



R Language - A Language for Data Analytics and Visualisation

Introduction



- ⌘ R is a programming language and software environment for statistical analysis, graphics representation and reporting.
- ⌘ The core of R is an interpreted computer language which allows branching and looping as well as modular programming using functions.
- ⌘ R allows integration with the procedures written in the C, C++, .Net, Python or FORTRAN languages for efficiency.

Introduction



- # The term “R” is used to refer to both the programming language to write scripts and the software (“environment”) that interprets the scripts written in R.
- # It is an alternative to statistical packages like SAS, SPSS, or Stata, which lets you perform a wide variety of data analysis, statistics, and visualization.

Introduction



- ⌘ R is freely available under the GNU (**GNU is Not Unix**) General Public License, and pre-compiled binary versions are provided for various operating systems like Linux, Windows and Mac.
- ⌘ R was initially written by **Ross Ihaka** and **Robert Gentleman** at the Department of Statistics of the University of Auckland in Auckland, New Zealand.
- ⌘ R made its first appearance in 1993.
- ⌘ A large group of individuals has contributed to R by sending code and bug reports.
- ⌘ Since mid-1997 there has been a core group (the "R Core Team") who can modify the R source code archive.

Introduction

- ⌘ R version 1.0.0 was released in the year 2000.
- ⌘ John chambers and his colleagues at Bell laboratories developed S language on which R was implemented.
- ⌘ R is a dialect (a particular form of a language which is peculiar to a specific region or social group)of S language.
- ⌘ As an internal statistical analysis, S was initiated in the year 1976.



Introduction



- # Earlier versions of S language (V 1 and V 2) do not contain functions for statistical modelling.
- # In 1998, for version 3 the syntax of the language was re-written in C and many changes were made.
- # In the year 1998, version 4 of the S language was released.
- # After V4 of S language, came R which is a analogous evolutionary model to the previously used versions of S language.

Evolution of R



Year	Description
1991	R made its first appearance
1993	R was introduced to the public
1995	Ross and Robert convinced by Martin Maechler, use the GNU General Public License to make R free software
1996	Public mailing list – R-help and R-devel is created
1997	R core group is formed which controls the source code of R
2000	R version 1.0.0 is released
2013	R version 3.0.2 is released
2017	Version 3.4.2 is released on 28 th September, 2017
2022	Latest Version 4.2.2 is released on 31 st October, 2022

Features of R



- # R is a programming language and software environment for statistical analysis, graphics representation and reporting.
- # The following are the important features of R:
 - ▣ R is a well-developed, simple and effective programming language which includes conditionals, loops, user defined recursive functions and input and output facilities.
 - ▣ R has an effective data handling and storage facility.
 - ▣ R provides a suite of operators for calculations on arrays, lists, vectors and matrices.
 - ▣ R provides a large, coherent and integrated collection of tools for data analysis.
 - ▣ R provides graphical facilities for data analysis and display either directly at the computer or printing at the papers.

Advantages of R



- ⌘ Free software
- ⌘ Interface with other languages and scripting capabilities
 - ↗ Interfaces virtually with other programming languages like Fortran, C, C++ and Python where old codes can be rewritten and tailored in R.
- ⌘ R visualization capabilities
 - ↗ R environment provides numerous visualization packages such as graphics, lattice and ggplot2.

Advantages of R



⌘ R role in academia

- ◻ R tool was first used by finest data researchers and academicians.
- ◻ It's widely used in academic because of its top analytical capabilities in the area of data mining, extraction and analysis.

⌘ Online help and discussion

- ◻ R provides online help and discussion to communities of developers who contribute to R through their UDF's (User Defined Functions)

Advantages of R



⌘ Number of packages

↗ It has around 10000 packages available from multiple repositories specializing in topics such as data mining, econometrics, bio-informatics and spatial analysis.

Disadvantages of R



- ⌘ The learning curve of R is very steep. It does take a while to get used to the functionalities of R programming in data analysis.
- ⌘ R is not so easy for a beginner and some prerequisites such as some knowledge of programming in a language and domain knowledge of the fields in which data analysis happens at a larger scale are required.

Disadvantages of R



- ⌘ Poor management of large data set. R has a file size limit of 2-4 GB. Objects of R always live in memory, therefore it takes up a large space of hard drive.
- ⌘ Complicated structure of packages in R.
- ⌘ Currently R can not be embedded in a web browser.

How R works



- # R is an interpreted language, not a compiled one.
- # Like most programming languages such as C, Fortran or Pascal the commands need to be executed after building a complete program.
- # Syntax used in R is simple.
 - # Linear regression can be done with the command `lm(y ~ x)` -> fitting a linear model with y as response and x as predictor.

R environment



- ⌘ R environment refers to the prerequisite software and packages required for the proper functioning of R system.
- ⌘ R environment consists of the following components
 - ─ R Studio
 - ─ R software
 - ─ Packages

R environment



❖ R environment provides

- ❖ Storage facility and effective data handling.
- ❖ List of operators for calculations on arrays and matrices.
- ❖ Integrated and coherent collection of intermediate tools for data analysis.
- ❖ Display on both screen and hardcopy as well as graphical facilities for data analysis.
- ❖ Input and output facilities, user-defined recursive functions, loops and conditions.

R Studio



- ⌘ RStudio is the opus of J.J. Allaire, a software engineer and the brain behind coldFusion programming language and web application server.
- ⌘ An Integrated Development Environment (IDE) for R which issues commands interactively.
- ⌘ Can be considered as a front end for R which includes a console, editor and various tools for plotting, history, debugging and workspace management.

Existing IDEs for R

Name	Platforms	Description
ESS	All	ESS (http://ess.r-project.org) is one of the most popular and widely used R interface.
Eclipse	All	Open source StatET plugin available on http://www.walware.de/goto/statet for download. This plugin helps in turning a Java-based multipurpose IDE known as Eclipse, into a full featured R IDE.
SciViews	All	It is an extension for komodo code editor as well as an API for R
JGR	All	It is a Java based editor which, with the help of JRI and rJava packages interacts with R. Furthermore a Deducer package adds a pack of data analysis tools.
Tinn-R	Windows	It is an extension for Tinn editor which allows users to interact with the underlying R process.
Notepad++	Windows	Notepad++ editor can interact with an R process after attaching with NpptoR extension.
RGui	Windows	It is the default interface for R in windows operating system

R Vs. R Studio



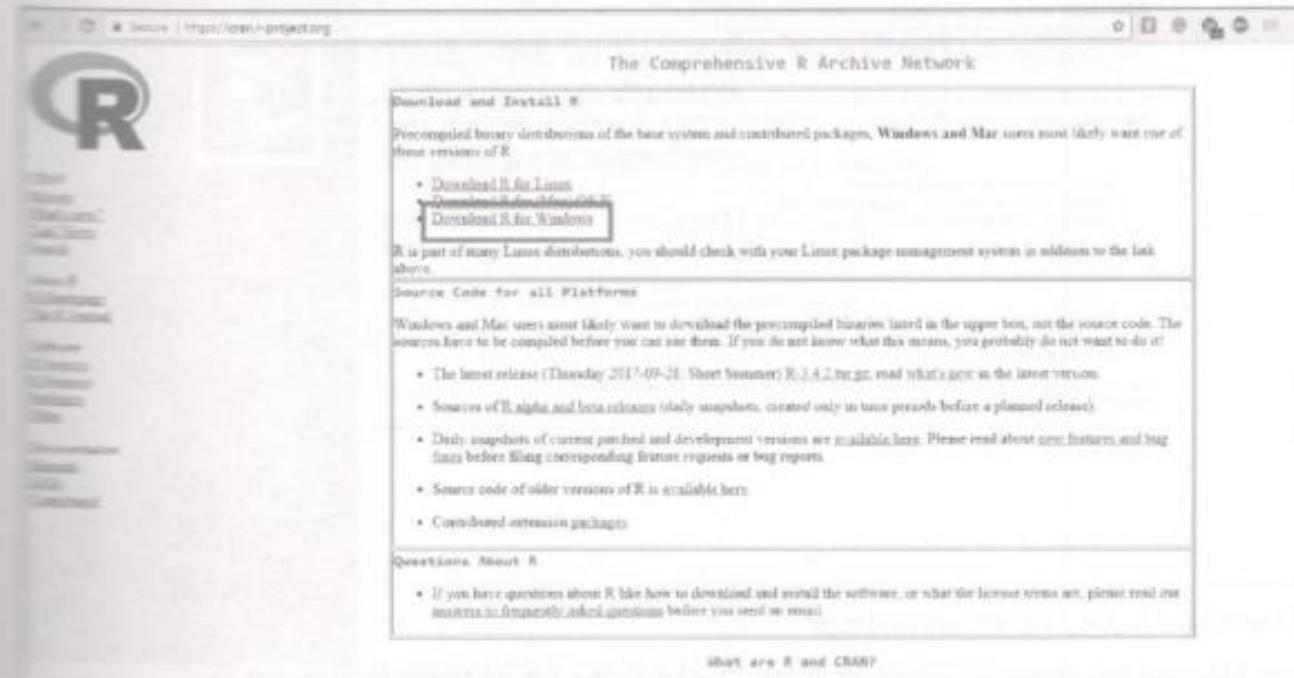
- ⌘ R and RStudio are not two different versions of the same thing.
- ⌘ One can't be substituted for the other. In fact, they work together.
- ⌘ R is a programming language for statistical calculation. And RStudio is an Integrated Development Environment (IDE) that helps you develop programs in R.
- ⌘ We can use R without using RStudio, but we can't use RStudio without using R, so R comes first.

Installing R and Rstudio

To install R, RStudio and R Commander in Windows, follow the given steps:

Installing R

1. Go to <http://cran.us.r-project.org>. Click on Download R for Windows as shown in Figure 2.



Installing R

2 Click on 'base' to proceed in R for Windows page as shown in Figure 3.

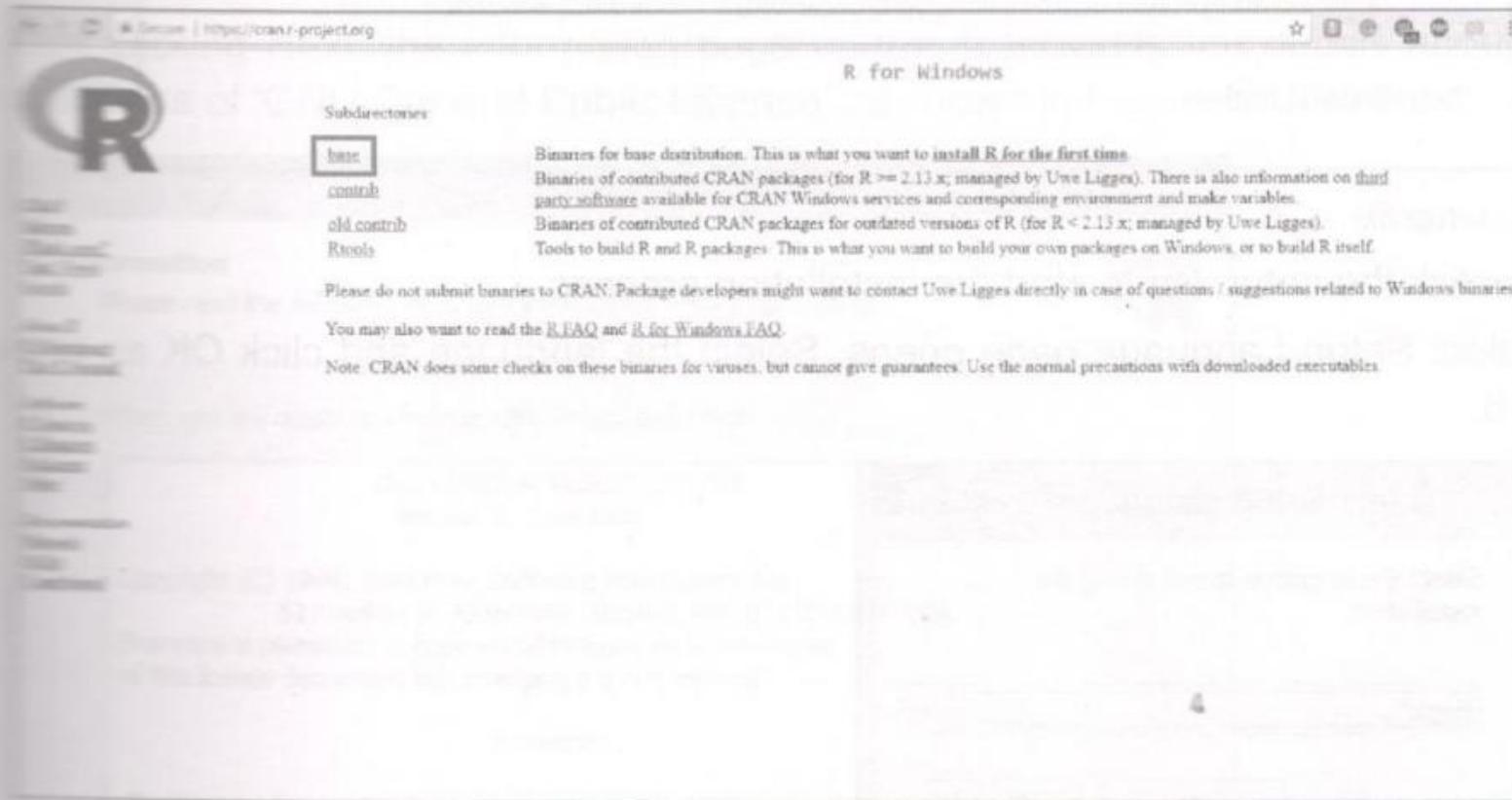


Figure 3: R for Windows page

Installing R

- Click on Download R 3.4.2 for Windows, to start the installation as shown in Figure 4.

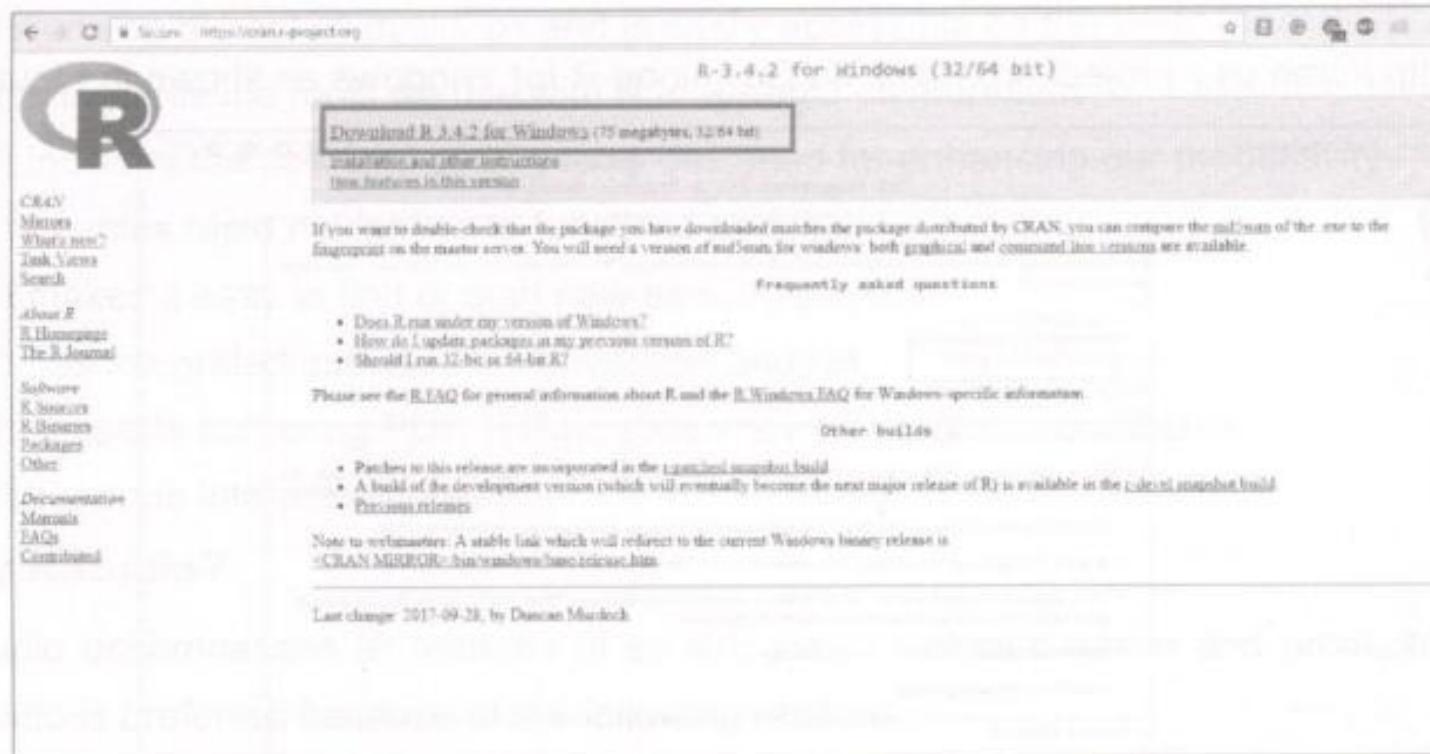


Figure 4: Download R-3.4.2 for Windows page

Installing R



4. The setup file will be downloaded in your root folder as shown in Figure 5.

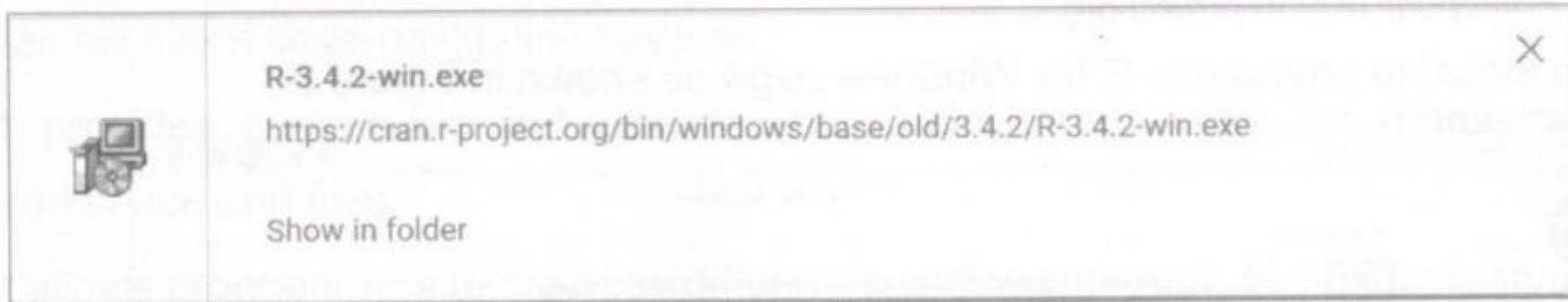


Figure 5: Setup file

Installing R

5. Double click the setup file to start the installation process.
6. The Select Setup Language page opens. Select the language and click OK as shown in Figure 6.

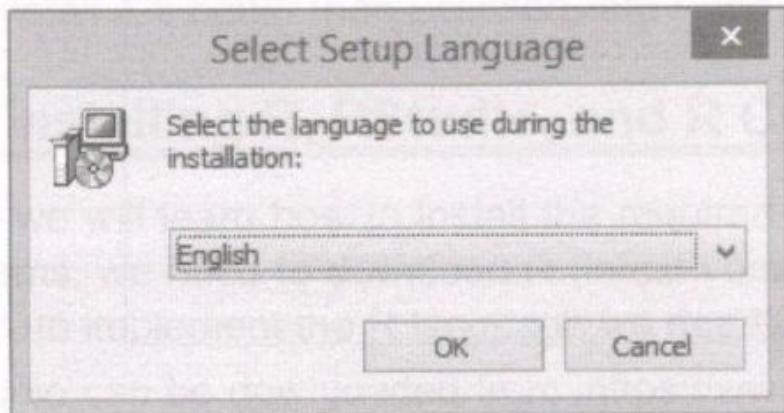


Figure 6: Select Setup Language page

Installing R

7. The Welcome to the R for Windows 3.4.2 Setup Wizard page appears as shown in Figure 7. Then click Next.



Figure 7: Welcome to the R for Windows 3.4.2 Setup Wizard page

Installing R

- After clicking Next, the information page opens. It provides the details about terms and conditions of 'GNU General Public License' as shown in Figure 8. Click Next.

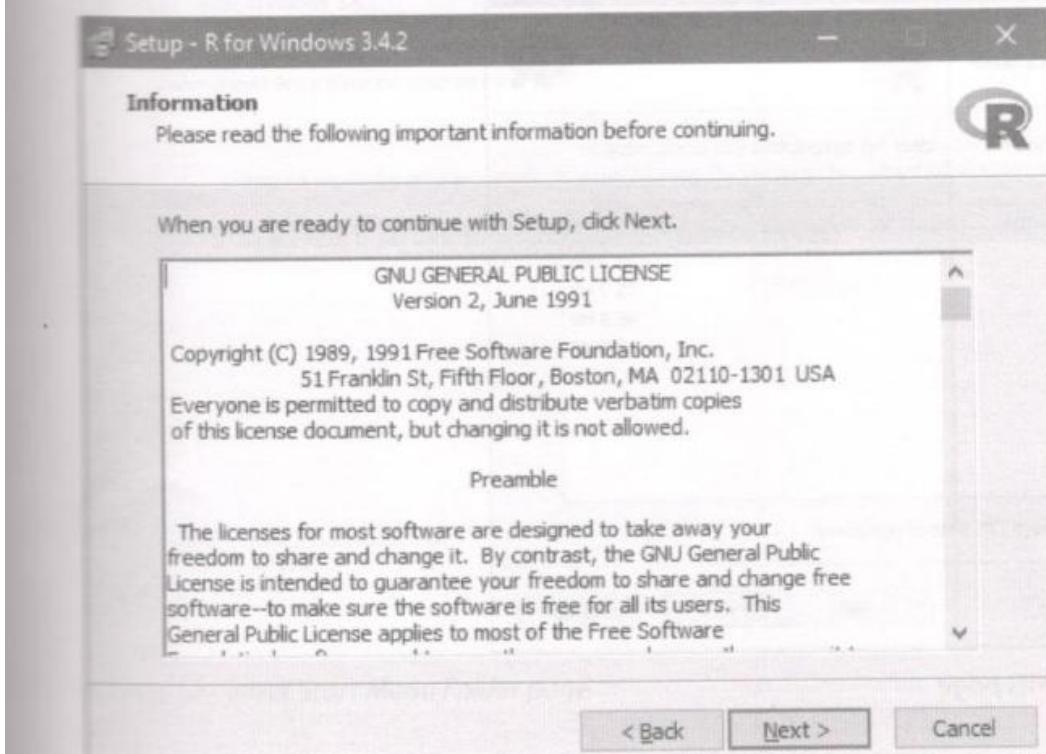


Figure 8: The Information page

Installing R

9. Select Destination Location page opens as shown in Figure 9. Select the destination folder where you want to save R program, then click Next.

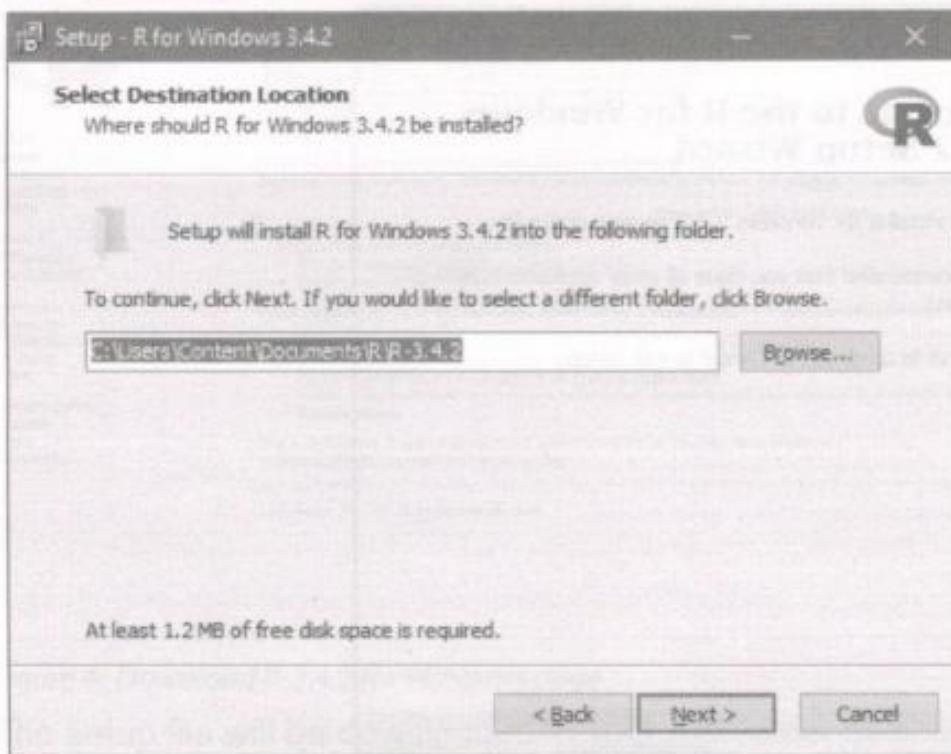


Figure 9: Select Destination Location page

Installing R

- After we have selected the destination folder, Select Components page opens as shown in Figure 10. Check all the boxes to select all the components and click Next.

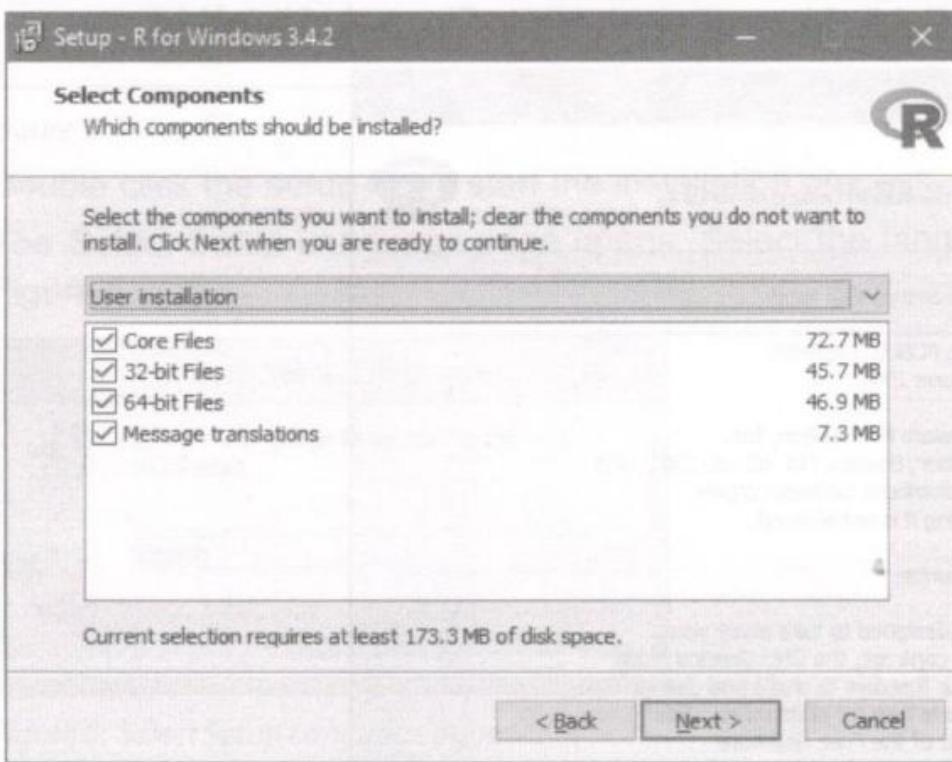


Figure 10: Select Components page

Installing R

- III. Startup options page opens as shown in Figure 11. Click on No (accept defaults) and then click Next.

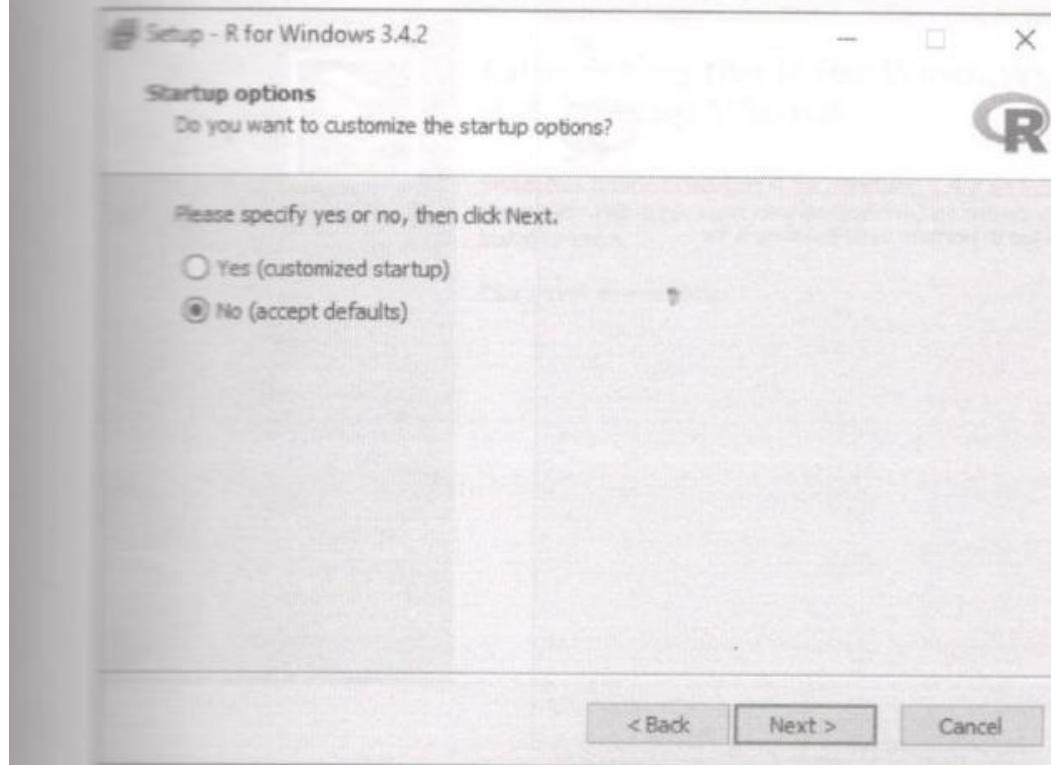


Figure 11: Startup options page

Installing R

- Select Start Menu Folder page opens as shown in Figure 12. Select the folder, where you want to save the shortcut of R. To change the location, you can use the 'Browse' option available at the page and Click Next.

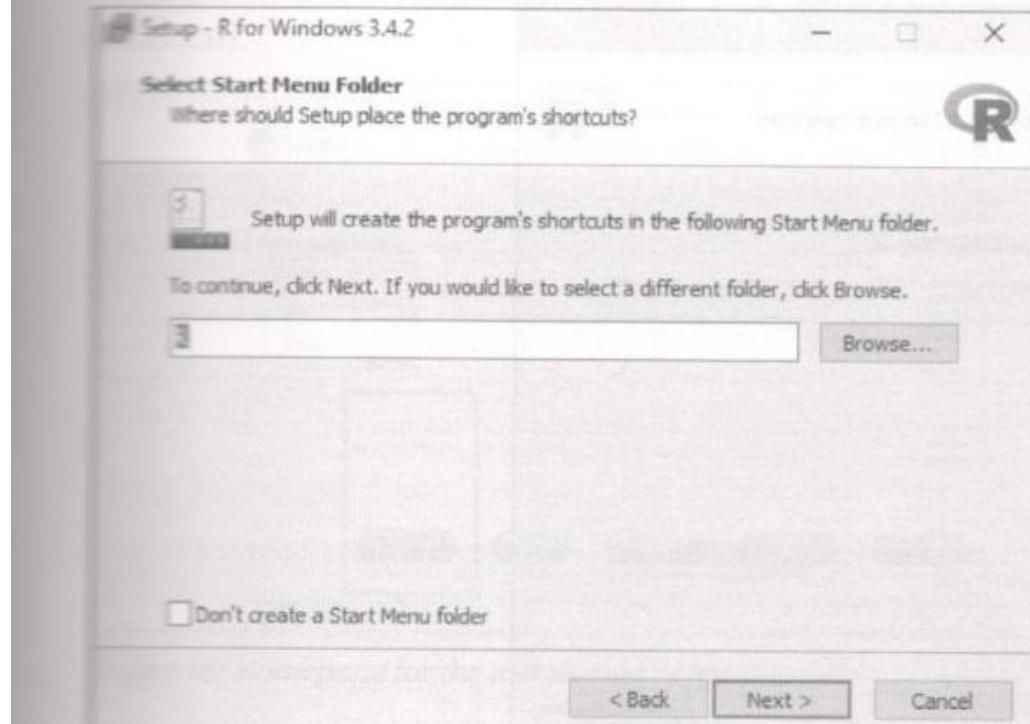


Figure 12: Select Start Menu Folder page

Installing R

13. Select Additional Tasks page opens as shown in Figure 13. Check the options you may want and click Next.

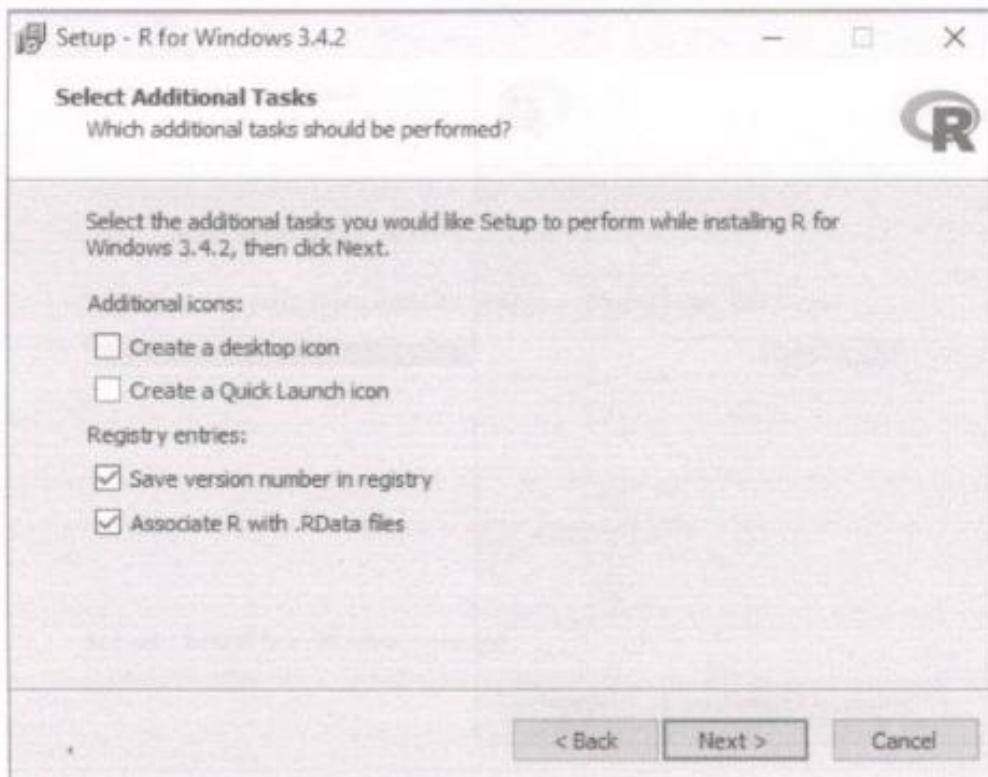


Figure 13: Select Additional Tasks page

Installing R

14. Installing page appears as shown in Figure 14. This is the page on which installation will begin.

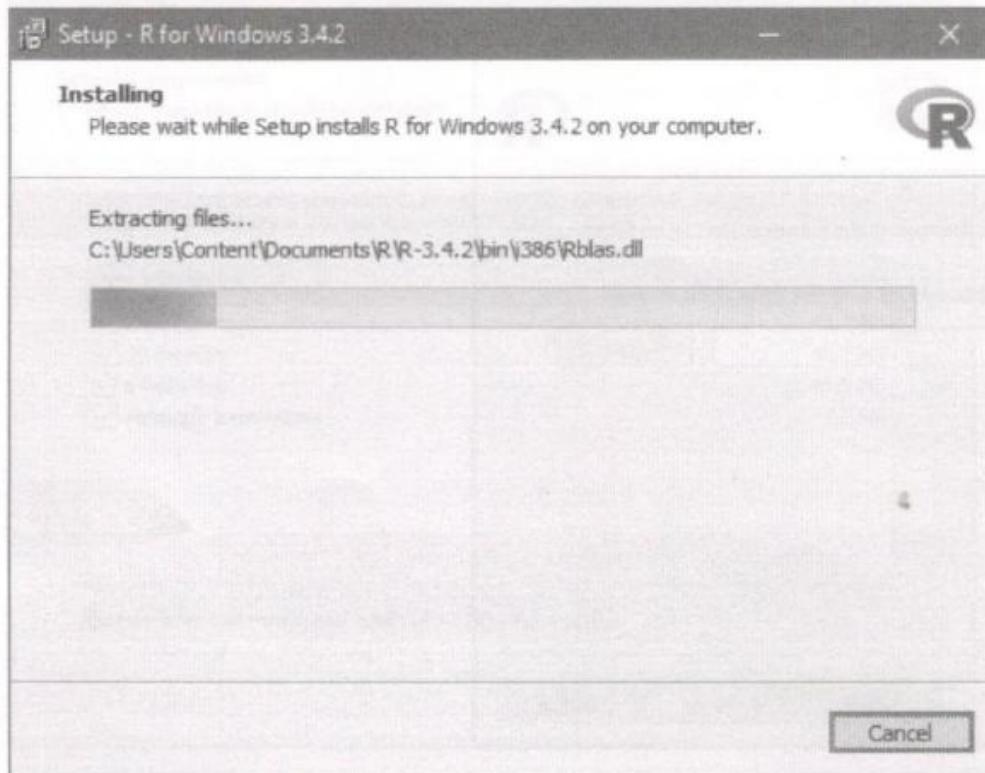


Figure 14: Installing page

Installing R

- Once installation is complete, click Finish to exit the setup page as shown in Figure 15.



Figure 15: Installation was successful

Installing Rstudio

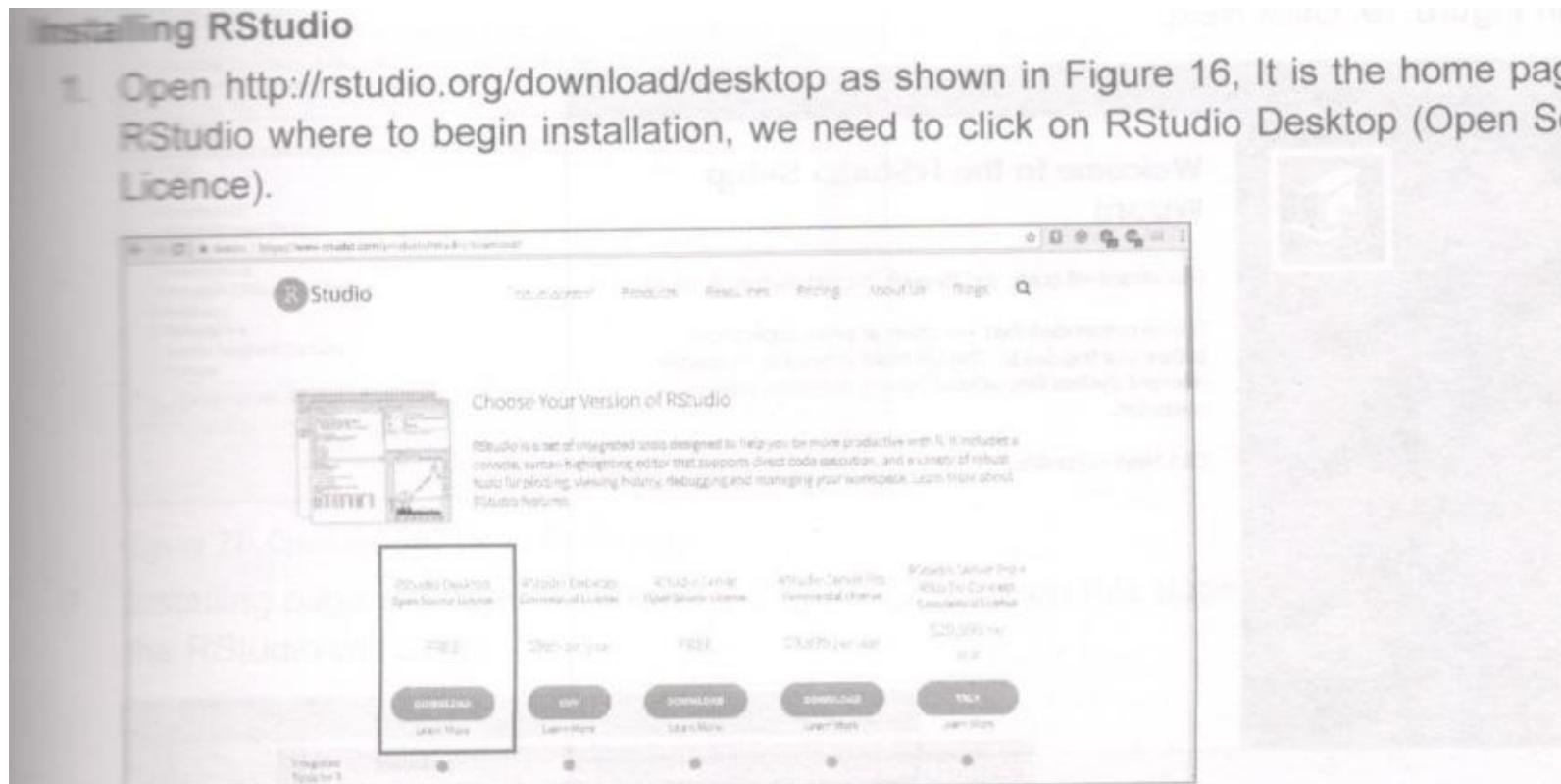


Figure 16: Homepage for the installation of RStudio

Installing Rstudio

2. Download the package depending on your operating system as shown in Figure 17.



Figure 17: RStudio download page

Installing Rstudio



3. A Setup file will be downloaded in your root folder as shown in Figure 18.

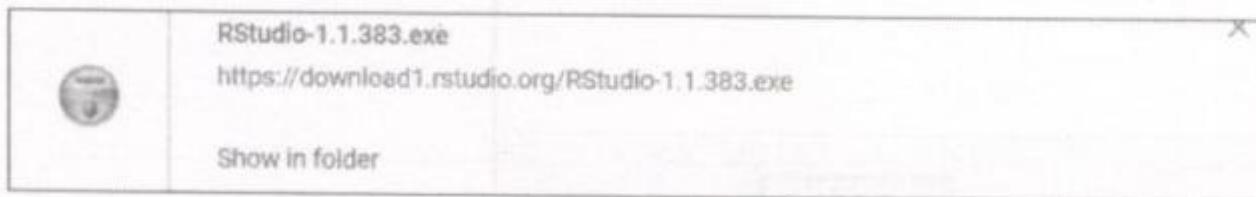


Figure 18: Setup file

Installing Rstudio

- Double click the setup file. Welcome to the RStudio Setup Wizard page will open as shown in Figure 19. Click Next.



Figure 19: Welcome to the RStudio Setup Wizard page

Installing Rstudio

- Choose Install Location page opens as shown in Figure 20. Choose the destination folder where we want to save RStudio and then click Next.

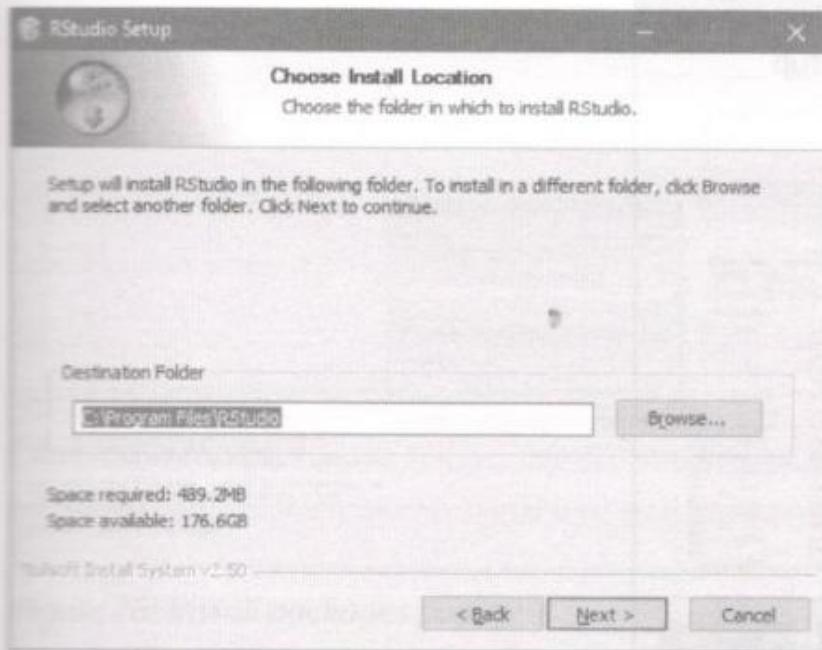


Figure 20: Choose Install Location page

Installing Rstudio

6. Choose Start Menu Folder opens as shown in Figure 21. Click Install to proceed.

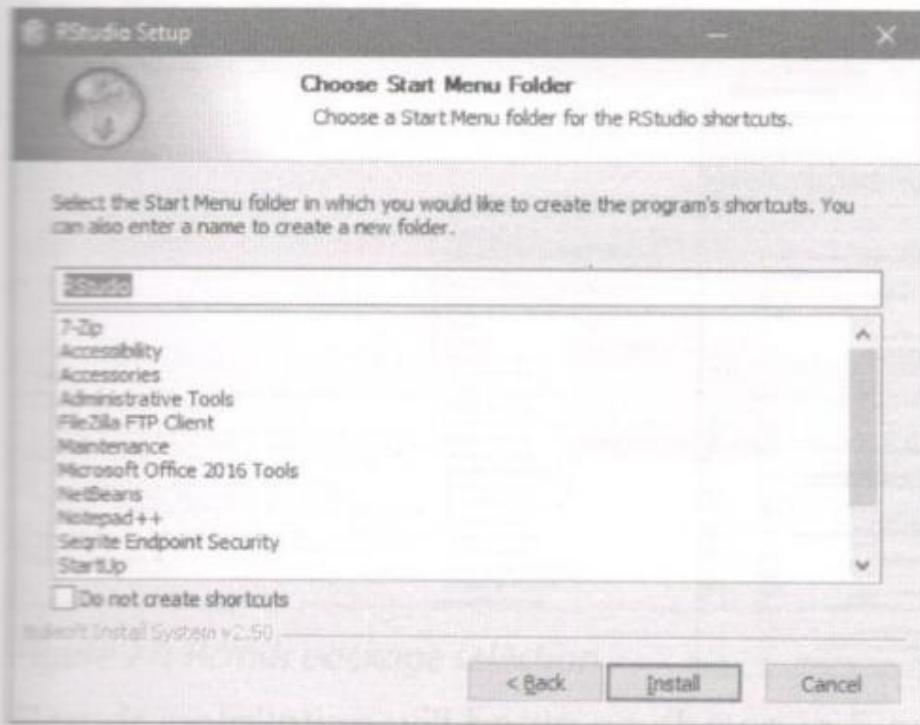


Figure 21: Choose Start Menu Folder page

Installing Rstudio

7. Installing page will open as shown in Figure 22. It is on this page where the installation of the RStudio will begin.

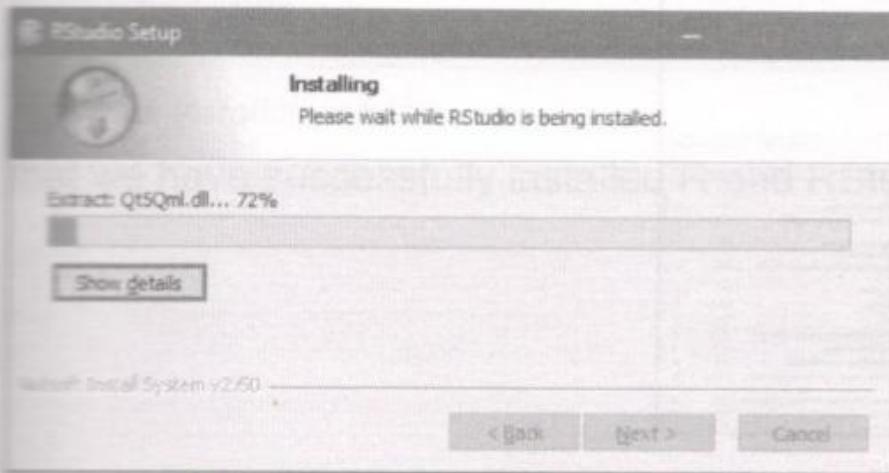


Figure 22: Installing page

Installing Rstudio

- Once RStudio has been installed as shown in Figure 23. Click Finish to exit the setup page.



Figure 23: RStudio is installed successfully

Installing R Commander

Installing R Commander

1. Open RStudio as shown in Figure 24.

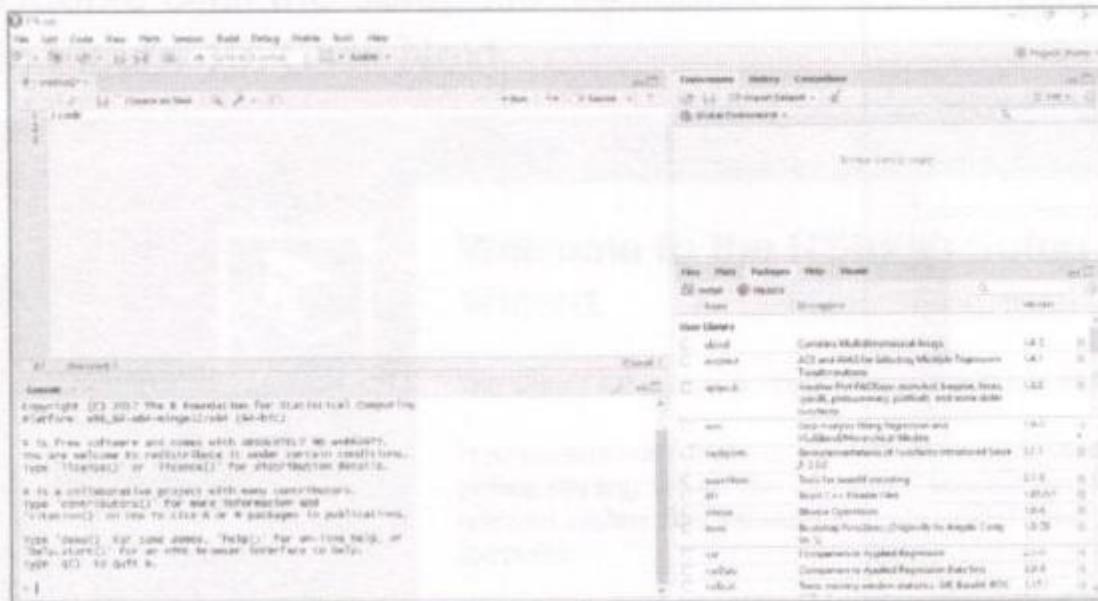


Figure 24: RStudio startup page

Installing R Commander

2. Go to the “Packages” tab and click on “Install Packages” as in shown Figure 25.

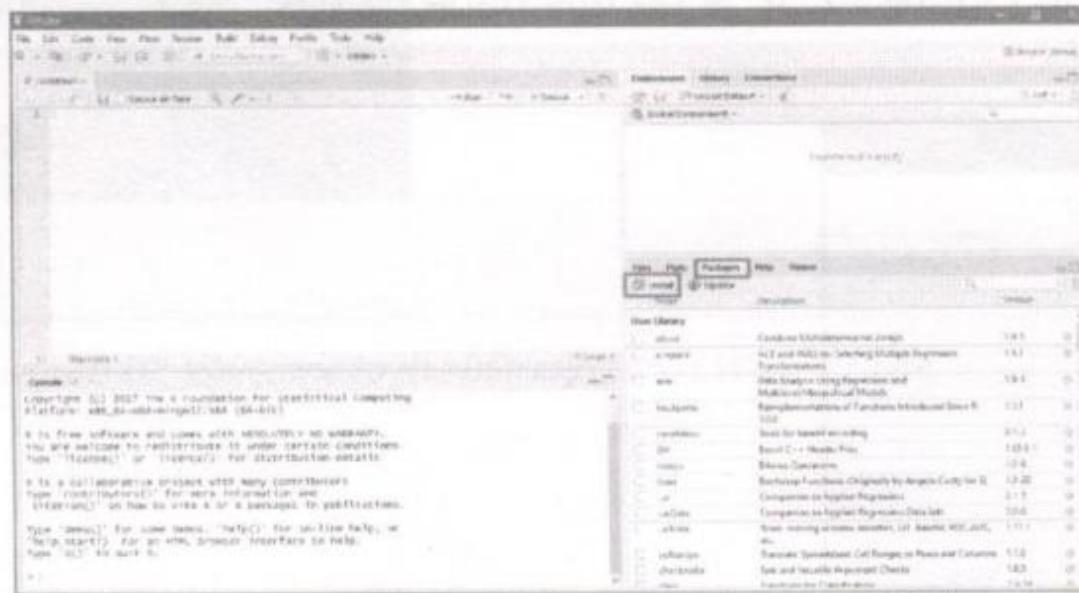


Figure 25: Installing Packages

Installing R Commander

- The first time that you will install the packages, you'll be prompted to choose a CRAN mirror as shown in Figure 26. R will download all necessary files from the server we select here.

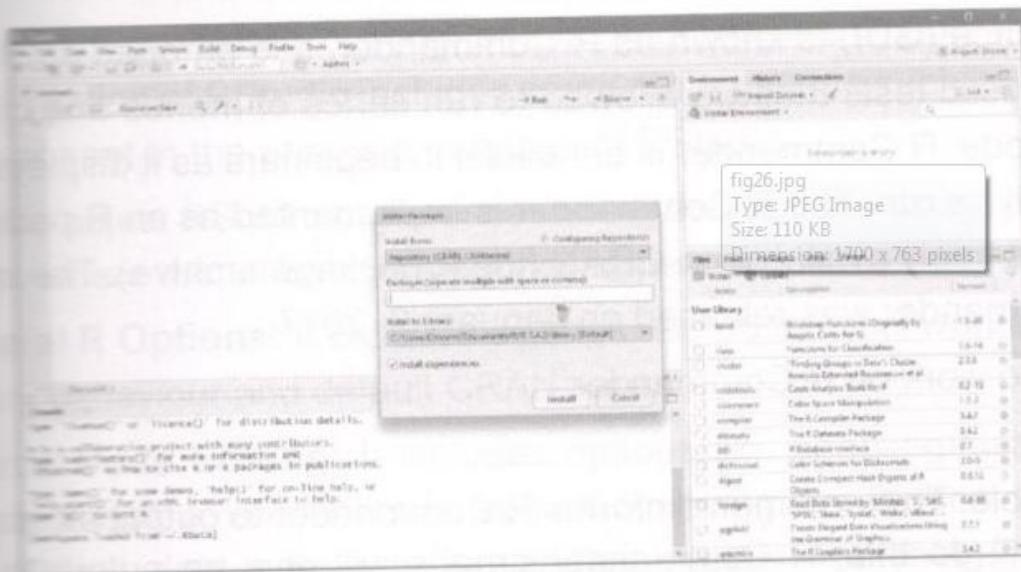


Figure 26: Install packages popup

Installing R Commander

- Start typing "Rcmdr" until we see it appear in a list. Select the first option and ensure that "Install dependencies" is checked as shown in Figure 27, and then click Install.

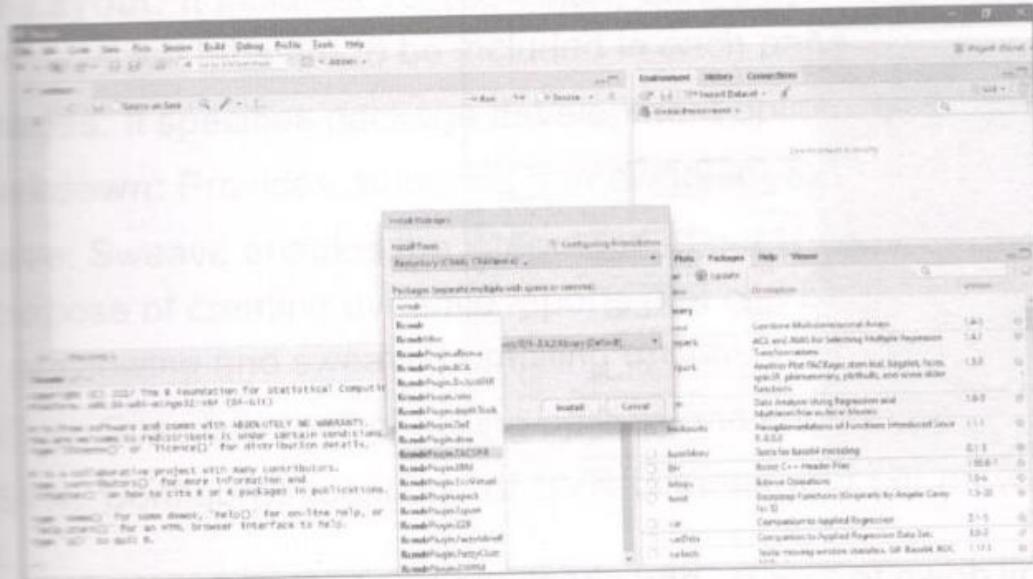
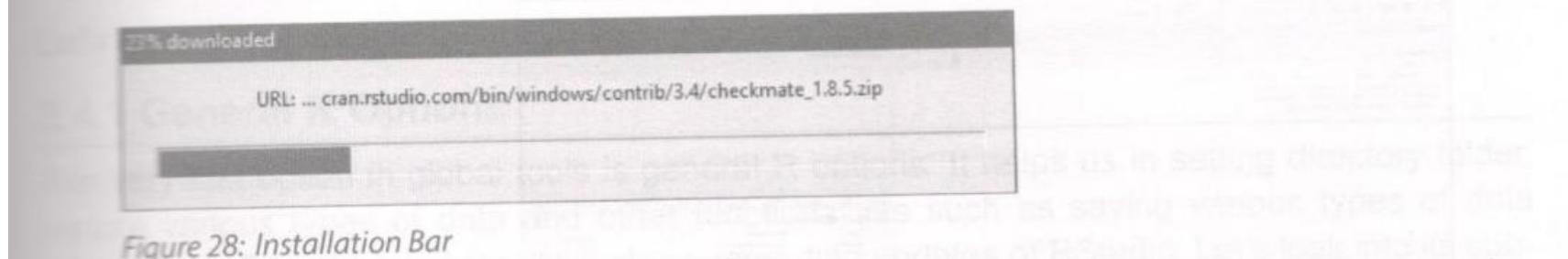


Figure 27: Rcmdr package selection

Installing R Commander



- Rcmdr installation will begin as shown in Figure 28.



```
1 cat("\f")  
2 a=10  
3 b=20  
4 c=a+b  
5 print(c)  
6  
7  
8  
9 I
```

1. Source or Script Window

1. This is where you write your programme.
2. Your Code will not be evaluated until you "Run" them to the console.

2. Console

1. This is where your code from the source is evaluated by R
2. You can, also use the console to perform quick calculations that you don't need to save.

a	10
c	30

3. Environment / History

1. Here you can see what object's are in your work space.
2. View Your Command History

4. Files/Plots/Packages/Help

1. Here you can see file directories, view plots, see your packages and access R Help

<input checked="" type="checkbox"/>	datasets	The R Datasets Package	3.6.0
<input type="checkbox"/>	foreign	Read Data Stored by 'Minitab', 'S', 'SAS', 'SPSS', 'Stata', 'Systat', 'Weka', 'dBase', ...	0.8-71



⌘ RStudio is divided into 4 “Panes”:

- the Source for your scripts and documents (top-left, in the default layout),
- the R Console (bottom-left),
- your Environment/History (top-right), and
- your Files/Plots/Packages/Help/Viewer (bottom-right).

Data Management in RStudio



⌘ Steps for creating an “R Project”

- └─ Start RStudio
- └─ Under the File menu, click on New project, choose New directory, then Empty project
- └─ As directory (or folder) name enter r-intro and create project as subdirecory of your desktop folder: ~/Desktop
- └─ Click on Create project
- └─ Under the Files tab on the right of the screen, click on New Folder and create a folder named data within your newly created working directory (e.g., ~/r-intro/data)
- └─ On the main menu go to Files > New File > R Script (or use the shortcut Shift + Cmd + N) to open a new file
- └─ Save the empty script as r-intro-script.R in your working directory.

Your working directory should now look like in Figure 1.2.

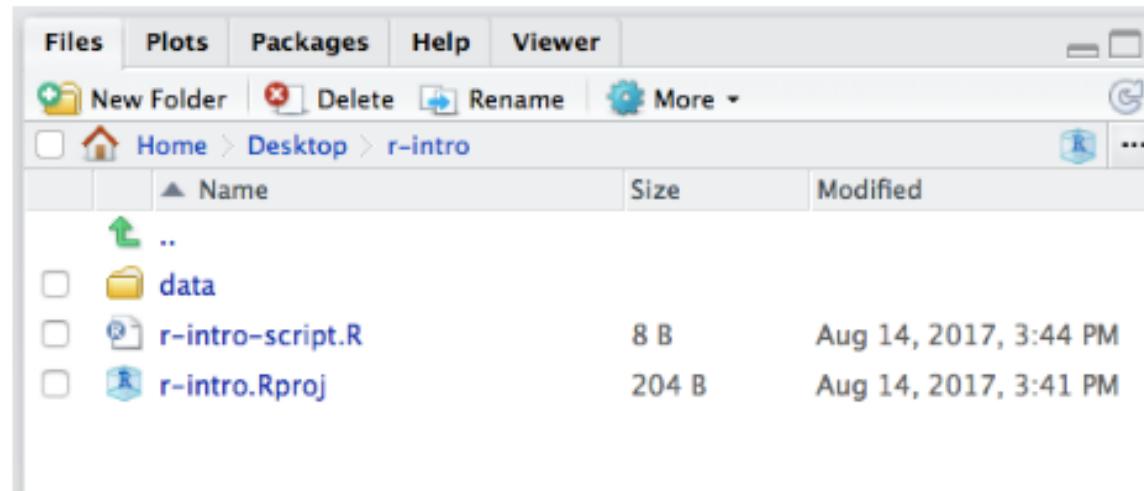


Figure 1.2: What it should look like at the beginning of this lesson

If you ever need to set a different working directory you can use the RStudio interface like seen in Figure 1.3.

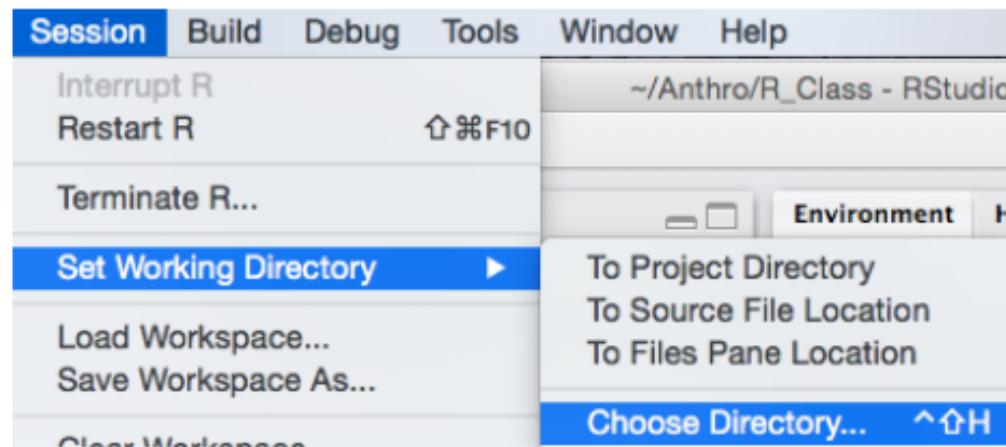


Figure 1.3: How to set a working directory with the RStudio interface



⌘ To check the current working directory

`getwd()`

⌘ To set a working directory in R go to the Console and type:

`setwd("Path/To/Your/Workingdirectory")`

Terminologies in R



⌘ Variables

Reserved memory locations which store the values provided by users.

⌘ Data types

Specifies the type of value that the variable holds

⌘ Operations

Operator works on one or more operands to get an output. Whole process is called as operation.

⌘ Loops

Used to execute multiple statements by using one or more conditions.

Terminologies in R



⌘ Strings

- ⌘ Used to represent characters/text in R.
- ⌘ Represented inside single quotes or double quotes

⌘ Functions

- ⌘ Slab of codes that are reusable and organized.
- ⌘ Used to perform a single, related action.

Data types in R



⌘ Numeric

⌘ Integer

⌘ Complex

⌘ Logical

⌘ Character

Data types in R



❖ Numeric Data type

❖ Decimal values are referred as numeric data types in R.

❖ This is the default working out data type.

```
g = 62.4      # assign a decimal value to g  
>g            # print the variable's value – g  
>class(g)
```

Data types in R



❖ Integer Data type

❖ To create any integer variable in R, invoke the `as.integer()` function to define any integer type data.

```
>s = as.integer(3)
```

```
>s      # print the value of s
```

```
>as.integer(3.14)  # drives in a numeric value
```

Data types in R



⌘ Complex data type

↗ A complex value for coding in R can be defined using the pure imaginary values 'i'.

```
>k = 1 + 2i    # creating a complex number
```

```
>k            # printing the value of k
```

Data types in R



❖ Logical data type

❖ A logical value is mostly created when comparison between variables are done.

```
> a = 4; b = 6          # sample values
```

```
> g = a > b          # is a larger than b?
```

```
> g                  # print the logical value
```

Data types in R



❖ Character Data type

- ❖ A character object can be used for representing string values in R.

```
>g="hai"
```

```
>g
```

- ❖ To convert objects into character values use the as.character() function

```
>g = as.character(62.48)
```

```
>g          # prints the character string
```

```
[1] "62.48"
```

Navigating ‘R’ workspace



- # Objects created during an R session will be stored in the system's memory.
- # Workspace
 - # Collection of all these objects.
- # Workspace will only be saved on a disk when we tell R to save it.
- # Our objects are volatile in nature, and can be deleted when we close the R without closing workspace or when during a session the system shuts down abruptly.



⌘ Everything we see or create in R is an object.

⌘ Five basic classes of objects

 ⌘ Character

 ⌘ Numeric (Real numbers)

 ⌘ Integer (Whole numbers)

 ⌘ Complex

 ⌘ Logical (True / False)



⌘ Closing R console

⌘q()

Prints the current working directory	getwd()
Lists the objects in the workspace	ls()
Change the working directory	setwd(new.dir)
List of help options	help(options)
List of available options	options()



```
>print("REVA")
```

```
[1] REVA
```

```
>print("Welcome to R Programming")
```

```
>print('Welcome to R programming')
```

```
>myString <- "Hello, World!"
```

```
>print ( myString)
```

```
[1] "Hello, World!"
```

Nuts and Bolts of using R



```
> 1 + 1
```

```
[1] 2
```

The digit '1' inside square brackets indicate that the index will start at the very first element. First element of the vector can be accessed with [1]

```
> 1 - 1
```

```
[1] 0
```

The operators available are + (addition), - (subtraction), / (division), * (multiplication), and ^ (raise to a power).



```
> variable1 = 1 + 1
```

```
> variable1
```

```
[1] 2
```

```
> variable1 <- 1 + 1
```

```
> variable1
```

```
[1] 2
```



```
> 1 + 2 / 3 - 2 * 6.5
```

```
[1] -11.33333
```

```
> 1 * (2 / (1 + 1))
```

```
[1] 1
```

```
> 1 * 2 / 1 + 1
```

```
[1] 3
```

Vectors

- ⌘ Used to store a sequence of data elements.
- ⌘ Can only store elements of similar data type.
- ⌘ A vector is a string of numbers; sometimes vectors are sequential numbers, other times vectors are random numbers — there is no restriction on the type or amount of numbers that a vector can contain.
- ⌘ Single element vector creation:
 - ↗ Use the function `c()`.
 - ↗ `C("xyz")`
 - ↗ `C(9.8)`
 - ↗ `C(898L)`
 - ↗ `C(FALSE)`
 - ↗ `C(3+9i)`

Vectors

⌘ Multiple element vector creation

- ⌘ Using colon

- ⌘ Using Sequence function

⌘ Using colon

```
> vector1 = 1:9
```

```
> vector1
```

```
[1] 1 2 3 4 5 6 7 8 9
```

The colon : operator creates a sequence of numbers from the left hand value to the right hand value, iterating in increments of 1.

Vectors



```
> 1.2:9
```

```
[1] 1.2 2.2 3.2 4.2 5.2 6.2 7.2 8.2
```

R started at 1.2 and increased this number by 1 until it reached the largest number less than or equal to 9.

Vectors



⌘ Another way to make vectors is with numbers you specifically select:

```
> vector2 = c(1, 3, 2, -8.1)
```

```
> vector2
```

```
[1] 1.0 3.0 2.0 -8.1
```

⌘ Numbers (separated by commas) can be inputted into the `c()` function to create a new vector with those numbers.

Vectors



⌘ Using sequence function

```
>c(seq(25,39,by=2))  
[1] 25, 27, 29, 31, 33, 35, 37, 39
```

Vectors



⌘ Accessing vector elements with positive integer indexes

```
>s=c("ax", "bx", "cx", "dx", "ex")
```

```
>s[4]
```

```
[1] dx
```

Vectors



```
>t <- c("Jan", "Feb", "Mar", "Apr", "May",
  "Jun", "Jul", "Aug", "Sep", "Oct", "Nov",
  "Dec")
```

```
> x <- t[c(2,9,4)]
```

```
>X
```

Rearranging the elements using : operator

```
X = c("ax", "bx", "cx", "dx", "ex")
```

```
X[2:5]
```

Vectors



⌘ Accessing vector elements with negative integers

⚠ Using negative integers as indexes will exclude or delete the specific element from the concatenation.

```
S = c("ax", "bx", "cx", "dx", "ex")
```

```
S[-5]
```

```
[1] "ax" "bx" "cx" "dx"
```

Vectors



⌘ Multiple exclusion from concatenation

```
t<- c("Jan", "Feb", "Mar", "Apr", "May",  
  "Jun", "Jul", "Aug", "Sep", "Oct", "Nov",  
  "Dec")
```

```
X<- t[c(-4, -6)]
```

Vectors



⌘ Accessing elements of named vectors

```
v=c("Jerry", "Jim")
```

```
v
```

```
[1] "Jerry" "Jim"
```

```
names(v)=c("1", "2")
```

```
v
```

```
1      2
```

```
"Jerry" "Jim"
```

Vectors



```
v=c("Jerry", "Jim")
```

```
v
```

```
[1] "Jerry" "Jim"
```

```
names(v)=c("1", "2")
```

```
v
```

```
1     2
```

```
"Jerry" "Jim"
```

```
v[c("2", "1")]
```

```
2     1
```

```
"Jim" "Jerry"
```



⌘ Accessing elements using logical index values

- Used to slice a certain portion of the vector to create a new vector based on conditional evaluation.
- In general, an index value of TRUE means to include the associated data member whereas FALSE means to exclude the data element from the new vector.



```
S = c("ax", "bx", "cx", "dx", "ex")
L = c(FALSE, TRUE, FALSE, TRUE, FALSE)
>S[L]
>[1] "bx"      "dx"
```



⌘ Vector Manipulation

- Combining vectors
- Vector Arithmetic
- Vector element recycling
- Vector element sorting



⌘Combining vectors

↗ Vectors are sometimes combined to merge the information contained in two or more vectors.

↗ Done by using `c()` function

↗ Syntax

`c(....)`

.... are the objects to be concatenated.



```
n = c(4, 1, 5)
```

```
s = c("ax", "bx", "cx")
```

```
c(n,s)
```

Output:

```
"4" "1" "5" "ax" "bx" "cx"
```



❖ Vector Arithmetic

- ❖ Arithmetic operations can be applied on vectors.
- ❖ To perform arithmetic operations, it is important that the included vectors in the arithmetic operations should be of the same length.

Arithmetic Operations	Examples
Creation of vector	<pre>vec_1 <- c(23,28,42,25,20,14) vec_2 <- c(24,14,50,81,14,12)</pre>
Performing addition on Vec1 and 2	<pre>addition_result <- vec_1 + vec_2 print(addition_result)</pre>
Performing subtraction on Vec1 and 2	<pre>sub_result <- vec_1 - vec_2 print(sub_result)</pre>
Performing multiplication on Vec1 and 2	<pre>mul_result <- vec_1 * vec_2 print(mul_result)</pre>
Performing division on Vec1 and 2	<pre>div_result <- vec_1 / vec_2 print(div_result)</pre>



⌘ Vector element recycling

- ─ R automatically recycles, or repeats, elements of the shorter Vector.
- ─ When applying an operation to two vectors that requires them to be the same length, R automatically recycles, or repeats, elements of the shorter one, until it is long enough to match the longer Vector.



⌘ Suppose we have two Vectors $c(1,2,4)$, $c(6,0,9,10,13)$, where the first one is shorter with only 3 elements. Now if we sum these two, we will get a warning message as follows.

```
>c(1,2,4) + c(6,0,9,10,13)  
[1] 7 2 13 11 15
```

Warning message:

In $c(1, 2, 4) + c(6, 0, 9, 10, 13)$: longer object length is not a multiple of shorter object length.



❖ Vector element sorting

- ❖ Arrange the elements in an organised sequence.
- ❖ To sort, R programming uses the `sort()` function.

```
Vec <- c(4.9, 5, 6, 1, 10, -11, 654)
```

Description	Commands
Sort the elements of the vector	<code>sort_vecresult <- sort(vec)</code> <code>print(sort_vecresult)</code>
Sort the elements in the reverse order	<code>revsort_vecresult <- sort(vec,</code> <code>decreasing = TRUE)</code> <code>print(revsort_vecresult)</code>
Sort the character elements of the vector	<code>vec <- c("green", "black", "red",</code> <code>"white")</code> <code>sort_vecresult <- sort(vec)</code> <code>print(sort_vecresult)</code>
Sort the character elements of the vector in decreasing order	<code>revsort_vecsesult <-sort(vec,</code> <code>decreasing = TRUE)</code> <code>print(revsort_vecresult)</code>



⌘ Deleting a vector

- When the data structure is no longer required, the allocated memory to the data structure is released.
- To delete a vector, null value is assigned to it.



```
X <- c(1, 5, 4, 9, 0)
```

```
X <- NULL
```

```
X
```

Vectors



⌘ When adding vectors, R adds the first element of one vector to the first element of the other, the second element of one vector to the second element of the other, etc.

Vectors



```
> vector3 = 1:9  
> vector3  
[1] 1 2 3 4 5 6 7 8 9
```

```
> vector4 = 9:1  
> vector4  
[1] 9 8 7 6 5 4 3 2 1
```

```
> vector3 + vector4  
[1] 10 10 10 10 10 10 10 10 10
```

Vectors



⌘ what happens if the vectors are not the same size?

```
vector5 = 1:3
```

```
> vector5  
[1] 1 2 3
```

```
> vector3  
[1] 1 2 3 4 5 6 7 8 9
```

```
> vector3 + vector5  
[1] 2 4 6 5 7 9 8 10 12
```

Vectors

- # Create the vectors:
 - # (a) (1, 2, 3, . . . , 19, 20)
 - # (b) (20, 19, . . . , 2, 1)
 - # (c) (1, 2, 3, . . . , 19, 20, 19, 18, . . . , 2, 1)
 - # (d) (4, 6, 3) and assign it to the name tmp.
 - # For parts (e), (f) and (g) look at the help for the function rep.
 - # (e) (4, 6, 3, 4, 6, 3, . . . , 4, 6, 3) where there are 10 occurrences of 4.
 - # (f) (4, 6, 3, 4, 6, 3, . . . , 4, 6, 3, 4) where there are 11 occurrences of 4, 10 occurrences of 6 and 10 occurrences of 3.
 - # (g) (4, 4, . . . , 4, 6, 6, . . . , 6, 3, 3, . . . , 3) where there are 10 occurrences of 4, 20 occurrences of 6 and 30 occurrences of 3.

Vectors

- ⌘ Create a vector of the values of $e^x \cos(x)$ at $x = 3, 3.1, 3.2, \dots, 6$
- ⌘ Create the following vectors:
 - ☒ $(0.1^{30}, 0.2^1, 0.1^6, 0.2^4, \dots, 0.1^{36}, 0.2^{34})$

$$\left(2, \frac{2^2}{2}, \frac{2^3}{3}, \dots, \frac{2^{25}}{25}\right)$$

$$\sum_{i=10}^{100} (i^3 + 4i^2).$$

$$\sum_{i=1}^{25} \left(\frac{2^i}{i} + \frac{3^i}{i^2} \right)$$

Vectors

- ⌘ Use the function paste to create the following character vectors of length 30:
 - ⌘ (a) ("label 1", "label 2", , "label 30").
Note that there is a single space between label and the number following.
 - ⌘ (b) ("fn1", "fn2", ..., "fn30").
In this case, there is no space between fn and the number following.

Matrices

⌘ Collection of homogenous data elements which is represented using a tabular format or as a collection of rows and columns.

⌘ Can contain elements of the same data type.

⌘ Syntax

matrix (data, nrow, ncol, byrow, dimnames)

where

data – set of input values for the matrix

nrow – number of rows

ncol – number of columns

byrow – logical clue; If FALSE (the default) the matrix is filled by columns, otherwise the matrix is filled by rows.

dimnames – defines the names for rows and columns of the matrix.

Matrices



⌘ Create of a matrix of 2 X 4 using the numeric data from 1 to 8.

```
matrix (1:8, nrow = 2, ncol = 4)
```

```
matrix (1:8, nrow=2)
```

Matrices



⌘ **dim** parameter

➤ To check the dimensions of the matrix.

```
M <- Matrix(nrow = 2, ncol = 4)
```

```
dim(M)
```

```
[1] 2 4
```

Matrices



⌘ dimnames

Used to define the names for the rows and columns of the matrix.

```
x <- matrix(1:8, nrow = 2, dimnames =  
list(c("ax", "by"), c("xa", "xb", "xc", "xd")))
```

```
x
```

colnames() : used to extract the name of columns in the matrix

rownames() : used to extract the name of rows in the matrix.

Matrices



```
colnames(x) <- c("X1", "X2", "X3", "X4")
```

```
rownames(x) <- c("A1", "A2")
```

```
x
```

colnames() and rownames() are used to rename the columns and rows of matrix.

Matrices



- ⌘ Matrix can also be created using cbind() and rbind() functions.
- ⌘ rbind() – declares the rows of the matrix
- ⌘ cbind() – declares the cols of the matrix

cbind(c(87,31), c(19,95))

rbind(c(13,95,21), c(70,98,60))

```
> cbind(c(87,31),c(19,95))
 [,1] [,2]
[1,] 87 19
[2,] 31 95
> rbind(c(13,95,21), c(70,98,60))
 [,1] [,2] [,3]
[1,] 13 95 21
[2,] 70 98 60
```

Matrices

- ⌘ Simple way to create a new matrix.
- ⌘ Can be defined as a collection of homogeneous data elements which is represented using a tabular format or as a collection of rows and columns.

```
>matrix(c(1, 2, 3, 4, 5, 6, 7, 8, 9), nrow = 3)
```

	[,1]	[,2]	[,3]
[1,]	1	4	7
[2,]	2	5	8
[3,]	3	6	9

Matrices



- ⌘ `matrix()` function takes the data given as input and the number of rows inputted (`nrow`) and makes a matrix by filling down each column from the left to the right.
- ⌘ `[1,]` means take all the elements; if it comes after the number it means all the elements in that row, if before the number all the elements in that column.

Matrices



- ⌘ Number of columns (`ncol`) can also be specified.

```
> matrix(1:8, ncol = 2)
```

```
 [,1] [,2]  
[1,]    1    5  
[2,]    2    6  
[3,]    3    7  
[4,]    4    8
```

Matrices



- ⌘ Mathematical vector and turn it into a matrix:

```
> matrix(vector3, nrow = 3)
```

	[,1]	[,2]	[,3]
[1,]	1	4	7
[2,]	2	5	8
[3,]	3	6	9

- ⌘ The numbers above and to the left side of the matrix give the dimensions of the matrix.

Matrices



- ⌘ If you try to create a matrix that has more elements than data provided, then R will give you a warning message and loop the data set until the matrix is filled:

```
> matrix(vector3, nrow = 2)
```

	[,1]	[,2]	[,3]	[,4]	[,5]
[1,]	1	3	5	7	9
[2,]	2	4	6	8	1

- ⌘ The bottom right corner of the matrix was created by looping the mathematical vector.

Matrices



```
> 1 + 1  
[1] 2  
> vector3  
[1] 1 2 3 4 5 6 7 8 9
```

This [1] is because a vector is equivalent to one row (or one column) of a matrix.

The answer to $1 + 1$ is a vector with one element (essentially a number) and vector3 is a vector with nine elements.

Vector with 9 elements is roughly equivalent to a matrix with 1 row and 9 columns.

Matrices

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

Adding, subtracting, multiplying, and dividing matrices works simply by performing the action on each element of the matrix:

```
> matrix1 = matrix(vector3, nrow = 3)
```

```
> matrix1 + 2
```

```
[ ,1] [ ,2] [ ,3]  
[1, ]    3     6     9  
[2, ]    4     7    10  
[3, ]    5     8    11
```

Matrices



- ❖ Adding two matrices works only when the matrices are of the same dimensions (same number of columns and rows) and the matrices are added element by element in the same way as vectors:

```
> matrix1 + matrix1  
[ ,1] [ ,2] [ ,3]  
[ 1, ]    2     8    14  
[ 2, ]    4    10    16  
[ 3, ]    6    12    18
```

Matrices



- ❖ In order to perform matrix multiplication in the linear algebra sense (not just multiply each element by a constant) the operation should be put in —%.

```
> matrix1%*%matrix1  
[ ,1 ] [ ,2 ] [ ,3 ]  
[ 1, ]    30    66   102  
[ 2, ]    36    81   126  
[ 3, ]    42    96   150
```

Matrices



- ❖ In contrast, element-by-element multiplication can also be performed:

```
> matrix1 * matrix1  
[.1] [.2] [.3]  
[1,] 1 16 49  
[2,] 4 25 64  
[3,] 9 36 81
```

Matrices



- ⌘ It is also very easy to take the transpose of a matrix (switch the rows and columns):

```
> t(matrix1)  
      [,1] [,2] [,3]  
[1,]    1    2    3  
[2,]    4    5    6  
[3,]    7    8    9
```

Manipulation



>Matrix1

	[.1]	[.2]	[.3]
[1,]	1	4	7
[2,]	2	5	8
[3,]	3	6	9

[1,] means take all the elements; if it comes after the number it means all the elements in that row, if before the number all the elements in that column.

Matrices



```
> matrix1[1, 3]
```

```
[1] 7
```

```
> matrix1[ 2, ]
```

```
[1] 2 5 8
```

Matrices



- ⌘ To remove the second column from the matrix

```
> matrix1[, -2]  
      [,1] [,2]  
[1,]    1    7  
[2,]    2    8  
[3,]    3    9
```

- ⌘ R takes the matrix, removes the second column, and shifts everything over.

Matrices



⌘ If you want to change the actual values of the data just access the part of the matrix you want to change and use the = operator as follows:

➤ `matrix1[1, 1] = 15`

Matrices



```
> matrix1[,2] = 1
```

```
> Matrix1
```

	[,1]	[,2]	[,3]
[1,]	15	1	7
[2,]	2	1	8
[3,]	3	1	9

```
> matrix1[,2:3] = 2
```

```
> matrix1
```

	[,1]	[,2]	[,3]
[1,]	15	2	2
[2,]	2	2	2
[3,]	3	2	2

Matrices



```
> matrix1[,2:3] = 4:9
```

```
>matrix1
```

	[,1]	[,2]	[,3]
[1,]	15	4	7
[2,]	2	5	8
[3,]	3	6	9

```
> matrix1[,1] = vector5
```

```
> matrix1
```

	[,1]	[,2]	[,3]
[1,]	1	4	7
[2,]	2	5	8
[3,]	3	6	9

Matrices



```
> matrix1 > 5
```

```
 [,1] [ ,2] [ ,3]  
[1,] FALSE FALSE TRUE  
[2,] FALSE FALSE TRUE  
[3,] FALSE TRUE TRUE
```

Matrices



⌘ Accessing elements of a matrix – integer vector as the index

⌘ `X <- matrix(1:20, nrow=4, ncol=5)`

Description	Command
To list all the elements in the matrix	<code>x[,]</code>
To display all the elements in the first row	<code>x[1,]</code>
To display all the elements from 1 st row to 3 rd row	<code>x[1:3,]</code>
To exclude 4 th row	<code>x[-4,]</code>
To display all the values in the second column	<code>X[, 2]</code>
To display all the values from the second col. to third col.	<code>X[, 2:3]</code>

Matrix computations

❖ Addition and Subtraction of matrices

❖ Primary condition is that, both the matrices should be of similar dimensions.

Operations	Commands
Create two matrices of order 2 X 3	<pre>mat_1 <- matrix(c(13,29,-10,3,4,9), nrow =2) print(mat_1) mat_2 <- matrix(c(10,2,20,4,13,10), nrow =2) print(mat_2)</pre>
Matrix addition	<pre>Result_add <- mat_1 + mat_2 cat("Result of Addition","\n") print(Result_add)</pre>
Matrix Subtraction	<pre>Result_sub <- mat_1 - mat_2 cat("Result of Subtraction", "\n") print(Result_sub)</pre>

❖ Multiplication and Division of matrices

◻ Dimensions of both matrices are similar.

Operations	Commands
Create two matrices of order 2X3	<pre>mat1<- matrix(c(3,2,-1,5,8,0,-4,1,70), nrow=3) print(mat1) mat2<- matrix(c(7,-3,9,6,1,0,3,2,1), nrow=3) print(mat2)</pre>
Matrix multiplication	<pre>resultmul<- mat1 * mat2 Cat("Result of multiplication", "\n") print(resultmul)</pre>
Matrix Division	<pre>resultdiv<- mat1 / mat2 Cat("Result of Division", "\n") print(resultdiv)</pre>



⌘ Modification of matrix elements

⌘ Element-based modification

↗ Done using assignment operations

Operation	Code
Creating a matrix	<code>x<-matrix(1:9, nrow=3, ncol=3)</code>
Modifying single element of a matrix	<code>x[2,2] <- 10</code> <code>X</code>
Modifying multiple elements of the matrix on the basis of conditional check	<code>x[x<5] <- 0</code> <code>x</code>

Arrays



- # Multi-dimensional data structure used to store homogenous data types.
- # In a 3-D array, the elements are arranged in the form of tables, rows and columns.
- # Represented in the form of (r x c x h)
- # Syntax

```
Array_name <- array(data, dim =  
c(row_size, col_size, matrices, dimnames))
```

Arrays



- Array_name – variable which specifies the name given to the array.
- Data – input vector for the array
- Row_size – specifies the number of rows in the array
- Column_size – specifies the number of columns in the array
- Matrices – specifies the number of tables in the array.
- Dimnames – used to define the names of rows, columns and matrices

Arrays



⌘ Creating an array

```
arr <- array(10:34, dim = c(4,3,2))
```

Arr

	[,1]	[,2]	[,3]
[1,]	10	14	18
[2,]	11	15	19
[3,]	12	16	20
[4,]	13	17	21

, , 2

	[,1]	[,2]	[,3]
[1,]	22	26	30
[2,]	23	27	31
[3,]	24	28	32
[4,]	25	29	33

Arrays



⌘ Creating an array with vector values

```
vector1<- c(21,10,36,12)
```

```
vector2 <- c(25,36,41,37,49,70)
```

```
arr<-array(c(vector1,vector2),dim=c(1,3,2))
```

```
print(arr)
```

```
, , 1  
[1,] [,1] [,2] [,3]  
      21   10   36
```

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

```
      12   25   36
```

Naming rows and columns



⌘ Rows, columns and matrices are either sequentially numbered or assigned NULL as their default name.

⌘ Syntax

```
Arr_name <- matrix(data, dim =  
list(rownames, colnames, matrixnames))
```

Arrays



```
A <- array(1:24, dim = c(3,4,2), dimnames  
= list(c("AR1", "AR2", "AR3"), c("AC1",  
"AC2", "AC3", "AC4"), c("AM1", "AM2")))  
, , AM1
```

```
Print(A)
```

	AC1	AC2	AC3	AC4
AR1	1	4	7	10
AR2	2	5	8	11
AR3	3	6	9	12

```
, , AM2
```

	AC1	AC2	AC3	AC4
AR1	13	16	19	22
AR2	14	17	20	23
AR3	15	18	21	24

Arrays



⌘ Accessing array elements

↗ Can be accessed using row, column and matrix indexes.

Array_name[row_position, col_position, matrix_level]

Arrays



```
A <- array(1:24, dim = c(3,4,2))
```

Operations	Commands
Access the element stored at 2 nd row and 3 rd column in matrix 1	Print(A[2,3,1])
Access all the elements in the second row in the 1 st matrix	Print(A[2, ,1])
Access all the elements stored in the 1 st column in the second matrix	Print(A[, 1, 2])
Access all the elements stored in the first matrix	Print(A[, ,1])
Access all the elements stored in the second matrix	Print(A[, , 2])



Operations	Commands
Access All the element of 2nd and 3rd row in array 'a'	print(a[c(2, 3), , 2])
Access All the element of 1st and 4th Column in array 'a'	print(a[, c(1, 4), 1])
Access All the element except 2nd row and 3rd Column in array 'a'	print(A[-2, -3, 2])

Manipulating array elements



```
# Adding and Subtracting Elements of Array in R
vect1 <- c(10, 20, 40 )
vect2 <- c(55, 67, 89, 96, 100)
A <- array(c(vect1, vect2), dim = c(3, 4, 2))
print(A)
mat.A <- A[, , 1]
mat.B <- A[, , 2]
print(mat.A + mat.B)
print(mat.B - mat.A)
```

Manipulating array elements



⌘ To perform the arithmetic operations, we are converting the multidimensional matrix into a one-dimensional matrix.

```
>  
> mat.A <- A[, , 1]  
> mat.B <- A[, , 2]  
>  
> print(mat.A + mat.B)  
      [,1] [,2] [,3] [,4]  
[1,]    77   155  136  109  
[2,]   109    77   155  136  
[3,]   136   109    77  155  
>  
> print(mat.B - mat.A)  
      [,1] [,2] [,3] [,4]  
[1,]    57    45   -56    69  
[2,]    69   -57   -45    56  
[3,]    56   -69    57    45
```

Try....

$$\mathbf{A} = \begin{bmatrix} 1 & 1 & 3 \\ 5 & 2 & 6 \\ -2 & -1 & -3 \end{bmatrix}$$

- (a) Check that $\mathbf{A}^3 = \mathbf{0}$ where $\mathbf{0}$ is a 3×3 matrix with every entry equal to 0.
- (b) Replace the third column of \mathbf{A} by the sum of the second and third columns.

Create the following matrix \mathbf{B} with 15 rows:

$$\mathbf{B} = \begin{bmatrix} 10 & -10 & 10 \\ 10 & -10 & 10 \\ \dots & \dots & \dots \\ 10 & -10 & 10 \end{bmatrix}$$

Calculate the 3×3 matrix $\mathbf{B}^T \mathbf{B}$. (Look at the help for `crossprod`.)

Try....

Create the following patterned matrix

$$\begin{pmatrix} 0 & 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 & 5 \\ 2 & 3 & 4 & 5 & 6 \\ 3 & 4 & 5 & 6 & 7 \\ 4 & 5 & 6 & 7 & 8 \end{pmatrix}$$

Lists



- # Used to store heterogeneous data.
- # Can be defined as a data structure that can store data elements from multiple data types.
- # Generic vector that store data elements such as strings, numbers, logical values and vectors.

Creating a list



⌘ The list function is used to create a list of data elements.

```
list.data <- list("Tutorial", "Gateway", TRUE,  
c(10, 20, 30), 95, 105.61)  
  
print(list.data)
```

```
> list.data <- list ("Tutorial", "Gateway", TRUE, c(10, 20, 30), 95, 105.61)  
> list.data  
[[1]]  
[1] "Tutorial"  
  
[[2]]  
[1] "Gateway"  
  
[[3]]  
[1] TRUE  
  
[[4]]  
[1] 10 20 30
```

Creating a list using vectors



```
# Creating three vectors  
vect.a <- c(10.25, 20.45, 30.75, 40.85)  
vect.b <- c(25, 50, 75, 100, 125)  
vect.c <- c("India", "China", "Japan", "Russia",  
          "USA")  
  
# Creating list  
list.data <- list(vect.a, vect.b, vect.c )  
print(list.data)
```

```
[[1]]  
[1] 10.25 20.45 30.75 40.85  
  
[[2]]  
[1] 25   50   75 100 125  
  
[[3]]  
[1] "India"  "China"  "Japan"  "Russia" "USA"
```

Creating R list using matrices and vectors



```
vect.a <- c(10.25, 30.75, 20.45, 40.85)
vect.b <- c("India", "Japan", "Russia", "China", "USA")
list.data <- list(vect.a, vect.b )
print(list.data)

A <- matrix(c(1:12), nrow = 3)
vect.c <- c(50, 75, 25, 100, 125)
list.mixed <- list(A, list.data, vect.c )
print(list.mixed)
```

[[1]] [,1] [,2] [,3] [,4]
[1,] 1 4 7 10
[2,] 2 5 8 11
[3,] 3 6 9 12

[[2]]
[[2]][[1]]
[1] 10.25 30.75 20.45 40.85

[[2]][[2]]
[1] "India" "Japan" "Russia" "China" "USA"

[[3]]
[1] 50 75 25 100 125

Creating Named List in R



⌘ Syntax:

```
list_Name <- c("index_Name1" = Value1,  
           "index_Name2" = Value2,...  
, "index_NameN" = ValueN )
```

```
list.data <- list("Company" = "Tutorial  
Gateway", "Flag" = TRUE, "prod" = c(10,  
20, 30), "val" = 95, "sale" = 105.61)  
print(list.data)
```

```
$company  
[1] "Tutorial Gateway"
```

```
$Flag  
[1] TRUE
```

```
$prod  
[1] 10 20 30
```

```
$val  
[1] 95
```

```
$sale  
[1] 105.61
```

Creating Named List in R using names function



⌘ Named function is very useful to assign names to Lists.

⌘ Syntax :

names(<list name>) <- c("name1", "name2", ..., "nameN").

```
# Create Named List in R Programming

vect.x <- c(10, 30, 50, 70)
vect.y <- c("India", "Japan", "UK", "Russia", "China", "USA")

list.a <- list(vect.x, vect.y )
# Assigning Names
names(list.a) <- c("Num_Vector", "Country")
print(list.a)

matrix.A <- matrix(c(1:12), 3, 4)
vect.z <- c(55, 75, 25, 105, 125)

list.mixed <- list(matrix.A, list.a, vect.z, "Tutorial Gateway")
names(list.mixed) <- c("Num_Matrix", "Inner_List", "Rand_vector", "Company")
print(list.mixed)
```

```
$Num_Matrix
 [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12

$Inner_List
$Inner_List$Num_Vector
[1] 10 30 50 70

$Inner_List$Country
[1] "India"  "Japan"  "UK"      "Russia" "china"  "USA"

$Rand_vector
[1] 55 75 25 105 125

$Company
[1] "Tutorial Gateway"
```

Accessing R List Elements



- ⌘ In R programming, elements in a List can be accessed using the index position.
- ⌘ Using this index value, we can access or alter/change each and every individual element present in the List.
- ⌘ Index value starts at 1 and ends at n where n is the number of elements in a list.

```
# Accessing R List Elements

vect.a <- c(10.25, 30.75, 20.45, 40.85)
vect.b <- c("India", "Japan", "Russia", "China", "USA")
vect.c <- c(50, 75, 25, 100, 125)

A <- matrix(c(1:12), 3, 4)

list.data <- list(A, vect.a, "Tutorial Gateway", vect.b, 95, vect.c )
print(list.data)

# Accessing First Element
print(list.data[1])

# Accessing Fourth Element
print(list.data[4])
```

```
> print(list.data[1])
[[1]]
 [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
```

```
> print(list.data[4])
[[1]]
[1] "India"  "Japan"  "Russia" "China"  "USA"
```

```
> print(list.data)
[[1]]
 [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12

[[2]]
[1] 10.25 30.75 20.45 40.85

[[3]]
[1] "Tutorial Gateway"

[[4]]
[1] "India"  "Japan" "Russia" "China" "USA"

[[5]]
[1] 95

[[6]]
[1] 50  75  25 100 125
```

Accessing R List items using Names



- ⌘ If we declared the list items with names or we assigned the names to list items, then we can use those names to access the List items.
- ⌘ Syntax : *<list name>\$Index_Name*

Accessing R List items using Names

```
# Accessing R List Elements

vect.x <- c(10, 30, 50, 70)
vect.y <- c("India", "Russia", "Japan", "UK", "China", "USA")

list.data <- list(vect.x, vect.y )
names(list.data) <- c("Numeric_Vector", "Country")

matrix.A <- matrix(c(1:12), 3, 4)
vect.z <- c(55, 75, 25, 105, 125)

list.mixed <- list(matrix.A, list.data, vect.z, "Tutorial Gateway")
names(list.mixed) <- c("Numeric_Matrix", "Nested_List", "Random_vector", "Company")
print(list.mixed)

# Accessing Vector.z Elements
print(list.mixed$Random_vector)

# Accessing Vector.z Elements
print(list.mixed$Numeric_Matrix)

# Accessing Nested List Elements
print(list.mixed$Nested_List)
```

Accessing R List items using Boolean vector

```
# Accessing R List Elements

vect.a <- c(10.25, 30.75, 20.45, 40.85)
vect.b <- c("India", "Japan", "Russia", "China", "USA")
vect.c <- c(50, 75, 25, 100, 125)

A <- matrix(c(1:12), 3, 4)

list.data <- list(A, vect.a, "Tutorial Gateway", vect.b, 95, vect.c )
print(list.data)

# Accessing List Element using Boolean Vector
print(list.data[c(FALSE, FALSE, TRUE, FALSE, TRUE, TRUE)]))

# Accessing All Element except 1
print(list.data[-1])

# Accessing All Element except 4
print(list.data[-4])

# Accessing All Element except 1 and 6th element
print(list.data[c(-1, -6)])
```

Manipulate R List Elements

```
# Manipulating R List Elements

vect.x <- c(10, 30, 50, 70)
vect.y <- c("India", "Russia", "Japan", "UK", "China", "USA")

list.data <- list(vect.x, vect.y )
names(list.data) <- c("Numeric_Vector", "Country")

matrix.A <- matrix(c(1:12), 3, 4)
vect.z <- c(55, 75, 25, 105, 125)

list.mixed <- list(matrix.A, list.data, 95, vect.z, "Tutorial Gateway")
names(list.mixed)   <-   c("Numeric_Matrix",    "Nested_List",    "favNum",      "Random_vector",
"Company")
print(list.mixed)

list.mixed$Company <- "Tutorialgateway.org"
print(list.mixed$Company)

list.mixed$Random_vector <- c(22, 44, 66, 88)
print(list.mixed$Random_vector)

list.mixed$Numeric_Matrix <- NULL
print(list.mixed)
```

Merge two lists in R programming

```
# Merging two Lists in R Programming

#Declared Two vector
vect.a <- c(10.25, 30.75, 20.45, 40.85)
vect.b <- c("India", "Japan", "Russia", "China", "USA")

# Creating List 1 from those two Vectors Vect.a, Vect.b
list.x <- list(vect.a, 95, vect.b )
print(list.x)

# Declared One 4 * 3 matrix
A <- matrix(c(1:12), 4, 3)

# Creating second list with one String, Matrix, and a vector
list.y <- list("Tutorial Gateway", A, c(5, 10, 15) )
print(list.y)

# Combining or Merging two List
list.z <- c(list.x, list.y)
print(list.z)
```

Decision Making and Branching statements



❖ If Statement

```
if (Boolean_Expression) {  
    Statement 1;  
    Statement 2;  
    .....  
    .....  
    Statement n;  
}
```

If Statement



```
number <- as.integer(readline(prompt="Please Enter any  
integer Value: "))
```

```
if (number > 1)
```

```
{
```

```
print("You have entered POSITIVE Number")
```

```
}
```

```
X <- 33
```

```
if(x>0)
```

```
{
```

```
Print("Positive number")
```

```
}
```

*Find the largest among three
numbers using simple if statement*

If..else



```
if (test_expression)
{
    statement1
}
else
{
    statement2
}
```

If..else



```
X <- -33
if (X > 0) {
print ("Positive number")
} else {
print("Negative number")
}
```

Else .. If ladder



```
if (test expression1) {  
    Statement 1  
} else if (test expression2) {  
    Statement 2  
} else if (test expression3) {  
    Statement 3  
}  
...  
else  
    Statement 4
```

Else .. If ladder



X<-0

```
if (X < 0) {
```

```
print("Negative Number")
```

```
} else if (X > 0) {
```

```
print("Positive Number")
```

```
} else
```

```
print("Zero")
```

Switch statement



⌘ Helps programmers in testing a variable against a list of values.

Syntax

```
switch (Expression, "Option 1", "Option 2", "Option 3", ....., "Option N")
```



```
switch (Expression,  
"Option 1" = Execute these statements when the expression result match Option 1,  
"Option 2" = Execute these statements when the expression result match Option 2,  
"Option 3" = When the expression result match Option 3, Execute these statements,  
.... ....  
"Option N" = When the expression result match Option N, Execute these statements,  
Default Statements )
```



```
X <- switch (3, "First", "Second", "Third",  
Fourth")
```

```
Print(x)
```

While loops



- # The While loop in R Programming is used to repeat a block of statements for a given number of times until the specified expression is False.
- # While loop in R starts with the expression, and if the expression is True, then statements inside the while loop will be executed.
- # If the specified expression is false, it won't be executed at least once.
- # It means, R while loop may execute zero or more time.

While loop



```
While ( Expression ) {  
    statement 1  
    statement 2  
    .....  
    statement N;  
    # Increment or Decrement the Values  
}  
#This statement is from Outside the While Loop
```

While loop



```
count<-0
sum<-0
number<-as.integer(readline(prompt="Enter integer
    number:"))
while(count <= number){
  sum <- sum + number
  count <- count + 1
}
print(paste("The total Sum of Numbers From the While
Loop is: ", sum))
```

For loop



- ⌘ The R For Loop is used to repeat a block of statements until there are no items in the Vector.

```
for (val in vector)  {  
    Statement 1  
    Statement 2  
    .....  
    Statement N  
}
```

For loop



```
school <- c("CSA", "CIT", "Civil", "Mech", "commerce")
for (str in school) {
  + print (paste ("Departments are:", str))
+ }
[1] "Departments are: CSA"
[1] "Departments are: CIT"
[1] "Departments are: Civil"
[1] "Departments are: Mech"
[1] "Departments are: commerce"
```

Repeat loop



- ⌘ The R Repeat executes the statements inside the code block multiple number times.
- ⌘ Repeat loop in R programming doesn't provide any condition to check so, we have to give the condition to exit from repeat loop.

Repeat loop



```
repeat {  
    statement 1  
    statement 2  
    .....  
    statement N  
    # Please provide Condition to exit or use Break Statement  
}  
#This statement is from Outside the Repeat Loop
```

Repeat loop



```
u<-35
v<-10
t<-0
repeat {
  + t<- u%%v
  + u<-v
  + v<-t
  + if (v ==0){
  + break
  + }
  + }
print(paste("GCD is:",u))
[1] "GCD is: 5"
```

R – Operators



⌘ An operator is the symbol that tells the compiler to perform specific arithmetic or logical manipulations.

⌘ Types of operators

- ─ Arithmetic Operators

- ─ Relational Operators

- ─ Logical Operators

- ─ Assignment Operators

- ─ Miscellaneous Operators



❖ Arithmetic Operators

❖ Operator : +

❖ Description : Add two vectors

```
A <- c(1,4,6)
```

```
B <- c(13,44,57)
```

```
Print(a + b)
```



❖ Arithmetic Operators

❖ Operator : -

❖ Description : Subtract one vector from another

```
A <- c(1,4,6)
```

```
B <- c(13,44,57)
```

```
Print(b - a)
```



❖ Arithmetic Operators

❖ Operator : *

❖ Description : Multiply two vectors

```
A <- c(1,4,6)
```

```
B <- c(13,44,57)
```

```
Print(a * b)
```



❖ Arithmetic Operators

↗ Operator : /

↗ Description : Division of two vectors

```
A <- c(1,4,6)
```

```
B <- c(13,44,57)
```

```
Print(a / b)
```



❖ Arithmetic Operators

❖ Operator : %%

❖ Description : Give the remainder of the first vector with the second

```
A <- c(1,4,6)
```

```
B <- c(13,44,57)
```

```
Print(a %% b)
```



❖ Arithmetic Operators

❖ Operator : % / %

❖ Description : Result of division of first vector
with second(quotient)

```
A <- c(1,4,6)
```

```
B <- c(13,44,57)
```

```
Print(a%/%b)
```

```
0 0 0
```



❖ Arithmetic Operators

❖ Operator : ^

❖ Description : First vector raised to the exponent of the second vector

```
A <- c(1,4,6)
```

```
B <- c(13,44,57)
```

```
Print(a ^ b)
```

Relational Operators



- ⌘ Each element of the first vector is compared with the corresponding element of the second vector.



❖ Relational Operators

❖ Operator : <

❖ Description : checks if each element of the first vector is less than the corresponding element of the second vector.

```
A <- c(12,34,44)
```

```
B <- c(65,4,16)
```

```
Print(a < b)
```

```
TRUE FALSE FALSE
```



❖ Relational Operators

❖ Operator : >

❖ Description : checks if each element of the first vector is greater than the corresponding element of the second vector.

```
A <- c(12,34,44)
```

```
B <- c(65,4,16)
```

```
Print(a > b)
```

```
FALSE TRUE TRUE
```



❖ Relational Operators

❖ Operator : ==

❖ Description : checks if each element of the first vector is equal to the corresponding element of the second vector.

```
A <- c(12,34,44)
```

```
B <- c(65,4,16)
```

```
Print(a == b)
```

```
FALSE FALSE FALSE
```



❖ Relational Operators

❖ Operator : \leq

❖ Description : checks if each element of the first vector is less than or equal to the corresponding element of the second vector.

A <- c(12,34,44)

B <- c(65,4,16)

Print(a \leq b)

TRUE FALSE FALSE



❖ Relational Operators

❖ Operator : \geq

❖ Description : checks if each element of the first vector is greater than or equal to the corresponding element of the second vector.

```
A <- c(12,34,44)
```

```
B <- c(65,4,16)
```

```
Print(a >= b)
```

```
FALSE  TRUE  TRUE
```



❖ Relational Operators

❖ Operator : !=

❖ Description : checks if each element of the first vector is unequal to the corresponding element of the second vector.

```
A <- c(12,34,44)
```

```
B <- c(65,4,16)
```

```
Print(a != b)
```

```
TRUE TRUE TRUE
```

❖ Logical Operators

❖ Operator : &

❖ Description : Element-wise logical AND operator. It combines each element of the first vector with the corresponding element of the second vector and gives a output TRUE if both the elements are TRUE.

```
A <- c(12,34,TRUE, 2+3i)
```

```
B <- c(65,4,FALSE, 2+3i)
```

```
Print(a & b)
```

```
TRUE  TRUE FALSE TRUE
```

❖ Logical Operators

❖ Operator : |

❖ Description : Element-wise logical OR operator. It combines each element of the first vector with the corresponding element of the second vector and gives a output TRUE if one of the element is TRUE.

```
A <- c(12,34,TRUE, 2+3i)
```

```
B <- c(65,4,FALSE, 2+3i)
```

```
Print(a | b)
```

```
TRUE TRUE TRUE TRUE
```

❖ Logical Operators

❖ Operator : !

❖ Description : Element-wise logical NOT operator. Takes each element of the vector and gives the opposite logical value.

```
A <- c(12,34,TRUE, 2+3i)
```

```
B <- c(65,4,FALSE, 2+3i)
```

```
Print(! b)
```

```
FALSE FALSE TRUE FALSE
```



⌘Miscellaneous operators

↖Operator : :

↖Description : Colon operator. It creates the series of numbers in sequence for a vector.

A <- 3 : 6

Print(A)

3 4 5 6



❖ Miscellaneous operators

❖ Operator : %in%

❖ Description : This operator is used to identify if an element belongs to a vector.

A <- 5

B <- 10

T <- 1:5

Print(A %in% T)

TRUE

Print(B %in% T)

FALSE

R - Functions



- ⌘ A function is a set of statements organized together to perform a specific task.
- ⌘ R has a large number of built-in functions and the user can create their own functions.
- ⌘ In R, a function is an object so the R interpreter is able to pass control to the function, along with arguments that may be necessary for the function to accomplish the actions.
- ⌘ The function in turn performs its task and returns control to the interpreter as well as any result which may be stored in other objects.

R - Functions



❖ Function Definition

❖ R function is created by using the keyword function.

❖ Syntax

```
function_name <- function (arg_1, arg_2,....)  
{  
  Function body  
}
```

Components of a function



❖ Function_name

- ❖ Actual name of the function. It is stored in R environment as an object with this name.

❖ Arguments

- ❖ An argument is a place holder. When a function is invoked, value has to be passed to an argument. Arguments are optional; i.e. a function may contain no arguments. Also arguments may have default values.

❖ Function body

- ❖ Contains a collection of statements that defines what the function does.

❖ Return value

- ❖ Last expression in the function body to be evaluated.

Built-in functions



❖ Built-in functions

- seq(), mean(), max(), sum(x) and paste(...), etc.
- Directly called by the user written programs.

Create a sequence of numbers from 1 to 10

```
print(seq(1:10))
```

Calculate mean of a number from 24 to 145

```
print(mean(24:145))
```

Calculate sum of numbers from 25 to 50

```
print(sum(25:50))
```

User Defined functions



❖ User Defined functions

❖ Specific to what a user wants and once created, they can be used like the built-in functions.

Function - example



```
# Function to find the square of a number
square.it <- function(x)
{
  square <- x * x
  return(square)
}
# call function to square a number
square.it (12)
```

Calling a function without an argument

```
#function without an argument
sqr_series <- function()
{
  for(i in 1:4)
  {
    print(i*i)
  }
}
# call function
sqr_series()
```

Calling a function with argument values (by position and by name)



⌘ The arguments to a function call can be supplied in the same sequence as defined in the function or they can be supplied in a different sequence but assigned to the names of the arguments.

Calling a function with argument values (by position and by name)

```
# create function with arguments
f1 <- function (x,y,z)
{
eq <- x/y+z
Print(eq)
}
#call by position of arguments
f1(34,2,10)
[1] 27
#call by name of arguments
f1(x=50, y=2, z=1)
[1] 26
```

Calling a function with default argument



```
# create a function with arguments
new.function <- function (p=14, q=7)
{
  result <- p * q
  print(result)
}
#call the function without giving any argument
new.function()
```

```
#call the function with giving new values of the argument
new.function(5,7)
```

Lazy evaluation of function



- ⌘ Arguments to functions are evaluated lazily, which means so they are evaluated only when needed by the function body.
- ⌘ Lazy evaluation is an evaluation strategy which holds the evaluation of an expression until its value is needed

Lazy evaluation of function



```
# demo for lazy evaluation of function
```

```
f2 <- function(x,y)
```

```
{
```

```
print(x*x)
```

```
print(y)
```

```
print(x)
```

```
}
```

```
f2(3)
```

```
[1] 9
```

```
Error in print(y) : argument "y" is missing, with no default
```

R - Strings



- ⌘ Any value written within a pair of single quote or double quotes in R is treated as a string.
- ⌘ Internally R stores every string within double quotes, even when you create them with a single quote.

Rules applied in string construction



- ⌘ The quotes at the beginning and the end of the string should be both double quotes or single quote. They can't be mixed.
- ⌘ Double quotes can be inserted into a string starting and ending with single quotes.
- ⌘ Single quotes can be inserted into a string starting and ending with double quotes.
- ⌘ Double quotes can not be inserted into a string starting and ending with double quotes.
- ⌘ Single quote can not be inserted into a string starting and ending with single quote.



⌘ Examples of valid strings

a <- 'start and end with single quote'

b <- "start and end with double quotes"

c <- "single quote ' in between double quotes"

d<- ' Double quotes " in between single quote'



Output:

“start and end with single quote”

“start and end with double quotes”

“single quote ‘ in between double quotes”

“ Double quotes \" in between single quote”



⌘ Examples of invalid strings

e<-'Mixed quotes"

f<- 'single quote ` inside single quote'

g <- "double quotes " inside double quotes"

Concatenating Strings – **paste()** function



- ❖ Many strings in R are combined using the **paste()** function.

Syntax:

paste(...., sep = "", collapse = NULL)

where

.... – any number of arguments to be combined.

sep – any separator between arguments (optional)

collapse – used to eliminate space in between two strings. But not the space within two words of one string.

Concatenating Strings – paste() function



```
X<- " WELCOME"
```

```
Y <- "R Programming"
```

```
Z <- "Chapter 1"
```

Try :

```
print(paste(X,Y,Z))
```

```
print(paste(X,Y,Z, sep="-----"))
```

```
print(paste(X,Y,Z, sep="-----", collapse=""))
```

```
print(paste(X,Y,Z, sep="", collapse =""))
```

Formatting numbers and strings

⌘ Numbers and strings can be formatted to a specific style using `format()` function.

Syntax

```
format(x, digits, nsmall, scientific, width, justify = c("left",  
"right", "centre", "none"))
```

Where

`x` – vector input

`digits` – total number of digits displayed

`nsmall` – minimum number of digits to the right of the decimal point.

`scientific` – set to TRUE to display scientific notation

`width` – indicates the minimum width to be displayed by padding
blanks in the beginning

`justify` – display of the string to left, right or centre

Formatting numbers and strings



```
#number padding with blank spaces at the  
beginning
```

```
A<- format(23.5, width=5)
```

```
Print(A)
```

```
" 23.5"
```

```
#justify string with center
```

```
A<- format("R-Book", width = 8, justify =  
"centre")
```

Formatting numbers and strings



```
#justify string with left
```

```
A<- format("R-Book", width = 10, justify =  
"left")
```

```
# format everything as string
```

```
A<- format(13)
```

Counting characters of a string – nchar() function

❖ nchar() function counts the number of characters in a string including spaces.

Syntax

nchar(x)

Where

x – vector input

Result <- nchar("count the number of characters")

Print(result)

```
C:/Users/Senthil/Desktop/r-intro1/ ↗
> result<- nchar("Count the number of characters")
> print(result)
[1] 30
> |
```

Changing the case – toupper() and tolower() functions



⌘ These functions change the case of characters of a string.

⌘ Syntax

`toupper(x)`

`tolower(x)`

```
> str<-toupper("R-Programming")
> print(str)
[1] "R-PROGRAMMING"
>
> str<- tolower("R-PROGRAMMING")
> print(str)
[1] "r-programming"
```

Where

x – vector input

Extracting parts of a string

– **substring()** function

⌘ **substring()** function extracts parts of a string.

⌘ **Syntax**

`substring(x,first,last)`

Where

x-character vector input

first – position of first character to be extracted

last – position of last character to be extracted.

```
> str<- substring(" R-Programming",1,9)
> print(str)
[1] " R-Progra"
> |
```

R – Data Frames



⌘ A Data Frame is a table or a 2-D array like structure in which each column contains values of one variable and each row contains one set of values from each column.

Data Frame



❖ Characteristics of Data Frame

- ❖ Column names should be non-empty
- ❖ Row names should be unique
- ❖ Data stored in a data frame can be of numeric, factor or character type.
- ❖ Each column should contain same number of data items.

Create Data Frame

```
> stud<- data.frame(  
+ sno = c(101,102,103,104,105),  
+ name = c("A", "B", "C", "D", "E"),  
+ marks = c(77,90,85,80,76),  
+ age = c(22,20,18,20,21));  
> print(stud)  
   sno name marks age  
1 101    A     77  22  
2 102    B     90  20  
3 103    C     85  18  
4 104    D     80  20  
5 105    E     76  21
```

Get the structure of the data frame



❖ Structure of the data frame can be seen by using str() function.

```
> str(stud)
'data.frame': 5 obs. of 4 variables:
 $ sno : num 101 102 103 104 105
 $ name: Factor w/ 5 levels "A","B","C","D",...: 1 2 3 4 5
 $ marks: num 77 90 85 80 76
 $ age : num 22 20 18 20 21
```

Summary of Data in Data Frame



⌘ Statistical summary and nature of the data can be obtained by applying `summary()` function.

```
> print(summary(stud))
```

sno	name	marks	age
Min. :101	A:1	Min. :76.0	Min. :18.0
1st Qu.:102	B:1	1st Qu.:77.0	1st Qu.:20.0
Median :103	C:1	Median :80.0	Median :20.0
Mean :103	D:1	Mean :81.6	Mean :20.2
3rd Qu.:104	E:1	3rd Qu.:85.0	3rd Qu.:21.0
Max. :105		Max. :90.0	Max. :22.0
.			

Extract data from data frame



⌘ Extract specific column from a data frame using column name.

```
> print(stud)
  sno name marks age
1 101   A    77  22
2 102   B    90  20
3 103   C    85  18
4 104   D    80  20
5 105   E    76  21
> a<-data.frame(stud$name, stud$age)
> print(a)
  stud.name stud.age
1           A      22
2           B      20
3           C      18
4           D      20
5           E      21
```

Extract data from data frame



⌘ Extract first three rows and all columns

```
> a <- stud[1:3,]
> print(a)
  sno name marks age
1 101    A     77  22
2 102    B     90  20
3 103    C     85  18
```

Extract data from data frame



⌘ Extract 3rd and 4th row with 2nd and 3rd column

```
> a<-stud[c(3,4), c(2,3)]  
> print(a)  
    name marks  
3     C     85  
4     D     80
```

Expand Data Frame



- ⌘ Data frame can be expanded by adding rows and columns.
- ⌘ To add column
 - ↗ Just add the column vector using new column name.

Expand Data Frame



```
> print(stud)
  sno name marks age
1 101   A     77  22
2 102   B     90  20
3 103   C     85  18
4 104   D     80  20
5 105   E     76  21
> stud$dept <- c("CSA", "CSE", "Civil", "EEE", "ECE")
> print(stud)
  sno name marks age dept
1 101   A     77  22  CSA
2 102   B     90  20  CSE
3 103   C     85  18 Civil
4 104   D     80  20  EEE
5 105   E     76  21  ECE
```

Expand Data Frame



⌘ To add new rows:

↗ To add more rows permanently to an existing data frame, put the new rows in the same structure as existing data frame and use the rbind() function.

Expand Data Frame

```
> print(stud)
  sno name marks age dept
1 101   A     77  22  CSA
2 102   B     90  20  CSE
3 103   C     85  18 Civil
4 104   D     80  20   EEE
5 105   E     76  21  ECE

> stud.new <- data.frame(
+ sno = c(106,107),
+ name = c('F','G'),
+ marks = c(90,65),
+ age = c(18,23),
+ dept = c("CSA","Arch"));
> stud_final <- rbind(stud,stud.new)
> print(stud_final)
  sno name marks age dept
1 101   A     77  22  CSA
2 102   B     90  20  CSE
3 103   C     85  18 Civil
4 104   D     80  20   EEE
5 105   E     76  21  ECE
6 106   F     90  18  CSA
7 107   G     65  23  Arch
```

R- Working with CSV files



- ⌘ In R, we can read data from files stored outside the R environment.
- ⌘ We can also write data into files which will be stored and accessed by the Operating system.
- ⌘ R can read and write into various file formats like .csv, .xls, .xml, etc



⌘ To read data from a .csv (comma separated value) and then write data into a .csv file, the file should be present in the current working directory so that R can read it.



⌘`getwd()` – to get your current working directory.

⌘`setwd()` – to set your new working directory.

Input as CSV file



- # CSV is a text file in which the values in the columns are separated by a comma.
- # Create the file by using windows notepad.
- # Save this file as input.csv using the Save As – All Files (*.*) option in notepad.

Reading a CSV file



- ⌘ To read a .csv file available in your current working directory use `read.csv()` function.

```
> getwd()
[1] "C:/Users/Senthil/Desktop/r-intro1"
> d<-read.csv("input.csv")
> print(d)
   Rno Age     Name    Dept
1  101  26   Saloni      IT
2  102  25   Idhant      HR
3  103  24 Reyanshh Finance
4  104  24   Tarush      IT
5  105  25 Madhurya      HR
```

Analyzing the .CSV file



⌘ By default, the `read.csv()` function gives the output as a data frame.

⌘ To check whether it is stored as a data frame

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

⌘ To check the number of columns and number of rows

```
> print(ncol(d))  
[1] 4  
> print(nrow(d))  
[1] 5
```

Analyzing the .CSV file



⌘ Get the maximum age:

```
> print(data)
  Rno Age      Name     Dept
1 101 26    Saloni      IT
2 102 25   Idhant      HR
3 103 24 Reyanshh Finance
4 104 24   Tarush      IT
5 105 25 Madhurya      HR
> max_age<-max(data$Age)
> print(max_age)
[1] 26
```

Analyzing the .CSV file

- ⌘ Get the details of the student with the maximum age:
- ⌘ We can fetch rows meeting specific filter criteria similar to a SQL where clause.

```
> print(data)
   Rno Age     Name    Dept
1 101  26 Saloni      IT
2 102  25 Idhant      HR
3 103  24 Reyanshh Finance
4 104  24 Tarush      IT
5 105  25 Madhurya    HR
```

Method - II

```
> info<-subset(data, Age == max(data$Age))
> print(info)
   Rno Age     Name    Dept
1 101  26 Saloni      IT
```

Method - I

```
> max_age <- max(data$Age)
> info<- subset(data, Age == max_age)
> print(info)
   Rno Age     Name    Dept
1 101  26 Saloni      IT
```

Analyzing the .CSV file

```
> print(data)
  Rno Age     Name   Dept
1 101  26 Saloni     IT
2 102  25 Idhant      HR
3 103  24 Reyanshh Finance
4 104  24 Tarush      IT
5 105  25 Madhurya    HR
```

- # Get all the students in the HR department.
- # Get students in IT department whose age is greater than 25.

Analyzing the .CSV file

```
> print(data)
   Rno Age     Name    Dept
1 101  26 Saloni      IT
2 102  25 Idhant      HR
3 103  24 Reyanshh Finance
4 104  24 Tarush      IT
5 105  25 Madhurya    HR

> info_dept <- subset(data, Dept == "HR")
> print(info_dept)
   Rno Age     Name    Dept
2 102  25 Idhant      HR
5 105  25 Madhurya    HR
```

⌘ Get all the students in the HR department

Analyzing the .CSV file



- # Get students in IT department whose age is greater than 25.

```
> info_age_dept <- subset(data, Age > 25 & Dept == "IT")
> print(info_age_dept)
   Rno  Age    Name Dept
1 101  26 Saloni   IT
```

Writing into a CSV file



- ⌘ R can create .csv file from existing data frame.
- ⌘ The write.csv() function is used to create the .csv file.
- ⌘ This file gets created in the working directory.

Writing into a CSV file



```
> #write filtered data into a new file  
> write.csv(info_dept, "INFO_HR_DEPT")  
  
> new_info <- read.csv("INFO_HR_DEPT")  
> print(new_info)  
   X Rno Age      Name Dept  
 1 2 102  25  Idhant    HR  
 2 5 105  25 Madhurya    HR
```

Here the column X comes from the data set. This can be dropped.

Writing into a CSV file



```
> write.csv(info_dept,"INFO_HR_DEPT", row.names = FALSE)
> new_info<-read.csv("INFO_HR_DEPT")
> print(new_info)
  Rno Age      Name Dept
1 102  25    Idhant   HR
2 105  25  Madhurya   HR
```

R-Excel file



- ⌘ Microsoft excel is the most widely used spreadsheet program which stores data in .xls or .xlsx format.
- ⌘ R can read directly from these files using some excel specific packages.
- ⌘ Few such packages are XLConnect, xlsx, gdata, etc.
- ⌘ We will be using xlsx package.

Install .xlsx package



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

```
> install.packages("xlsx")
also installing the dependencies 'rJava', 'xlsxjars'

trying URL 'http://cran.rstudio.com/bin/windows/contrib/3.6/rJava_0.9-11.zip'
Content type 'application/zip' length 832080 bytes (812 KB)
downloaded 812 KB

trying URL 'http://cran.rstudio.com/bin/windows/contrib/3.6/xlsxjars_0.6.1.zip'
Content type 'application/zip' length 9485571 bytes (9.0 MB)
downloaded 9.0 MB

trying URL 'http://cran.rstudio.com/bin/windows/contrib/3.6/xlsx_0.6.3.zip'
Content type 'application/zip' length 379601 bytes (370 KB)
downloaded 370 KB

package 'rJava' successfully unpacked and MD5 sums checked
package 'xlsxjars' successfully unpacked and MD5 sums checked
package 'xlsx' successfully unpacked and MD5 sums checked
```

⌘ Load the library file

```
> library("openxlsx")
Warning message:
package 'openxlsx' was built under R version 3.6.3
```

⌘ Create an Excel file with two sheets

Rno	Age	Name	Dept	Name	City
1	26	Aarya	IT	Aarya	Kochi
2	25	Siddharth	HR	Siddharth	Chennai
3	24	Tarush	Finance	Tarush	Mumbai
4	24	Diva	IT	Diva	Delhi
5	25	Richa	HR	Richa	Jaipur

Sheet Name : Employee

Sheet Name : City

⌘ Save the excel file as “input.xlsx”.

Reading the Excel file

- ⌘ Use `read.xlsx()` function to read the excel file.
- ⌘ To read the `input.xlsx` file and first sheet use the following command:

```
> data <- read.xlsx("input.xlsx",1)
```

```
> print(data)
   Rno Age     Name Dept
1   1  26    Aarya   IT
2   2  25 Siddharth  HR
3   3  24   Tarush Finance
4   4  24     Diva    IT
5   5  25   Richa    HR
```

R Charts and Graphs – Pie Charts



- # R programming language has numerous libraries to create charts and graphs.
- # A Pie chart is a representation of values as slices of a circle with different colors.
- # The slices are labeled and the numbers corresponding to each slice is also represented in the chart.
- # Here, Pie chart is created using the **pie()** function.

R Charts and Graphs – Pie Charts



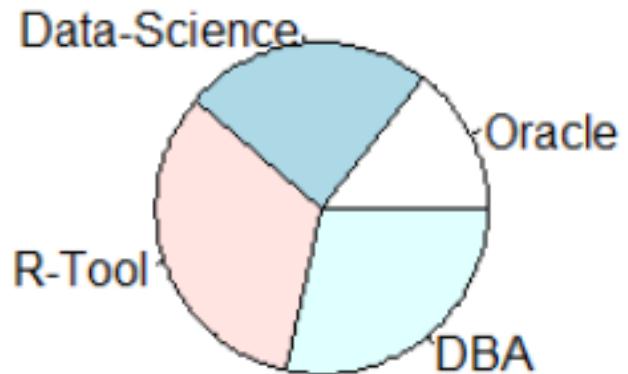
❖ Syntax

pie(x, labels, radius, main, col, clockwise)

Where

- ❖ **x** is a vector containing the numeric values used in the pie chart.
- ❖ **labels** is used to give description to the slices.
- ❖ **radius** indicates the radius of the circle of the pie chart.(value between -1 and +1).
- ❖ **main** indicates the title of the chart.
- ❖ **col** indicates the color palette.
- ❖ **clockwise** is a logical value indicating if the slices are drawn clockwise or anti clockwise.

```
> x<-c(34,56,76,66)
> y<-c("Oracle", "Data Science", "R-Tool", "DBA")
> # Assign name to a file
> png(file="course1.jpg")
> #Plot the pie chart
> pie(x,labels=y)
```



png - This package provides an easy and simple way to read, write and display bitmap images stored in the PNG format

To change the border color

```
> x<-c(34,56,76,66)
> y<-c("oracle", "data science", "R-tool", "DBA")
> pie(x,labels=y,border="red")
```

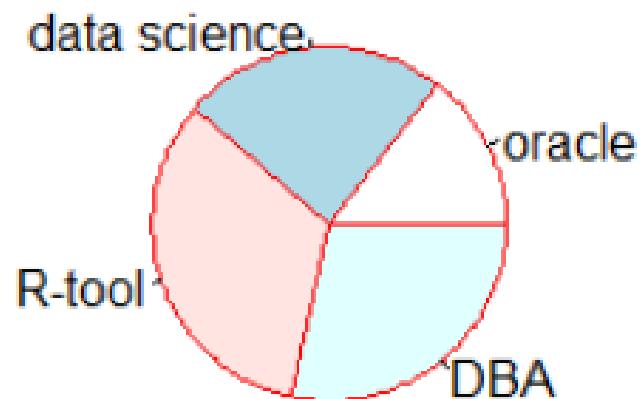


Chart title and colors

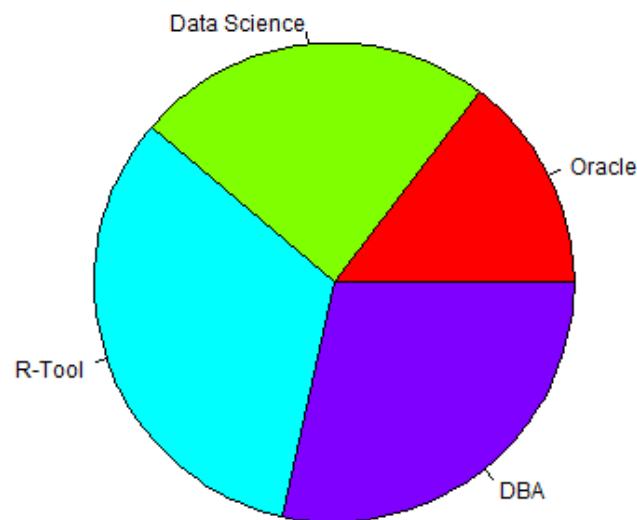


- ⌘ **main** parameter is used to add a title to the chart and **col** which makes use of rainbow color pallet while drawing the chart.
- ⌘ Length of pallet should be same as the number of values we have for the chart.

```
> #Create Data for Pie Chart  
> x<- c(34,56,76,66)  
> y <- c("Oracle", "Data Science", "R-Tool", "DBA")  
> #Give name to a file  
> png(file="Course_title.jpg")  
> #Plot the pie chart  
> pie(x,y,main="Course_Piechart", col=rainbow(length(x)))  
> #To save the file  
> dev.off()
```

dev.off() - without it you'll get a partial plot or nothing at all

Course_Piechart



Slice Percentages and Chart legend

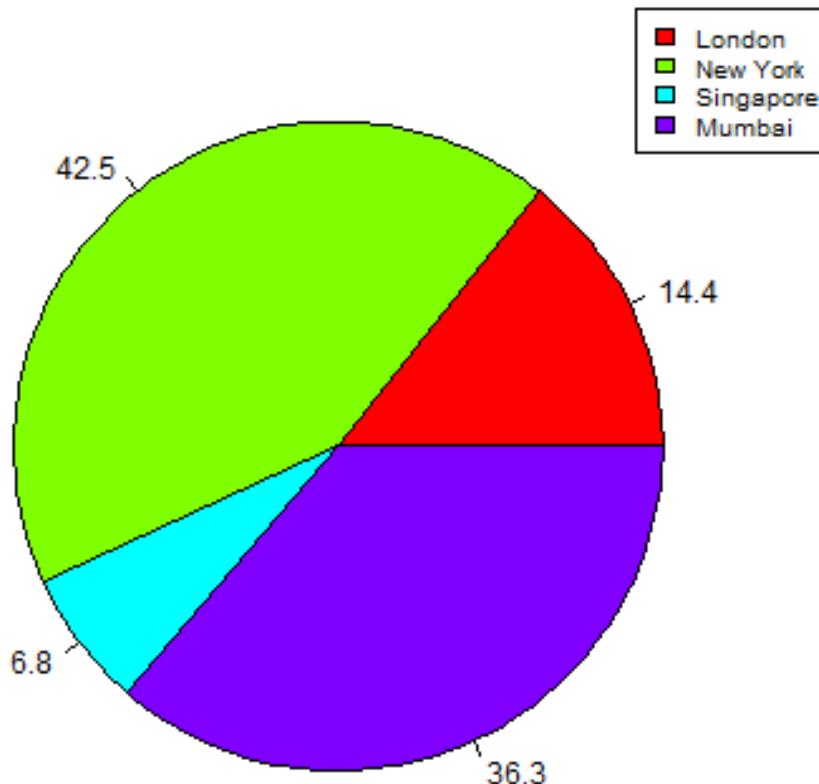


```
> # Create data for the graph
> x <- c(21, 62, 10,53)
> labels <- c("London", "New York", "Singapore", "Mumbai")
> piepercent<- round(100*x/sum(x), 1)
> # Give the chart file a name
> png(file = "city_percentage_legends.jpg")
> # Plot the chart
> pie(x, labels = piepercent, main = "City pie chart", col = rainbow(length(x)))
> legend("topright", c("London", "New York", "Singapore", "Mumbai"), cex = 0.8,
+        fill = rainbow(length(x)))
> # Save the file
> dev.off()
```

Cex - number indicating the amount by which plotting text and symbols should be scaled relative to the default

Legends

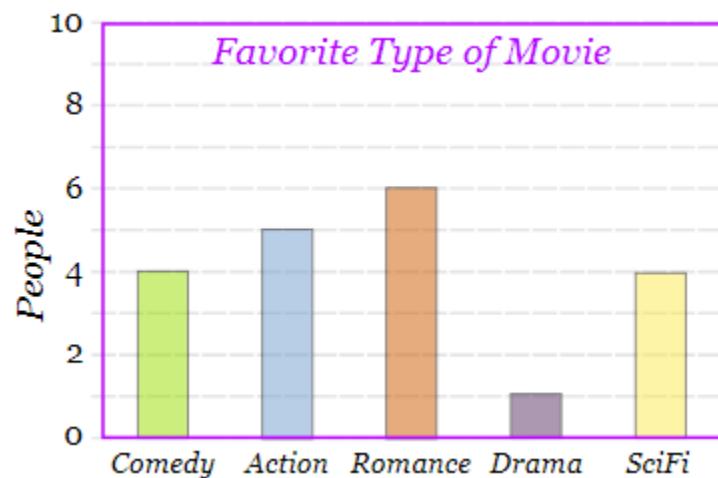
City pie chart



BarCharts

⌘ A **Bar Charts** (also called Bar Graph) is a graphical display of data using bars of different heights.

Table: Favorite Type of Movie				
Comedy	Action	Romance	Drama	SciFi
4	5	6	1	4



Bar Charts



- # The Barplot or Bar Chart in R Programming is handy to compare the data visually.
- # By seeing this R barplot or bar chart, one can understand, which product is performing better compared to others.
- # For example, if we want to compare the sales between different product categories, product color, we can use this R bar chart.

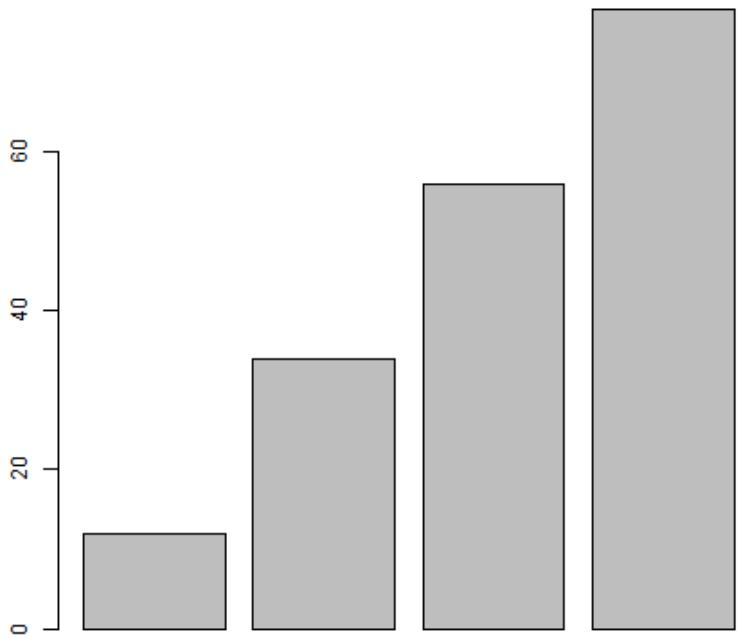
⌘ Syntax

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

Following is the description of the parameters used –

- **H** is a vector or matrix containing numeric values used in bar chart.
- **xlab** is the label for x axis.
- **ylab** is the label for y axis.
- **main** is the title of the bar chart.
- **names.arg** is a vector of names appearing under each bar.
- **col** is used to give colors to the bars in the graph.

```
> # Create the data for Bar Chart  
> h <- c(12,34,56,78)  
>  
> # Give the chart file a name  
> png(file="barchart.jpg")  
>  
> # Plot the bar chart  
> barplot(h)  
>  
> # Save the file  
> dev.off()  
null device
```



Bar Chart Labels, Titles and Colors



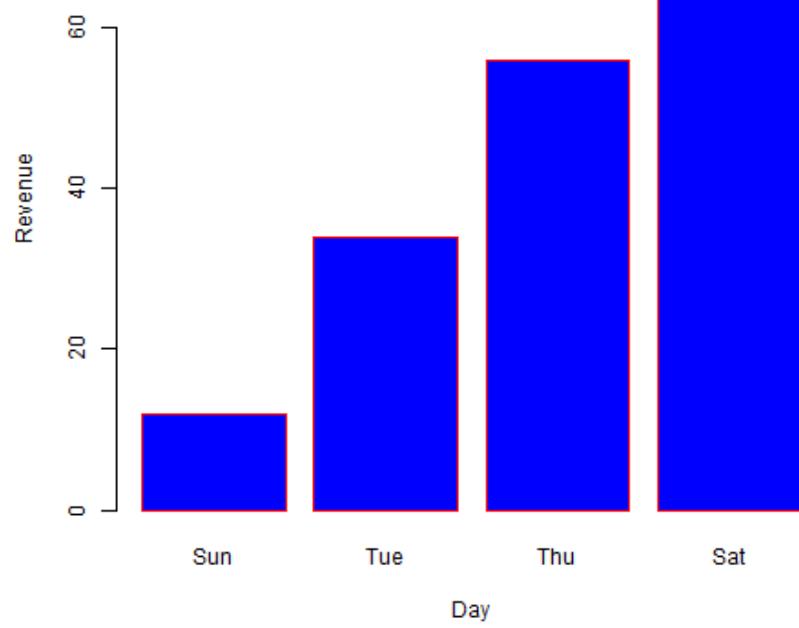
- ⌘ The main parameter is used to add title.
- ⌘ The col parameter is used to add colors to the bars.
- ⌘ The args.name is a vector having same number of values as the input vector to describe the meaning of each bar.

Bar Chart Labels, Titles and Colors



```
> #Create Data for Bar Chart
> h <- c(12,34,56,78)
> #Assign labels
> m <- c("Sun", "Tue", "Thu", "Sat")
> # Give the chart a name
> png(file="barchart_day.jpg")
> #Plot the chart
> barplot(h, names.arg=m, xlab="Day", ylab="Revenue", col="blue", main="Day Chart", border="Red")
> #To save the file
> dev.off()
null device
```

Day Chart



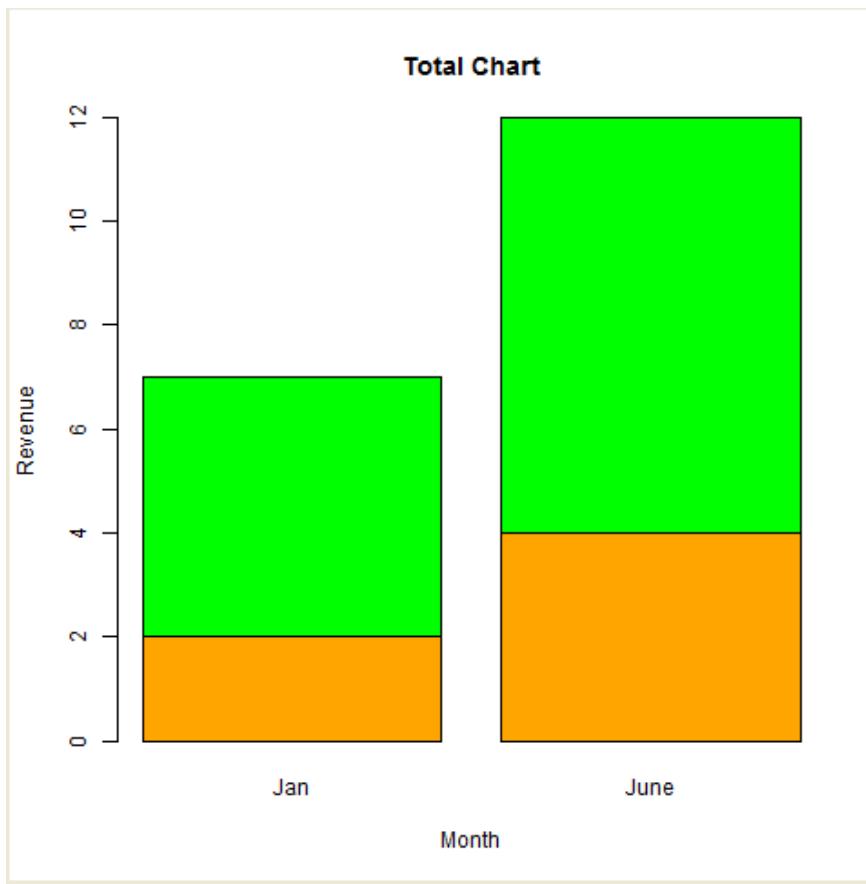
Group Bar Chart and Stacked Bar Chart



- ⌘ We can create bar chart with group of bars and stacks in each bar by using a matrix as input values.
- ⌘ More than two variables are represented as a matrix which can be used to create group bar chart or stacked bar chart.

```
> # Create the input vectors  
> f <- c("orange", "green")  
> r <- c("South", "East")  
> m <- c("Jan", "June")  
> #Create matrix of values  
> values <- matrix(c(2,4,5,8), nrow=2, ncol=2, byrow=TRUE)  
> #To assign a name to the chart  
> png(file = "barchart_stacked.jpg")  
> #To create Bar chart  
> barplot(values, main = "Total Chart", names.arg = m, xlab="Month", ylab = "  
Revenue", col=f)  
> #To save the file  
> dev.off()  
null device  
1
```

	Jan	June
South	2	4
East	5	8



Summarizing data



⌘ Measures of location

- ◻ Mean is a measure of location.
- ◻ Mode
 - ◻ most common value in X
- ◻ Median
 - ◻ Value that has an equal number of data points above and below it.
 - ◻ Easy if 'n' is an odd number.
 - ◻ When 'n' is even – defined as halfway between the two middle values.
- ◻ Quartiles
 - ◻ The **median** divides the data into a lower half and an upper half.
 - ◻ The **lower quartile** is the middle value of the lower half.
 - ◻ The **upper quartile** is the middle value of the upper half.

Summarizing data

⌘ **Example :**

⌘ Find the median, lower quartile and upper quartile of the following numbers.

☒ 12, 5, 22, 30, 7, 36, 14, 42, 15, 53, 25

⌘ **Solution:**

⌘ First, arrange the data in ascending order:

5, 7, 12, 14, 15, 22, 25, 30, 36, 42, 53
 ↑ ↑ ↑
 lower quartile median upper quartile

☒ Median (middle value) = 22

☒ Lower quartile (middle value of the lower half) = 12

☒ Upper quartile (middle value of the upper half) = 36

⌘ If there is an even number of data items, then we need to get the average of the middle numbers:

Summarizing data

Example:

- Find the median, lower quartile, upper quartile, interquartile range and range of the following numbers.
- 12, 5, 22, 30, 7, 36, 14, 42, 15, 53, 25, 65

Solution:

- First, arrange the data in ascending order:

5, 7, 12, 14, 15, 22, 25, 30, 36, 42, 53, 65
lower quartile or first quartile median or second quartile upper quartile or third quartile

- Lower quartile or first quartile** = $\frac{12+14}{2} = 13$

- Median or second quartile** = $\frac{22+25}{2} = 23.5$

- Upper quartile or third quartile** = $\frac{36+42}{2} = 39$

Interquartile range

- Upper quartile – lower quartile
- $39 - 13 = 26$

Range

- largest value – smallest value
- $65 - 5 = 60$

- When evaluating the quartiles, always remember to first arrange the data in increasing order.

Boxplots



⌘ Boxplots

✉ Popular way of visualizing distribution.

✉ *Five number summary*

✉ *Q_1 , Median and Q_3 together contain no information about the endpoints of the data, a fuller summary of the shape of the distribution can be obtained by providing the lowest and highest values as well.*

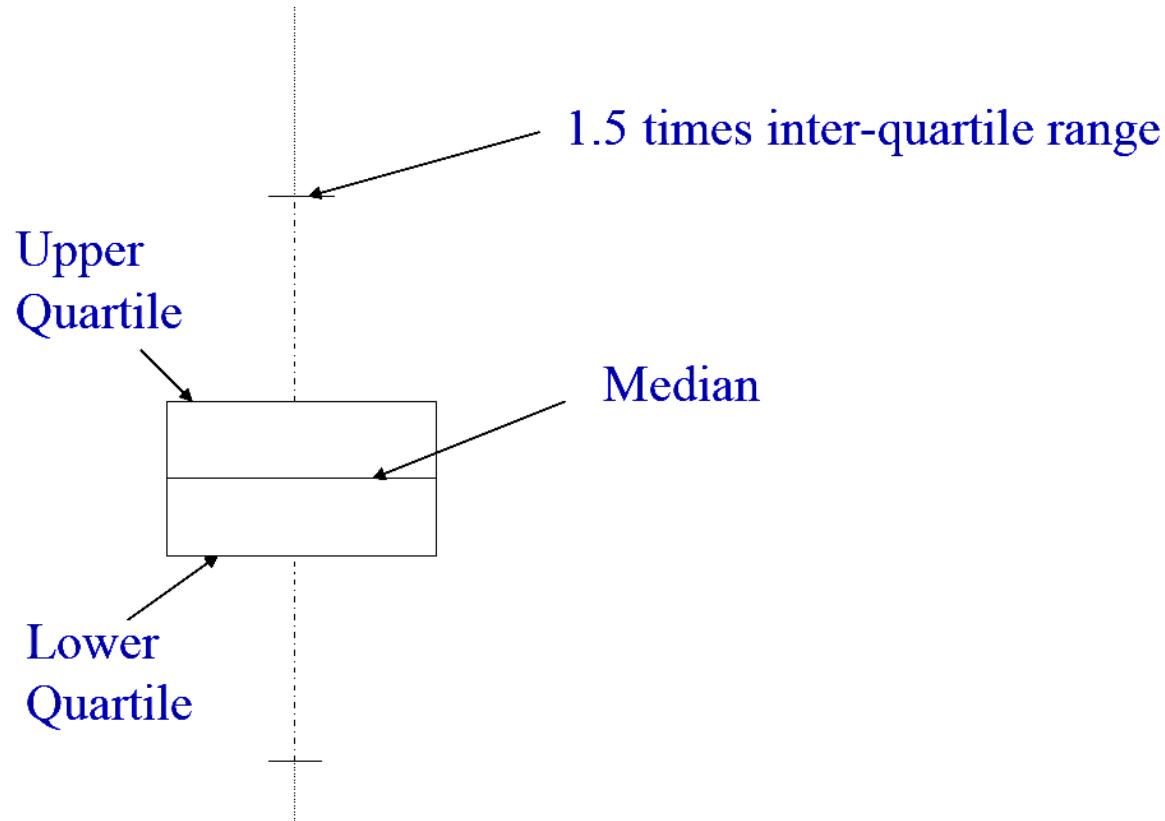
✉ *Consists of the median, the quartiles Q_1 and Q_3 , and the smallest and the largest observations written in the order Minimum, Q_1 , Median, Q_3 , Maximum.*

Tools for displaying relationships between two variables



- ⌘ Incorporates the *five number summary*.
 - ↗ Ends of the box are at the quartiles, so that the box length is the Interquartile Range (IQR).
 - ↗ Median is marked by a line within the box.
 - ↗ Two lines (called whiskers) outside the box extend to the smallest (Minimum) and largest (Maximum) observations.

BoxPlots





⌘ The owner of a Restaurant wants to find out more about where his patrons are coming from. One day he decided to gather data about the distance (in miles) that people commuted to get to his restaurant. People reported the following distances travelled.

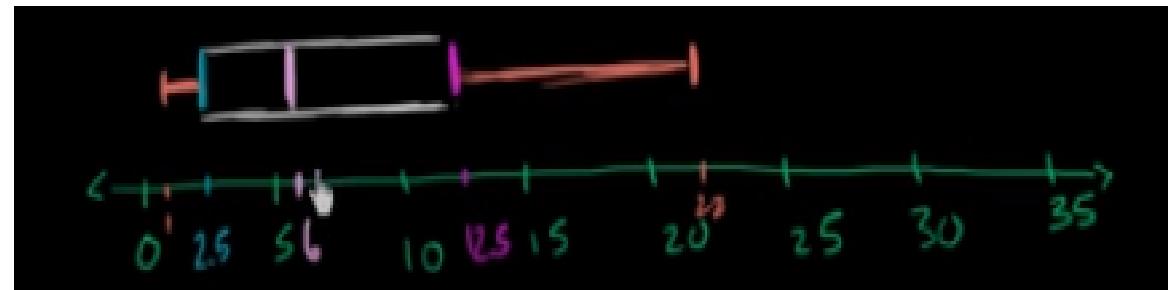
↗ 14, 6, 3, 2, 4, 15, 11, 8, 1, 7, 2, 1, 3, 4, 10, 22, 20

⌘ He wants to create a graph that helps him understand the **spread of distances** (and the **median distance**) that people travel.

⌘ Before calculating Median, the list should be sorted.

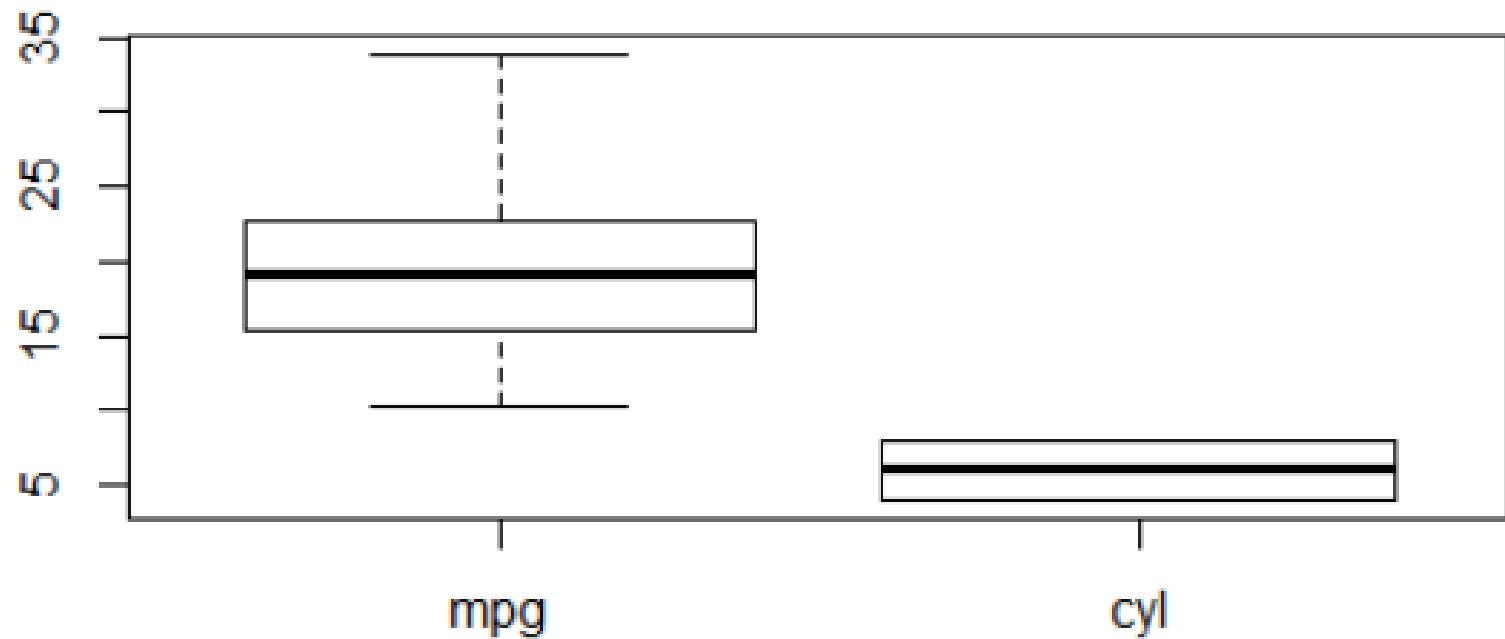
1,1,2,2,3,3,4,4,6,7,8,10,11,14,15,20,22

create?
2.5
median
12.5
1,1,2,2,3,3,4,4,6,7,8,10,11,14,15,20,22

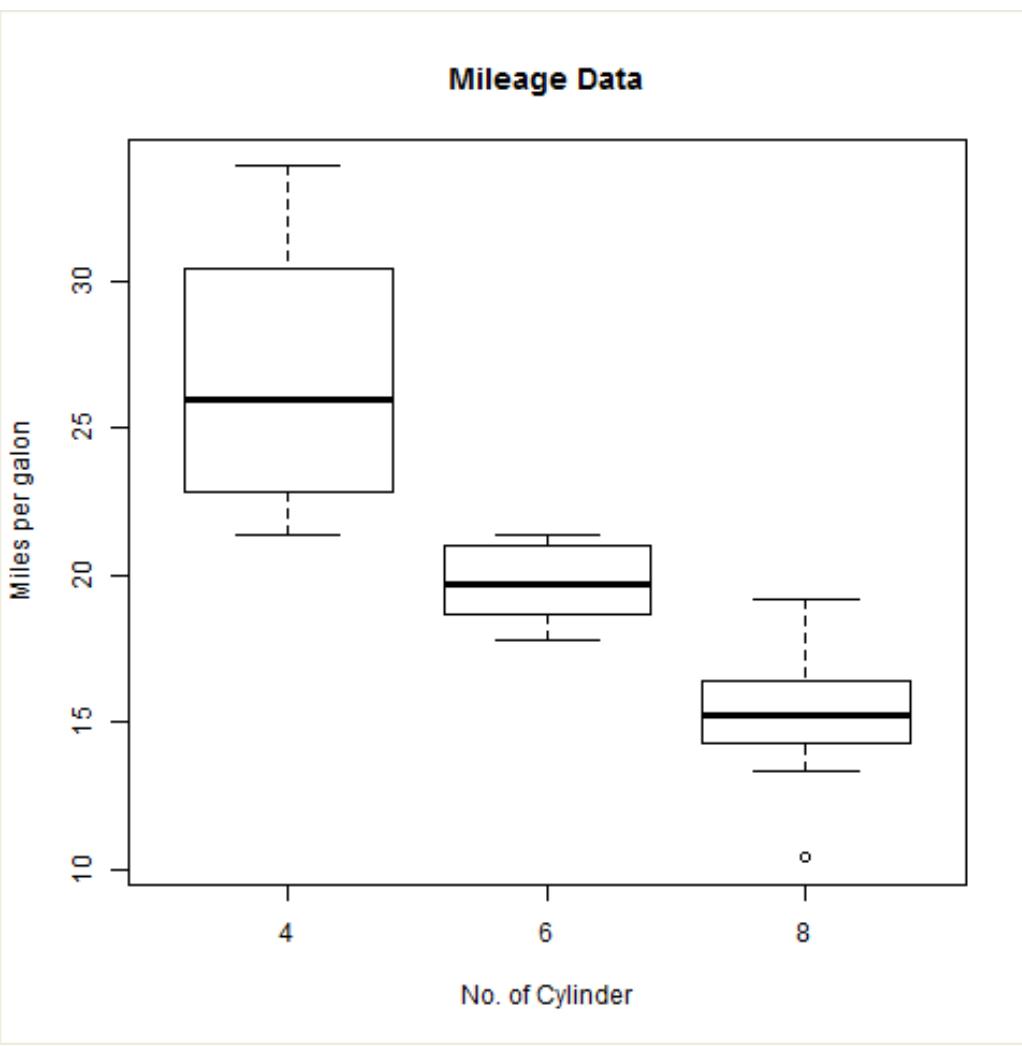


```
> head(mtcars)
      mpg cyl disp hp drat    wt  qsec vs am gear carb
Mazda RX4     21.0   6 160 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag 21.0   6 160 110 3.90 2.875 17.02  0  1    4    4
Datsun 710    22.8   4 108  93 3.85 2.320 18.61  1  1    4    1
Hornet 4 Drive 21.4   6 258 110 3.08 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
> ip <- mtcars[,c('mpg','cyl')]
> print(head(ip))
      mpg cyl
Mazda RX4     21.0   6
Mazda RX4 Wag 21.0   6
Datsun 710    22.8   4
Hornet 4 Drive 21.4   6
Hornet Sportabout 18.7   8
Valiant       18.1   6
> summary(ip)
      mpg                      cyl
Min. :10.40                Min. :4.000
1st Qu.:15.43               1st Qu.:4.000
Median :19.20                Median :6.000
Mean   :20.09                Mean   :6.188
3rd Qu.:22.80               3rd Qu.:8.000
Max.   :33.90                Max.   :8.000
```

boxplot(ip)



```
> #Assign a name to the chart  
> png(file="boxplot.jpg")  
> boxplot(mpg~cyl, data=mtcars, xlab = "No. of Cylinder", ylab="Miles per gallon", main="Mileage Data")  
> dev.off()
```



Histograms



- # A histogram represents the frequencies of values of a variable bucketed into ranges.
- # Histogram is similar to bar chart but the difference is it groups the values into continuous ranges.
- # Each bar in histogram represents the height of the number of values present in that range.
- # R creates histogram using **hist()** function.
 - # This function takes a vector as an input and uses some more parameters to plot histograms.



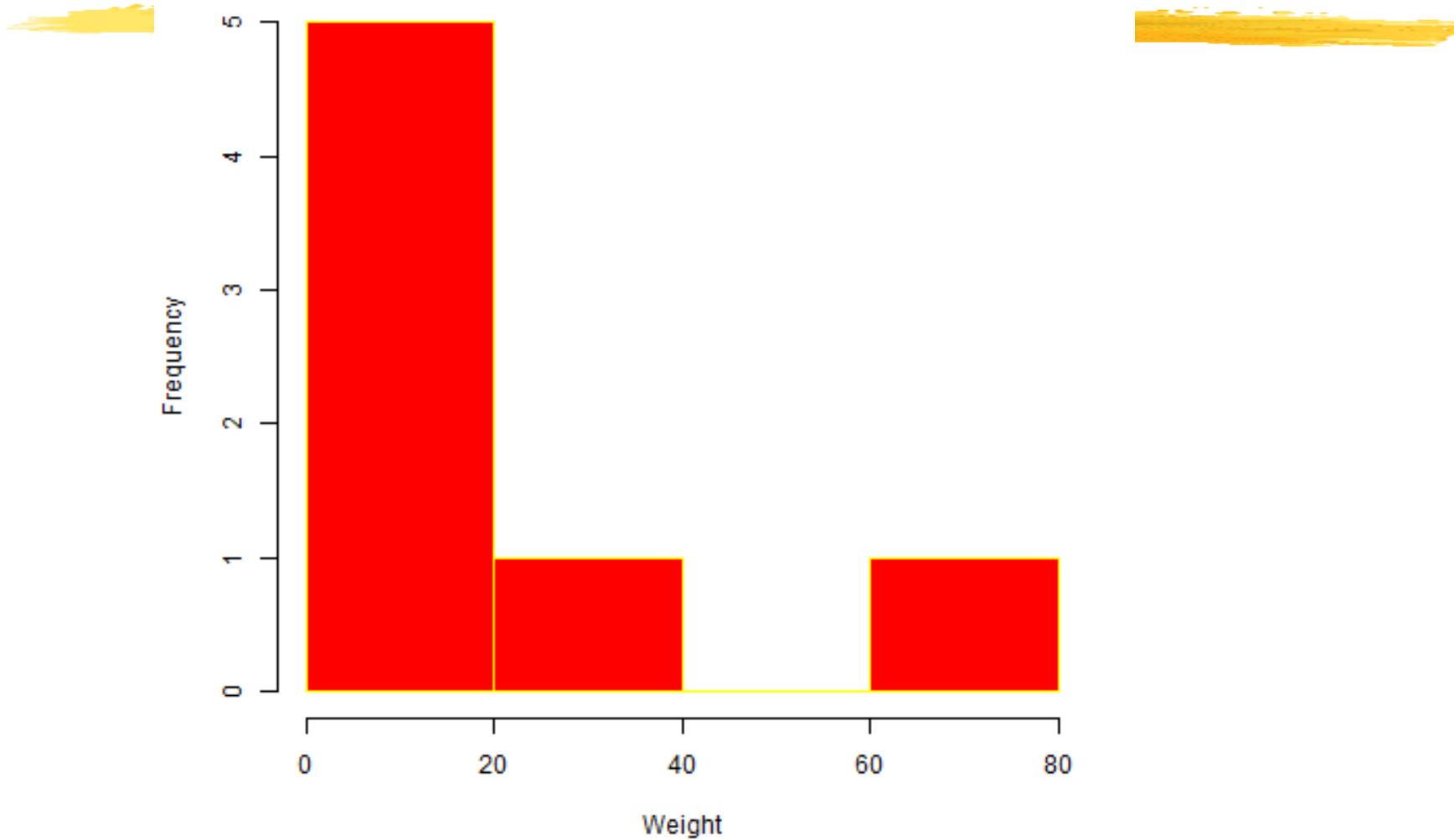
```
hist(v,main,xlab,xlim,ylim,breaks,col,border)
```

Following is the description of the parameters used –

- **v** is a vector containing numeric values used in histogram.
- **main** indicates title of the chart.
- **col** is used to set color of the bars.
- **border** is used to set border color of each bar.
- **xlab** is used to give description of x-axis.
- **xlim** is used to specify the range of values on the x-axis.
- **ylim** is used to specify the range of values on the y-axis.
- **breaks** is used to mention the width of each bar.

```
> # Create Data for a graph  
> v <- c(3,4,5,23,4,5,66)  
> # Assign the chart a file name  
> png (file="Histogram.jpg")  
> # Create the histogram  
> hist(v, xlab="Weight", col="red", border="yellow")  
> dev.off()  
null device  
      1
```

Histogram of v



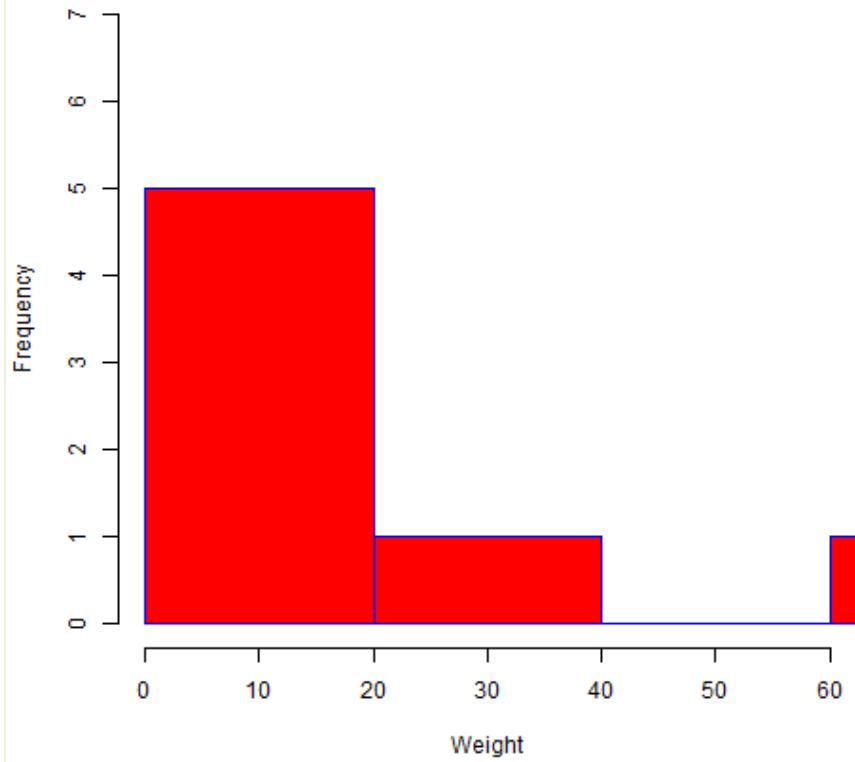
Range of x and y values



- ⌘ To specify the range of values allowed in X axis and Y axis, we can use the xlim and ylim parameters.
- ⌘ The width of each of the bar can be decided by using breaks.

```
> # Create Data for a graph  
> v <- c(3,4,5,23,4,5,66)  
> # Assign the chart a file name  
> png(file="Histogram_labels.jpg")  
> #Create the Histogram  
> hist(v,xlab="Weight", col="red", border="blue", xlim=c(0,60), ylim=c(0,7),  
breaks = 3)  
> dev.off()  
null device  
      1
```

Histogram of v



R – Line Graphs



- # A line chart is a graph that connects a series of points by drawing line segments between them.
- # These points are ordered in one of their coordinate (usually the x-coordinate) value.
- # Line charts are usually used in identifying the trends in data.
- # The **plot()** function in R is used to create the line graph.

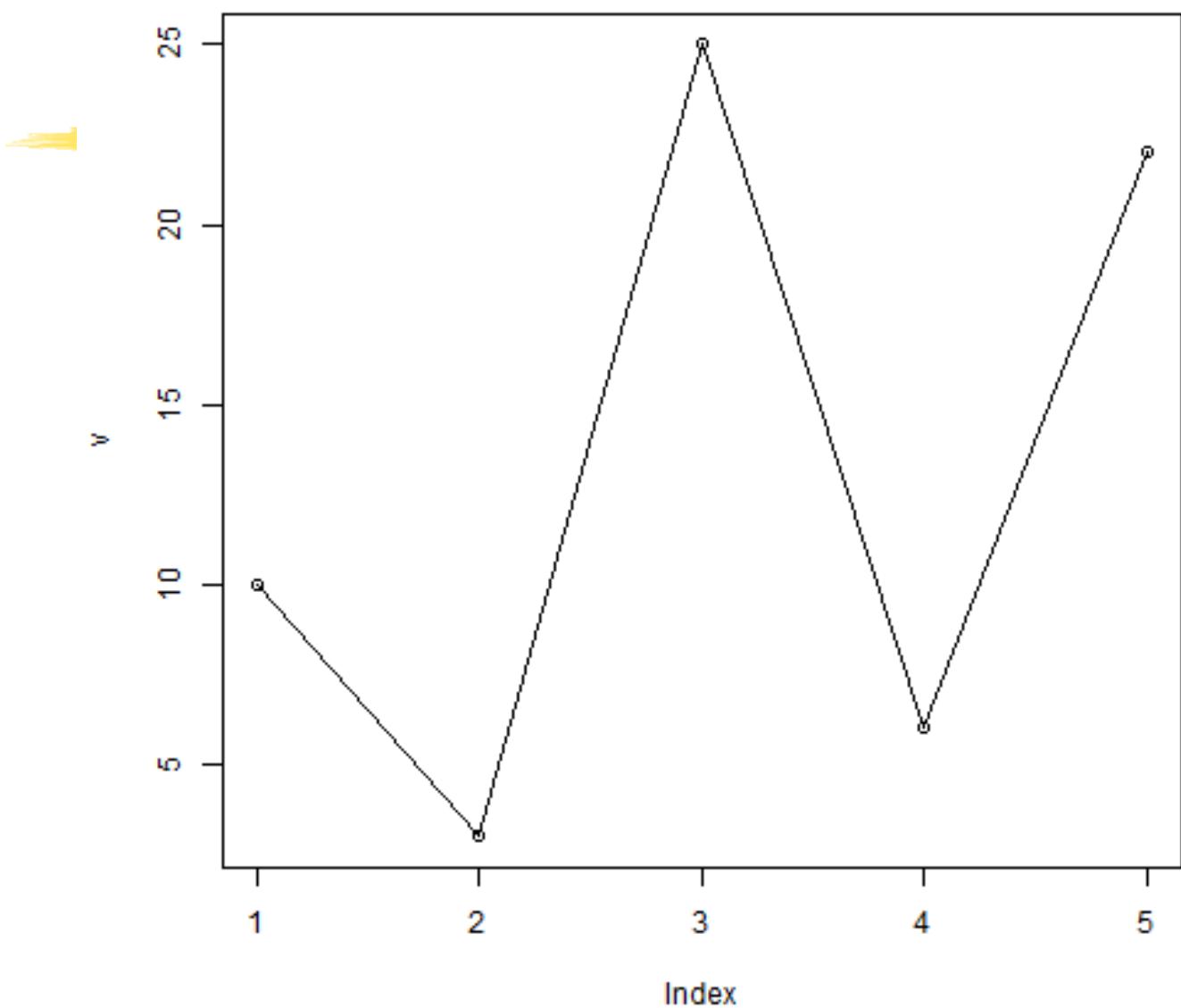
```
plot(v,type,col,xlab,ylab)
```

Following is the description of the parameters used –

- **v** is a vector containing the numeric values.
- **type** takes the value "p" to draw only the points, "l" to draw only the lines and "o" to draw both points and lines.
- **xlab** is the label for x axis.
- **ylab** is the label for y axis.
- **main** is the Title of the chart.
- **col** is used to give colors to both the points and lines.

1

```
> # Create data for line chart
> v <- c(10,3,25,6,22)
> # Assign the chart a file name
> png(file="Line_Chart.jpg")
> #Create Line chart
> plot(v,type="o")
> dev.off()
null device
1
```

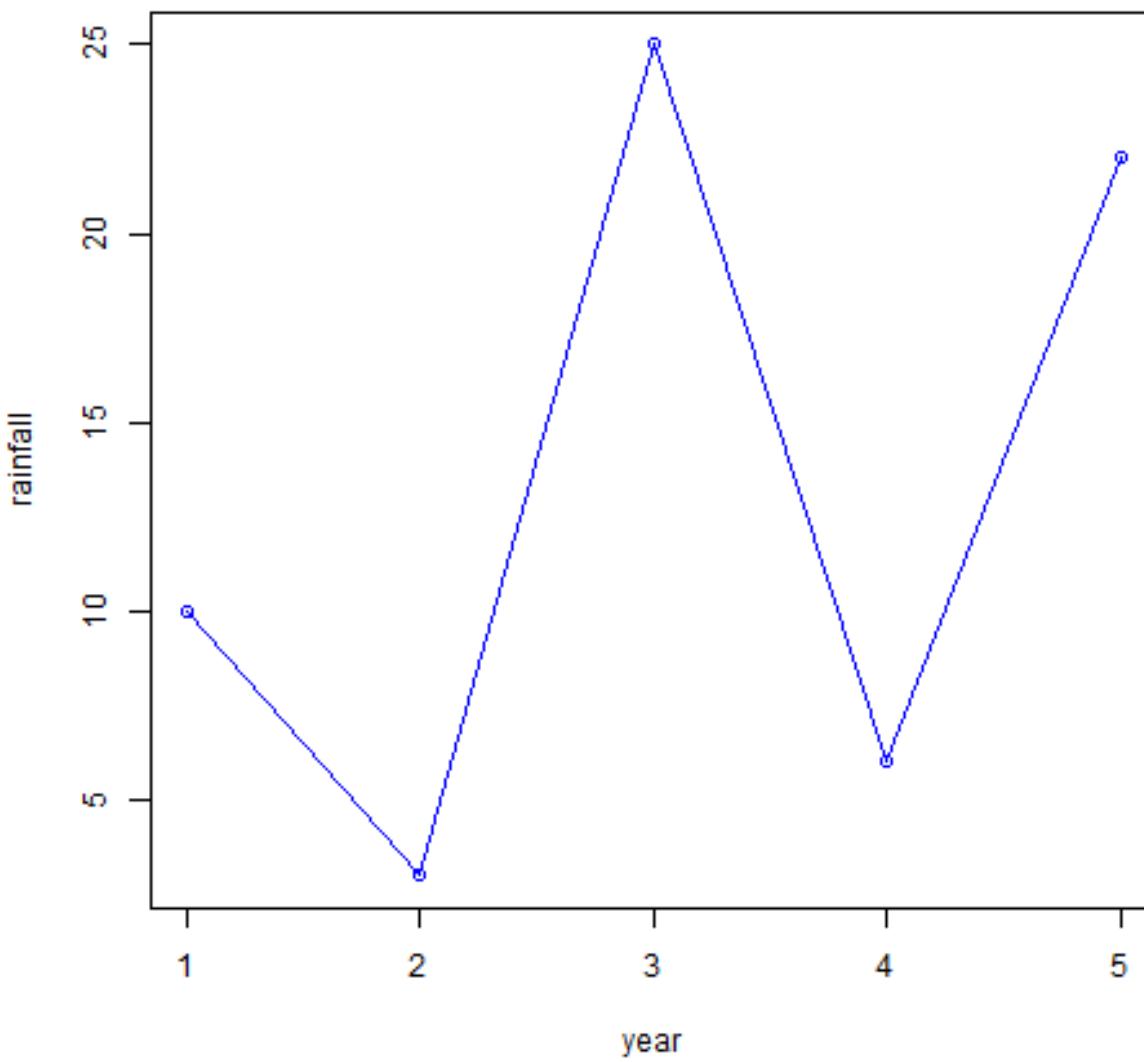


Line Chart – Title, Color and Labels



```
> plot(v, type="l")
> # Create data for line chart
> v <- c(10,3,25,6,22)
> # Assign the chart a file name
> png(file="linechart_title.jpg")
> # Create Line Chart
> plot(v,type="o",col="blue",xlab="year", ylab="rainfall", main = "Rainfall Chart")
> dev.off()
```

Rainfall Chart

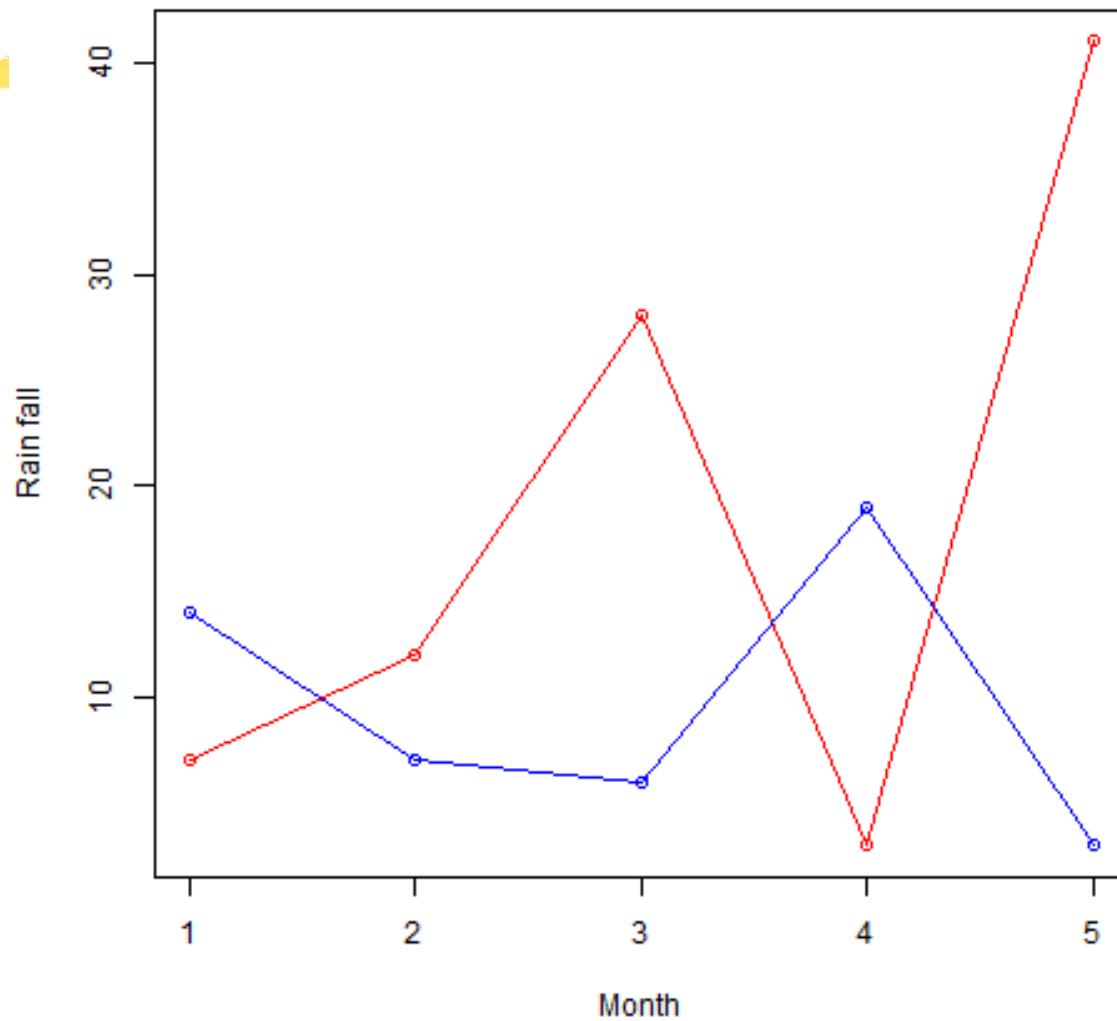


Line Chart – Multiple Lines



```
> # Create the Data for the Chart  
> v <- c(7,12,28,3,41)  
> t <- c(14,7,6,19,3)  
> # Assign the chart a file name  
> png(file="Linechart_Multiple_lines.jpg")  
> #Plot the line chart  
> plot(v,type="o", col="red",xlab="Month", ylab="Rain fall", main = "Rain_fal  
l_chart")  
> lines(t,type="o", col="blue")  
> dev.off()
```

Rain_fall_chart



Correlation



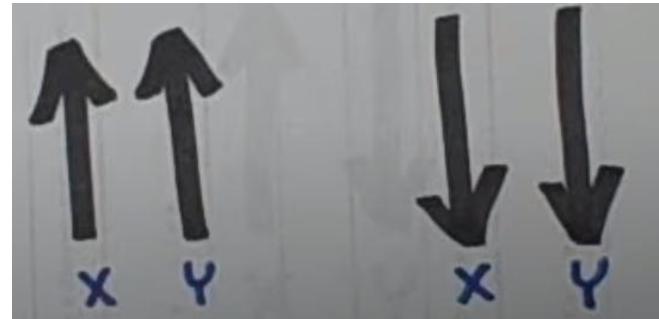
- ⌘ Study of strength of relationship between variables.
- ⌘ The word Correlation is made of **Co-** (meaning "together"), and **Relation**.
- ⌘ Correlation is **Positive** when the values **increase** together.
- ⌘ Correlation is **Negative** when one value **decreases** as the other increases.

Correlation



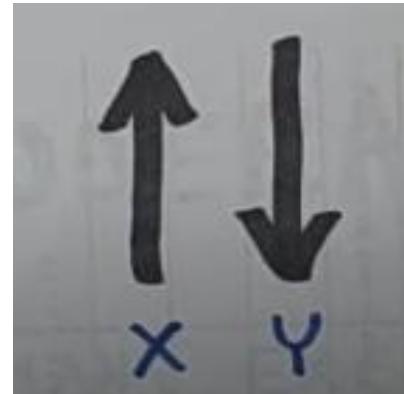
- # The correlation values ranges from -1 to 1
- # It is divided into three Types
 - ↗ **Positive Correlation** – when the value of one variable increases with respect to another.
 - ↘ **Negative Correlation** – when the value of one variable decreases with respect to another.
 - ↔ **No Correlation** – when there is no linear dependence or no relation between the two variables.

⌘ Correlation is **Positive** when the values increase together.

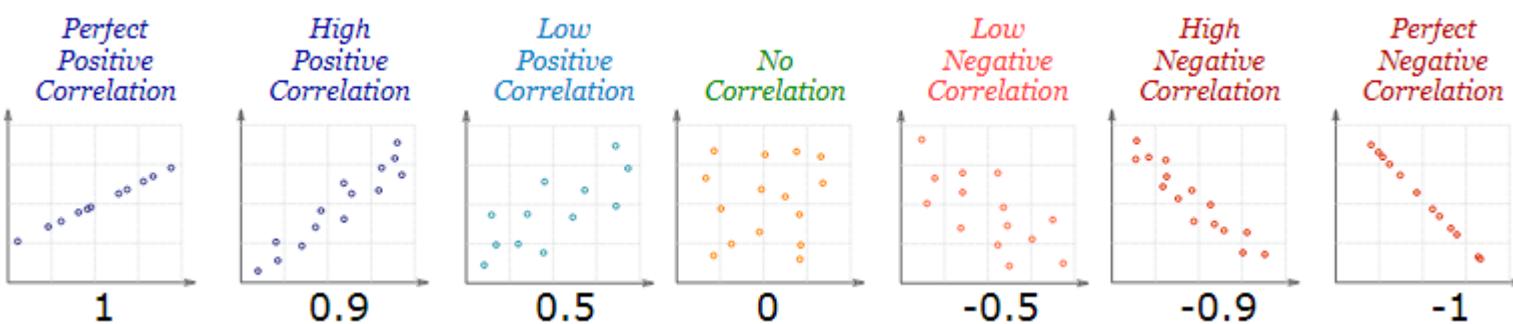


- ⌘ As the temperature goes up, **ice cream** and **cool drink** sales also go up.
- ⌘ The more time you spend running on a treadmill, the more calories you will burn.

- 
- ⌘ Correlation is **Negative** when one value decreases as the other increases



- ⌘ If a train increases speed, the length of time to get to the final point decreases.
- ⌘ If it is darker outside, more light is needed inside.
- ⌘ As one exercises more, his body weight becomes less.



Scatterplots



- # A Scatter (XY) Plot has points that show the relationship between two sets of data.
- # Each point represents the values of two variables.
- # One variable is chosen in the horizontal axis and another in the vertical axis.
- # The simple scatterplot is created using the **plot()** function.

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

Following is the description of the parameters used –

- **x** is the data set whose values are the horizontal coordinates.
- **y** is the data set whose values are the vertical coordinates.
- **main** is the title of the graph.
- **xlab** is the label in the horizontal axis.
- **ylab** is the label in the vertical axis.
- **xlim** is the limits of the values of x used for plotting.
- **ylim** is the limits of the values of y used for plotting.
- **axes** indicates whether both axes should be drawn on the plot.

⌘ Dataset : mtcars

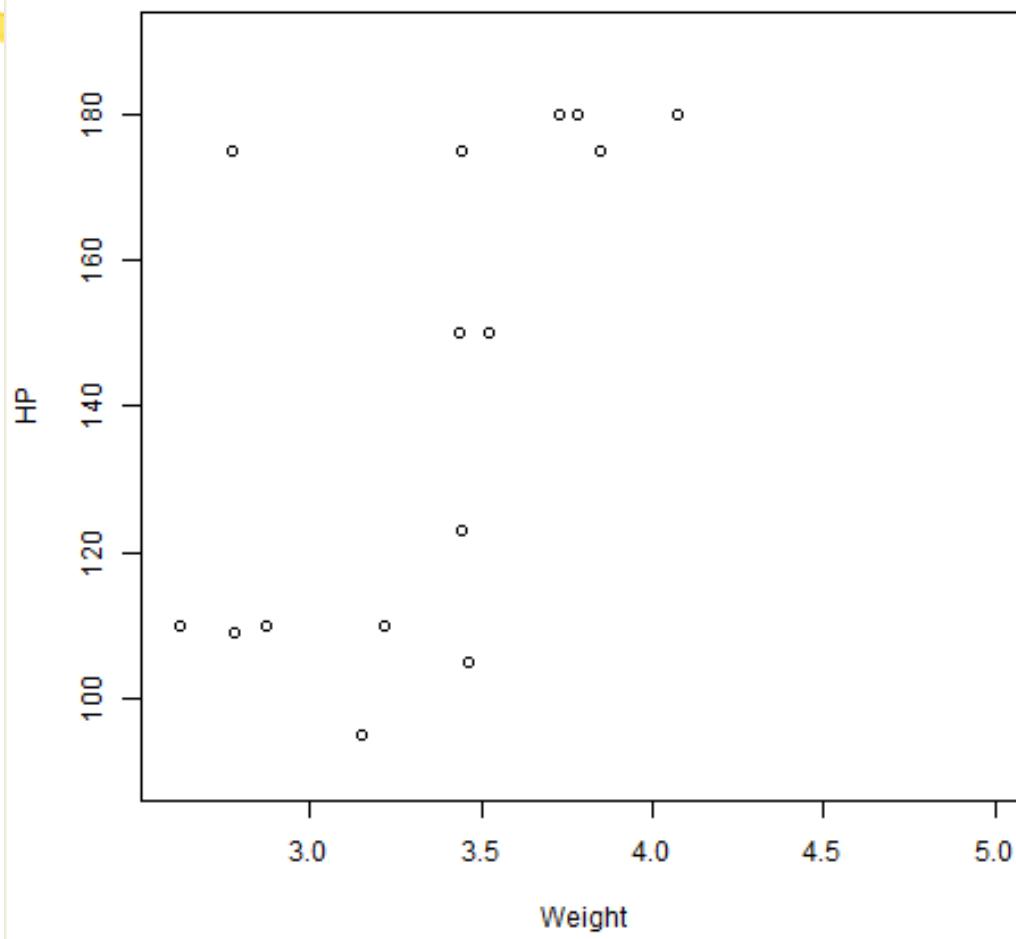
⌘ Columns : 'wt' and 'hp'

```
> ip <- mtcars[,c('wt','hp')]  
> print(head(ip))  
          wt   hp  
Mazda RX4       2.620 110  
Mazda RX4 Wag   2.875 110  
Datsun 710      2.320  93  
Hornet 4 Drive  3.215 110  
Hornet Sportabout 3.440 175  
Valiant        3.460 105
```

```
> View(ip)
> #To assign a name to the scatterplot
> png(file="Scatterplot.jpg")
> #Draw Scatterplot for Horsepower and Weight
> plot(x=ip$wt, y=ip$hp, xlab="Weight", ylab="HP", xlim=c(2.6,5), ylim=c(90,190), main="Weight Vs HP")
> #To Save the file
> dev.off()
```

Plot the chart for cars with weight between 2.6 to 5 and hp from 90 to 190

Weight Vs HP



Regression

- ⌘ To predict the unknown value from known value.
- ⌘ Regression analysis is a statistical, predictive modelling technique used to study the relationship between a dependent variable and one or more independent variables.

X	0	2	4	6
Y	0	4	?	12

Dependent variable – ?
Independent Variable - ?

Find the value of 'Y' when X = 4

⌘ To study the relationship between two or more variables using Regression

e.g.1 Relationship b/w advertising expenditure of sales

adv. expenditure ↑

sales also ↑

e.g.2: relationship b/w no. of hours practice

hours of practice ↑

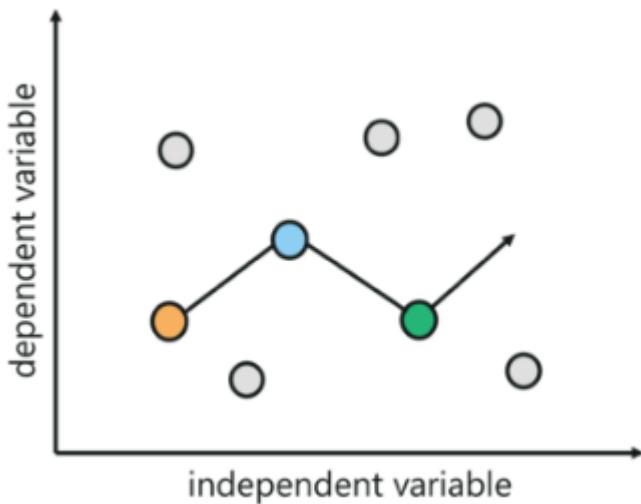
no of errors ↓

- 
- ⌘ Objective - To develop a model to show how the variables are related and to predict?
 - ⌘ eg. Predict sales for a given level of advertising
 - └ Dependent variable (y)
 - └ variables we're trying to predict.
 - └ Independent variable (x)
 - └ variable we use to predict the dependent variable.
 - ⌘ Dependent variable (y) → sales, no of errors
 - ⌘ Independent variable (x) → adv. exp, hours of Practice.

⌘ Input Dataset - Housing price data set of New York City.

↗ This data set contains information such as, size, locality, number of bedrooms in the house, etc.

⌘ Task - to predict the price of the house.





⌘ Independent (or) Predictor variables

- ↳ Values not depending on any variable.
- ↳ number of bedrooms, the size of the house and so on.

⌘ These predictor variables are used to predict the response variable.

⌘ Dependent (or) Response variable

- ↳ values depend on the values of the independent variable.
- ↳ Price of the house is the dependent variable.

CLASSIFICATION VS REGRESSION



Student Profile



Student Profile

Types of Regression Analysis

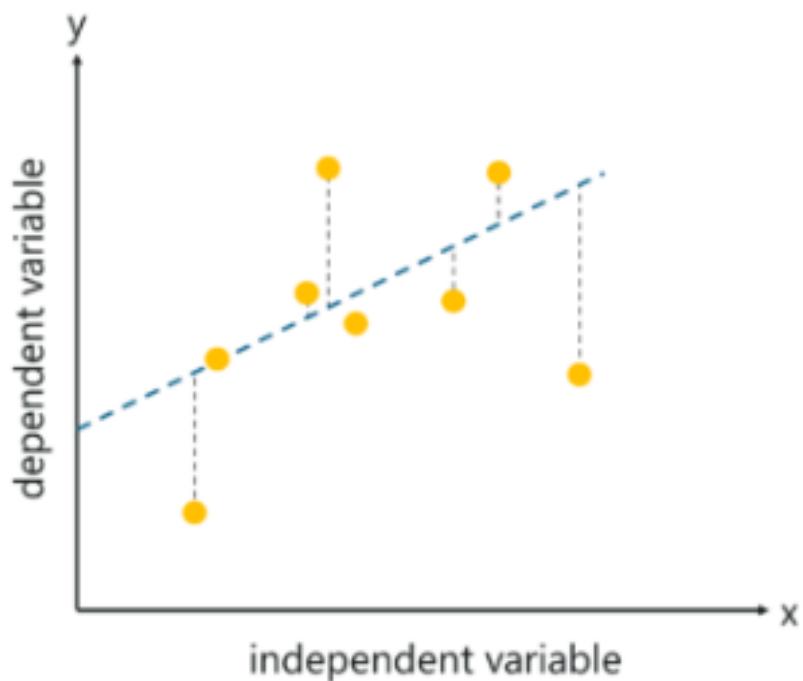


- # Linear Regression
- # Logistic Regression
- # Polynomial Regression

Linear Regression



- # It is one of the most basic and widely used machine learning algorithms.
- # It is a predictive modeling technique used to predict a continuous dependent variable, given one or more independent variables.
- # Simple linear regression
 - ↗ One independent and one dependent variable.
- # Multiple linear regression
 - ↗ More than one independent variable and one dependent variable.



- 
- ⌘ Here, the relationship between the dependent and independent variable is always linear thus, when we try to plot their relationship, we'll observe more of a straight line than a curved one.
 - ⌘ Equation used to represent a linear regression model:

$$y = \beta_0 + \beta_1 x + \epsilon$$



⌘ Multiple linear regression

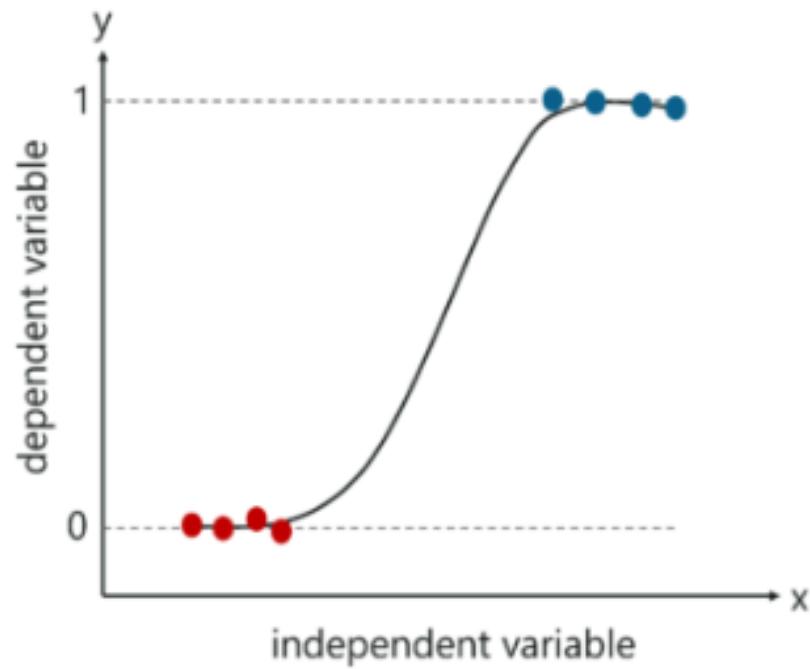
- ↗ Extension of linear regression into relationship between more than two variables.
- ↗ In simple linear relation we have one predictor and one response variable, but in multiple regression we have more than one predictor variable and one response variable.

$$y = a + b_1x_1 + b_2x_2 + \dots + b_nx_n$$

Logistic Regression



- ⌘ Logistic Regression is a machine learning algorithm used to solve classification problems.
- ⌘ Logistic Regression is a predictive analysis technique used to predict a dependent variable, given a set of independent variables, *such that the dependent variable is categorical.*

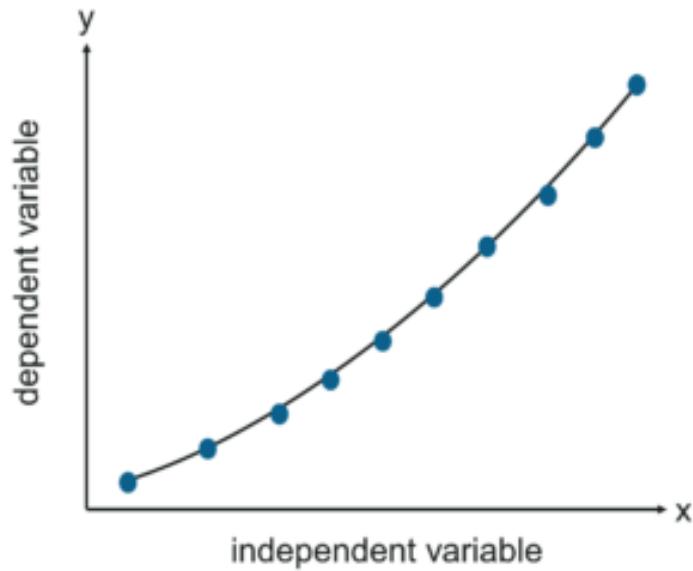


$$\log \left(\frac{Y}{1 - Y} \right) = C + B_1 X_1 + B_2 X_2 + \dots$$

Polynomial Regression



- ⌘ Polynomial Regression is a method used to handle non-linear data.
- ⌘ Non-linearly separable data is basically when you cannot draw out a straight line to study the relationship between the dependent and independent variables.



The reason it is called 'Polynomial' regression is that the power of some independent variables is more than 1.

$$Y = b_0 + b_1 x + b_2 x^2 + \dots + b_n x^n + \varepsilon$$

Simple Linear Regression

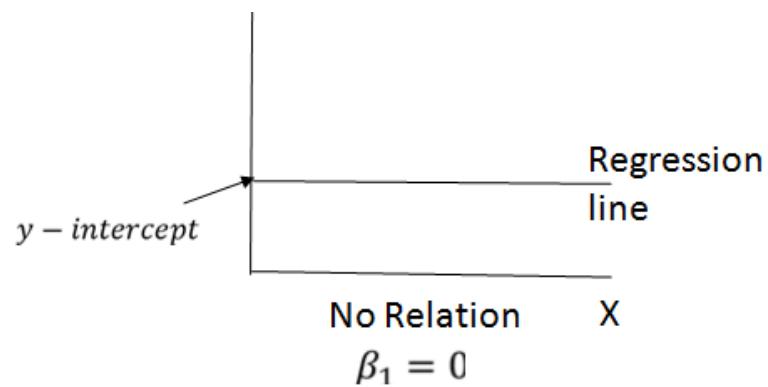
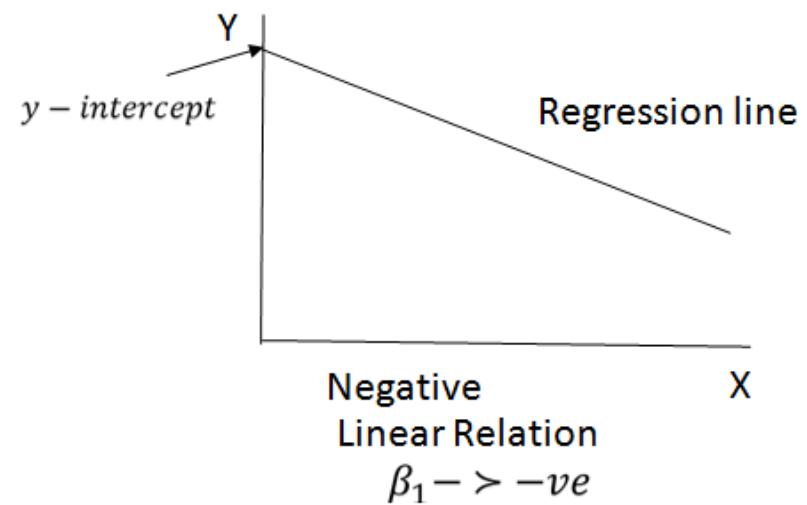
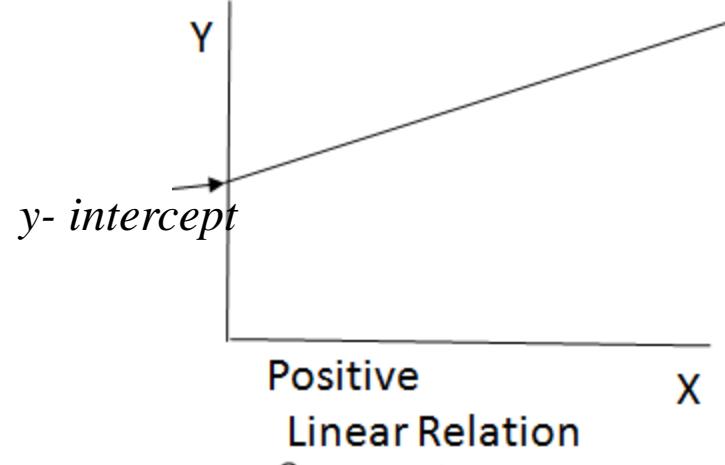


- ⌘ One Independent variable and one output variable.
- ⌘ It is named as linear, because the relationship is approximated using a straight line.

$$y = \beta_0 + \beta_1 x + \epsilon$$

β_0 -> y – intercept of the regression line

β_1 -> slope of the reg. line



Slope – tells whether the line is increasing / decreasing; how steep it is

$\beta_0, \beta_1 \rightarrow$ Popl. parameter

$b_0, b_1 \rightarrow$ sample

For samples

$$\hat{y} = b_0 + b_1 x$$

\hat{y} = predicted value of 'y' for gn 'x' value

b_0 = y- intercept of the reg. line

b_1 = slope of the reg. line

Hours Studied (x)	Grade on Exam (y)
2	69
9	98
5	82
5	77
3	71
7	84
1	55
8	94
6	84
2	64

Least Squares method

- Fit the points in the graph

$$\hat{y} = b_0 + b_1 x$$

\hat{y} = Predict grade on the exam

b_0 = y- intercept of the reg. line

b_1 = slope of the reg. line

X = no. of hours studies

Calculating Slope

$$b_1 = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sum(x_i - \bar{x})^2}$$

x_i - value of the independent var for the i^{th} observation

y_i - value of the independent var for the i^{th} observation

\bar{x} - mean value of the independent var for the i^{th} observation

\bar{y} - mean value of the independent var for the i^{th} observation

Regression equation : $\hat{y} = b_0 + b_1 x$

To calculate b_0 : $b_0 = \bar{y} - b_1 \bar{x}$

x_i	y_i	$(x_i - \bar{x})$ 4.8	$(y_i - \bar{y})$ 77.8	$(x_i - \bar{x})(y_i - \bar{y})$	$(x_i - \bar{x})^2$
2	69	-2.8	-8.8	24.64	7.84
9	98	4.2	20.2	84.84	17.64
5	82	0.2	4.2	0.84	0.04
5	77	0.2	-0.8	-0.16	0.04
3	71	-1.8	-6.8	12.24	3.24
7	84	2.2	6.2	13.64	4.84
1	55	-3.8	-22.8	86.64	14.44
8	94	3.2	16.2	51.84	10.24
6	84	1.2	6.2	7.44	1.44
2	64	-2.8	-13.8	38.64	7.84
48	778			320.6	67.6

$$\sum x_i = 48$$

$$\sum y_i = 778$$

$$\bar{x} = 4.8 \text{ (48/10)}$$

$$\bar{y} = 77.8 \text{ (778/10)}$$

$$b_1 = \frac{\sum (x_i - \bar{x})(y_i - \bar{y})}{\sum (x_i - \bar{x})^2} = \frac{320.6}{67.6} = 4.74$$

$$\hat{y} = b_0 + b_1 x$$

$$\begin{aligned}b_0 &= \bar{y} - b_1 \bar{x} \\&= 77.8 - 4.74(4.8) \\&= 55.048\end{aligned}$$

$$\hat{y}_i = \mathbf{55.048 + 4.74 x}$$



⌘ To predict the grade when no. of hours studied = 3

$$\hat{y}_i = 55.048 + 4.74 (3)$$

$$\hat{y}_i = \mathbf{69.268}$$

Steps to Establish a Regression



- ⌘ Carry out the experiment of gathering a sample of observed values of number of hours studied and corresponding grade.
- ⌘ Create a relationship model using the **lm()** functions in R.
- ⌘ Find the coefficients from the model created and create the mathematical equation using the coefficients.
- ⌘ Get a summary of the relationship model to know the average error in prediction. Also called **residuals**.
- ⌘ To predict the grade for new persons, use the **predict()** function in R.



⌘Input Data:

```
> #values of Hours studied  
> x <- c(2,9,5,5,3,7,1,8,6,2)  
> #values of Grade  
> y <- c(69,98,82,77,71,84,55,94,84,64)
```

⌘ lm() function

↗ This function creates the relationship model between the predictor and the response variable.

```
lm(formula,data)
```

Following is the description of the parameters used –

- **formula** is a symbol presenting the relation between x and y.
- **data** is the vector on which the formula will be applied.

```
> #values of Hours studied  
> x <- c(2,9,5,5,3,7,1,8,6,2)  
> #values of Grade  
> y <- c(69,98,82,77,71,84,55,94,84,64)  
> #Apply the lm() function  
> relation <- lm (y ~ x)  
> print(relation)
```

Call:
lm(formula = y ~ x)

Coefficients:
(Intercept) x
55.036 4.743

- 
- ⌘ Is this enough to actually use this model?
 - ⌘ How do we ensure that the model generated is statistically significant?
 - ↗ Soln : p-Values
 - ↗ we can consider a linear model to be statistically significant only when both these p-Values are less than the pre-determined statistical significance level of 0.05.

- 
- # Whenever there is a p-value, there is always a Null and Alternate Hypothesis associated.
 - # Null Hypothesis (H_0)
 - ◻ The beta coefficients associated with the variables is equal to zero.
 - # Alternate Hypothesis (H_1)
 - ◻ The coefficients are not equal to zero. (i.e. there exists a relationship between the independent variable in question and the dependent variable).

⌘ Get the Summary of Relationship

```
> print(summary(relation))

Call:
lm(formula = y ~ x)

Residuals:
    Min      1Q  Median      3Q     Max 
-4.778 -1.442  0.395  1.558  4.479 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept) 55.0355   2.0880   26.36 4.61e-09 ***
x            4.7426   0.3825   12.40 1.67e-06 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 3.145 on 8 degrees of freedom
Multiple R-squared:  0.9505,    Adjusted R-squared:  0.9444 
F-statistic: 153.7 on 1 and 8 DF,  p-value: 1.67e-06
```



⌘ R-squared

- └ Statistical measure which shows how close the data are to the fitted regression line.
- └ Known as the coefficient of determination, or the coefficient of multiple determination for multiple regression.

⌘ R-squared = Explained variation / Total variation

⌘ R-squared is always between 0 and 100%:

- └ 0% indicates that the model explains none of the variability of the response data around its mean.
- └ 100% indicates that the model explains all the variability of the response data around its mean.



⌘ predict() function

- ↗ To predict the value for new records

To predict the grade when number of hours studied = 3

```
> a <- data.frame(x=3)
> result <- predict(relation,a)
> print(result)
  1
69.26331
```

- 
- ⌘ How good is the prediction?
 - ⌘ How well the regression line fits the data?
 - ⌘ Solution : coefficient of determination

$$r^2 = \text{SSR} / \text{SST}$$

$\text{SSR} = \text{Sum of Square due to regression} = \sum(\widehat{y}_0 - \bar{y})^2$

$\text{SST} = \text{Total sum of Squares} = \sum(y_i - \bar{y})^2$

$\text{SSE} = \text{Sum of Squares due to Error} = \sum(y_i - \widehat{y}_i)^2$

$$\text{SST} = \text{SSR} + \text{SSE}$$

x_i	y_i	Predicted Grades $\hat{y}_i = 55.048 + 4.748x_i$	Error $y_i - \hat{y}_i$	Sq Error $(y_i - \hat{y}_i)^2$	Deviation $y_i - \bar{y}$	Sq deviation $(y_i - \bar{y})^2$
2	69	64.528	4.472	19.9988	-8.8	77.44
9	98	97.708	0.292	0.0852	20.02	408.04
5	82	78.748	3.252	10.5755	4.2	17.64
5	77	78.748	-1.748	3.0555	-0.8	0.64
3	71	69.268	1.732	2.9998	-6.8	46.24
7	84	88.228	-4.228	17.8759	6.2	38.44
1	55	59.788	4.788	22.9249	-22.8	519.84
8	94	92.968	1.032	1.0650	16.2	262.44
6	84	83.488	0.512	0.2621	6.2	368.44
2	64	64.528	-0.528	0.2788	-13.8	190.44
			SSE	79.1215	SST	1599.6

$$\text{SSE} = 79.1215$$

$$\text{SST} = 1599.6$$

$$\text{SSR} = \text{SST} - \text{SSE}$$

$$= 1520.4785$$

$$\text{SST} = \text{SSR} + \text{SSE}$$

$$\text{SSR} = \text{SST} - \text{SSE}$$

$$= 1599.6 - 79.1215$$

$$= 1520.4785$$

$$r^2 = \frac{1520.4785}{1599.6} = 0.9505$$

r^2 = % of variability in 'y' can be explained by 'x'

r^2 = 95.05 % of the variability in grades can be explained by the number of hours studied

Correlation coefficient -> measures the strength of the relationships b/w 'x' & 'y'

r' -> -1 to 1

$r = +1$ -> perfect positive linear relationship

$r = -1$ -> perfect negative linear relationship

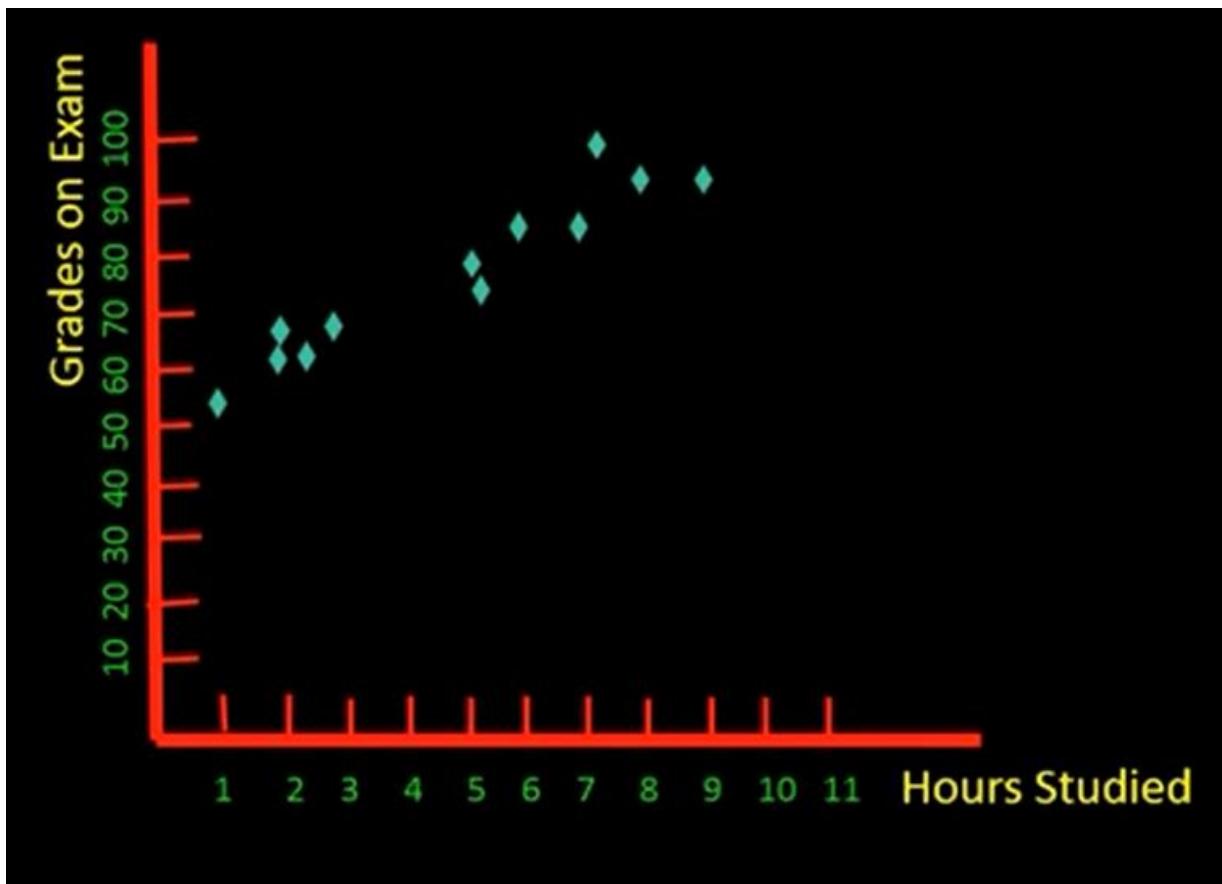
$r=0$ -> no linear relationship

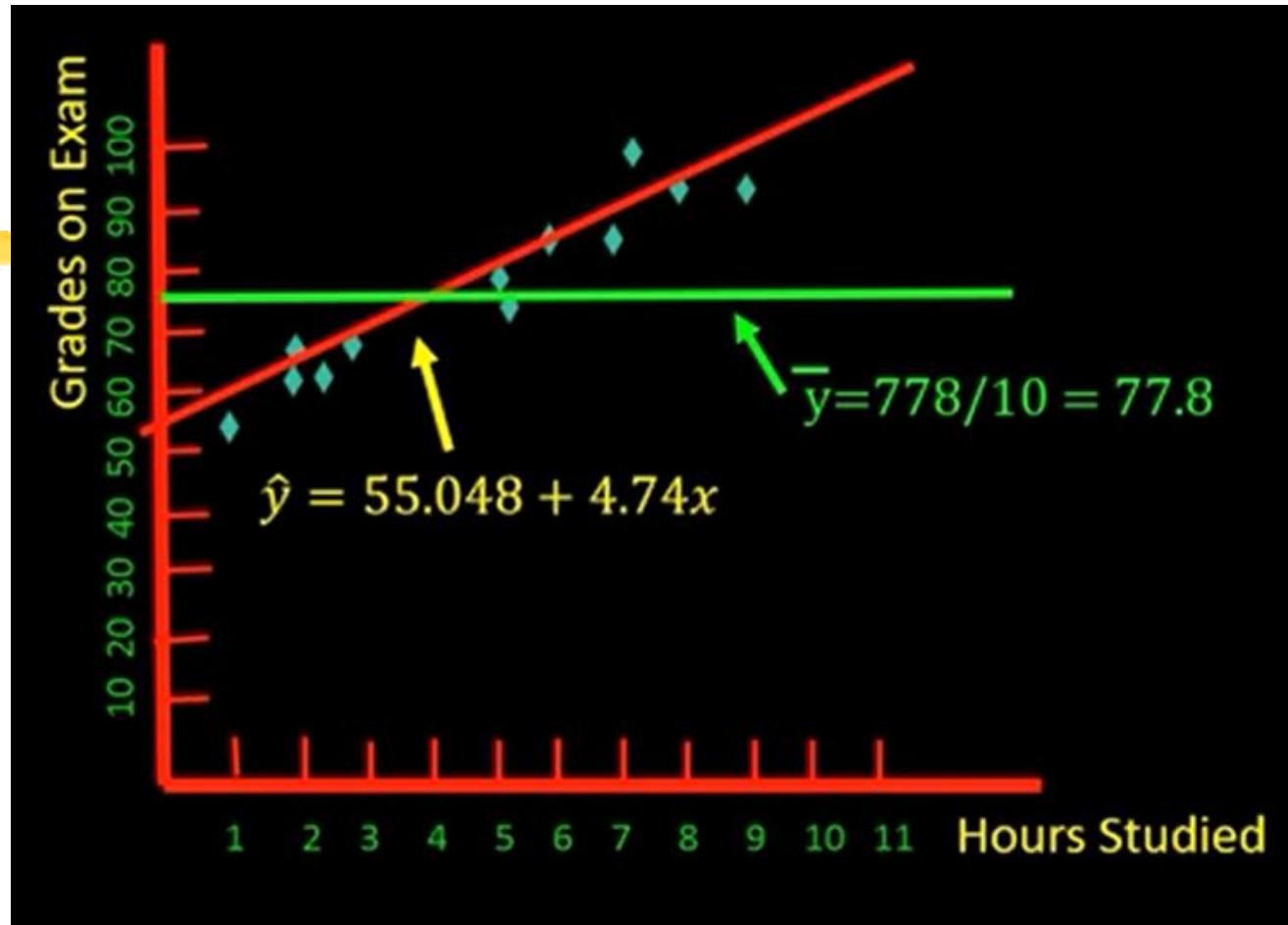
$$r_{xy} = (\text{sign of } b_1) \sqrt{\text{Coefficient of determination}}$$

$$= (\text{sign of } b_1) \sqrt{r^2}$$

$$= + \sqrt{.9505}$$

$$r = +.9749 \text{ (very strong relationship b/w 'x' & 'y')}$$

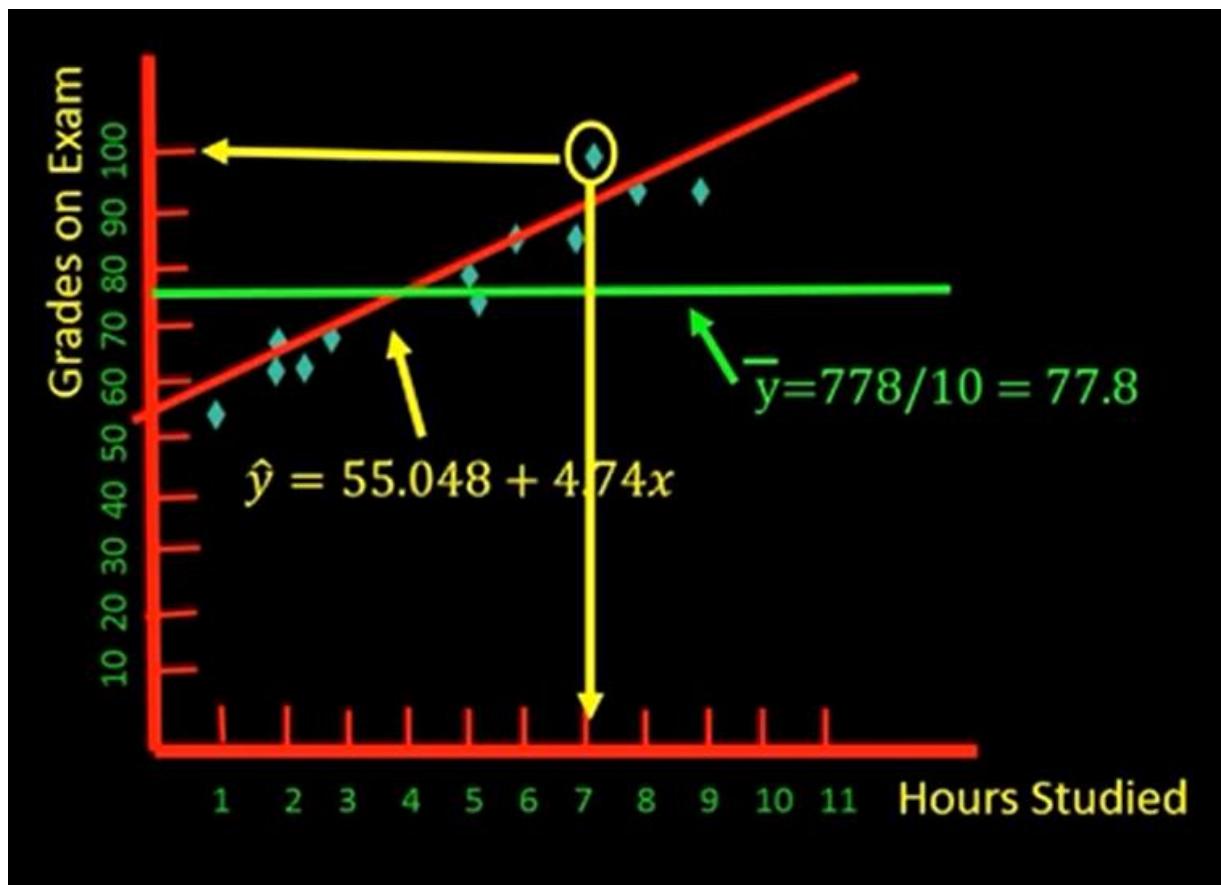


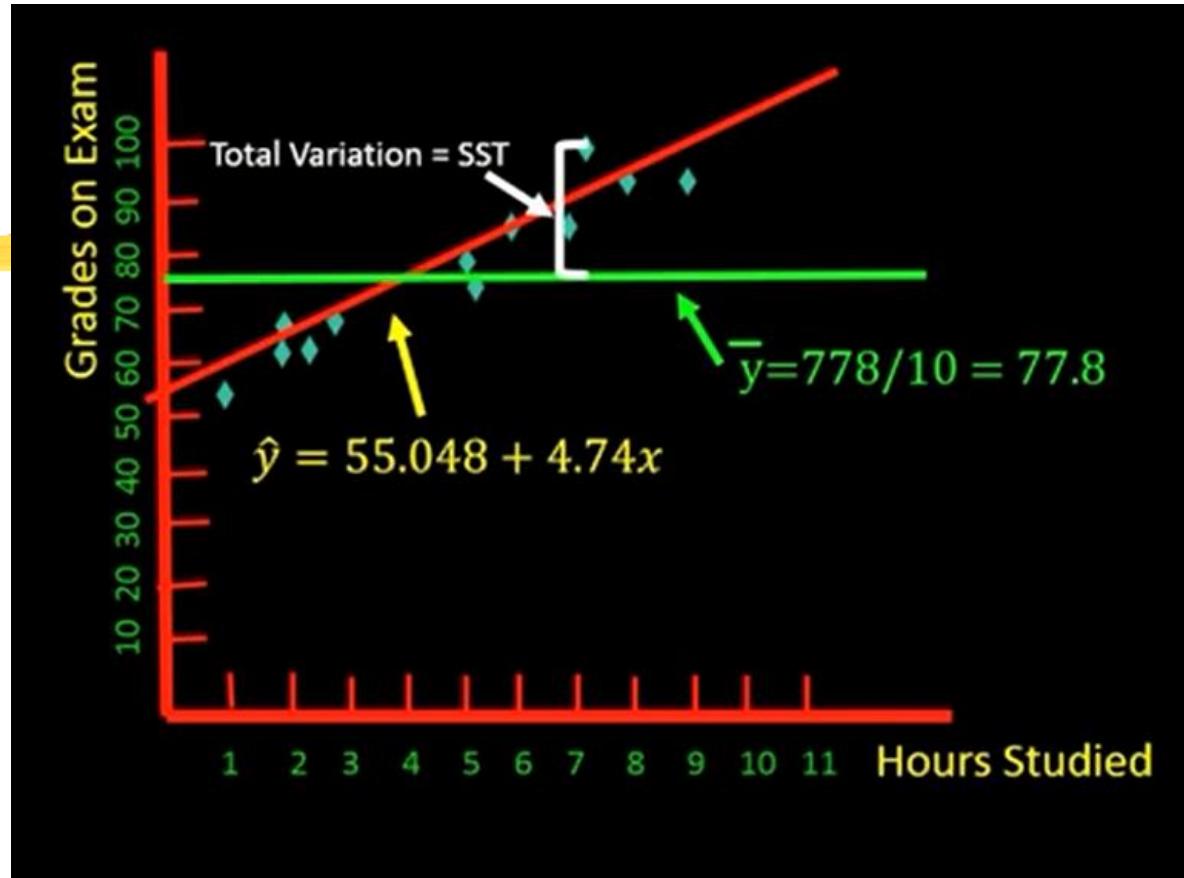


$$\hat{y} = b_0 + b_1 x$$

$$\hat{y} = 55.048 + 4.74x$$

$$\bar{y} = 778/10 = 77.8$$

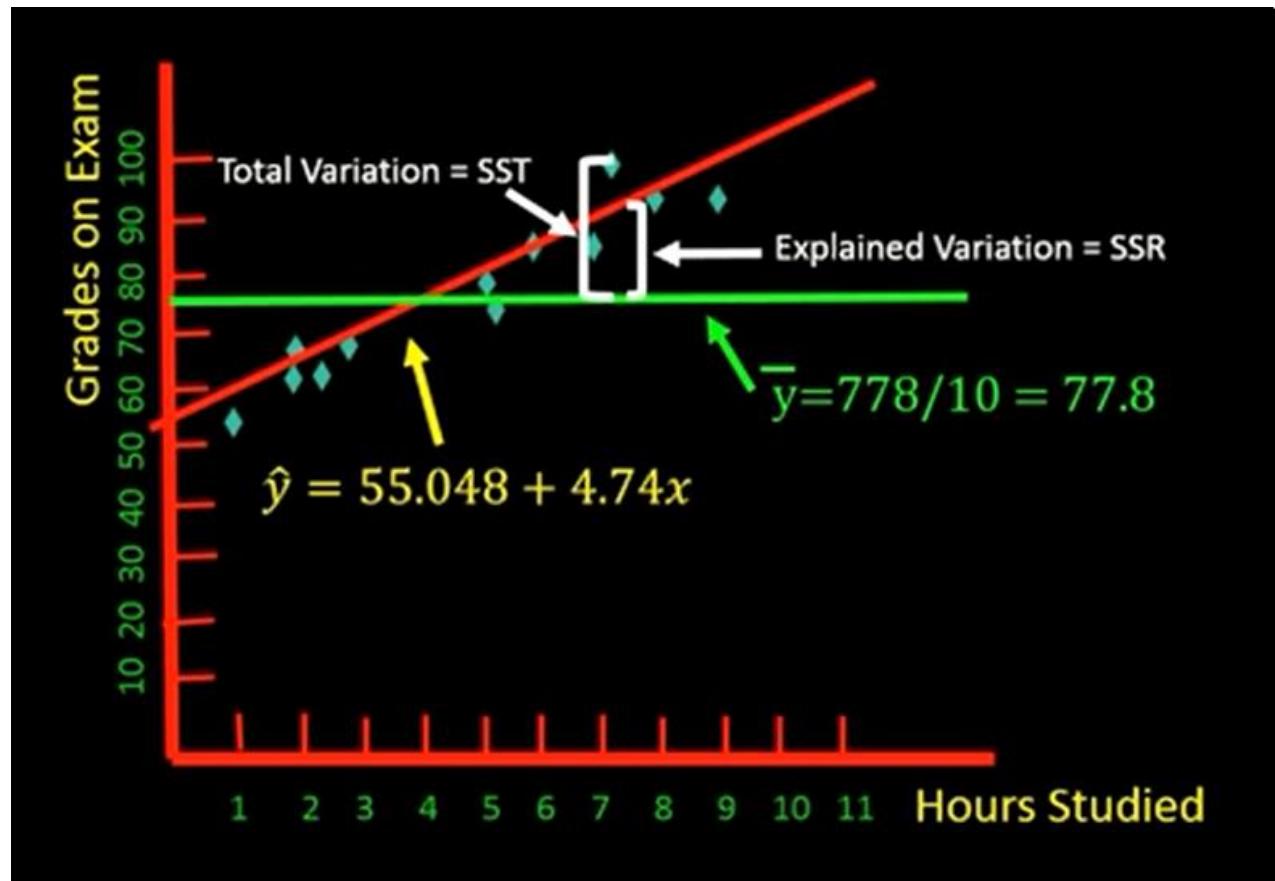




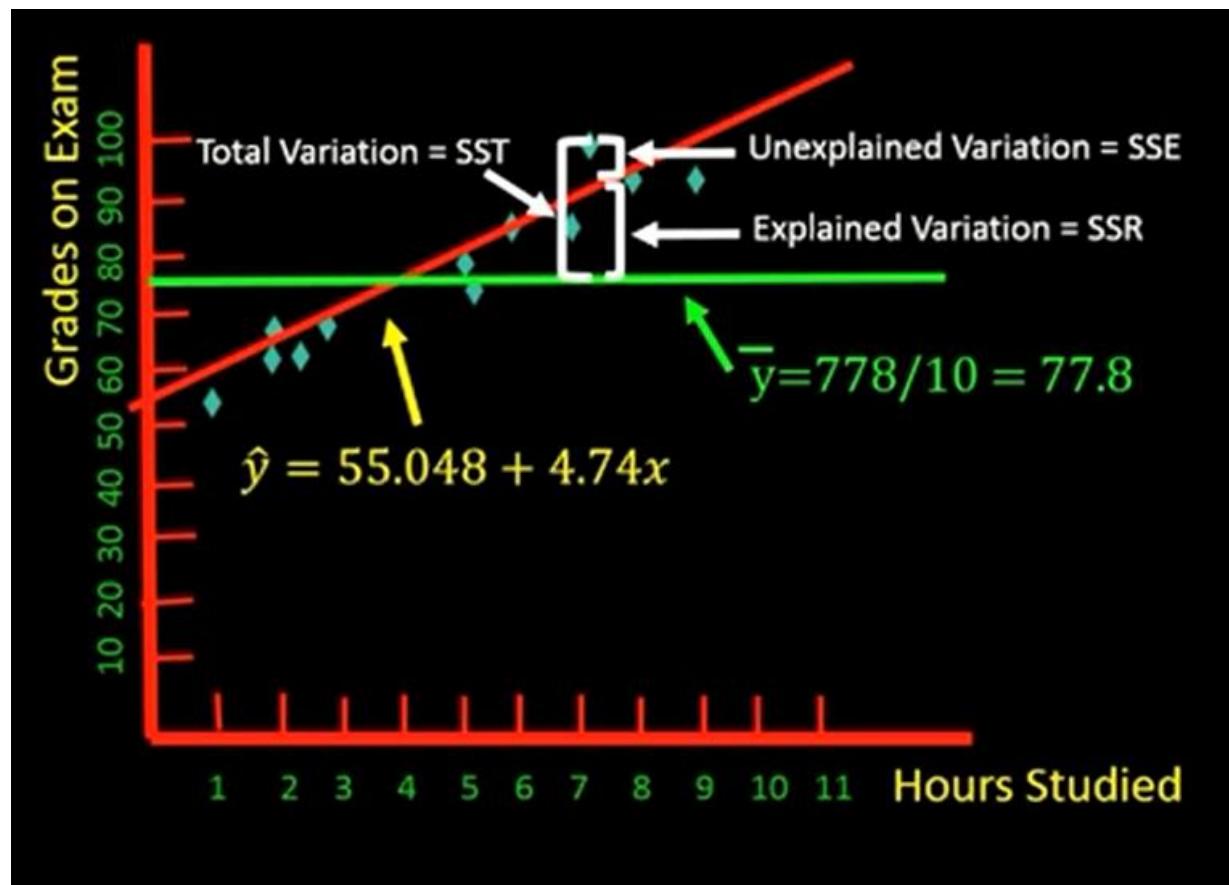
$$r^2 = \text{SSR/SST}$$

$$\text{SST} = \sum (y_i - \bar{y})^2$$

$$\begin{aligned}\hat{y} &= 55.048 + 4.74(7) \\ \hat{y} &= 88.228\end{aligned}$$



$$SSR = \sum(\hat{y}_i - \bar{y})^2$$



$$SSE = \sum (y_i - \hat{y}_i)^2$$

Multiple Regression



- ⌘ Multiple regression is an extension of linear regression into relationship between more than two variables.
- ⌘ In simple linear relation we have one predictor and one response variable, but in multiple regression we have more than one predictor variable and one response variable.


$$y = a + b_1x_1 + b_2x_2 + \dots + b_nx_n$$

Following is the description of the parameters used –

- **y** is the response variable.
- **a, b₁, b₂...b_n** are the coefficients.
- **x₁, x₂, ...x_n** are the predictor variables.



❖ Steps to apply the multiple linear regression in R

- ❖ Collect the data
- ❖ Capture the data in R
- ❖ Apply the multiple linear regression in R
- ❖ Make a prediction

⌘ Step 1:

↗ Goal is to predict the stock_index_price (the dependent variable) of a fictitious economy based on two independent/input variables:

- ☒ Interest_Rate
- ☒ Unemployment_Rate

$$Y = b_0 + b_1 * X_1 + b_2 * X_2$$

Where,

b_0 - Y- intercept

X_1 - Interest_Rate

X_2 - Unemployment_Rate

Year	Month	Interest_Rate	Unemployment_Rate	Stock_Index_Price
2017	12	2.75	5.3	1464
2017	11	2.5	5.3	1394
2017	10	2.5	5.3	1357
2017	9	2.5	5.3	1293
2017	8	2.5	5.4	1256
2017	7	2.5	5.6	1254
2017	6	2.5	5.5	1234
2017	5	2.25	5.5	1195
2017	4	2.25	5.5	1159
2017	3	2.25	5.6	1167
2017	2	2	5.7	1130
2017	1	2	5.9	1075
2016	12	2	6	1047
2016	11	1.75	5.9	965
2016	10	1.75	5.8	943
2016	9	1.75	6.1	958
2016	8	1.75	6.2	971
2016	7	1.75	6.1	949
2016	6	1.75	6.1	884
2016	5	1.75	6.1	866
2016	4	1.75	5.9	876
2016	3	1.75	6.2	822
2016	2	1.75	6.2	704
2016	1	1.75	6.1	719



⌘ Step 2: Capture the data in R

```
> mydata <- read.csv("C:/users/Senthil/desktop/r-intro1/stock.csv",header = TRUE)
```



⌘ Step 3: Apply multiple linear regression in R

```
model <- lm(Dependent variable ~ First independent Variable + Second independent variable + ...)  
summary(model)
```

```
> mydata <- read.csv("C:/users/Senthil/desktop/r-intro1/stock.csv",header = TRUE)
> model1 <- lm (Stock_Index_Price ~ Interest_Rate + Unemployment_Rate,data=mydata)
> summary(model1)
```

```
call:  
lm(formula = Stock_Index_Price ~ Interest_Rate + Unemployment_Rate,  
    data = mydata)  
  
Residuals:  
    Min      1Q  Median      3Q     Max  
-158.205 -41.667   -6.248   57.741  118.810  
  
Coefficients:  
              Estimate Std. Error t value Pr(>|t|)  
(Intercept) 1798.4     899.2   2.000  0.05861 .  
Interest_Rate 345.5     111.4   3.103  0.00539 **  
Unemployment_Rate -250.1     117.9  -2.121  0.04601 *  
---  
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1  
  
Residual standard error: 70.56 on 21 degrees of freedom  
Multiple R-squared:  0.8976,    Adjusted R-squared:  0.8879  
F-statistic: 92.07 on 2 and 21 DF,  p-value: 4.043e-11
```

⌘ Multiple linear regression equation

Stock_Index_Price = (**Intercept**) + (**Interest_Rate coef**) * X₁ + (**Unemployment_Rate coef**) * X₂

Stock_Index_Price = (**1798.4**) + (**345.5**) * X₁ + (**-250.1**) * X₂

- 
- # Step 4: Make a prediction.
 - # Predict the stock index price for the following data:
 - # Interest Rate = 1.5 (i.e., $X_1 = 1.5$)
 - # Unemployment Rate = 5.8 (i.e., $X_2 = 5.8$)

```
Stock_Index_Price = (1798.4) + (345.5)*(1.5) + (-250.1)*(5.8)
```

```
> a <- data.frame(Interest_Rate=15, Unemployment_Rate=5.8)
> result <- predict(model,a)
> result
  1
5530.655
```

Example – II

```
> # Create the Data set  
> Sales <- read.csv("C:/users/senthil/desktop/r-intro1/rev.csv")
```

```
> summary(sales)  
      V1          V2          V3          V4  
 0 : 2  111798 : 2  0 : 3  121248.7116: 2  
153606 : 2  115667 : 2  231886 : 2  180257.1795: 2  
84533 : 2  115775 : 2  365162 : 2  100404.0021: 1  
1000 : 1  118282 : 2  100537 : 1  100435.6109: 1  
100306 : 1  119943 : 2  101017 : 1  100556.066 : 1  
100405 : 1  122941 : 2  101170 : 1  100589.3834: 1  
(Other):992 (Other):989 (Other):991 (Other) :993
```

```
> set.seed(2)  
> #Splitting the dataset into Training and Test  
> library(caTools)  
> split <- sample.split(sales, SplitRatio=0.7)  
> train <- subset(sales, split="TRUE")  
> test <- subset(sales, split="FALSE")  
>
```

```
> library(caTools)  
Warning message:  
package 'caTools' was built under R version 3.6.3  
> split <- sample.split(sales, SplitRatio=0.7)  
> split  
[1] TRUE FALSE FALSE TRUE  
> train <- subset(sales, split=TRUE)  
> test <- subset(sales, split = FALSE)
```

```
> model <- lm (Profit ~ ., data = train)
> summary(model)

Call:
lm(formula = Profit ~ ., data = train)

Residuals:
    Min      1Q  Median      3Q     Max 
-33534       0       0       0    17276 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept) 5.012e+04  1.313e+03   38.161 < 2e-16 ***
Paid         8.057e-01  7.605e-03  105.951 < 2e-16 ***
Organic     -2.681e-02  1.097e-02   -2.445   0.0146 *  
Social       2.723e-02  3.535e-03    7.701 3.24e-14 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 1984 on 996 degrees of freedom
Multiple R-squared:  0.9975,    Adjusted R-squared:  0.9975 
F-statistic: 1.335e+05 on 3 and 996 DF,  p-value: < 2.2e-16
```

- 
- ⌘ By using the `sample.split()` we are actually creating a vector with two values TRUE and FALSE.
 - ⌘ By setting the SplitRatio to 0.7, you are splitting the original Revenue dataset of 1000 rows to 70% training and 30% testing data.



TEST RESULTS

AVERAGE

MEAN



$\text{Marks(Student A)} + \text{Marks(Student B)} + \text{Marks(Student C)} + \text{Marks(Student D)} + \text{Marks(Student E)}$

5 (Number of Students)

TEST RESULTS

TEACHER 1:

Students	Marks
A	19
B	17
C	9
D	13
E	12
Total	70

$$\frac{70}{5} = 14$$



TEACHER 2:

Students	Marks
A	14
B	15
C	16
D	13
E	12
Total	70

$$\frac{70}{5} = 14$$

•

MEAN SQUARED ERROR / RESIDUAL:

TEACHER 1:

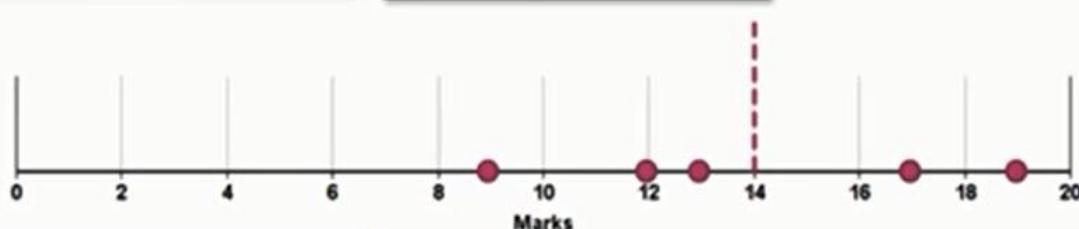
Students	Marks
A	19
B	17
C	9
D	13
E	12
Total	70

Students	Residuals
A	$19 - 14 = 5$
B	$17 - 14 = 3$
C	$9 - 14 = -5$
D	$13 - 14 = -1$
E	$12 - 14 = -2$
Total	0

$$\text{MSE} = \frac{5^2 + 3^2 + (-5)^2 + (-1)^2 + (-2)^2}{5} = \frac{25 + 9 + 25 + 1 + 4}{5} = \frac{64}{5}$$

$$\text{MSE} = \frac{64}{5}$$

$$\text{MSE} = 12.4$$



$$\frac{70}{5} = 14$$



DECISION TIME!

$$\text{MSE} = 5^2 + 2^2 + (-5)^2 + (-1)^2 + (-2)^2 = 25 + 4 + 25 + 1 + 4$$

12.8

$$\text{MSE} = 0^2 + 1^2 + 2^2 + (-1)^2 + (-2)^2 = 0 + 1 + 4 + 1 + 4$$

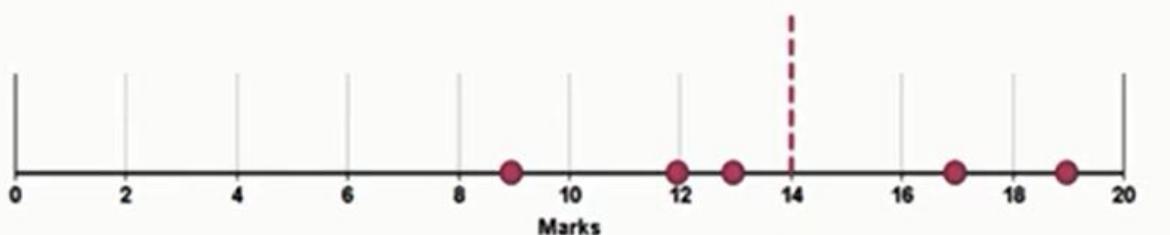
2



You're Hired!

HIGHLIGHTS

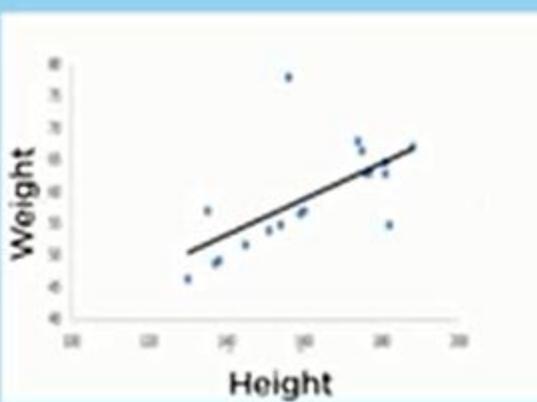
- Less MSE \rightarrow More efficient, your model is
- Your Model is centered around the observed values



Logistic Regression

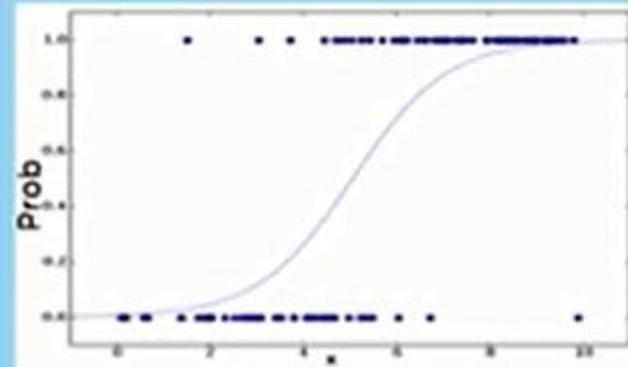
Linear Regression

When there is a linear relation between a dependent variable (continuous) and an independent variable (continuous or discrete)



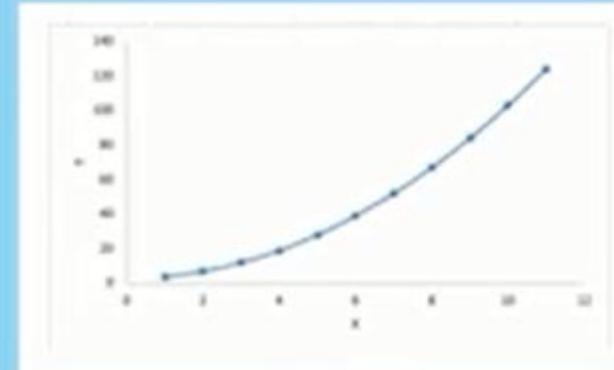
Logistic Regression

When the Y value in the graph is categorical in nature (i.e. Yes/No) and depends on the X variable



Polynomial Regression

When the relation between the dependent variable Y and the independent variable X is in nth degree of X





Linear Regression answers the question
"how much?"



Linear Regression is used to predict the
continuous variable



Logistic Regression answers "will it happen or
not"



Logistic Regression is used when the response
variable has only two outcomes (i.e. Yes/No , 1/ 0)

Let's try to understand this with an example

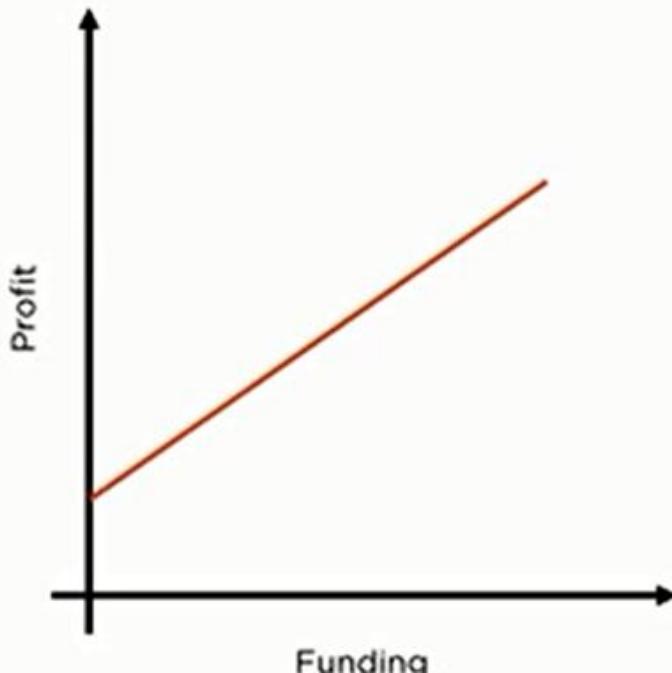
Trying to figure out whether a start-up will be profitable or not?



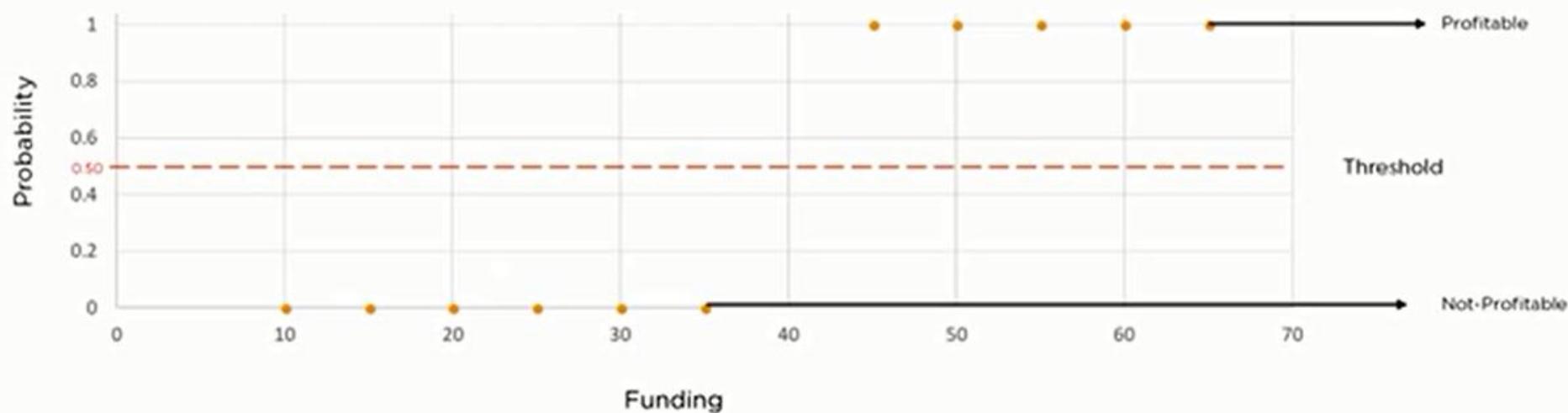
Profitable

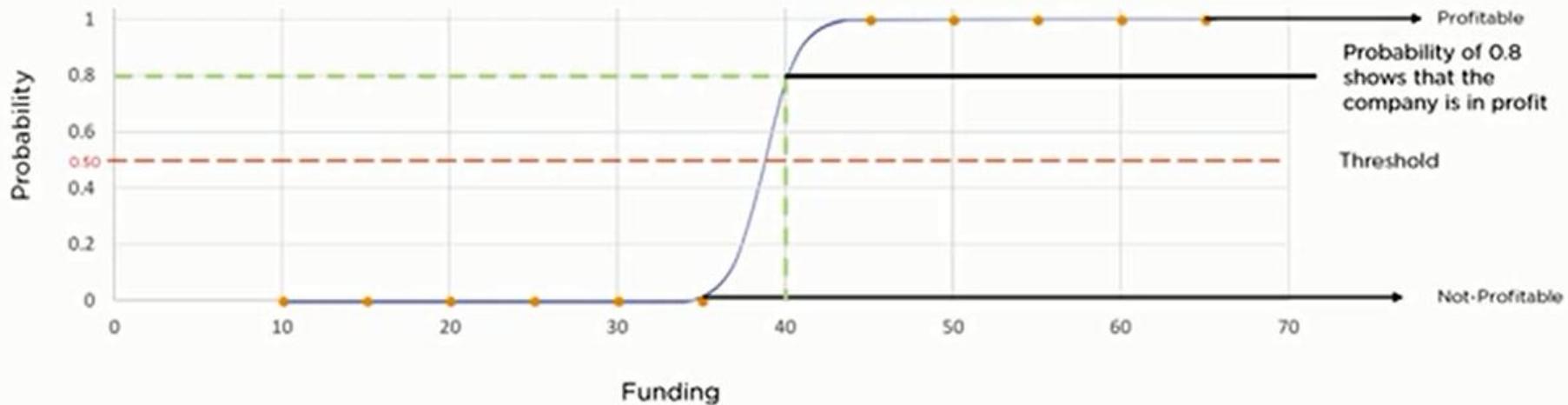
Not-Profitable

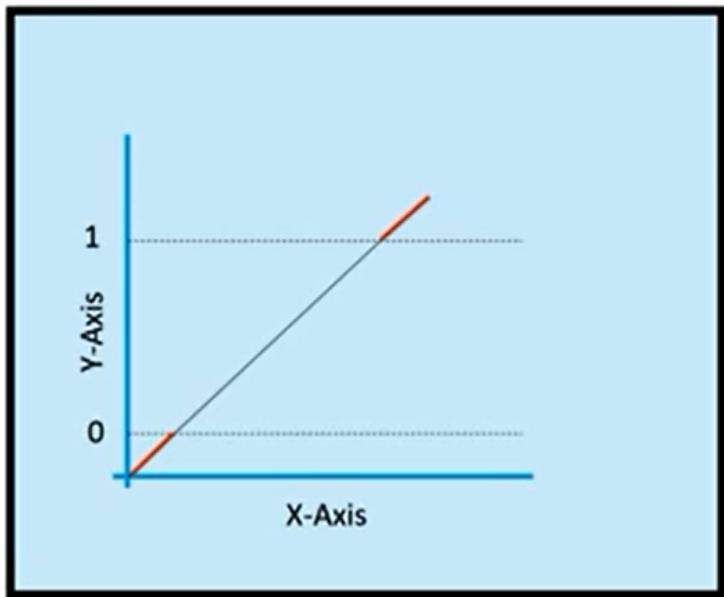
But this graph does not tell whether the startup will be profitable (Yes or No), it only states that with an increase in funding, the profit also increases



Hence, we make use of Logistic Regression which has two outcomes
(in our case profitable and not profitable)



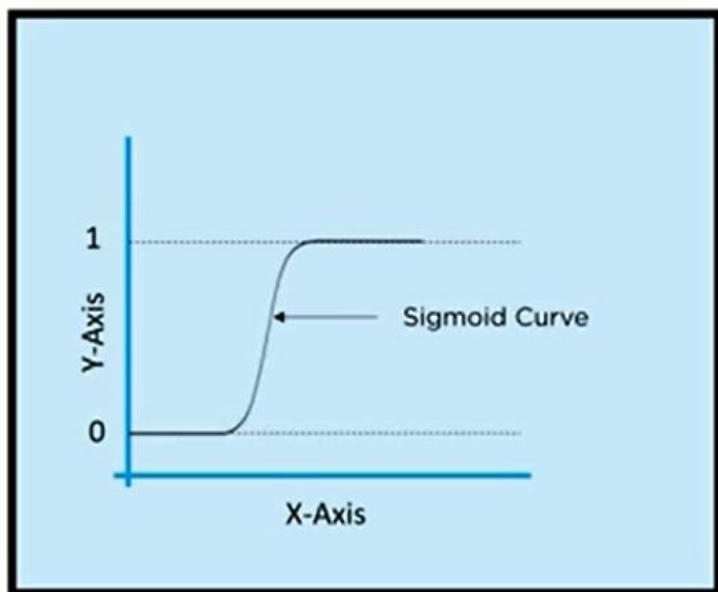




This is a Linear Regression graph

Using Linear Regression we cannot divide the output into categories

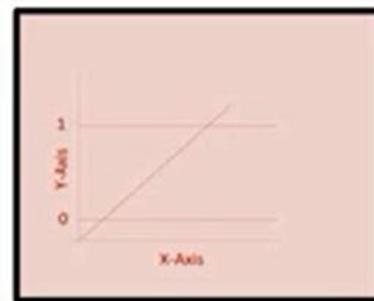
To divide the results into two categories, the linear line has to be clipped between 0 and 1



Once we clip the line we see that the resulting curve cannot be represented in a Linear equation

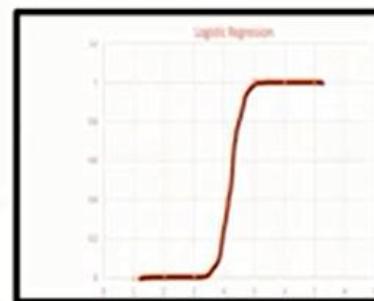
Hence, we make use of Sigmoid function

$$y = b_0 + b_1 * x$$



$$p = \frac{1}{1 + e^{-y}}$$

$$\ln\left(\frac{p}{1-p}\right) = b_0 + b_1 * x$$



Use case – College Admission using Logistic Regression

Problem Statement

We are given a dataset and we need to predict whether a candidate will get admission in desired college or not based on:

1. GPA
2. College Rank



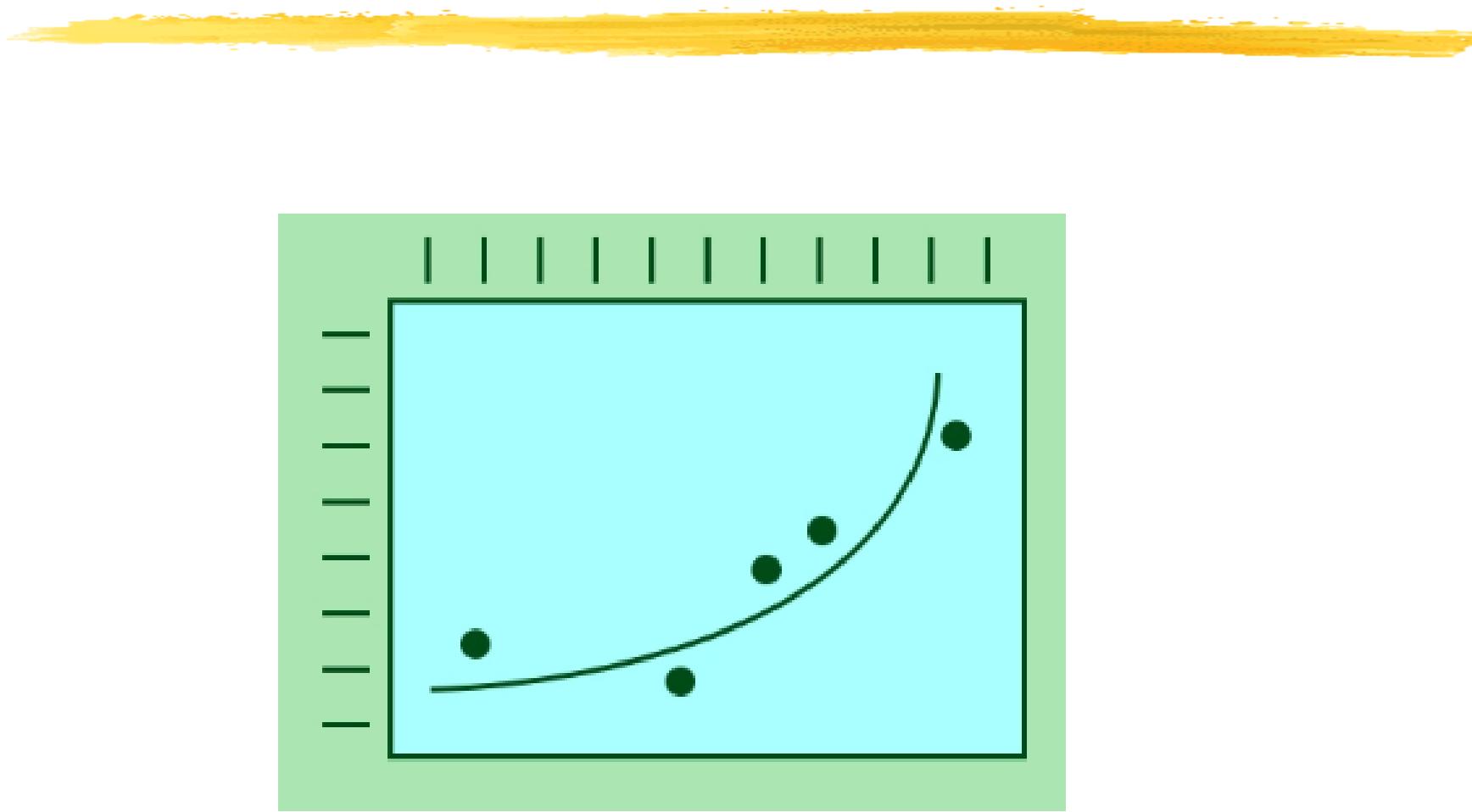


Polynomial Regression



- ⌘ Polynomial Regression is a special case of linear regression where the relationship between X and Y is modeled using a polynomial, rather than a line....
- ⌘ It can be used when the relationship between X and Y is nonlinear, although this is still considered to be a special case of Multiple Linear Regression.

- 
- ⌘ But what if your linear regression model cannot model the relationship between the target variable and the predictor variable?
 - ⌘ In other words, what if they don't have a linear relationship?





⌘ Linear Regression

$$Y = \theta_0 + \theta_1 x$$

⌘ Polynomial Regression

$$Y = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \dots + \theta_n x^n$$

θ_0 is the bias,

$\theta_1, \theta_2, \dots, \theta_n$ are the weights in the equation of the polynomial regression, and

n is the degree of the polynomial

Co-variance



- # Covariance is a measure of how much two random variables vary together
- # Lie between -infinity and +infinity
- # Measure of correlation

$$Covari(x, y) = \frac{\sum_{i=1}^n (x_i - x') (y_i - y')}{n - 1}$$

Correlation



- ⌘ Correlation is a statistical measure that indicates how strongly two variables are related.
- ⌘ Lie between -1 and +1

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

Covariance shows you how the two variables differ, whereas Correlation shows you how the two variables are related.

- ⌘ Pearson's correlation is a parametric measure of the linear association between two numeric variables.
- ⌘ Spearman's rank correlation is a non-parametric measure of the monotonic association (increase [or decrease] in the same direction, but not always at the same rate) between two numeric variables.
- ⌘ Kendall's rank correlation is another non-parametric measure of the association, based on concordance or discordance (refer to comparing two pairs of data points to see if they "match.") of x-y pairs.

K-means clustering in R

Can you distinguish between the 3 species of IRIS flower
using Machine Learning



Iris- Setosa



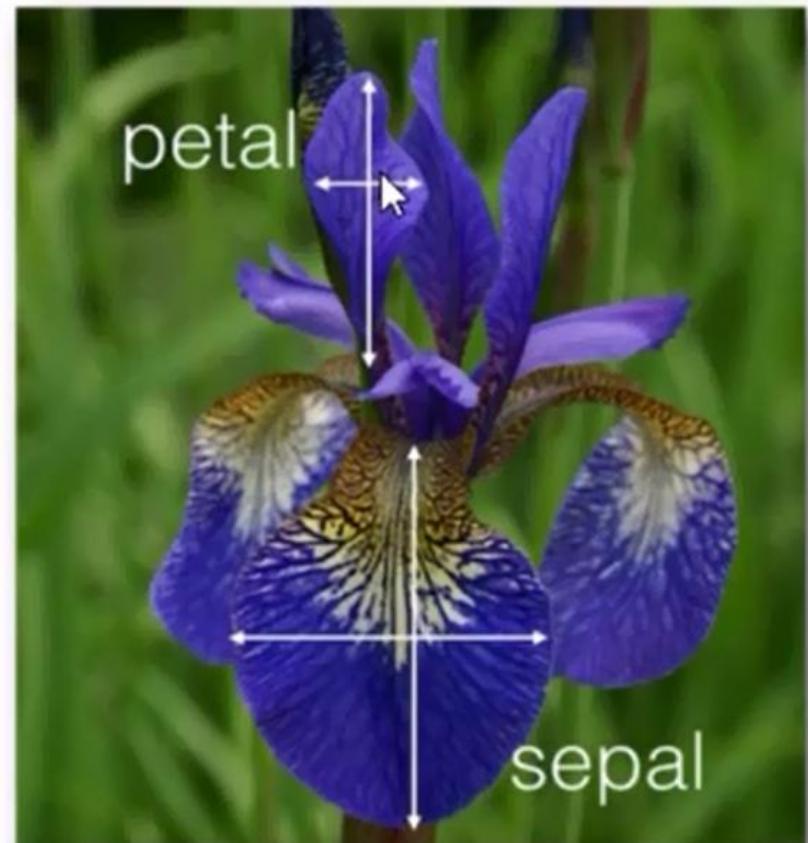
Iris- Versicolor



Iris- Virginica

Data Available

- 150 'Iris' flowers
- 4 flowers' Measurements



k-means Clustering



⌘ Features

- └ Initial set of clusters randomly chosen
- └ Iteratively, items are moved among sets of clusters until the desired set is reached
- └ High degree of similarity among elements in a cluster is obtained
- └ Given a cluster $K_i = \{t_{i1}, t_{i2}, \dots, t_{im}\}$,
The cluster mean : $m_i = (1/m)(t_{i1} + \dots + t_{im})$

⌘ Strength

- └ Time complexity $O(tkn)$
- └ Often terminates at a local optimum

⌘ Weakness

- └ Does not work on categorical data
- └ Only convex-shaped cluster are found
- └ Need to specify k , the number of clusters, in advance
- └ Unable to handle noisy data and outliers

⌘ K-modes

- └ One variation of K-means
- └ Handle categorical data
- └ Uses modes, instead of using means

k-means Clustering

Input:

$D = \{t_1, t_2, \dots, t_n\}$ // Set of elements

A // Adjacency matrix showing distance between elements.

k // Number of desired clusters.

Output:

K // Set of clusters.

K-Means Algorithm:

assign initial values for means m_1, m_2, \dots, m_k ;

repeat

 assign each item t_i to the cluster which has the closest mean ;

 calculate new mean for each cluster;

until convergence criteria is met;

k-means Clustering

- ⌘ $\{2,4,10,12,3,20,30,11,25\}$, $k=2$
- ⌘ Randomly assign means : $m_1=2$, $m_2=4$

m_1	m_2	K_1	K_2
2	4	{2,3}	{4,10,12,20,30,11,25}
2.5	16	{2,3,4}	{10,12,20,30,11,25}
3	18	{2,3,4,10}	{12,20,30,11,25}
4.75	19.6	{2,3,4,10,11,12}	{20,30,25}
7	25	{2,3,4,10,11,12}	{20,30,25}

- ⌘ Stop as the clusters with these means are the same
- ⌘ Our answer
 - ℳ $K_1 = \{2,3,4,10,11,12\}$, $K_2 = \{20,30,25\}$

Hypothesis Testing

- # Hypothesis – Idea that can be tested.
- # It is a statistical method that is used in making statistical decisions using experimental data.
- # Hypothesis Testing is basically an assumption that we make about the population parameter.
- # **Null hypothesis (H_0):**
 - ↗ A null hypothesis is a type of hypothesis used in statistics that proposes that no statistical significance exists in a set of given observations.
- # **Alternative hypothesis (H_a or H_1):**
 - ↗ An Alternative hypothesis is a type of hypothesis used in statistics that proposes that there is statistical significance exists in a set of given observations.

t - test



- ⌘ Consider a telecom company that has two service centers in the city.
- ⌘ The company wants to find whether the average time required to service a customer is the same in both stores.
- ⌘ The company measures the average time taken by 50 random customers in each store.
- ⌘ Store A takes 22 minutes while Store B averages 25 minutes.
- ⌘ *Can we say that Store A is more efficient than Store B in terms of customer service?*

- 
- ⌘ *This is where the t-test comes into play.*
 - ⌘ *It helps us to understand if the difference between two sample means is actually real or simply due to chance.*



⌘ Types of t-tests

- ─ One sample t-test
- ─ Independent two-sample t-test
- ─ Paired sample t-test



⌘ One sample t-test

- ↗ In a one-sample t-test, we compare the average (or mean) of one group against the set average (or mean).
- ↗ This set average can be any theoretical value (or it can be the population mean).

- ⌘ Consider the following example – A research scholar wants to determine if the average eating time for a (standard size) burger differs from a set value.
- ⌘ Let's say this value is 10 minutes.
- ⌘ How do you think the research scholar can go about determining this?





❖ He/she can broadly follow the below steps:

- ❖ Select a group of people
- ❖ Record the individual eating time of a standard size burger
- ❖ Calculate the average eating time for the group
- ❖ Finally, compare that average value with the set value of 10.


$$t = \frac{m - \mu}{s/\sqrt{n}}$$

where,

- ◻ t = t-statistic
- ◻ m = mean of the group
- ◻ μ = theoretical value or population mean
- ◻ s = standard deviation of the group
- ◻ n = group size or sample size

- 
- ⌘ Once we have calculated the t-statistic value, the next task is to compare it with the critical value of the t-test.
 - ⌘ We can find this in the below t-test table against the degree of freedom ($n-1$) and the level of significance.
 - ⌘ Degrees of freedom
 - ℳ the number of values that are free to vary in a data set

Implementing the One-Sample t-test in R



- ⌘ A mobile manufacturing company has taken a sample of mobiles of the same model from the previous month's data.
- ⌘ They want to check whether the average screen size of the sample differs from the desired length of 10 cm.

- 
- ⌘ t-statistic -> -0.39548.
 - ⌘ Degree of freedom here is 999 and the confidence interval is 95%.
 - ⌘ The t-critical value is 1.962.
 - ⌘ **Since the t-statistic is less than the t-critical value, we fail to reject the null hypothesis and can conclude that the average screen size of the sample does not differ from 10 cm.**
 - ⌘ We can also verify this from the p-value, which is greater than 0.05.
 - ⌘ Therefore, we fail to reject the null hypothesis at a 95% confidence interval.

Independent Two-Sample t-test



- # The two-sample t-test is used to compare the means of two different samples.
- # Let's say we want to compare the average height of the male employees to the average height of the females.
- # Of course, the number of males and females should be equal for this comparison.
- # This is where a two-sample t-test is used.

$$t = \frac{m_A - m_B}{\sqrt{\frac{s^2}{n_A} + \frac{s^2}{n_B}}}$$

where,

m_A and m_B are the means of two different samples

n_A and n_B are the sample sizes

s^2 is an estimator of the common variance of two samples, such as:

$$s^2 = \frac{\sum (x - m_A)^2 + \sum (x - m_B)^2}{n_A + n_B - 2}$$

- 
- ⌘ For this section, we will work with data about two samples of the various models of a mobile phone.
 - ⌘ We want to check whether the mean screen size of sample 1 differs from the mean screen size of sample 2.

- 
- ⌘ We can confirm that the t-statistic is again less than the t-critical value so we fail to reject the null hypothesis.
 - ⌘ Hence, we can conclude that there is no difference between the mean screen size of both samples.
 - ⌘ We can verify this again using the p-value.
 - ↗ It comes out to be greater than 0.05, therefore we fail to reject the null hypothesis at a 95% confidence interval.
 - ↗ There is no difference between the mean of the two samples.

Paired t-test



- ⌘ Here, we measure one group at two different times.
- ⌘ We compare separate means for a group at two different times or under two different conditions.
- ⌘ A certain manager realized that the productivity level of his employees was trending significantly downwards.
- ⌘ This manager decided to conduct a training program for all his employees with the aim of increasing their productivity levels.

- 
- ⌘ How will the manager measure if the productivity levels increased?
 - ↗ Just compare the productivity level of the employees before versus after the training program.
 - ⌘ Here, we are comparing the same sample (the employees) at two different times (before and after the training).


$$t = \frac{m}{s/\sqrt{n}}$$


where,

- ❖ t = t-statistic
 - ❖ m = mean of the group
 - ❖ s = standard deviation of the group
 - ❖ n = group size or sample size
- ❖ Degree of freedom = n – 1

- 
- ⌘ The manager of a tyre manufacturing company wants to compare the rubber material for two lots of tyres.
 - ⌘ One way to do this – check the difference between average kilometers covered by one lot of tyres until they wear out.

- 
- ⌘ The p-value is less than 0.05.
 - ⌘ **We can reject the null hypothesis at a 95% confidence interval and conclude that there is a significant difference between the means of tyres before and after the rubber material replacement.**
 - ⌘ *The negative mean in the difference depicts that the average kilometers covered by tyre 2 are more than the average kilometers covered by tyre 1.*

Association Rules



- ⌘ Association rules are used to show the relationships between data items.
- ⌘ Association rules detect common usage of data items.
- ⌘ E.g. The purchasing of one product when another product is purchased represents an association rule.

Association Rules



- ⌘ Association rules have most direct application in the retail businesses.
- ⌘ Association rules used to assist in marketing, advertising, floor placements and inventory control.
- ⌘ Eg:
 - └ From the transaction history several association rules can be derived.
 - └ E.g. 100% of the time that PeanutButter is purchased, so is bread.
 - └ 33% of the time PeanutButter is purchased, Jelly is also purchased.

Association Rules - Example

Transaction	Items
t_1	Bread, Jelly, PeanutButter
t_2	Bread, PeanutButter
t_3	Bread, Milk, PeanutButter
t_4	Beer, Bread
t_5	Beer, Milk

Association Rules - Example



- # Database in which Association rule is to be found can be viewed as a set of tuples, where each tuple contains a set of items.
- # Here, each tuple represents the list of items purchased at one time.
- # *Support:*
 - ↗ The Support of an item (or set of items) is the percentage of transactions in which that item (or items) occurs.

Association Rule - Definition

- # Given a set of items $I = \{I_1, I_2, \dots, I_m\}$ and a database of transactions $D = \{t_1, t_2, \dots, t_m\}$ where $t_i = \{I_{i1}, I_{i2}, \dots, I_{ik}\}$ and $I_{ij} \in I$, an association rule is an implication of the form $X \rightarrow Y$ where $X, Y \subseteq I$ are sets of items called itemsets and $X \cap Y = \emptyset$.
- # The ***Support(S)*** for an association rule $X \rightarrow Y$ is the percentage of transactions in the database that contains $X \cup Y$.
- # **NOTE:** Support of $X \Rightarrow Y$ is same as support of $X \cup Y$.

Association Rule - Introduction



⌘ ***Confidence (or) Strength***

↗ The Confidence (or) Strength (α) for an Association rule $X \rightarrow Y$ is the ratio of number of transactions that contain $X \cup Y$ to the number of transactions that contain X.

Selecting Association Rules



- # The selection of association rules is based on Support and Confidence.
- # Confidence measures the strength of the rule, Whereas support measures how often it should occur in the database.
- # Typically large confidence values and a smaller support are used.
- # Rules that satisfy both minimum support and minimum confidence are called strong rules

Association Rule - Example

TABLE 6.2: Support of All Sets of Items Found in Table 6.1

Set	Support	Set	Support
Beer	40	Beer, Bread, Milk	0
Bread	80	Beer, Bread, PeanutButter	0
Jelly	20	Beer, Jelly, Milk	0
Milk	40	Beer, Jelly, PeanutButter	0
PeanutButter	60	Beer, Milk, PeanutButter	0
Beer, Bread	20	Bread, Jelly, Milk	0
Beer, Jelly	0	Bread, Jelly, PeanutButter	20
Beer, Milk	20	Bread, Milk, PeanutButter	20
Beer, PeanutButter	0	Jelly, Milk, PeanutButter	0
Bread, Jelly	20	Beer, Bread, Jelly, Milk	0
Bread, Milk	20	Beer, Bread, Jelly, PeanutButter	0
Bread, PeanutButter	60	Beer, Bread, Milk, PeanutButter	0
Jelly, Milk	0	Beer, Jelly, Milk, PeanutButter	0
Jelly, PeanutButter	20	Bread, Jelly, Milk, PeanutButter	0
Milk, PeanutButter	20	Beer, Bread, Jelly, Milk, PeanutButter	0
Beer, Bread, Jelly	0		

Association Rule - Example

$X \Rightarrow Y$	s	α
Bread \Rightarrow PeanutButter	60%	75%
PeanutButter \Rightarrow Bread	60%	100%
Beer \Rightarrow Bread	20%	50%
PeanutButter \Rightarrow Jelly	20%	33.3%
Jelly \Rightarrow PeanutButter	20%	100%
Jelly \Rightarrow Milk	0%	0%

Apriori algorithm



- ⌘ Most well known Association rule algorithm and is used in most commercial products.
- ⌘ Uses the property called ***Large Itemset property***.
 - └ Any subset of a large itemset must be large.

- 
- ⌘ To perform Association Rule Mining in R, we use the arules and the arulesViz packages in R.
 - ⌘ If you don't have these packages installed in your system, please use the following commands to install them.

```
> install.packages("arules")
> install.packages("arulesViz")
```

Decision tree based algorithms

Given:

- ☒ $D = \{t_1, \dots, t_n\}$ where $t_i = \langle t_{i1}, \dots, t_{ih} \rangle$
- ☒ Database schema contains $\{A_1, A_2, \dots, A_h\}$
- ☒ Classes $C = \{C_1, \dots, C_m\}$

Decision or Classification Tree is a tree associated with D such that

- ☒ Each internal node is labeled with attribute, A_i
 - ☒ Each arc is labeled with predicate which can be applied to attribute at parent
 - ☒ Each leaf node is labeled with a class, C_j
- ⌘ Solving the Classification problem using Decision trees is a two step process:
- ☒ Decision tree Induction
 - ☒ Construct a DT using training data
 - ☒ For each t_i belongs to D, apply the DT to determine its class.

Decision tree based algorithms

Input:

D //Training data

Output:

T //Decision Tree

DTBuild Algorithm:

//Simplistic algorithm to illustrate naive approach to building DT

$T = \emptyset;$

Determine best splitting criterion;

$T =$ Create root node node and label with splitting attribute;

$T =$ Add arc to root node for each split predicate and label;

for each arc do

$D =$ Database created by applying splitting predicate to D ;

if stopping point reached for this path then

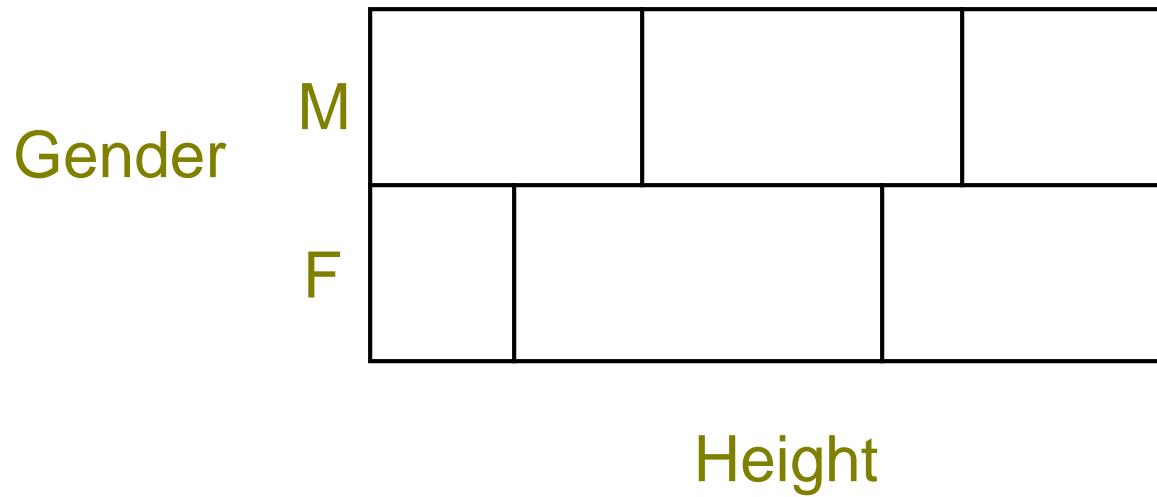
$T' =$ Create leaf node and label with appropriate class;

else

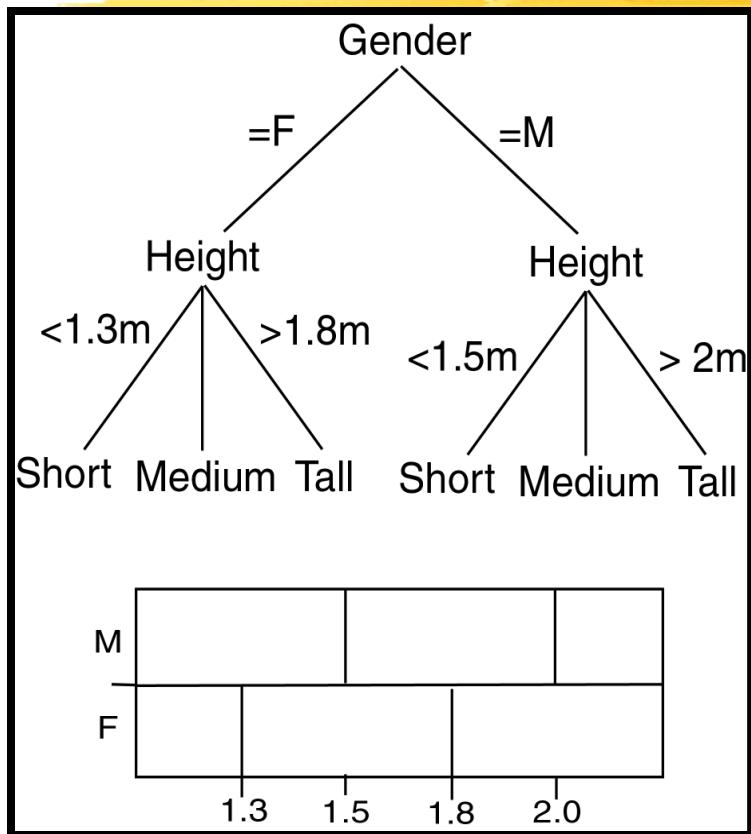
$T' = DTBuild(D);$

$T =$ Add T' to arc;

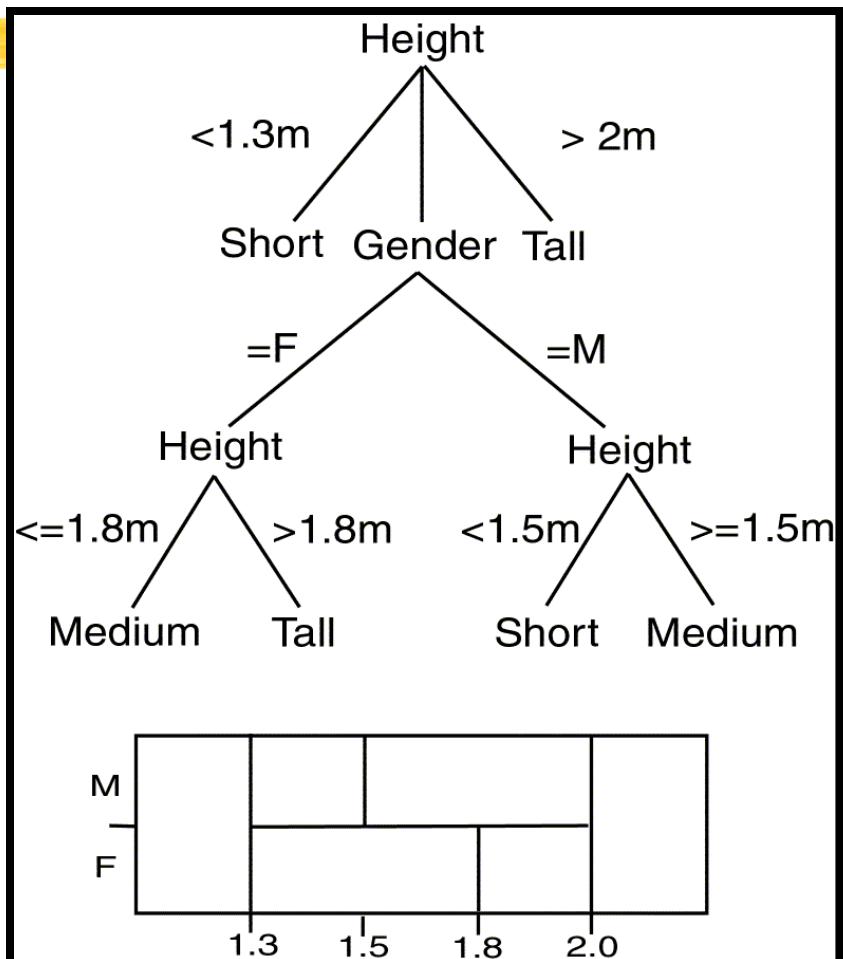
DT Splits Area



Comparing DT's



Balanced

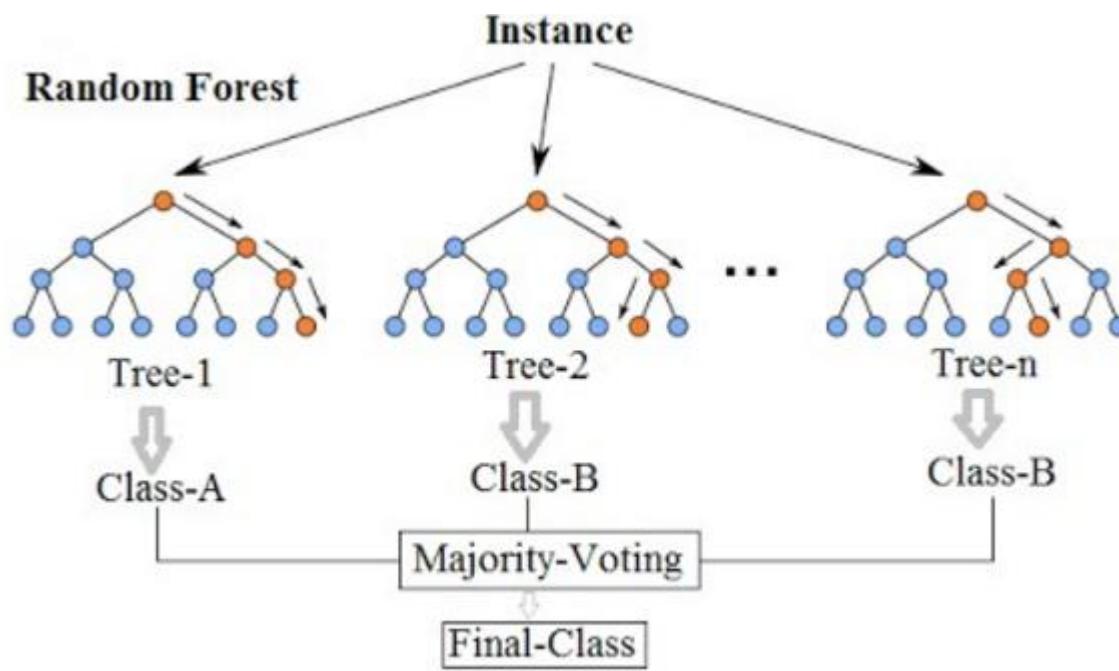


Deep

Random Forests



- ⌘ Random forest is a supervised learning algorithm which is used for both classification as well as regression.
- ⌘ Mainly used for classification problems.
- ⌘ Forest is made up of trees and more trees means more robust forest.
- ⌘ Similarly, random forest algorithm creates decision trees on data samples and then gets the prediction from each of them and finally selects the best solution by means of voting.
- ⌘ It is an ensemble method which is better than a single decision tree because it reduces the over-fitting by averaging the result.



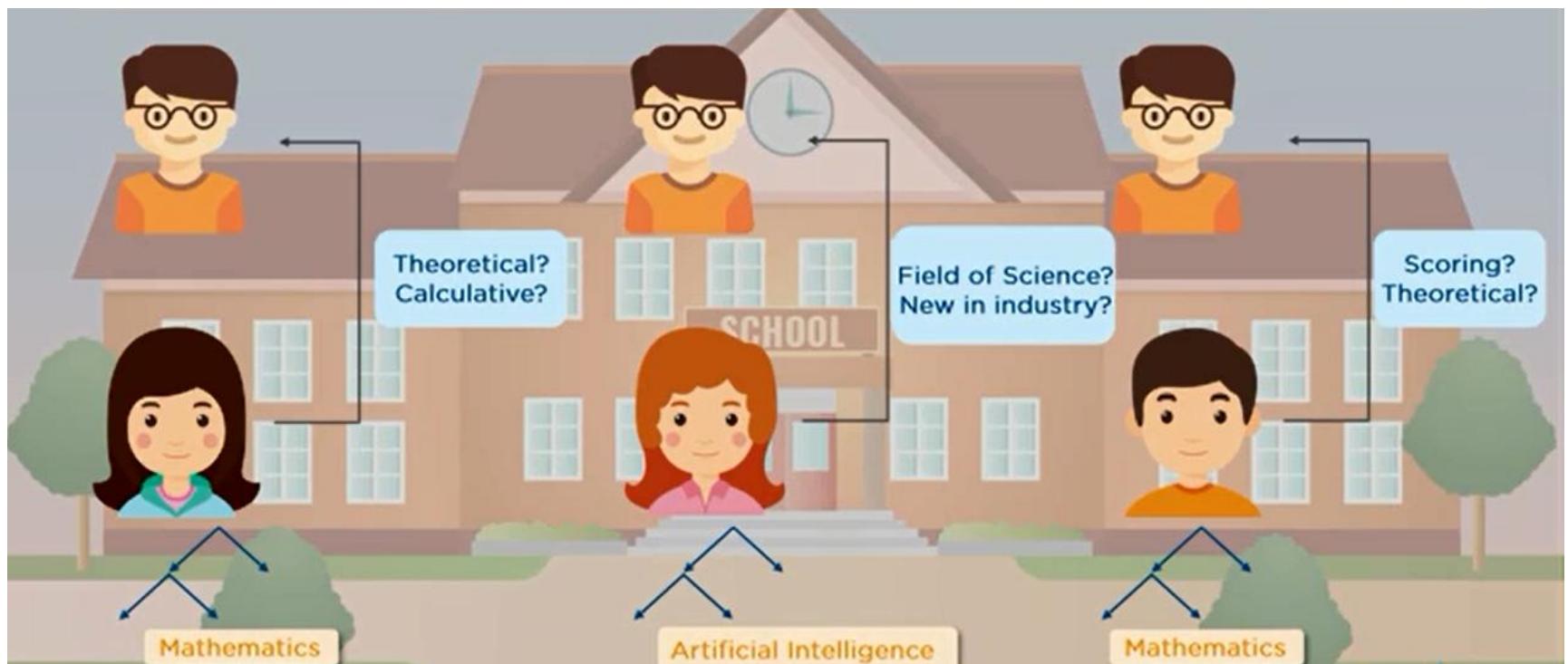
Random Forest



- ⌘ Random forest is an ensemble machine learning algorithm.
- ⌘ It operates by building multiple decision trees.
- ⌘ They work for both
 - ↗ Classification
 - ↗ Regression



Sam is confused on
which course to take up



- 
- ⌘ In Banking, it is used to predict fraudulent customers.
 - ⌘ It is used in analysing symptoms of the patients and detecting the disease.
 - ⌘ In e-commerce, the recommendations are based on customer activity.
 - ⌘ Stock market trends can be analysed to predict profit or loss.

ANOVA

- # **Analysis of Variance (ANOVA)** is a parametric statistical technique used to compare datasets.
- # This technique was invented by R.A. Fisher, and is thus often referred to as Fisher's ANOVA, as well.
- # It is similar in application to techniques such as t-test, in that it is used to compare means and the relative variance between them.
- # However, analysis of variance (ANOVA) is best applied where more than 2 populations or samples are meant to be compared.



❖ **One way analysis:**

- ❖ When we are comparing more than three groups based on one factor variable, then it said to be one way analysis of variance (ANOVA).
- ❖ For example, if we want to compare whether or not the mean output of three workers is the same based on the working hours of the three workers.



❖ Two way analysis

- ❖ When factor variables are more than two, then it is said to be two way analysis of variance (ANOVA).
- ❖ For example, based on working condition and working hours, we can compare whether or not the mean output of three workers is the same.



⌘ K-way analysis

When factor variables are k, then it is said to be the k-way analysis of variance (ANOVA).

- 
- # Group1 <- c(2,3,7,2,6) - stress level during regular time
 - # Group2 <- c(10,8,7,5,10) - stress when layoffs are announced.
 - # Group3 <- c(10,13,14,13,15) - stress level after announcement of layoffs.
 - # Check if there is any statistical relationship between these groups of employees?
 - # Null hypothesis
 - ◻ There is no statistically significant relationship between three groups.

Regression Vs. ANOVA



- ⌘ Regression is the statistical model that is used to predict a continuous outcome on the basis of one or more **continuous** predictor variables.
- ⌘ Whereas, ANOVA is the statistical model that you use to predict a continuous outcome on the basis of one or more **categorical** predictor variables.

Dealing with Missing values



- ⌘ A common task in data analysis is dealing with missing values.
- ⌘ In R, missing values are often represented by NA or some other value that represents missing values (i.e. 99).

Test for missing values



- ⌘ To identify missing values use `is.na()` which returns a logical vector with TRUE in the element locations that contain missing values represented by NA.
- ⌘ `is.na()` will work on vectors, lists, matrices, and data frames.

```
# vector with missing data
x <- c(1:4, NA, 6:7, NA)
x
## [1] 1 2 3 4 NA 6 7 NA

is.na(x)
## [1] FALSE FALSE FALSE FALSE  TRUE FALSE FALSE  TRUE

# data frame with missing data
df <- data.frame(col1 = c(1:3, NA),
                  col2 = c("this", NA, "is", "text"),
                  col3 = c(TRUE, FALSE, TRUE, TRUE),
                  col4 = c(2.5, 4.2, 3.2, NA),
                  stringsAsFactors = FALSE)
```

```
# identify NAs in full data frame
is.na(df)
##          col1   col2   col3   col4
## [1,] FALSE FALSE FALSE FALSE
## [2,] FALSE  TRUE FALSE FALSE
## [3,] FALSE FALSE FALSE FALSE
## [4,]  TRUE FALSE FALSE  TRUE

# identify NAs in specific data frame column
is.na(df$col4)
## [1] FALSE FALSE FALSE  TRUE
```



⌘ For data frames, a convenient shortcut to compute the total missing values in each column is to use `colSums()`.

```
colSums(is.na(df))  
## col1 col2 col3 col4  
##     1     1     0     1
```

Recode missing values



```
# recode missing values with the mean
# vector with missing data
x <- c(1:4, NA, 6:7, NA)
x
## [1] 1 2 3 4 NA 6 7 NA

x[is.na(x)] <- mean(x, na.rm = TRUE)

round(x, 2)
## [1] 1.00 2.00 3.00 4.00 3.83 6.00 7.00 3.83
```

```
# data frame with missing data
df <- data.frame(col1 = c(1:3, NA),
                  col2 = c("this", NA, "is", "text"),
                  col3 = c(TRUE, FALSE, TRUE, TRUE),
                  col4 = c(2.5, 4.2, 3.2, NA),
                  stringsAsFactors = FALSE)

df$col4[is.na(df$col4)] <- mean(df$col4, na.rm = TRUE)
df
##   col1 col2  col3 col4
## 1     1 this  TRUE  2.5
## 2     2 <NA> FALSE  4.2
## 3     3   is  TRUE  3.2
## 4    NA text  TRUE  3.3
```

Exclude missing values



- ⌘ We can exclude missing values in a couple different ways.
- ⌘ First, if we want to exclude missing values from mathematical operations use the `na.rm = TRUE` argument.
- ⌘ If you do not exclude these values most functions will return an NA.

```
# A vector with missing values
x <- c(1:4, NA, 6:7, NA)

# including NA values will produce an NA output
mean(x)
## [1] NA

# excluding NA values will calculate the mathematical operation for all n
mean(x, na.rm = TRUE)
## [1] 3.833333
```

Non-linear least square



- # When modeling real world data for regression analysis, we observe that it is rarely the case that the equation of the model is a linear equation giving a linear graph.
- # Most of the times, the plot of the model gives a curve rather than a line.
- # The goal of both linear and non-linear regression is to adjust the values of the model's parameters to find the line or curve that comes closest to your data.

- 
- ⌘ In Least Square regression, we establish a regression model in which the sum of the squares of the vertical distances of different points from the regression curve is minimized.
 - ⌘ We generally start with a defined model and assume some values for the coefficients.
 - ⌘ We then apply the **nls()** function of R to get the more accurate values along with the confidence intervals.



```
nls(formula, data, start)
```

Following is the description of the parameters used –

- **formula** is a nonlinear model formula including variables and parameters.
- **data** is a data frame used to evaluate the variables in the formula.
- **start** is a named list or named numeric vector of starting estimates.



Working with R scripts



- ⌘ R-studio's built-in text editor.
- ⌘ Before entering commands in console they are scripted in R script.
- ⌘ Necessary to document or automate a task.
- ⌘ It helps to
 - ↗ Reproduce work
 - ↗ Instruct on how to perform a task
 - ↗ Save time through rapid iteration
 - ↗ Evaluates incremental changes
 - ↗ Reduces the chance of human error

Working with R scripts



- ⌘ R-studio's built-in text editor.
- ⌘ Before entering commands in console they are scripted in R script.
- ⌘ Necessary to document or automate a task.
- ⌘ It helps to
 - ↗ Reproduce work
 - ↗ Instruct on how to perform a task
 - ↗ Save time through rapid iteration
 - ↗ Evaluates incremental changes
 - ↗ Reduces the chance of human error



⌘ Basic tips for scripting

- ─ Place each function on a separate line
- ─ If a function has long list of arguments, place each argument on a separate line
- ─ A command can be executed from the text editor by placing the cursor on a line and typing Ctrl+Enter or by clicking the run button.
- ─ An entire R script file can be executed by clicking the source button.

```
> # Get Input values  
> ip <- mtcars[,c('wt','hp')]  
> print(head(ip))  
          wt  hp  
Mazda RX4     2.620 110  
Mazda RX4 Wag 2.875 110  
Datsun 710    2.320  93  
Hornet 4 Drive 3.215 110  
Hornet Sportabout 3.440 175  
Valiant       3.460 105  
> View(ip)
```



```
> final <- cbind(actual = sales$Profit, predicted = pred)
> View(final)
```



⌘ANOVA is a parametric method appropriate for comparing the means for 2 or more independent populations.

Poisson Regression



- ⌘ Poisson Regression models are best used for modeling events where the outcomes are counts. Or, more specifically, *count data*: discrete data with non-negative integer values that count something, like the number of times an event occurs during a given timeframe or the number of people in line at the grocery store.
- ⌘ *Count data* can also be expressed as *rate data*, since the number of times an event occurs within a timeframe can be expressed as a raw count (i.e. "In a day, we eat three meals") or as a rate ("We eat at a rate of 0.125 meals per hour").

- 
- ⌘ Poisson Regression helps us analyze both count data and rate data by allowing us to determine which explanatory variables (X values) have an effect on a given response variable (Y value, the count or a rate).
 - ⌘ For example, Poisson regression could be applied by a grocery store to better understand and predict the number of people in a line.

Poisson Distribution

