and predicted values of the dependent variable y), each of which is given by, for any candidate parameter values α and β,

$$\widehat{\varepsilon}_i = y_i - \alpha - \beta x_i \tag{4.3}$$

There are few assumptions which need to be satisfied before performing *simple linear regression*. As a matter of caution we first need to do few tests for following assumptions.

1. Normality of residuals
2. Linearity of the data
3. Homogeneity of residual variance

These assumptions can be checked either before or after performing regression analysis. Usually, in R we need to obtain the model to check these assumptions.

In BHEL and COL datasets we have few dependent and independent variables. Let us first make assumption. The nulll hypothesis is that (H_0) *Net Profit*depends on *Financial Leverage* for BHEL.

```
> bhelregfit <- lm(bhel$Return.on.Assets.ROA. ~ bhel$Return.on.Equity.ROE., data = bhel

> summary(bhelregfit)

Call:
lm(formula = bhel$Return.on.Assets.ROA. ~ bhel$Return.on.Equity.ROE.,
    data = bhel)

Residuals:
       Min         1Q     Median         3Q        Max
-0.0102200 -0.0043886 -0.0000612  0.0020013  0.0167574

Coefficients:
                           Estimate Std. Error t value Pr(>|t|)
(Intercept)                0.004242   0.003267   1.298    0.217
bhel$Return.on.Equity.ROE. 0.336404   0.016544  20.334 3.08e-11 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.007223 on 13 degrees of freedom
Multiple R-squared:  0.9695,Adjusted R-squared:  0.9672
F-statistic: 413.5 on 1 and 13 DF,  p-value: 3.082e-11
```

The object `bhelregfit` represents the regression model. This is an object which has attributes associated with it. Suppose if we wish to see *fittend values*, we just have to perform *tab search*, this means write the name of the object in R

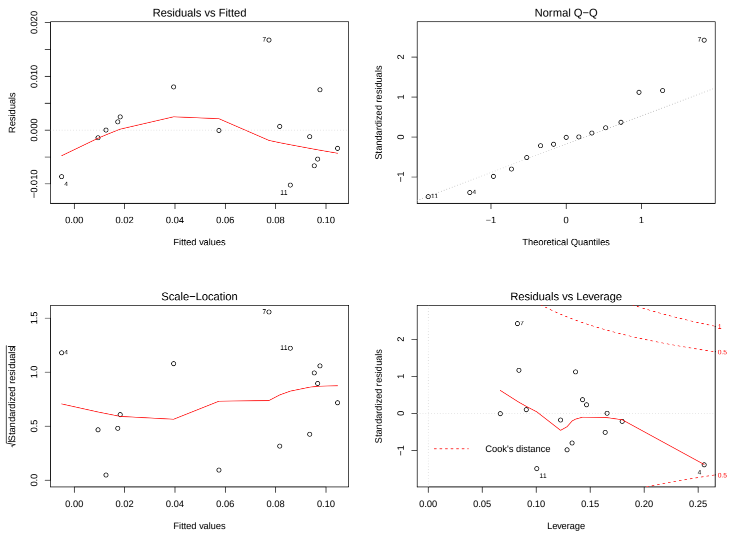


Figure 4.2: Simple linear regression for BHEL

Console followed by *dollar* sign ($) and then press tab to see what attributes are associated with this object. The *fitted values* can be retrieved as shown in the below chunk.

```
bhelregfit$fitted.values
bhelregfit$residuals
```

`bhelregfit$residuals` has residuals in it. These two terms are very important for regrssion diagnostics. Now coming to assumptions; we need to use the model object to make few plots these plots helps us to verify the regression assumptions.

```
par(mfrow=c(2, 2))
plot(bhelregfit)
```

The plot *Residula vs Fitted* must not have any discting patter and points should hang along horizontal line. This type of visual satisfies assumptions related to linearity. In the sample data most of the points are above and below this line and there is a little *fan effect*. This type of effect shows heteroskedasticity. Heteroskedasticity refers unreasonable dispersion in the data.

The second one is *Normal QQ* plot. The points in this plot should wander around the diagonal line. The proximity of points to diagonal line determines the extent to which this data is bivariate normal.

The third plot is *Scale Location* plot. The points in this plot should be distributed roughly equal above and blow the horizontal line. Otherwise, it is subjected to heteroscedasticity.

The last plot is *Residual vs Leverage*. This plot is rather more crucial for regression analysis. This plot helps us in identifying *outliers*. Outlier is a point that is extreme in outcome variable. The presence of outliers may affect the interpretation of the model, because it increases the RSE. RSE is known as *Residual Standard Error* which helps in evaluating fitness of the model to data. Outliers can be identified by examining the standardized residual (or studentized residual), which is the residual divided by its estimated standard error. Standardized residuals can be interpreted as the number of standard errors away from the regression line. Observations whose standardized residuals are greater than 3 in absolute value are possible outliers. A data point has high leverage, if it has extreme predictor x values. This can be detected by examining the leverage statistic or the *hat value*. A value of this statistic above $2(p+1)/n$ indicates an observation with high leverage; where, p is the number of predictors and n is the number of observations.

In our analysis points 4, 7 and 11 are outlier as well as influential. What are these points.

```
> bhel[c(4, 7, 11),]
   Year Total.Debt Total.Assets Financial.Levergae Net.Profit.Margin Return.on.Assets.
4  2016    33859.2      66912.5          0.5060220           -0.0356                -
0.0137
7  2013    39854.2      70298.3          0.5669298            0.1348               0.
11 2009    28591.9      41530.7          0.6884522            0.1197               0.
   Return.on.Equity.ROE.     Size Age  Growth Inflation.Rate
4                -0.0276 4.825507  52 #DIV/0!            4.9
7                 0.2173 4.846945  49 #DIV/0!           10.9
11                0.2425 4.618369  45 #DIV/0!           10.9
```

The data related to years 2009, 2013 and 2016 seems to be outliers. Something, went wrong in these years with respect to study variables. The financial leverage of the unit must have been disturbed in these years.

Going back to summary of regression fit. There are very few measures that needs interpretation. From the fit it is clear that both intercept and slope are statistically statistically significant. RSE is close to zero and it should be. Both *Multiple R Squared* and *Adjusted R Squared* are 0.7052. F statistic is statistically significant, this means there exist significant differences between variances of dependent and independent variables.

Chapter 5

Multivariate Analysis

Multivariate analysis (MVA) is based on the statistical principle of multivariate statistics, which involves observation and analysis of more than one statistical outcome variable at a time. Typically, MVA is used to address the situations where multiple measurements are made on each experimental unit and the relations among these measurements and their structures are important.

5.1 Multivariate correlation

The coefficient of multivariate correlation is a measure of how well a given variable can be predicted using a linear function of a set of other variables. It is the correlation between the variable's values and the best predictions that can be computed linearly from the predictive variables.

The coefficient of multivariate correlation takes values between 0 and 1; a higher value indicates a better predictability of the dependent variable from the independent variables, with a value of 1 indicating that the predictions are exactly correct and a value of 0 indicating that no linear combination of the independent variables is a better predictor than is the fixed mean of the dependent variable.

Performing multiple correlation in R is very simple using the same function which we used earlier to perform correlation. Suppose if we wish to perform multivariate correlation for BHEL.

```
> cor(bhel)
                      Total.Debt Total.Assets Financial.Levergae Net.Profit.Margin
Total.Debt             1.00000000   0.93647307         -0.3704883        -0.1445202
Total.Assets           0.93647307   1.00000000         -0.6714818        -0.4383073
Financial.Levergae    -0.37048833  -0.67148177          1.0000000         0.8521805
Net.Profit.Margin     -0.14452020  -0.43830726          0.8521805         1.0000000
Return.on.Assets.ROA. -0.08847959  -0.39671605          0.8571531         0.9874792
Return.on.Equity.ROE. -0.15364726  -0.47258245          0.9226214         0.9684636
Size                   0.94813880   0.98668678         -0.6090168        -0.4003935
Age                    0.66587081   0.85954525         -0.8753358        -0.7317318
Inflation.Rate         0.37616318   0.08618236          0.5763551         0.6384682
                      Return.on.Assets.ROA. Return.on.Equity.ROE.      Size       Age
Total.Debt                      -0.08847959            -0.1536473 0.9481388 0.6658708
Total.Assets                    -0.39671605            -0.4725824 0.9866868 0.8595453
```

```
Financial.Levergae             0.85715307           0.9226214 -0.6090168 -0.8753358
Net.Profit.Margin              0.98747920           0.9684636 -0.4003935 -0.7317318
Return.on.Assets.ROA.          1.00000000           0.9846404 -0.3546275 -0.7144300
Return.on.Equity.ROE.          0.98464045           1.0000000 -0.4130344 -0.7600879
Size                          -0.35462751          -0.4130344  1.0000000  0.8516004
Age                           -0.71443002          -0.7600879  0.8516004  1.0000000
Inflation.Rate                 0.67292428           0.6691278  0.1666337 -0.2180469
                      Inflation.Rate
Total.Debt                0.37616318
Total.Assets              0.08618236
Financial.Levergae        0.57635505
Net.Profit.Margin         0.63846818
Return.on.Assets.ROA.     0.67292428
Return.on.Equity.ROE.     0.66912782
Size                      0.16663367
Age                      -0.21804690
Inflation.Rate            1.00000000
```

There are strong associations in the output. The correlation between *Total Debts* observed to have strong and positive relationship with *Total Assets, Size* and *Age*. Interestingly *Assets* and *Debts* negatively associated with *Financial Leverage* and *Net Profit*. Interestingly, *Net Profit Margin* strongly correlated with *Financial Leverage* and *Returns*. *Age* of the unit negatively associated with *Leverage* and *Profit*. Interestingly *Inflation Rate* is positively correlated with most of the variables except *Age* of the unit.

5.2 Multiple regression

Multiple linear regression attempts to model the relationship between two or more explanatory variables and a response variable by fitting a linear equation to observed data. Every value of the independent variable x is associated with a value of the dependent variable y. The population regression line for p explanatory variables $x1, x2, ..., xp$ is defined to be $y = 0 + 1x1 + 2x2 + ... + pxp$. This line describes how the mean response y changes with the explanatory variables. The observed values for y vary about their means y and are assumed to have the same standard deviation. The fitted values $b0, b1, ..., bp$ estimate the parameters $0, 1, ..., p$ of the population regression line.

Suppose if we make assumption that *Net Profits* are influenced by *Leverage* and *Leverage* is being influenced by *Inflatin Rate*.

```
> bhelmrout <- lm(bhel$Net.Profit.Margin ~ bhel$Financial.Levergae*bhel$Inflation.Rate
> summary(bhelmrout)

Call:
lm(formula = bhel$Net.Profit.Margin ~ bhel$Financial.Levergae *
    bhel$Inflation.Rate, data = bhel)

Residuals:
      Min        1Q    Median        3Q       Max
-0.067071 -0.012431  0.005056  0.015492  0.035938

Coefficients:
```

```
                                             Estimate Std. Error t value Pr(>|t|)
(Intercept)                                  -0.50579    0.16580  -
3.051  0.01104 *
bhel$Financial.Levergae                       0.95750    0.28940   3.309  0.00697 **
bhel$Inflation.Rate                           0.04268    0.02452   1.741  0.10961
bhel$Financial.Levergae:bhel$Inflation.Rate -0.06307    0.04025  -
1.567  0.14545
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.02912 on 11 degrees of freedom
Multiple R-squared:  0.8027,Adjusted R-squared:  0.7489
F-statistic: 14.92 on 3 and 11 DF,  p-value: 0.0003417
```

Both *Intercept* and *Slope* for *Financial leverage* are statistically significant. *Inflation Rate* is not statistically significant. The interaction between *Financial Leverage* and *Inflation rate* is also not significant in the model.

5.2.1 Formula notations for multiple regression

Doing multiple regression in R is highly intuitive process. We need to understand the way *formula* is being used in the `lm()` function. There are different ways to perform multiple regression in R using variety of symbols in model definition.

Symbol	Example	Meaning
+	+X	include this variable
-	-X	delete this variable
:	X : Z	include the interaction between these variables
$*$	X $*$ Y	include these variables and the interactions between them
|	X|Z	conditioning: include x given z
ˆ |	$(X + Z + W)^3$	include these variables and all interactions up to three way
I	I(X * Z)	as is: include a new variable consisting of these variables multiplied
1	X-1	intercept: delete the intercept (regress through the origin)

5.3 Canonical correlation

canonical-correlation analysis (CCA), also called canonical variates analysis, is a way of inferring information from cross covariance matrices. If we have two vectors X = (X1, ..., Xn) and Y = (Y1, ..., Ym) of random variables, and there are correlations among the variables, then canonical-correlation analysis will find *linear combinations* of X and Y which have maximum correlation with each other. Canonical correlation analysis, is a general procedure for investigating the relationships between two sets of variables. The method was first introduced by Harold Hotelling in 1936, although in the context of angles between flats the mathematical concept was published by Jordan in 1875.

Performing Canonical Correlation Analysis in R is very simple and we can do that using a special function `cancor()`. This function is available from base package *stats*.

We will use a special dataset related to HR domain. The dataset has 9 columns and 30 rows. Rows represents individuals (employees). Columns represents three factors namely (1) training, (2) evaluation, (3) performance. Each of these factors were measured using three variables. Let us try to see if *evaluation* as a factor can be correlated with *performance*.

```
> cancor(hrdf[, 4:6], hrdf[, 7:9])
$cor
[1] 0.91766337 0.33775257 0.01134242

$xcoef
             [,1]        [,2]        [,3]
perf1  0.099495536 -0.17230672 -0.08335387
perf2 -0.004155827  0.01357719  0.17361185
perf3  0.074894036  0.17810525 -0.03487290

$ycoef
             [,1]        [,2]        [,3]
impact1 0.03533550 -0.07593333 -0.18601247
impact2 0.12219985  0.15077381  0.04980896
impact3 0.01824095 -0.15374649  0.08074854

$xcenter
   perf1    perf2    perf3
2.800000 2.600000 3.233333

$ycenter
 impact1  impact2  impact3
3.300000 3.133333 2.300000
```

From the result it is clear that the association at very first instance is strong and the association gradually reduced to third training session. Ther effect of training on employee performance at the end is not good.

5.4 MANOVA

Multivariate analysis of variance (MANOVA) is a procedure for comparing multivariate sample means. As a multivariate procedure, it is used when there are two or more dependent variables, and is typically followed by significance tests involving individual dependent variables separately.

1. Do changes in the independent variable(s) have significant effects on the dependent variables?
2. What are the relationships among the dependent variables?
3. What are the relationships among the independent variables?

In our BHEL dataset we have few dependent and independent variables. Suppose if we make an assumption that *Net Profit* together with other variables such as *RoA, RoE* depends on *Financial Leverage.* We might be able to test such assumption using MANOVA.

```
> finmanova
Call:
   manova(cbind(bhel$Net.Profit.Margin, bhel$Return.on.Assets.ROA.,
    bhel$Return.on.Equity.ROE.) ~ bhel$Financial.Levergae, data = bhel)

Terms:
                bhel$Financial.Levergae  Residuals
resp 1                       0.03434617 0.01294882
resp 2                       0.01634522 0.00590191
resp 3                        0.1622386  0.0283545
Deg. of Freedom                       1         13

Residual standard errors: 0.03156047 0.02130711 0.04670241
Estimated effects may be unbalanced
```

The above table shows response (dependent variable) wise estimates and their residuals. This table may not help us much but we can make sense from below tables.

```
\begin{verbatim}
> summary(finmanova)
                        Df  Pillai approx F num Df den Df    Pr(>F)
bhel$Financial.Levergae  1 0.94764    66.36      3     11 2.485e-07 ***
Residuals               13
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Above table shows the ANOVA like tables for MANOVA, because MANOVA is only an extension to ANOVA. At least in R. From the P Value it is clear that the overall model is statistically significant. This means *Net Profit, RoA* and *RoE* are significantly explained by *Financial Leverage.*

```
> summary.aov(finmanova)
```

```
 Response 1 :
                        Df   Sum Sq  Mean Sq F value    Pr(>F)
bhel$Financial.Levergae  1 0.034346 0.034346  34.482 5.482e-05 ***
Residuals               13 0.012949 0.000996
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

 Response 2 :
                        Df    Sum Sq  Mean Sq F value    Pr(>F)
bhel$Financial.Levergae  1 0.0163452 0.016345  36.003 4.444e-05 ***
Residuals               13 0.0059019 0.000454
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

 Response 3 :
                        Df   Sum Sq  Mean Sq F value    Pr(>F)
bhel$Financial.Levergae  1 0.162239 0.162239  74.383 9.728e-07 ***
Residuals               13 0.028354 0.002181
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Above table shows response wise relationships. All relationships are statistically significant.

5.5 Factor analysis

Factor analysis is a statistical method used to describe variability among observed, correlated variables in terms of a potentially lower number of unobserved variables called factors. For example, it is possible that variations in six observed variables mainly reflect the variations in two unobserved (underlying) variables. Factor analysis searches for such joint variations in response to unobserved latent variables. The observed variables are modelled as linear combinations of the potential factors, plus *error* terms. Factor analysis aims to find independent latent variables.

In matrix terms, we have

$$x - \mu = LF + \varepsilon \tag{5.1}$$

If we have n observations, then we will have the dimensions $x_{p\times n}, L_{p\times k}$, and $F_{k\times n}$. Each column of x and F denotes values for one particular observation, and matrix L does not vary across observations.

Also we will impose the following assumptions on F:

1. F and ε are independent.

2. E(F)=0. (E is Expectation)
3. Cov(F)=I (to make sure that the factors

are uncorrelated).

Any solution of the above set of equations following the constraints for F is defined as the factors, and L as the loading matrix. Suppose $\text{Cov}(x-\mu)=\Sigma$. Then note that from the conditions just imposed on F, we have

$$\text{Cov}(x-\mu)=\text{Cov}(LF+\varepsilon) \tag{5.2}$$

or

$$\Sigma = L\text{Cov}(F)L^T+\text{Cov}(\varepsilon) \tag{5.3}$$

In R doing factor analysis is very easy but using packages. There is one base function called `factanal()`, this is a high level function that does most of the work for us. However, there are couple of things to do before performing factor analysis. Doing statistical analysis is pretty much in practice in social sciences community *willy nilly* of its requirement. The following are few prerequisites to perform factor analysis.

1. Reliability analysis for uniqueness
2. KMO test for partial correlations
3. Bartlett's test of spherecity

5.5.1 Alpha

Suppose that we measure a quantity which is a sum of K components (K-items or testlets): $X=Y_1+Y_2+\cdots+Y_K$ Cronbach's α is defined as

$$\alpha=\frac{K\bar{c}}{(\bar{v}+(K-1)\bar{c})} \tag{5.4}$$

where K number of variables, $\bar{v}$ the average variance of each component (item), and $\bar{c}$ the average of all covariances between the components across the current sample.

We need to make few functions to compute Cronbach alpha. I simply call it as *alpha*.

```
alpha <- function(df){

  varmat <- diag(var(df))
  varmat
  v = mean(varmatvec)

  covarmat <- cov(df)
  covarmat
  covarmat.up <- upper.tri(covarmat, diag = FALSE)
  covarmat.up
  covarmatvec <- covarmat[covarmat.up]
  c = mean(covarmatvec)

  k = length(df)

  out <- (k*c)/(v + k*c)
  return(out)
}
```

Now it is time to test. I am going to use a dataset with a name *customer_data.csv* for factor analysis.

```
> custdf <- read.csv(file.choose())
> alpha(custdf)
[1] 0.585699
```

Oh! Nefarious. The data did not satisfy reliability test.

5.5.2 Bartlett's test

Now let us try *Bartlett's test.* We don't need to worry much there is base function `bartlett.test()` in *stats* package.

```
> bartlett.test(custdf)

Bartlett test of homogeneity of variances

data:  custdf
Bartlett's K-squared = 0.84989, df = 8, p-value = 0.999
```

As I mentioned **bartlett's test** is done to test spherecity, which means that *the data set used is not an identity matrix.*

5.5.3 KMO test

The third requirement is *Kaiser-Meyer-Olkin (KMO) Test.* KMO test in short helps us to test for partial correlations. The methodology for KMO test is as follows:

The Kaiser-Meyer-Olkin (KMO) measure of sample adequacy (MSA) for variable xj is given by the formula

$$KMO = \frac{\sigma_i r^2}{\sigma_i r^2 = \sigma_i u^2} \tag{5.5}$$

where the correlation matrix is r and the partial covariance matrix is u. The overall KMO measure of sample adequacy is given by the above formula taken over all combinations and $i \neq j$. There is no function available for KMO test in base package. So we need to create a user defined functon.

```
kmo.test <- function(df){
  cormat <- cor(custdf)
  cormat.up <- upper.tri(cor(custdf), diag = FALSE)
  r2 <- sum(cormat[cormat.up]^2)

  covmat <- cov(custdf)
  covmat.up <- upper.tri(covmat, diag = FALSE)
  u2 <- sum(covmat[covmat.up]^2)

  out <- (r2/(r2+u2))

  return(out)
}
```

Let us test our function now.

```
> kmo.test(custdf)
[1] 0.1766494
```

The criteria for interpretation is as follows: [16]

KMO	Interpretation
0.9 and above	Marvelous
0.8 to 0.9	Meritorious
0.7 to 0.6	Middling
0.6 to 0.7	Mediocre
0.5 to 0.6	Miserable
Under 0.5	Unacceptable

The KMO value is not close to 1. So it is not possible for us to make factor analysis with this data. We can also make *scree plot* using base `screeplot()` function which is available from *stats* package.

```
screeplot(princomp(custdf, 3), type = "lines", col = "blue")
```

From the diagram it is clear that 3 factor solution might suits to customer satisfaction data.

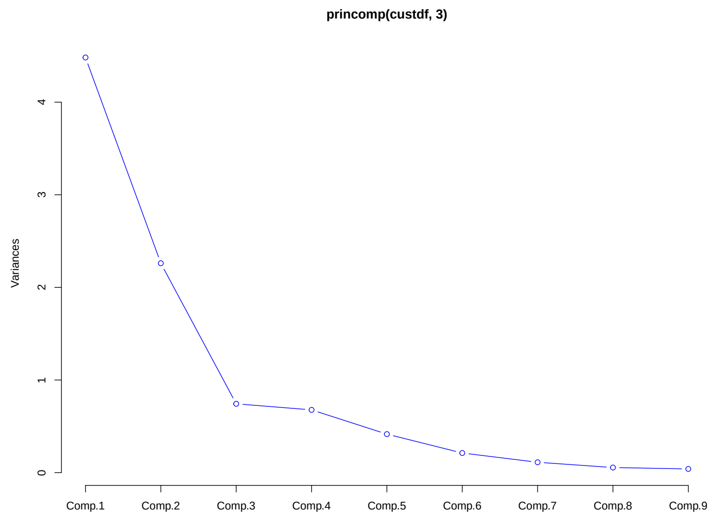


Figure 5.1: Screeplot for *Customer Data*

5.5.4 Factor analysis

Since we finished all prerequisites, we might be able to proceed to factor analysis. As it was mentioned earlier the base function `factanal()` is useful to perform factor analysis.

```
> factanal(custdf, 3)

Call:
factanal(x = custdf, factors = 3)

Uniquenesses:
 per1  per2  per3  att1  att2  att3  sat1  sat2  sat3
0.033 0.232 0.195 0.614 0.224 0.005 0.492 0.005 0.178

Loadings:
     Factor1 Factor2 Factor3
per1  0.946   0.255
per2  0.837   0.162   0.203
per3  0.865   0.131   0.197
att1  0.591  -0.121   0.151
att2          0.867   0.131
att3  0.359   0.452   0.814
sat1  0.414   0.574
sat2          0.982   0.154
```

```
sat3  0.143   0.810   0.383

               Factor1 Factor2 Factor3
SS loadings      3.028   3.027   0.967
Proportion Var   0.336   0.336   0.107
Cumulative Var   0.336   0.673   0.780

Test of the hypothesis that 3 factors are sufficient.
The chi square statistic is 29.84 on 12 degrees of freedom.
The p-value is 0.00295
```

From the output result it is clear that three factor solution is not appropriate to our customer satisfaction data. The chisquare value is statistically significant.

Notes

[15]Gerd Gigerenzer, Stefan Krauss, and Oliver Vitouch wrote a popular article with a name "Th e Null Ritual What You Always Wanted to Know About Signifi cance Testing but Were Afraid to Ask"; this will help demystify few doubts about hypothesis testing. Available at https://library.mpib-berlin.mpg.de/ft/gg/GG_Null_2004.pdf

[16]Charles Z., "Validity of Correlation Matrix and Sample Size", Retrieved from http://www.real-statistics.com/multivariate-statistics/factor-analysis/validity-of-correlation-matrix-and-sample-size/