CSE1006
Foundations of Data Analytics

Module - 4

Data Analysis

- **Data Import:** Reading Data, Writing Data in R, data cleaning and summarizing with dplyr package, Exploratory Data
- **Analysis:** Box plot, Histogram, Pie graph, Line chart,Barplot,Scatter Plot

Importing Data in R Script

- We can read external datasets and operate with them in our R environment by importing data into an R script.
- R offers a number of functions for importing data from various file formats.
- First, let's consider a data set that we can use for the demonstration. For this demonstration, we will use two examples of a single dataset, one in **.csv** form and another **.txt**

data1 ->

A	B	C
6.475	6	62.1
10.125	18	74.7
9.55	16	69.7
11.125	14	71
4.8	5	56.9
6.225	11	58.7
4.95	8	63.3
7.325	11	70.4
8.875	15	70.5
6.8	11	59.2

Reading a Comma-Separated Value(CSV) File:

Method 1: Using read.csv() Function Read CSV Files into R

Syntax: read.csv(file.choose(), header)
The function read.csv() has two parameters:

file.choose(): It opens a menu to choose a CSV file from the desktop.
header: It is to indicate whether the first row of the dataset is a variable name or not. Apply T/True if the variable name is present else put F/False.

```
# import and store the dataset in data1
data1 <- read.csv(file.choose(), header=T)
data1     # display the data
```

Reading a Comma-Separated Value(CSV) File:

Method 2: Using read.table() Function

Syntax: read. table(file.choose(), header, sep=“ , ”)

This function specifies how the dataset is separated, in this case we take sep=”, “ as an argument.

```
# import and store the dataset in data2
data2 <- read.table(file.choose(), header=T, sep=“,”)
# display the data
data2
```

Reading a Comma-Separated Value(CSV) File:

Output:

	A	B	C
1	6.475	6	62.1
2	10.125	18	74.7
3	9.550	16	69.7
4	11.125	14	71.0
5	4.800	5	56.9
6	6.225	11	58.7
7	4.950	8	63.3
8	7.325	11	70.4
9	8.875	15	70.5
10	6.800	11	59.2

Reading a Tab-Delimited(txt) File in R Programming Language:

Method 1: Using read.delim() Function

Syntax: read.delim(file.choose(), header)

The function has read.delim() two parameters:
file.choose(): It opens a menu to choose a CSV file from the desktop.

header: It is to indicate whether the first row of the dataset is a variable name or not. Apply T/True if the variable name is present else put F/False.

```
# import and store the dataset in data3
data3 <- read.delim(file.choose(), header=T)
# display the data
data3
```

Reading a Tab-Delimited(txt) File in R Programming Language:

Method 2: Using read.table() Function

Syntax: read. table(file.choose(), header, sep=“ \t ”)

This function specifies how the dataset is separated, in this case we take sep=”\t“ as an argument.

```
# import and store the dataset in data2
data4 <- read.table(file.choose(), header=T, sep=“\t”)
# display the data
data4
```

Reading a Tab-Delimited(txt) File in R Programming Language:

Output:

	A	B	C
1	6.475	6	62.1
2	10.125	18	74.7
3	9.550	16	69.7
4	11.125	14	71.0
5	4.800	5	56.9
6	6.225	11	58.7
7	4.950	8	63.3
8	7.325	11	70.4
9	8.875	15	70.5
10	6.800	11	59.2

Reading a excel File in R Programming Language:

Method 1: Using read_excel() from readxl

read_excel() function is basically used to import/read an Excel file and it can only be accessed after importing the readxl library in R language.

Syntax: read_excel(path)
The read_excel() method extracts the data from the Excel file and returns it as an R data frame.

```
library(readxl)
Data_gfg <- read_excel("Data_gfg.xlsx")
Data_gfg
```

Reading a excel File in R Programming Language:

Method 1: Using read.xlsx() from xlsx

read.xlsx() function is imported from the xlsx library of R language and used to read/import an excel file in R language.

Syntax: read.xlsx(path)

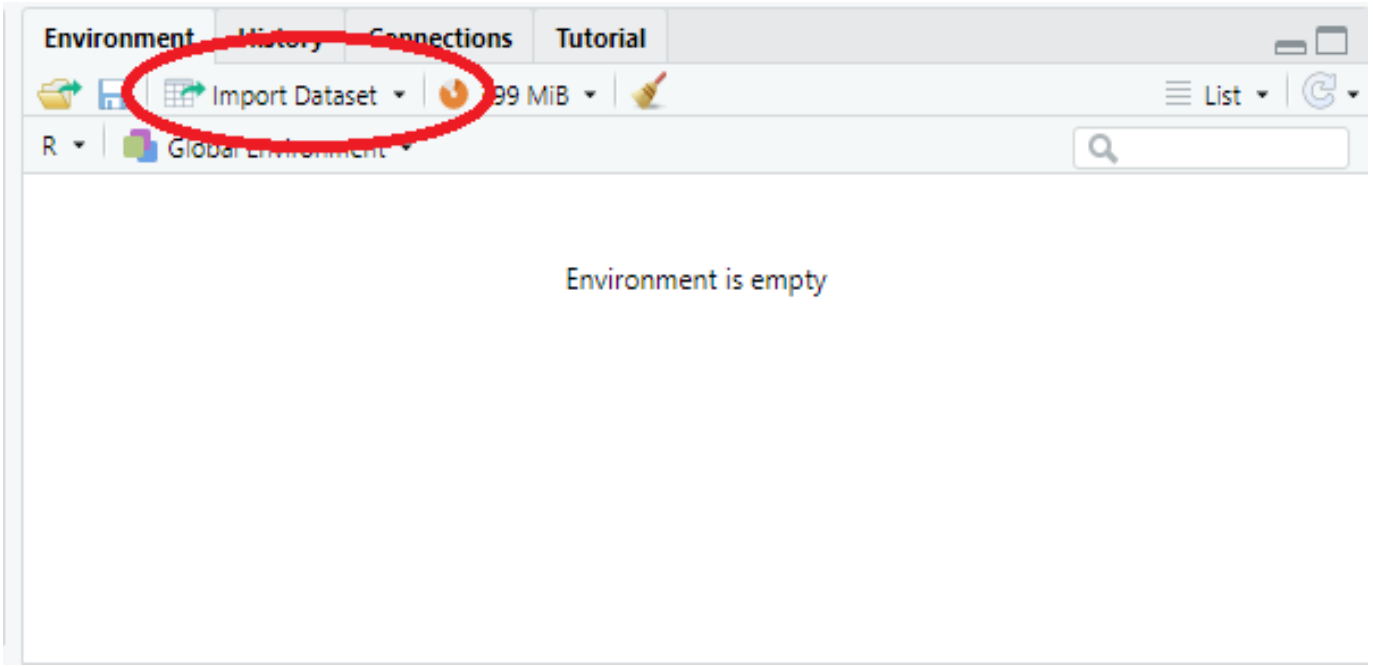
The read_excel() method extracts the data from the Excel file and returns it as an R data frame.

```
Data_gfg <-read.xlsx('Data_gfg.xlsx’)
Data_gfg
```

Here we are going to import data through R studio with the following steps.

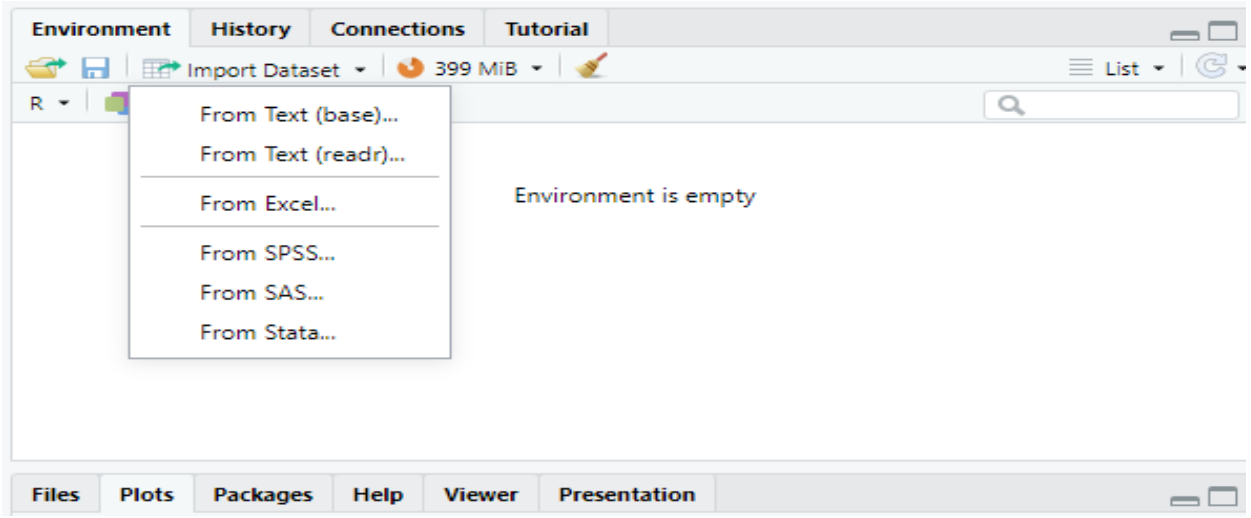
Steps:

1. From the Environment tab click on the Import Dataset Menu.



Steps:

2. Select the file extension from the option.



3. In the third step, a pop-up box will appear, either enter the file name or browse the desktop.
4. The selected file will be displayed on a new window with its dimensions.
5. In order to see the output on the console, type the filename.

Exporting Data from R Scripts

- When a program is terminated, the entire data is lost.
- Storing in a file will preserve one's data even if the program terminates.
- If one has to enter a large number of data, it will take a lot of time to enter them all. However, if one has a file containing all the data, he/she can easily access the contents of the file using a few commands in R.

Exporting data to a text file:

- One of the important formats to store a file is in a text file. R provides various methods that one can export data to a text file.

write.table():

- The R base function **write.table()** can be used to export a data frame or a matrix to a text file.
- In This section of R studio we get the data saved as the name that we gave in the code. and when we select that files we get this type of output.

write.table():

Syntax: write.table(x, file,sep = ” “, dec = “.”, row.names = TRUE, col.names = TRUE)

Parameters:

x: a matrix or a data frame to be written.

file: a character specifying the name of the result file.

sep: the field separator string, e.g., sep = “\t” (for tab-separated value).

dec: the string to be used as decimal separator. Default is “.”

row.names: either a logical value indicating whether the row names of x are to be written along with x, or a character vector of row names to be written.

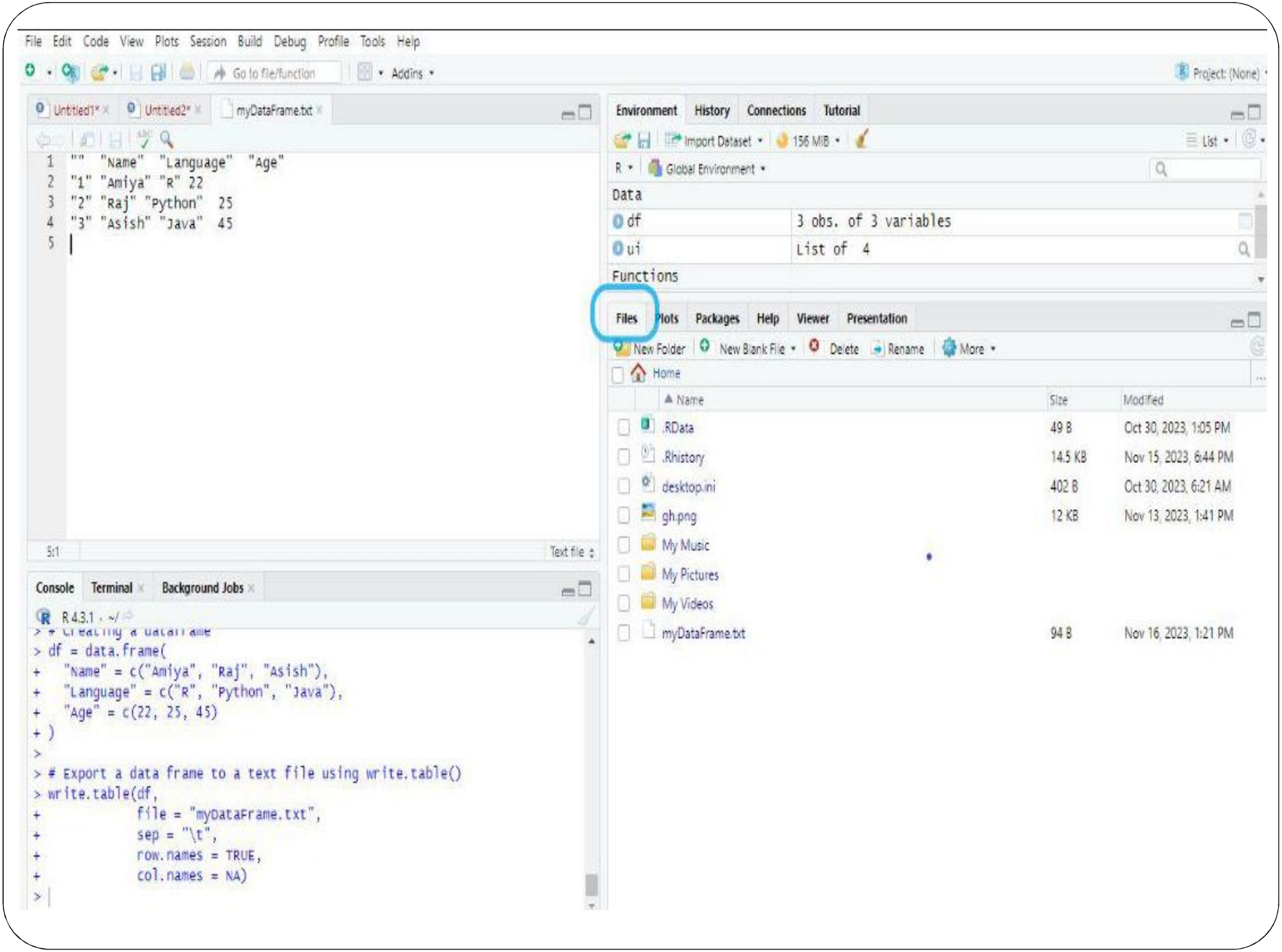
col.names: either a logical value indicating whether the column names of x are to be written along with x, or a character vector of column names to be written.

write.table():

Example:

```
# R program to illustrate Exporting data from R
# Creating a dataframe
df = data.frame( "Name" = c("Amiya", "Raj", "Asish"),
                 "Language" = c("R", "Python", "Java"),
                  "Age" = c(22, 25, 45))

# Export a data frame to a text file using write.table()
    write.table(df,  file = "myDataFrame.txt",
                     sep = "\t",
                     row.names = TRUE,
                     col.names = NA)
```


File Edit Code View Plots Session Build Debug Profile Tools Help
Go to file/function
Addins
Project: (None)
Untitled1*
Untitled2*
myDataFrame.txt
1 "" "Name" "Language" "Age"
2 "1" "Amiya" "R" 22
3 "2" "Raj" "Python" 25
4 "3" "Asish" "Java" 45
5
5:1
Text file
Environment
History
Connections
Tutorial
Import Dataset
156 MiB
List
R
Global Environment
Data
df
3 obs. of 3 variables
ui
List of 4
Functions
Files
Plots
Packages
Help
Viewer
Presentation
New Folder
New Blank File
Delete
Rename
More
Home
Name
Size
Modified
.RData
49 B
Oct 30, 2023, 1:05 PM
.Rhistory
14.5 KB
Nov 15, 2023, 6:44 PM
desktop.ini
402 B
Oct 30, 2023, 6:21 AM
gh.png
12 KB
Nov 13, 2023, 1:41 PM
My Music
My Pictures
My Videos
myDataFrame.txt
94 B
Nov 16, 2023, 1:21 PM
Console
Terminal
Background Jobs
R 4.3.1 · ~/
> df = data.frame(
+ "Name" = c("Amiya", "Raj", "Asish"),
+ "Language" = c("R", "Python", "Java"),
+ "Age" = c(22, 25, 45)
+)
>
> # Export a data frame to a text file using write.table()
> write.table(df,
+ file = "myDataFrame.txt",
+ sep = "\t",
+ row.names = TRUE,
+ col.names = NA)
>

write_tsv():

This **write_tsv()** method is also used for to export data to a tab separated (“\t”) values by using the help of readr package.

Syntax: write_tsv(file, path)

Parameters:

file: a data frame to be written

path: the path to the result file

Example:

```
# R program to illustrate Exporting data from R
# Importing readr library
library(readr)
# Creating a dataframe
df = data.frame( "Name" = c("Amiya", "Raj", "Asish"),
                 "Language" = c("R", "Python", "Java"),
                 "Age" = c(22, 25, 45) )
 # Export a data frame using write_tsv()
write_tsv(df, path = "MyDataFrame.txt")
```

Output:

```
Name    Language    Age
Amiya     R     22
Raj     Python      25
Asish     Java      45
```

write.csv():

This write.csv() method is recommendable for exporting data to a csv file. It uses “.” for the decimal point and a comma (“, ”) for the separator.

Syntax: write.csv(file, path)

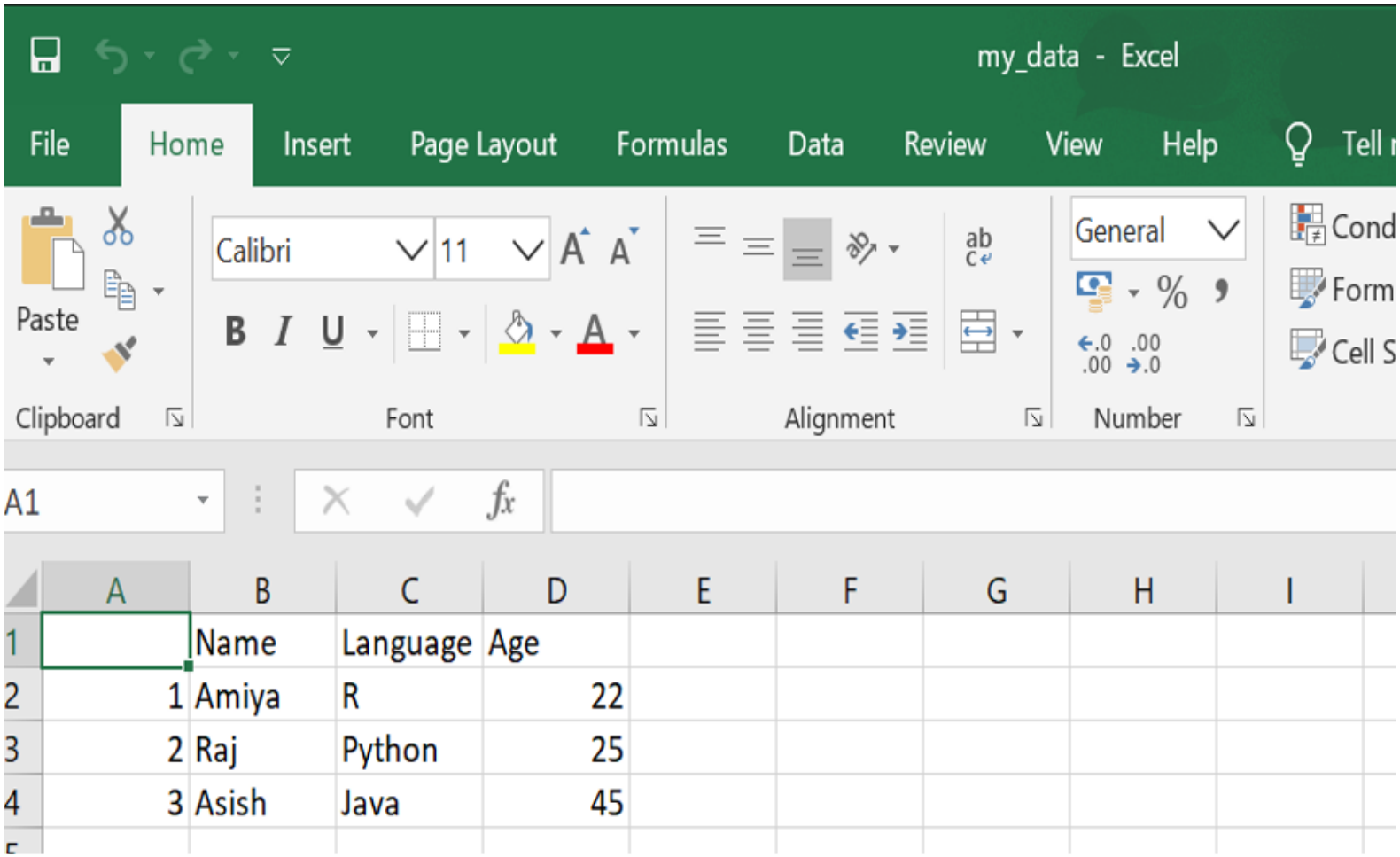
Parameters:

file: a data frame to be written

path: the path to the result file

Example:

```
# R program to illustrate Exporting data from R
# Importing readr library
library(readr)
# Creating a dataframe
df = data.frame( "Name" = c("Amiya", "Raj", "Asish"),
                 "Language" = c("R", "Python", "Java"),
                 "Age" = c(22, 25, 45) )
 # Export a data frame using write.csv()
write.csv(df, file = "My_Data.csv")
```


my_data - Excel
File
Home
Insert
Page Layout
Formulas
Data
Review
View
Help
Paste
Calibri
11
General
Clipboard
Font
Alignment
Number
A1
A B C D E F G H I
1 Name Language Age
2 1 Amiya R 22
3 2 Raj Python 25
4 3 Asish Java 45

write.csv2():

This method is much similar as write.csv() but it uses a comma (“, ”) for the decimal point and a semicolon (“;”) for the separator.

Syntax: write.csv2(file, path)

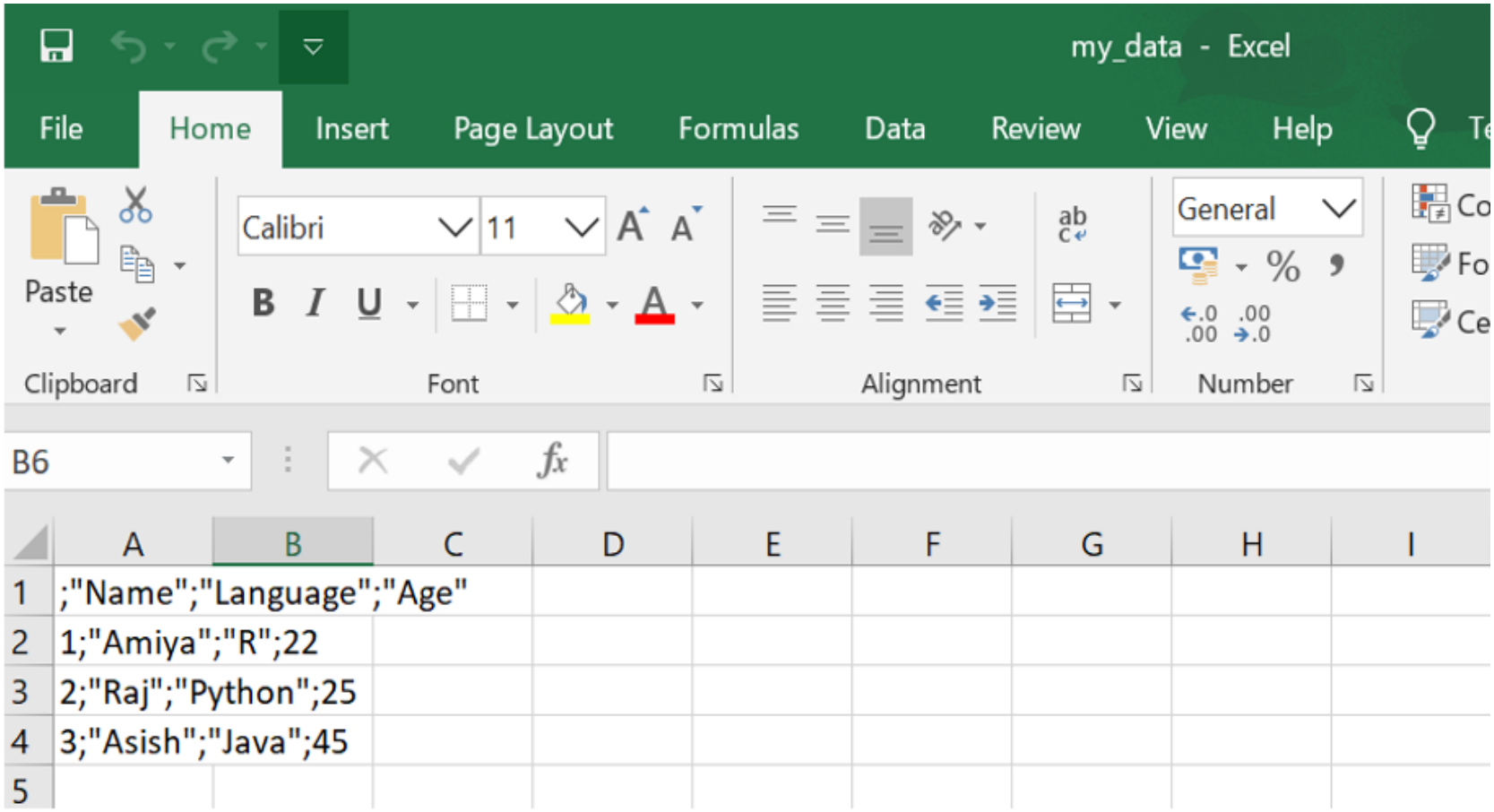
Parameters:

file: a data frame to be written

path: the path to the result file

Example:

```
# R program to illustrate Exporting data from R
# Importing readr library
library(readr)
# Creating a dataframe
df = data.frame( "Name" = c("Amiya", "Raj", "Asish"),
                 "Language" = c("R", "Python", "Java"),
                 "Age" = c(22, 25, 45) )
 # Export a data frame using write_tsv()
write.csv2(df, file = "My_Data.csv")
```


my_data - Excel
File
Home
Insert
Page Layout
Formulas
Data
Review
View
Help
Paste
Calibri
11
General
Clipboard
Font
Alignment
Number
B6
;"Name";"Language";"Age"
1;"Amiya";"R";22
2;"Raj";"Python";25
3;"Asish";"Java";45

Data cleaning and summarizing with dplyr package

dplyr is a powerful R-package to transform and summarize tabular data with rows and columns.

- The package contains a set of functions (or "verbs") that perform common data manipulation operations such as filtering for rows, selecting specific columns, re-ordering rows, adding new columns and summarizing data.
- In addition, dplyr contains a useful function to perform another common task which is the "split-apply-combine" concept.

Install and load dplyr:

To install dplyr: install.packages("dplyr")

To load dplyr: library(dplyr)

The below are some of the most common dplyr functions:

rename() : rename columns
recode() : recode values in a column
select() : subset columns
filter() : subset rows on conditions
mutate() : create new columns by using information from other columns
summarise() : create summary statistics on grouped data
arrange() : sort results
count() : count discrete values
group_by() : allows for group operations in the "split-apply-combine" concept

Note: %>%: the "**pipe**" operator is used to connect multiple verb actions together into a pipeline

rename(): It is often necessary to rename variables to make them more meaningful.

Example :

```
#library(dplyr)
Data <- data.frame (
  Training = c("Strength", "Stamina", "Other"),
  Pulse = c(100, 150, 120),
  Duration = c(60, 30, 45)
)
Data
dplyr::rename(Data,PULSE1=Pulse,DURATION1=Duration)
```

```
   Training Pulse Duration
1 Strength   100       60
2  Stamina   150       30
3    Other   120       45
   Training PULSE1 DURATION1
1 Strength    100        60
2  Stamina    150        30
3    Other    120        45
```

select(): The select() function is used to pick specific variables or features of a DataFrame or tibble.

- It selects columns based on provided conditions like contains, matches, starts with, ends with, and so on.

Syntax: select(data,col1,col2,…)

This function returns an object of the same type as data.

Example:

```
dplyr::select(Data,Training)
dplyr::select(Data,Pulse,Training)
```

```
  Training
1 Strength
2  Stamina
3    Other
  Pulse Training
1   100 Strength
2   150  Stamina
3   120    Other
```

select(): Select column list either by name or index number

```
Data <- data.frame (
  Training = c("Strength", "Stamina", "Other"),
  Pulse = c(100, 150, 120),
  Duration = c(60, 30, 45))
dplyr::rename(Data,PULSE1=Pulse,DURATION1=Duration)
dplyr::select(Data,Training)
dplyr::select(Data,Pulse,Training)
dplyr::select(Data,1,3)
dplyr::select(Data,1:3)
```

```
   Training PULSE1 DURATION1
1 Strength    100         60
2  Stamina    150         30
3    Other    120         45
  Pulse Training
1   100 Strength
2   150  Stamina
3   120    Other
  Training Duration
1 Strength       60
2  Stamina       30
3    Other       45
  Training Pulse Duration
1 Strength   100       60
2  Stamina   150       30
3    Other   120       45
```

select(): Some additional options to select columns based on a specific criteria include

1. ends_with() = Select columns that end with a character string
2. contains() = Select columns that contain a character string
3. matches() = Select columns that match a regular expression
4. one_of() = Select columns names that are from a group of names

Example:

select(Data, starts_with(“Mo"))

```
d <- data.frame(   name=c("Abhi", "Bhavesh","Chaman", "Dimri"),
                   age=c(7, 5, 9, 16),
                   ht=c(46, NA, NA, 69),
                   school=c("yes", "yes", "no", "no"))

# startswith() function to print only ht data
select(d, starts_with("sc"))

# -startswith() function to print everything except ht data
select(d, -starts_with("ht"))

# Printing column 2 to 4
select(d, 2: 4)

# Printing data of column heading containing 'a'
select(d, contains("a"))

# Printing data of column heading which matches 'na'
select(d, matches("na"))
```

filter():

- The filter() function is used to produce a subset of the data frame, retaining all rows that satisfy the specified conditions.
- The filter() method in R programming language can be applied to both grouped and ungrouped data.
- The expressions include comparison operators (==, >, >=) , logical operators (&, |, !, xor()) , range operators (between(), near()) as well as NA value check against the column values.

Syntax: filter(df , condition)

Parameters :

df: The data frame object

condition: filtering based upon this condition

filter() Example:

```
#library(dplyr)
df=data.frame(x=c(12,31,4,66,78),
              y=c(22.1,44.5,6.1,43.1,99),
              z=c(TRUE,TRUE,FALSE,TRUE,TRUE))
df
# condition
dplyr::filter(df, x<50 & z==TRUE)
```

```
   x    y     z
1 12 22.1  TRUE
2 31 44.5  TRUE
3  4  6.1 FALSE
4 66 43.1  TRUE
5 78 99.0  TRUE
   x    y    z
1 12 22.1 TRUE
2 31 44.5 TRUE
```

```
# Create a data frame with missing data
d <- data.frame(name = c("Abhi", "Bhavesh", "Chaman", "Dimri"),
                age = c(7, 5, 9, 16),
                ht = c(46, NA, NA, 69),
                school = c("yes", "no", "yes", "no"))
print(d)

# Finding rows with NA value
rows_with_na <- d %>% filter(is.na(ht))
print(rows_with_na)

# Finding rows with no NA value
rows_without_na <- d %>% filter(!is.na(ht))
print(rows_without_na)

d %>% filter(age == 9, school == "yes")
```

mutate(): mutate() function in R Programming Language is used to add new variables in a data frame which are formed by performing operations on existing variables.

Syntax: mutate(x, expr)

- In R there are five types of main function for mutate that are discribe as below. we will use dplyr package in R for all mutate functions.
  - ✓ mutate()
  - ✓ transmute()
  - ✓ mutate_all() - to apply a transformation to all variables in a data frame simultaneously.
  - ✓ mutate_at()
  - ✓ mutate_if() - to apply a transformation to variables in a data frame based on a specific condition.

mutate() Example:

```
library(dplyr)

# Create a data frame
data <- data.frame(
name = c("Abhi", "Bhavesh", "Chaman", "Dimri"),
            age = c(7, 5, 9, 16),

            ht = c(46, NA, NA, 69),

            school = c("yes", "yes", "no", "no") )
data

# Calculating a variable x3 which is sum of height and age
 printing with ht and age

dplyr::mutate(d, x3 = ht + age)
```

```
     name age ht school
1    Abhi   7 46    yes
2 Bhavesh   5 NA    yes
3  Chaman   9 NA     no
4   Dimri  16 69     no
```

```
     name age ht school x3
1    Abhi   7 46    yes 53
2 Bhavesh   5 NA    yes NA
3  Chaman   9 NA     no NA
4   Dimri  16 69     no 85
```

transmute() Example:

```
# Use transmute to create a new variable 'age_in_months' and drop the 'age' variable

result <- transmute(data,
                    name = name,
                    age_in_months = age * 12,
                    ht,
                    school)
print(result)
```

```
     name age_in_months ht school
1    Abhi            84 46    yes
2 Bhavesh            60 NA    yes
3  Chaman           108 NA     no
4   Dimri           192 69     no
```

summarise_all():

- The summarise_all method in R is used to affect every column of the data frame.
- The output data frame returns all the columns of the data frame where the specified function is applied over every column.

Syntax: summarise_all(data, function)

Arguments :

data – The data frame to summarise the columns of

function – The function to apply on all the data frame columns.

Summarise_all() Example:

```
# creating a data frame
df <- data.frame(col1=c(1:10),col2=c(11:20))
print("original dataframe")
print(df)
print("summarised dataframe")
dplyr::summarise_all(df, mean)
```

```
[1] "original dataframe"
   col1 col2
1     1   11
2     2   12
3     3   13
4     4   14
5     5   15
6     6   16
7     7   17
8     8   18
9     9   19
10   10   20
[1] "summarised dataframe"
  col1 col2
1  5.5 15.5
```

arrange():

- arrange() function in R Language is used for reordering of table rows with the help of column names as expression passed to the function.

Syntax: arrange(x, expr)

Parameters:

x: data set to be reordered

expr: logical expression with column name

Example:

```
#library(dplyr)
d <- data.frame( name = c("Abhi", "Bhavesh", "Chaman", "Dimri"),
                 age = c(7, 5, 9, 16) )
# Arranging name according to the age
d2<- dplyr::arrange(d, age)
print(d2)
```

```
     name age
1 Bhavesh   5
2    Abhi   7
3  Chaman   9
4   Dimri  16
```

arrange(): arrange() function in R Language is used for reordering of table rows with the help of column names as expression passed to the function.

To arrange in a descending order:

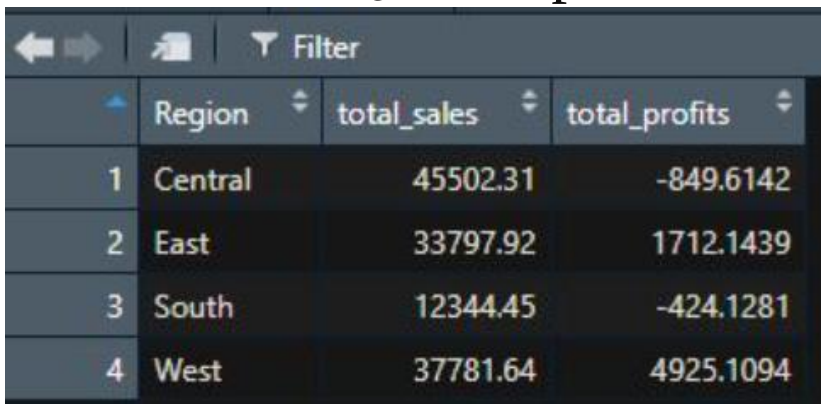
```
arrange(d1, desc(age))
```

To arrange in order using col1 and then by col2:

```
arrange(d, age,rollno)
```

Group_by():

- Group_by() function belongs to the dplyr package in the R programming language, which groups the data frames.
- Group_by() function alone will not give any output. It should be followed by summarise() function with an appropriate action to perform. It works similar to GROUP BY in SQL and pivot table in excel.



	Region	total_sales	total_profits
1	Central	45502.31	-849.6142
2	East	33797.92	1712.1439
3	South	12344.45	-424.1281
4	West	37781.64	4925.1094

Example:

```
library(dplyr)
    df = read.csv("Sample_Superstore.csv")
    df_grp_region = df %>% group_by(Region)  %>%
    summarise(total_sales = sum(Sales),
  total_profits = sum(Profit),.groups = 'drop')
```

```
data <- data.frame(Department = c("HR", "IT", "IT", "HR",
"Finance"),
                   Salary = c(50000, 70000, 75000, 55000, 60000))

                   grouped_summary <- data %>%
                   group_by(Department) %>%
        summarize(Avg_Salary = mean(Salary), Count = n())
        # count_data <- count(data, Department)
        print(grouped_summary)
```

	Department	Avg_Salary	Count
	<chr>	<dbl>	<int>
1	Finance	60000	1
2	HR	52500	2
3	IT	72500	2

Example 1:
Consider the previous example dataframe, find the average salary for each department, but only for employees earning more than $55,000, and sort results in descending order.

```
result <- data %>%
  filter(Salary > 55000) %>%
  group_by(Department) %>%
  summarize(Avg_Salary = mean(Salary))
%>%
  arrange(desc(Avg_Salary))

print(result)
```

Example 2:

Create the following sales dataset (Sales.csv).

Transaction_ID	Product	Category	Sales	Date
T001	Laptop	Electronics	1500	2024-01-10
T002	Shirt	Clothing	50	2024-01-11
T003	Phone	Electronics	800	2024-01-12
T004	TV	Electronics	1200	2024-01-15
T005	Jeans	Clothing	70	2024-01-18

i) Import the dataset (Sales.csv) into R.

ii) Filter sales data only from the "Electronics" category.

Group the data by Product and calculate the total sales for each product.

iii) Arrange the results in descending order of total sales.

iv) Export the processed data to a new CSV file named "electronics_sales.csv".

```
library(dplyr)
library(readr)

# Load sales data
sales_data <- read_csv("Sales.csv")

# Process the sales data
electronics_sales <- sales_data %>%
  filter(Category == "Electronics") %>%   # Filter Electronics category
  group_by(Product) %>%                   # Group by Product
  summarize(Total_Sales = sum(Sales)) %>% # Calculate total sales
  arrange(desc(Total_Sales))              # Sort in descending order

# Save processed data
write_csv(electronics_sales, "electronics_sales.csv")

# Print result
print(electronics_sales)
```

Example 3:

Create the following sales dataset (employee_data.xlsx).

Employee_ID	Name	Department	Salary	Experience
E001	Alice	IT	7000	6
E002	Bob	HR	5000	4
E003	Charlie	IT	8000	8
E004	David	Sales	6000	5
E005	Eva	HR	5500	3

i) Import the dataset (employee_data.xlsx) into R.

ii) Filter employees with at least 5 years of experience.

iii) Create a new column named "Annual_Salary" by multiplying Salary by 12.

iv) Group by Department and calculate the average annual salary.

v) Arrange the departments in descending order of average annual salary.

vi) Save the final processed dataset as filtered_employee_data.xlsx"

```
library(dplyr)
library(readxl)
library(writexl)
# Load employee data
employee_data <- read_excel("employee_data.xlsx")

# Process employee data
filtered_employees <- employee_data %>%
  filter(Experience >= 5) %>%  # Filter employees with at least 5 years of experience
  mutate(Annual_Salary = Salary * 12) %>%  # Create Annual_Salary column
  group_by(Department) %>%  # Group by Department
# Compute average salary
  summarize(Average_Annual_Salary = mean(Annual_Salary)) %>%
arrange(desc(Average_Annual_Salary))  # Sort in descending order

# Save processed data
write_xlsx(filtered_employees, "filtered_employee_data.xlsx")

# Print result
print(filtered_employees)
```

Exploratory Data Analysis

- Exploratory Data Analysis or EDA is a statistical approach or technique for analysing data sets in order to summarize their important and main characteristics generally by using some visual aids.
- The EDA approach can be used to gather knowledge about the following aspects of data:
  - ✓ Main characteristics or features of the data.
  - ✓ The variables and their relationships.
  - ✓ Finding out the important variables that can be used in our problem.

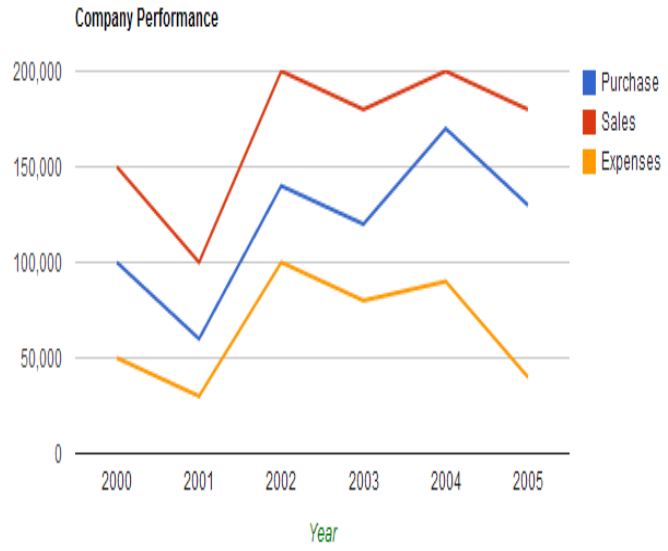
Exploratory Data Analysis in R:

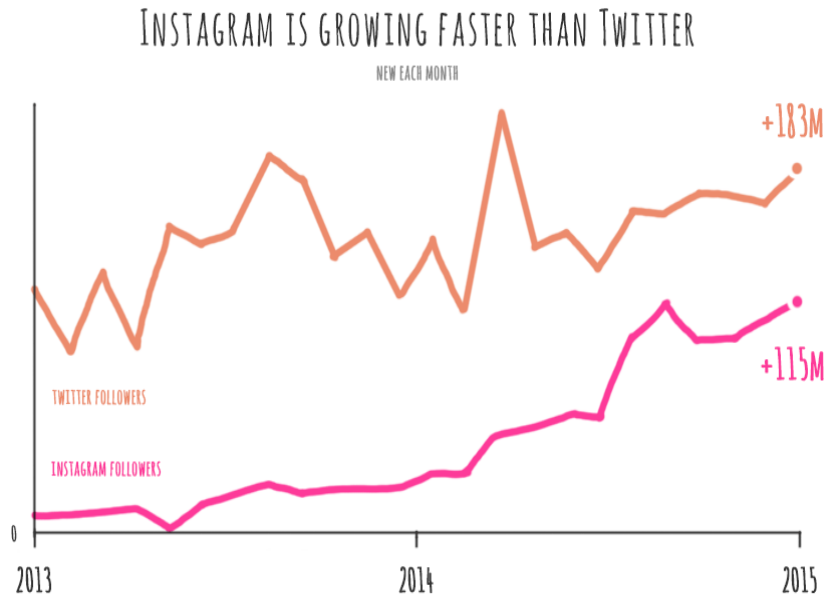
In R Language, we are going to perform EDA under two broad classifications:

- ✓ Descriptive Statistics, which includes mean, median, mode, inter-quartile range, and so on.
- ✓ Graphical Methods, which includes Box plot, Histogram, Pie graph, Line chart, Barplot, Scatter Plot and so on.

Diagrammatic representation of data:

- The diagrammatic representation of data is one of the best and attractive way of presenting data.
- It caters both educated and uneducated section of the society.





Histograms

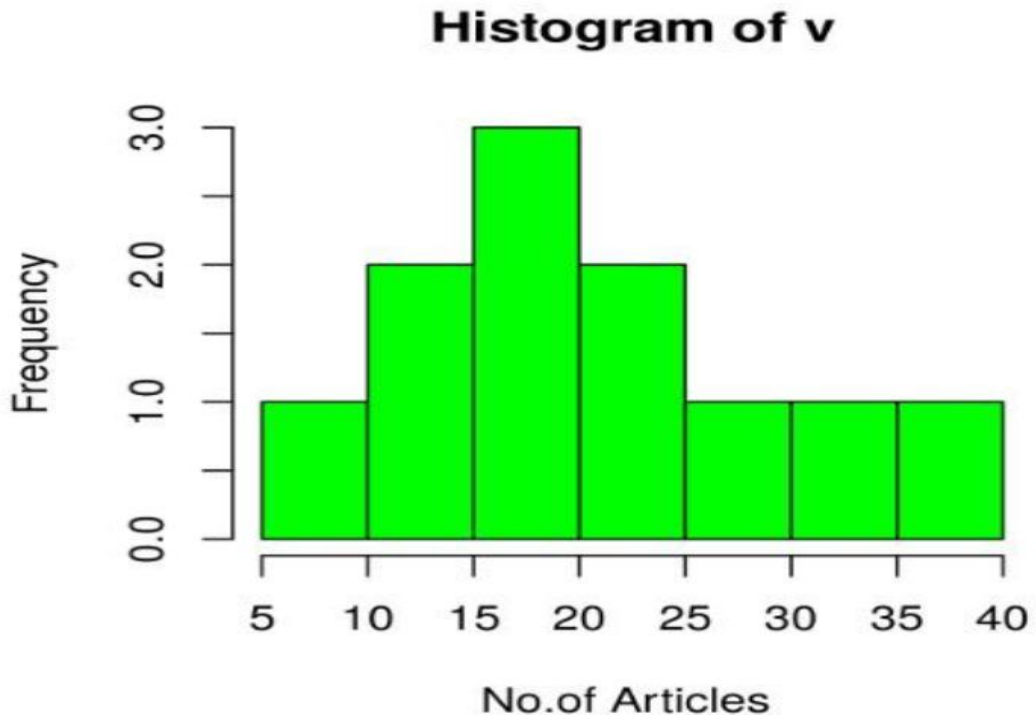
- A histogram contains a rectangular area to display the statistical information which is proportional to the frequency of a variable and its width in successive numerical intervals.
- We can create histograms in R Programming Language using the **hist()** function.

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

- ✓ v: This parameter contains numerical values used in histogram.
- ✓ main: This parameter main is the title of the chart.
- ✓ col: This parameter is used to set color of the bars.
- ✓ xlab: This parameter is the label for horizontal axis.
- ✓ border: This parameter is used to set border color of each bar.
- ✓ xlim: This parameter is used for plotting values of x-axis.
- ✓ ylim: This parameter is used for plotting values of y-axis.
- ✓ breaks: This parameter is used as width of each bar.

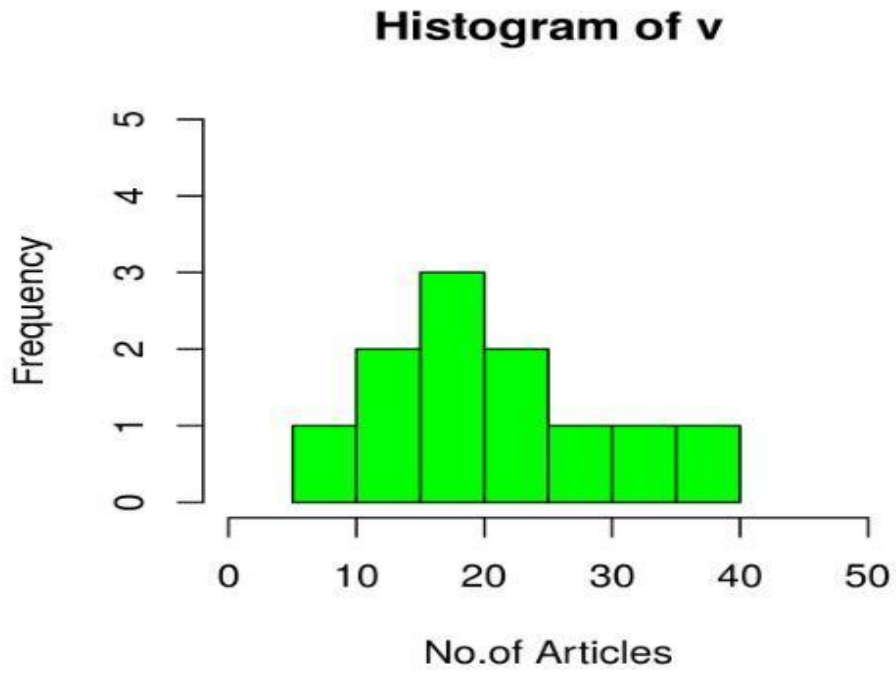
Example:

```
# Create data for the graph.
v <- c(19, 23, 11, 5, 16, 21, 32, 14, 19, 27, 39)
# Create the histogram.
hist(v, xlab = "No.of Articles ",col = "green", border = "black")
```



Example:

```
# Create data for the graph.
v <- c(19, 23, 11, 5, 16, 21, 32, 14, 19, 27, 39)
# Create the histogram.
hist(v, xlab="No.of Articles ",col = "green", border = "black“,xlim=c(0,50),ylim=c(0,5),break=5)
```



Pie graph

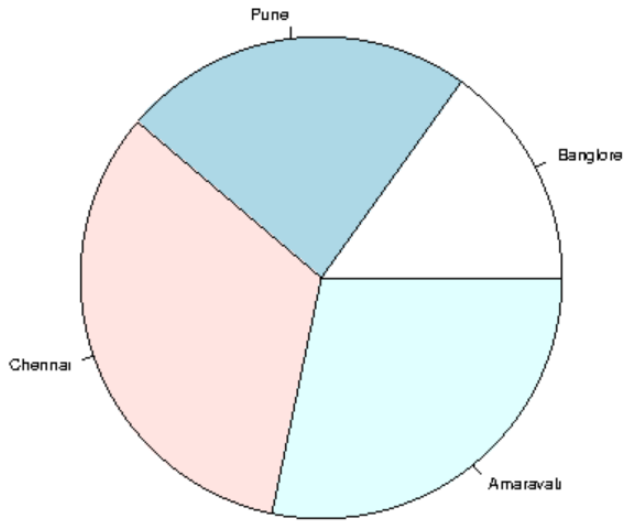
- A pie chart is a circular statistical graphic, which is divided into slices to illustrate numerical proportions.
- It depicts a special chart that uses “pie slices”, where each sector shows the relative sizes of data.
- A circular chart cuts in the form of radii into segments describing relative frequencies or magnitude also known as a circle graph.

Syntax: pie(x, labels, main, col, clockwise)

- ✓ **x**: This parameter is a vector that contains the numeric values which are used in the pie chart.
- ✓ **labels**: This parameter gives the description to the slices in pie chart.
- ✓ **main**: This parameter is representing title of the pie chart.
- ✓ **clockwise**: This parameter contains the logical value which indicates whether the slices are drawn clockwise or in anti-clockwise direction.
- ✓ **col**: This parameter give colours to the pie in the graph.

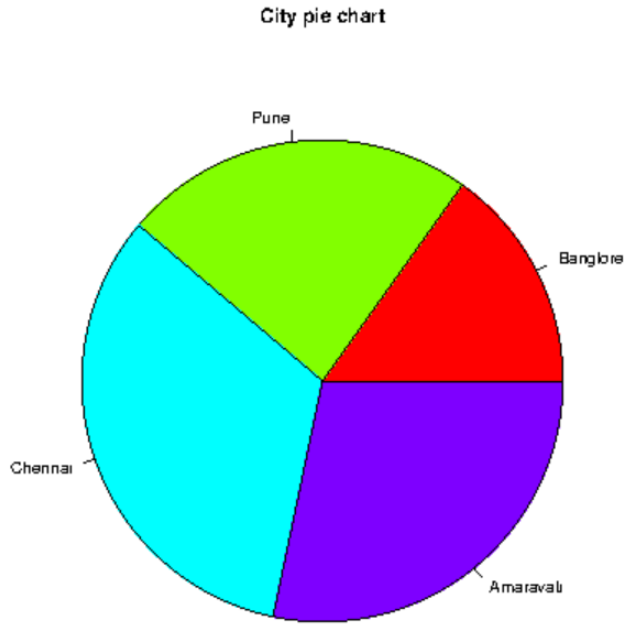
Example:

```
Temp<- c(23, 36, 50, 43)
Cities <- c("Banglore", "Pune", "Chennai", "Amaravati")
# Plot the chart.
pie(Temp, Cities)
```



Example:

```
Temp<- c(23, 36, 50, 43)
Cities <- c("Banglore", "Pune", "Chennai", "Amaravati")
# Plot the chart.
pie(Temp, Cities, main = "City pie chart",
          col = rainbow(length(Temp)) )
```



Barplot

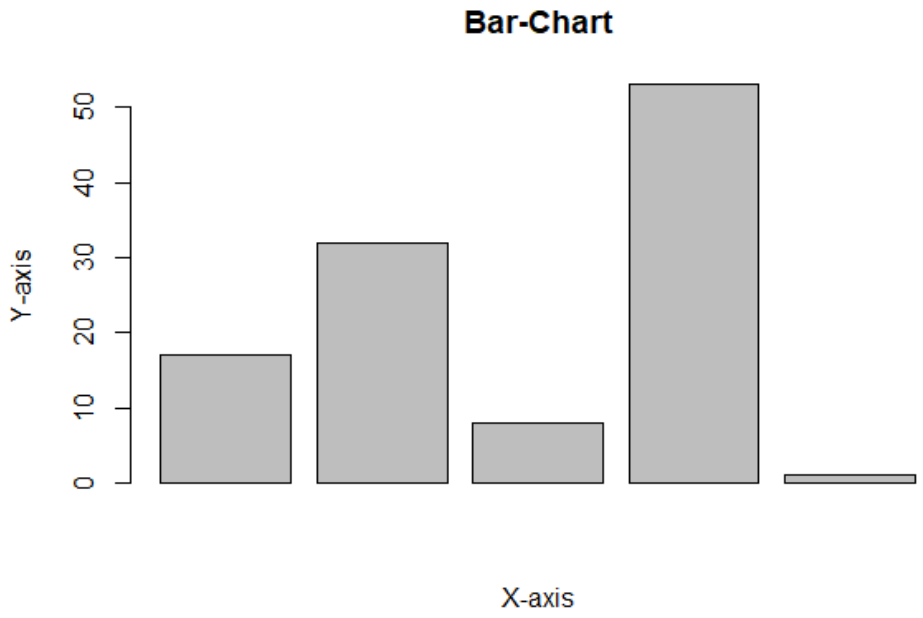
- Bar charts are a popular and effective way to visually represent categorical data in a structured manner.
- R uses the barplot() function to create bar charts. Here, both vertical and Horizontal bars can be drawn.

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

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

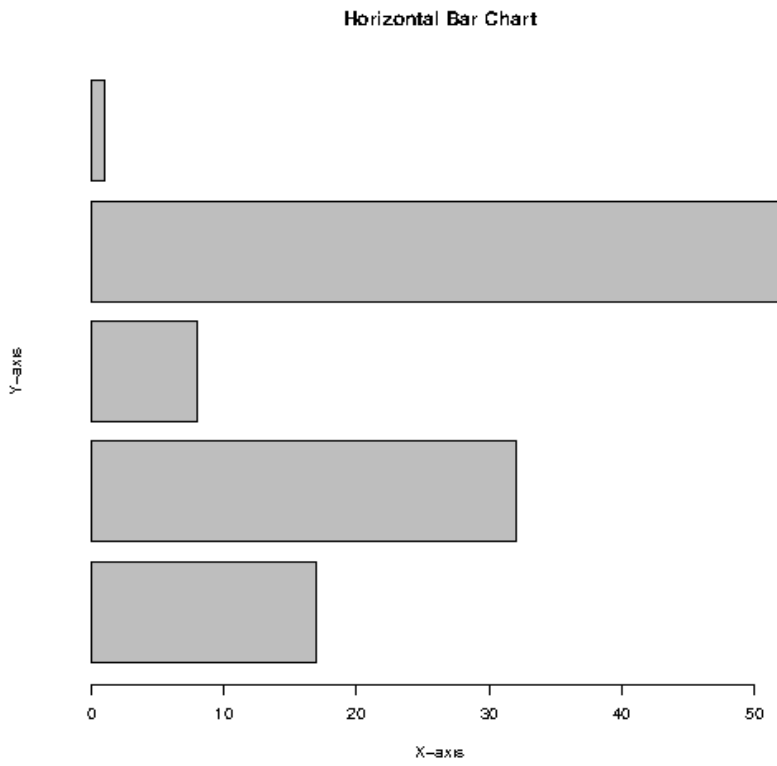
Example:

```
bitmap(file="out.png")
# Create the data for the chart
A <- c(17, 32, 8, 53, 1)
# Plot the bar chart
barplot(A, xlab = "X-axis", ylab = "Y-axis", main ="Bar-Chart")
```



Example:

```
# Create the data for the chart
A <- c(17, 32, 8, 53, 1)
# Plot the bar chart
barplot(A, horiz = TRUE, xlab = "X-axis", ylab = "Y-axis", main ="Horizontal Bar Chart" )
```



Scatter Plots

- A "scatter plot" is a type of plot used to display the relationship between two numerical variables, and plots one dot for each observation.
- It needs two vectors of same length, one for the x-axis (horizontal) and one for the y-axis (vertical).

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

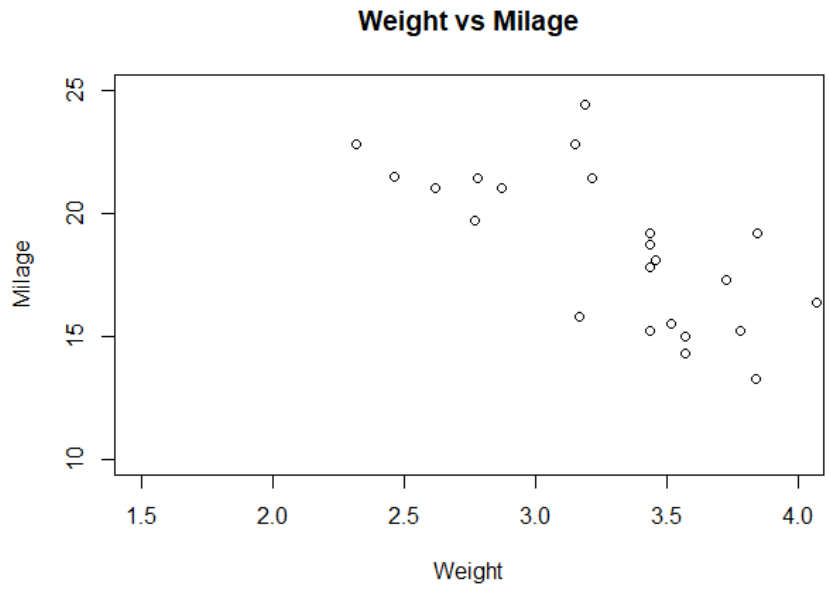
- ✓ **x**: This parameter sets the horizontal coordinates.
- ✓ **y**: This parameter sets the vertical coordinates.
- ✓ **xlab**: This parameter is the label for horizontal axis.
- ✓ **ylab**: This parameter is the label for vertical axis.
- ✓ **main**: This parameter main is the title of the chart.
- ✓ **xlim**: This parameter is used for plotting values of x.
- ✓ **ylim**: This parameter is used for plotting values of y.
- ✓ **axes**: This parameter indicates whether both axes should be drawn on the plot.

Example:

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

# Plot the chart for cars with weight between 1.5 to 4 and mileage
 between 10 and 25.

plot(x = input$wt, y = input$mpg,
    xlab = "Weight",
    ylab = "Milage",
    xlim = c(1.5, 4),
    ylim = c(10, 25),
    main = "Weight vs Milage"
)
```



Line Graphs

- A line graph is a chart that is used to display information in the form of a series of data points.
- It utilizes points and lines to represent change over time.
- Line graphs are drawn by plotting different points on their X coordinates and Y coordinates, then by joining them together through a line from beginning to end.

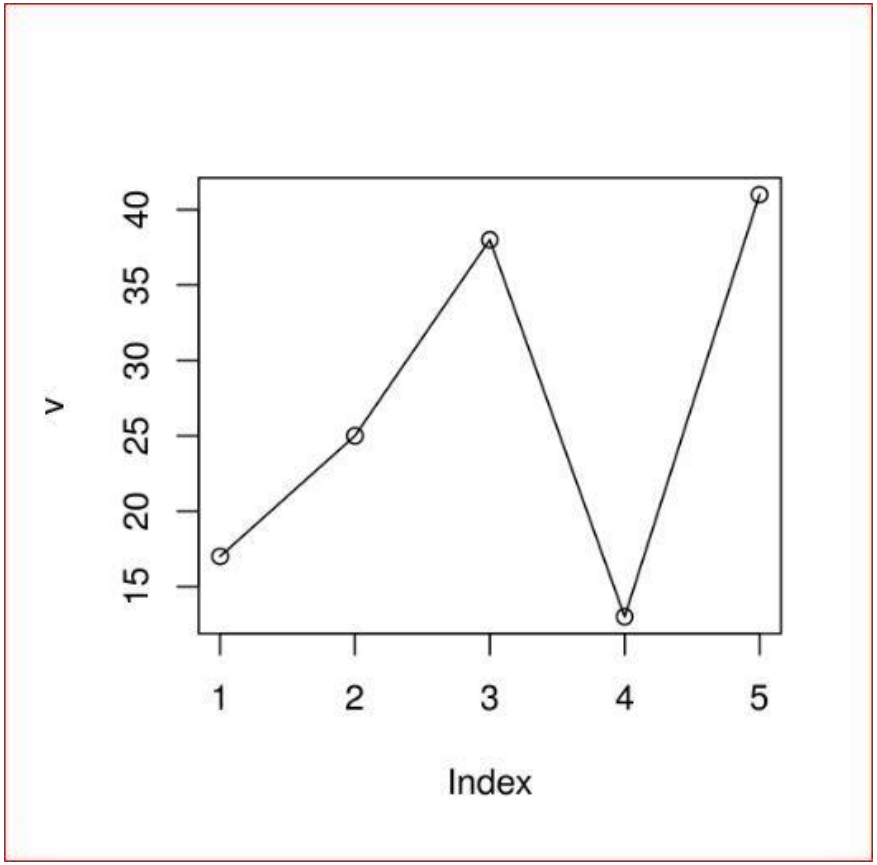
Syntax: plot(v, type, col, xlab, ylab)

- ✓ **v**: This parameter is a contains only the numeric values
- ✓ **type**: This parameter has the following value:
  - ✓ "**p**" : This value is used to draw only the points.
  - ✓ "**l**" : This value is used to draw only the lines.
  - ✓ "**o**": This value is used to draw both points and lines
- ✓ **xlab**: This parameter is the label for x axis in the chart.
- ✓ **ylab**: This parameter is the label for y axis in the chart.
- ✓ **main**: This parameter main is the title of the chart.
- ✓ **col**: This parameter is used to give colors to both the points and lines.

Example:

```
# Create the data for the chart.
v <- c(17, 25, 38, 13, 41)

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



Boxplots

A box graph is a chart that is used to display information in the form of distribution by drawing boxplots for each of them.

This distribution of data is based on five sets (minimum, first quartile, median, third quartile, and maximum).

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

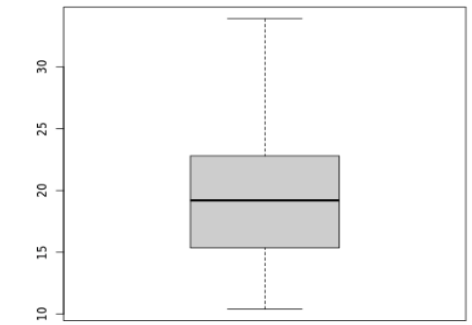
- ✓ **x**: This parameter sets as a vector or a formula.
- ✓ **data**: This parameter sets the data frame.
- ✓ **notch**: This parameter is the label for horizontal axis.
- ✓ **varwidth**: This parameter is a logical value. Set as true to draw width of the box proportionate to the sample size.
- ✓ **main**: This parameter is the title of the chart.
- ✓ **names**: This parameter are the group labels that will be showed under each boxplot.

Example:

```
# use head() to load first six rows of mtcars dataset
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

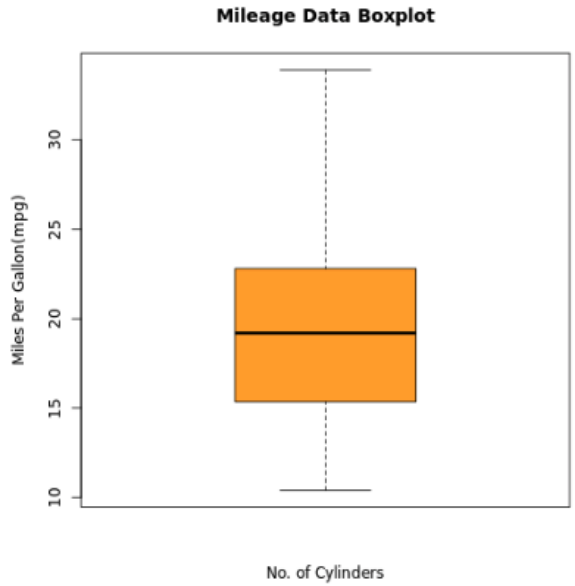
```
# boxplot for mpg reading of mtcars dataset
boxplot(mtcars$mpg)
```



Example:

```
# add title, label, new color to boxplot

boxplot(mtcars$mpg,
        main="Mileage Data Boxplot",
        ylab="Miles Per Gallon(mpg)",
        xlab="No. of Cylinders",
        col="orange")
```



Example 1:

Create the following dataset (sales.csv).

Employee_ID	Department	Salary	Experience
101	IT	75000	5
102	HR	52000	3
103	Sales	60000	4
104	IT	80000	6
105	HR	50000	2
106	Sales	68000	5
107	IT	90000	7
108	HR	55000	4
109	Sales	72000	6
110	IT	85000	6

i) Plot a histogram of Salary to analyze the salary distribution of employees. Identify the most common salary range.

ii) Create a pie chart showing the proportion of employees in each department. Which department has the most employees?

iii) Generate a scatter plot of Experience vs. Salary and determine whether experience influences salary.

iv) Compute and visualize the average salary per department using a bar chart. Which department offers the highest average salary?

v) Create a box plot to visualize the salary distribution per department. Identify any outliers.