Week 1: Installation of R and R-Studio on windows and understand the basics of R

Introduction to R programming:

R is a programming language and free software developed by Ross Ihaka and Robert Gentleman in 1993. R possesses an extensive catalog of statistical and graphical methods. It includes machine learning algorithms, linear regression, time series, and statistical inference to name a few. Most of the R libraries are written in R, but for heavy computational tasks, C, C++ and Fortran codes are preferred. R is not only entrusted by academic, but many large companies also use R programming language, including Uber, Google, Airbnb, Facebook and so on**.**

What is R used for?

- Statistical inference
- Data analysis
- Data Visualization.
- Machine learning algorithm

Getting R

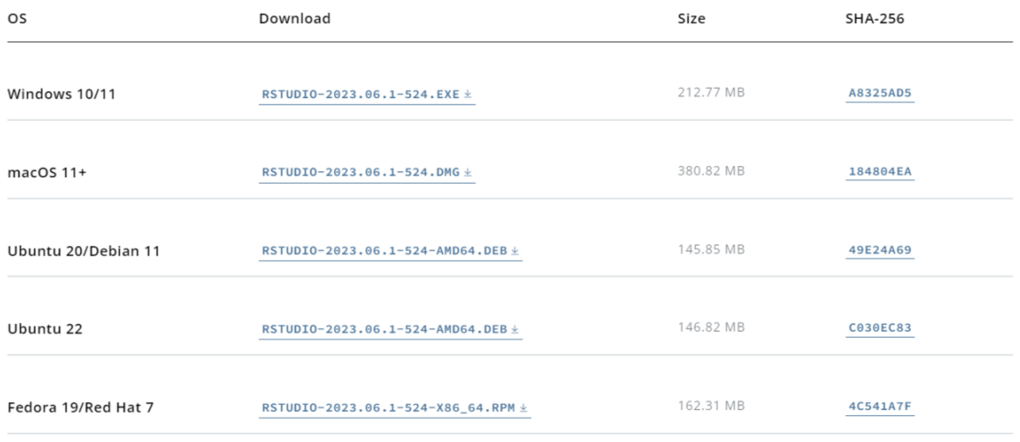
R can be downloaded from one of the "CRAN" (Comprehensive R Archive Network) sites. In the US, the main site is at http://cran.us.r-project.org/. Look in the "Download and Install R" area. Click on the appropriate link based on your operating system.

Installation of R-Studio on windows:

To download and install RStudio, follow the directions below

1. Navigate to RStudio's download site

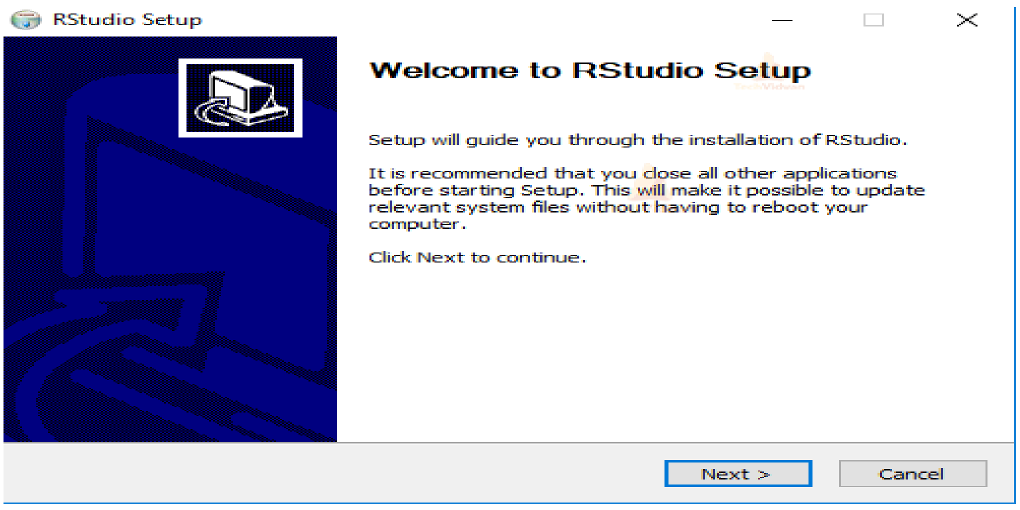
2. Click on the appropriate link based on your OS (Windows, Mac, Linux and many others).



OS	Download	Size	SHA-256
Windows 10/11	RSTUDIO-2023.06.1-524.EXE ⤓	212.77 MB	A8325AD5
macOS 11+	RSTUDIO-2023.06.1-524.DMG ⤓	380.82 MB	184804EA
Ubuntu 20/Debian 11	RSTUDIO-2023.06.1-524-AMD64.DEB ⤓	145.85 MB	49E24A69
Ubuntu 22	RSTUDIO-2023.06.1-524-AMD64.DEB ⤓	146.82 MB	C030EC83
Fedora 19/Red Hat 7	RSTUDIO-2023.06.1-524-X86_64.RPM ⤓	162.31 MB	4C541A7F

3. Run the .exe and follow the installation instructions.

4. Click **Next** on the welcome window.


RStudio Setup

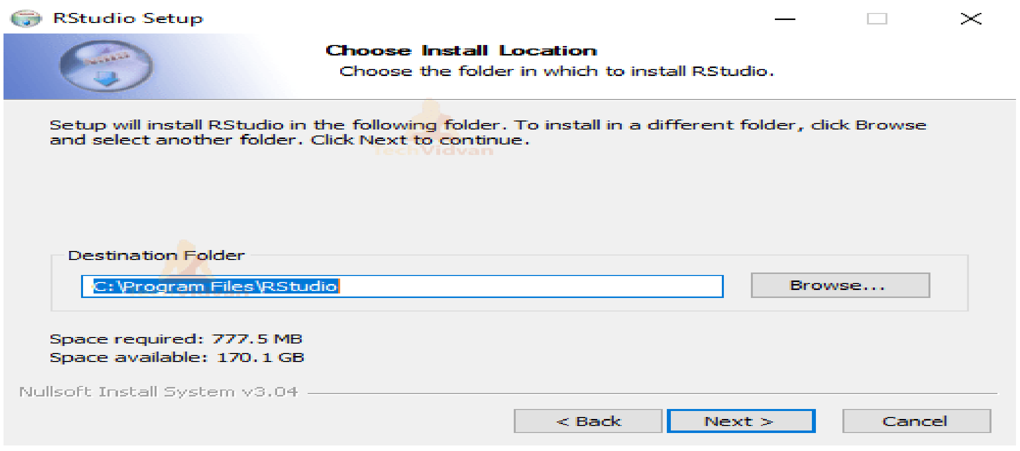
Welcome to RStudio Setup

Setup will guide you through the installation of RStudio.

It is recommended that you close all other applications before starting Setup. This will make it possible to update relevant system files without having to reboot your computer.

Click Next to continue.

5. Enter/browse the path to the installation folder and click Next to proceed


RStudio Setup

Choose Install Location

Choose the folder in which to install RStudio.

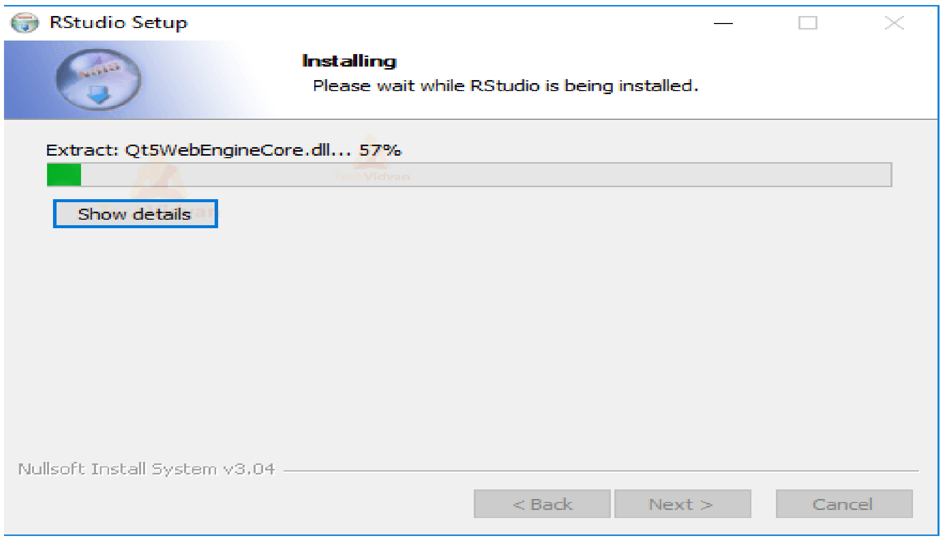
Setup will install RStudio in the following folder. To install in a different folder, click Browse and select another folder. Click Next to continue.

Destination Folder: C:\Program Files\RStudio

Space required: 777.5 MB
Space available: 170.1 GB

Nullsoft Install System v3.04

6. Wait for the installation process to complete.


RStudio Setup

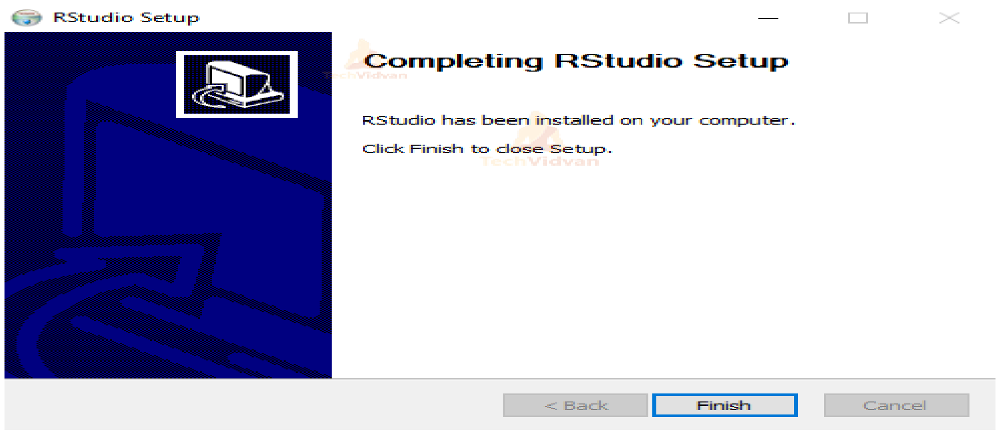
Installing

Please wait while RStudio is being installed.

Extract: Qt5WebEngineCore.dll... 57%

Nullsoft Install System v3.04

7. Click Finish to end the installation

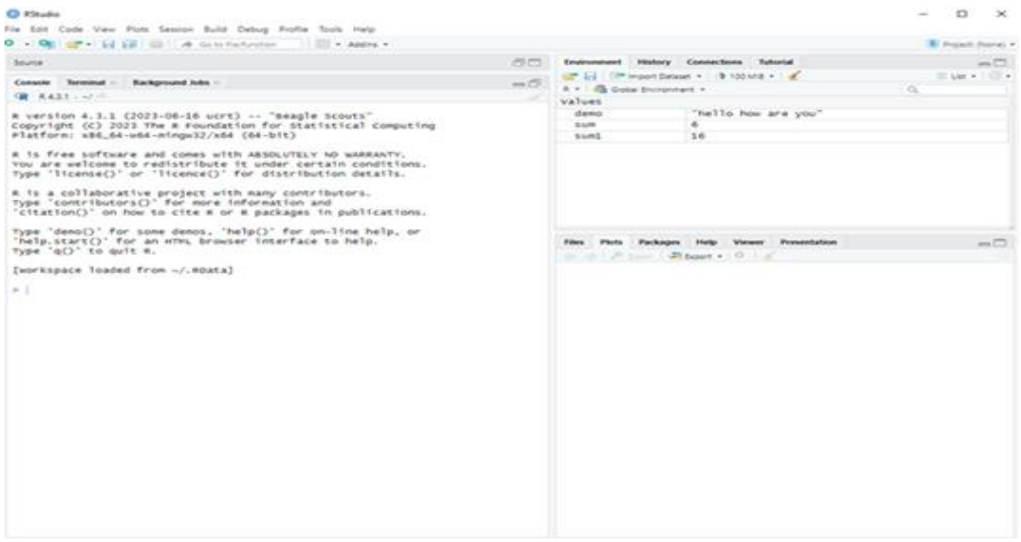


Completing RStudio Setup

RStudio has been installed on your computer.

Click Finish to close Setup.

8. The RStudio Interface looks like this::


RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
Go to file/function
Addins
Project: (None)
Source
Console Terminal Background Jobs
R 4.3.1 · ~/
Environment History Connections Tutorial
Import Dataset
100 MiB
List
Global Environment
Files Plots Packages Help Viewer Presentation
Zoom Export

```
R version 4.3.1 (2023-06-16 ucrt) -- "Beagle Scouts"
Copyright (C) 2023 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)

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

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

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

[Workspace loaded from ~/.RData]

> 
```

```
Values
  demo          "hello how are you"
  sum           6
  sum1          16
```

- **Console (left) -** The way R works is you write a line of code to execute some kind of task on a data object. The R Console allows you to run code interactively. The screen prompt > is an invitation from R to enter its world. This is where you type code in, press enter to execute the code, and see the results.

- **Environment, History, and Connections tabs (upper-right)**

  - Environment - shows all the R objects that are currently open in your workspace. This is the place, for example, where you will see any data you've loaded into R. When you exit RStudio, R will clear all objects in this window. You can also click on 🧹 to clear out all the objects loaded and created in your current session.

  - History - shows a list of executed commands in the current session.

    - Connections - you can connect to a variety of data sources, and explore the objects and data inside the connection

- **Files, Plots, Packages, Help and Viewer tabs (lower-right)**
    - Files - shows all the files and folders in your current working directory (more on what this means later).
    - Plots - shows any charts, graphs, maps and plots you've successfully executed.
    - Packages - tells you all the R packages that you have access to Help - shows help documentation for R commands that you've called up.
    - Viewer - allows you to view local web content (won't be using this much).

Installing Packages:

The most common place to get packages from is CRAN. To install packages from CRAN you use **install.packages("package name").** For instance, if you want to install the ggplot2 package, which is a very popular visualization package, you would type the following in the console:-

Syntax:-

```
# install package from CRAN
install.packages("ggplot2")
install.packages(c('readr', 'ggplot2', 'tidyr'))
```

Loading Packages:

Once the package is downloaded to your computer you can access the functions and resources provided by the package in two different ways:

```
# load the package to use in the current R session
library(packagename)
```

Basic Syntax in R

Programs are usually written in R scripts and then executed in the console window

- Create a new R script file

- Write the following code, select the block of code you want to run, and then press **Ctrl+Enter** or click on **'Run'**

mystring <- "Hello, World!"

print(mystring)

Output:

[1] "Hello, World!"

Variables in R Programming

A variable is a name given to a memory location, which is used to store values in a computer program. Variables in R programming can be used to store numbers (real and complex), words, matrices, and even tables. R is a dynamically programmed language which means that unlike other programming languages, we do not have to declare the data type of a variable before we can use it in our program.

For a variable to be valid, it should follow these rules

- It should contain letters, numbers, and only dot or underscore characters.
- It should not start with a number (eg:- 2iota)
- It should not start with a dot followed by a number (eg:- .2iota)
- It should not start with an underscore (eg:- _iota)
- It should not be a reserved keyword.

Reserved Keywords in R

Following are the reserved keywords in R				
for	in	repeat	while	function
if	else	next	break	TRUE
FALSE	NULL	Inf	NaN	NA
NA_integer_	NA_real_	NA_complex_	NA_character_	…

NA:– Not Available is used to represent missing values.

NULL:- It represents a missing or an undefined value.

NaN:– It is a short form for Not a Number (eg:- 0/0).

TRUE/FALSE: – These are used to represent Logical values.

Basic Data Types in R

Basic data types in R can be divided into the following types:

- numeric - (10.5, 55, 787)
- integer - (1L, 55L, 100L, where the letter "L" declares this as an integer)
- complex - (9 + 3i, where "i" is the imaginary part)
- character (a.k.a. string) - ("k", "R is exciting", "FALSE", "11.5")
- logical (a.k.a. boolean) - (TRUE or FALSE)

Numeric Data type in R

```
# A simple R program to illustrate Numeric data type

# Assign a decimal value to x

x = 5.6

# print the class name of variable

print(class(x))

# print the type of variable

print(typeof(x))
```

Test Output

When R stores a number in a variable, it converts the number into a "double" value or a decimal type with at least two decimal places. This means that a value such as "5" here, is stored as 5.00 with a type of double and a class of numeric. And also y is not an integer here can be confirmed with the is.integer() function.

Integer Data type in R

```
# A simple R program to illustrate integer data type

# Create an integer value

x = as.integer(5)

 # print the class name of x

print(class(x))

# print the type of x
```

```
print(typeof(x))
 # Declare an integer by appending an L suffix.
y = 5L
# print the class name of y
print(class(y))
 # print the type of y
print(typeof(y))
```

Test Output

Logical Data type in R

```
# A simple R program to illustrate logical data type
# Sample values
x = 4
y = 3
 # comparing two values
z = x > y
 # print the logical value
print(z)
 # print the class name of z
print(class(z))
# print the type of z
print(typeof(z))
```

Test Output:

Complex Data type in R

```
# A simple R program to illustrate complex data type

# Assign a complex value to x

x = 4 + 3i

# print the class name of x

print(class(x))

 # print the type of x

print(typeof(x))
```

Test Output :

Character Data type in R

```
# A simple R program to illustrate character data type

# Assign a character value to char

char = "welcome to R programming"

 # print the class name of char

print(class(char))

 # print the type of char

print(typeof(char))
```

Test Output :

Type verification

To do that, you need to use the prefix “is.” before the data type as a command. The syntax for that is, **is.data_type()** of the object you have to verify.

Syntax:

```
is.data_type(object)
```

eg:

```
# Integer
print(is.integer(3L))
```

Output: [1] TRUE

Coerce or convert the data type of an object to another

Syntax

```
as.data_type(object)
```

eg:

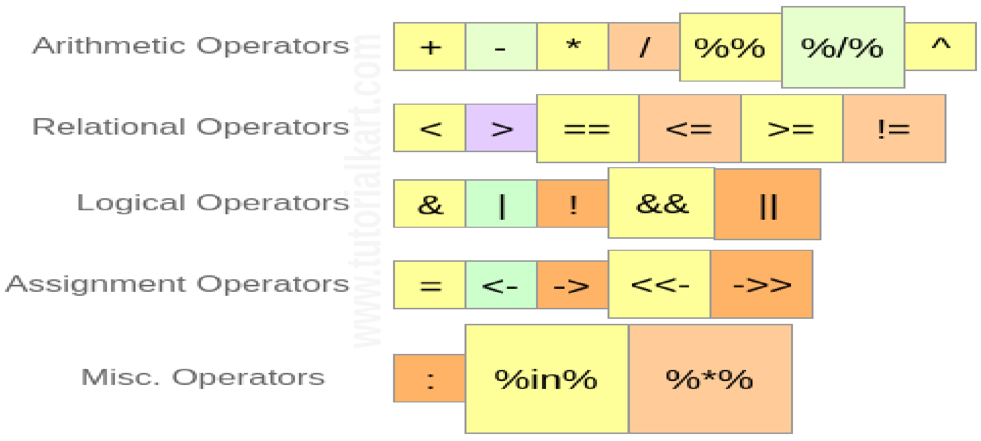
```
# Logical
print(as.numeric(TRUE))
 # Integer
print(as.complex(3L))
 # Numeric
print(as.logical(10.5))
 # Complex
print(as.character(1+2i))
 # Can't possible
print(as.numeric("12-04-2020"))
```

Test Output:

R Operators

R divides the operators in the following groups:

- Arithmetic operators
- Assignment operators
- Comparison operators
- Logical operators
- Miscellaneous operators



Worksheet

Exercise: Write R-script to demonstrate different types of operators in R.

Week2: Working with R Objects: Vectors, lists, Factors and Data Frame.

R – Objects

The frequently used ones are −

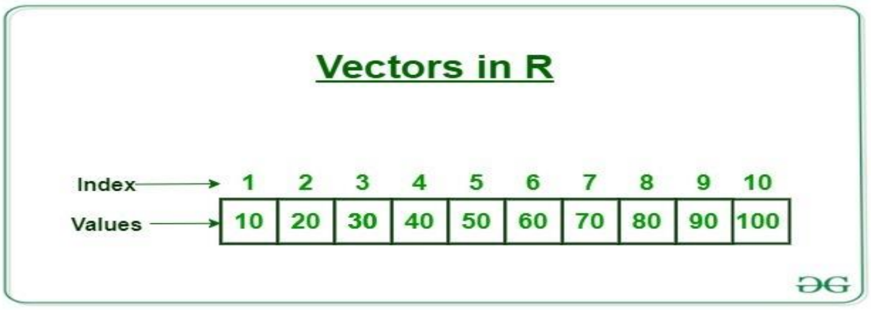
- Vectors
- Lists
- Matrices
- Arrays
- Factors
- Data Frames

R has five basic or "atomic" classes of objects:

- character
- numeric (real numbers)
- integer
- complex
- logical (True/False)

Vectors

Atomic vectors are one of the basic types of objects in R programming. Atomic vectors can store homogeneous data types such as character, doubles, integers, raw, logical, and complex. A single element variable is also said to be vector.



Operations on Vectors:

- **For creating vectors:** c( ) function, seq( ) function , use':' to create a vector.
- **Length of R vector:** Find the length of the vector length(x)
- **Accessing R vector elements:** with an index number, by passing a range of values
- **Modifying a R vector:**
- **Deleting a R vector:** set NULL to the vector
- **Sorting elements of a R Vector:** sort( ) function

Exercise: Create vectors v: elements ranging from 5 to 13, t: storing days of a week and perform following operations on vector elements.

a) **Print the 2,3,and 6th element from vector t**
b) **Sort the elements of vector v**
c) **Perform different operations on vectors v1, v2: add, substract, multiply ,division**

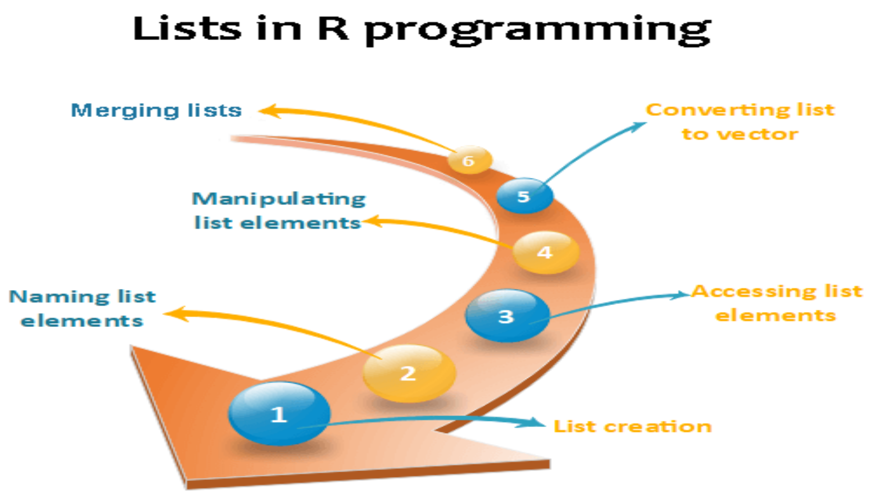
```
# Creating a VECTOR of sequence from 5 to 13.

v <- 5:13

print(v)

# Create vector with elements from 5 to 9 incrementing by 0.4.

print(seq(5, 9, by = 0.4))

# Accessing vector elements using position.

t <- c("Sun","Mon","Tue","Wed","Thurs","Fri","Sat")

u <- t[c(2,3,6)]

print(u)

# Accessing vector elements using logical indexing.

v <- t[c(TRUE,FALSE,FALSE,FALSE,FALSE,TRUE,FALSE)]

print(v)

# Accessing vector elements using 0/1 indexing.

y <- t[c(0,0,0,0,0,0,1)]

print(y)

#Vector Manipulation

#Create two vectors.

v1 <- c(3,8,4,5,0,11)

v2 <- c(4,11,0,8,1,2)

# Vector addition.

add.result <- v1+v2

print(add.result)

# Vector subtraction.
```

```
sub.result <- v1-v2

print(sub.result)

# Vector multiplication.

multi.result <- v1*v2

print(multi.result)

# Vector division.

divi.result <- v1/v2

print(divi.result)

#Vector Element Sorting

v <- c(3,8,4,5,0,11, -9, 304)

# Sort the elements of the vector.

sort.result <- sort(v)

print(sort.result)

# Sort the elements in the reverse order.

revsort.result <- sort(v, decreasing = TRUE)

print(revsort.result)
```

Output:

Lists

A list is an R-object which can contain many different types of elements inside it such as number, vectors, string and another list inside it.



Exercise: Create empList with the following components: empId(1, 2, 3, 4) , empName("Debi", "Sandeep", "Subham", "Shiba"), numberOfEmp= 4. Perform following operations on List elements.

- **a) Print the employee names**
- **b) Modify total staff = 5**
- **c) Add the component empAge = (34, 23, 18, 45) to the list and remove the record of sandeep from the list.**
- **d) Create a list of elements 1:5 and convert it to vector and matrix.**

```
# Creating a list by naming all its components

empId = c(1, 2, 3, 4)

empName = c("Debi", "Sandeep", "Subham", "Shiba")

numberOfEmp = 4

empList = list(

  "ID" = empId,

  "Names" = empName,

  "Total Staff" = numberOfEmp

  )

print(empList)

 # Accessing components by names

cat("Accessing name components using $ command\n")

print(empList$Names)
```

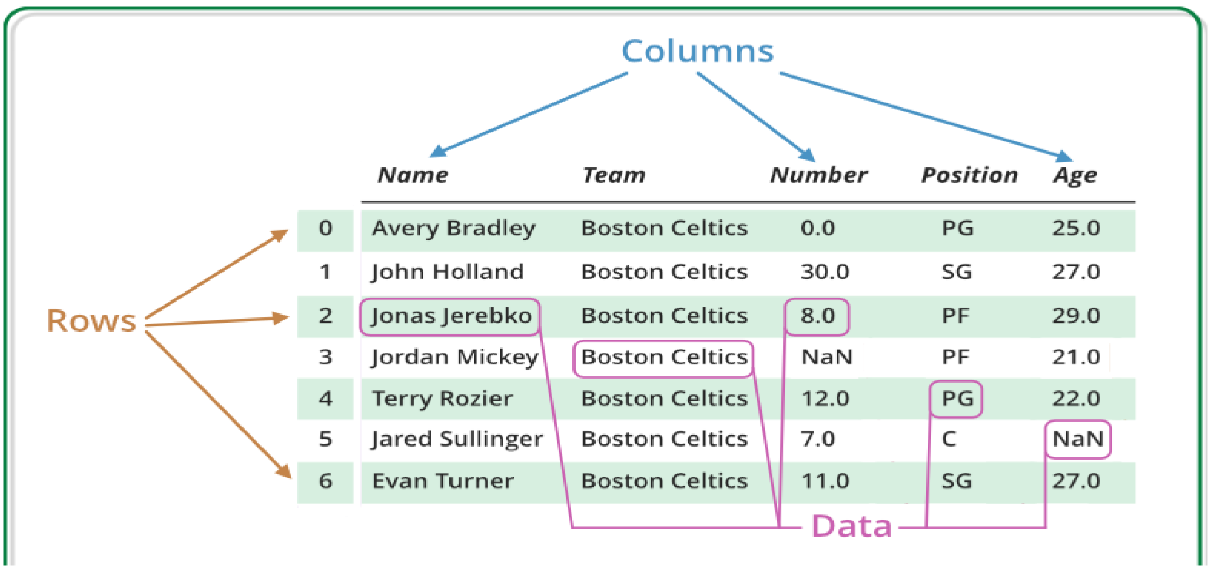
```
# Accessing a inner level components by indices
cat("Accessing Sandeep from name using indices\n")
print(empList[[2]][2])
 # Accessing another inner level components by indices
 # Modifying the top-level component
empList$`Total Staff` = 5
 # Modifying inner level component
empList[[1]][5] = 5
empList[[2]][5] = "Kamala"
print(empList)
# Creating another list
empAge = c(34, 23, 18, 45)
 # Concatenation of list using concatenation operator
empList = c(empName, empAge)
 cat("After concatenation of the new list\n")
print(empList)
# Deleting a top level components
cat("After Deleting Total staff components\n")
print(empList[-3])
 # Deleting a inner level components
cat("After Deleting sandeep from name\n")
print(empList[[2]][-2])
#Merging two list
# Create two lists.
lst1 <- list(1,2,3)
lst2 <- list("Sun","Mon","Tue")
 # Merge the two lists.
new_list <- c(lst1,lst2)
 # Print the merged list.
print(new_list)
```

```
# Converting List to Vector
#Create lists.
lst <- list(1:5)
print(lst)
# Convert the lists to vectors.
vec <- unlist(lst)
 print(vec)
# List to matrix
lst1 <- list(list(1, 2, 3),
            list(4, 5, 6))
 # Print list
cat("The list is:\n")
print(lst1)
cat("Class:", class(lst1), "\n")
 # Convert list to matrix
mat <- matrix(unlist(lst1), nrow = 2, byrow = TRUE)
 # Print matrix
cat("\nAfter conversion to matrix:\n")
print(mat)
```

Output:

R – Data Frames

A data frame is a two-dimensional array-like structure or a table in which a column contains values of one variable, and rows contains one set of values from each column. A data frame is a special case of the list in which each component has equal length.



	Name	Team	Number	Position	Age
0	Avery Bradley	Boston Celtics	0.0	PG	25.0
1	John Holland	Boston Celtics	30.0	SG	27.0
2	Jonas Jerebko	Boston Celtics	8.0	PF	29.0
3	Jordan Mickey	Boston Celtics	NaN	PF	21.0
4	Terry Rozier	Boston Celtics	12.0	PG	22.0
5	Jared Sullinger	Boston Celtics	7.0	C	NaN
6	Evan Turner	Boston Celtics	11.0	SG	27.0

Exercise: Create Data Frame “my_data_frame” of following information:

animal <- sheep, pig

year <- 2019:2021,

weight <- 110, 120, 140, NA, 300, 800

height <- 2.2, 2.4, 2.7, 2, 2.1, 2.3

condition <- “excellent", "good", NA, "excellent", "good", "average"

Perform following operations on Data frame.

a) **Print class and structure of my_data_frame**
b) **Get the summary statistics for each variable of my_data_frame**
c) **Add the new observation :animal = "pig", year = 2018, weight = 200, height = 1.9, condition = "excellent"**
d) **Print the following output using subsetting : i) 110 ii) 2019 2020 2021 2019 2020 2021 iii) 2 sheep 2020 120 2.4 good TRUE**

```
iv)     animal   weight
        1  sheep    110
        2  sheep    120
        3  sheep    140
        4    pig     NA
        5    pig    300
        6    pig     80
```

animal <- rep(c("sheep", "pig"), c(3,3))

year <- rep(2019:2021, 2)

weight <- c(110, 120, 140, NA, 300, 800)

height <- c(2.2, 2.4, 2.7, 2, 2.1, 2.3)

condition <- c("excellent", "good", NA, "excellent", "good", "average")

condition <- factor(condition, ordered = TRUE, levels = c("average", "good", "excellent"))

healthy <- c(rep(TRUE, 5), FALSE)

my_data_frame <- data.frame(animal, year, weight, height, condition, healthy)

my_data_frame

#After creating the data frame, it is useful to examine its class using the class() function and #structure using the str() function.

```
class(my_data_frame)
```
str(my_data_frame)

#Using `summary()` on a data frame, you get the summary statistics for each variable.

summary(my_data_frame)

#Adding Observations or Variables in Data Frames

#you need to put the additional observations or variables into a new data frame, and use #the `rbind()` function.

new_obs <- data.frame(animal = "pig", year = 2018, weight = 200, height = 1.9, condition = "excellent", healthy = TRUE)

rbind(my_data_frame, new_obs)

#Subsetting Data Frames

my_data_frame[1, 3]

my_data_frame[2, 4]

my_data_frame[, 2]

my_data_frame[2,]

my_data_frame[c(1,3), -c(3,4)]

my_data_frame[, c("animal", "weight")]

Output:

Week 3: Importing Data into R from Different Sources

Importing data in R programming means that we can read data from external files, write data to external files, and can access those files from outside the R environment. File formats like CSV, XML, xlsx, JSON, and web data can be imported into the R environment to read the data and perform data analysis, and also the data present in the R environment can be stored in external files in the same file formats.

Reading CSV Files:

--When you are working in R it is useful to know your working directory. This is the folder in which R will save or look for files by default. You can see your working directory by typing: **getwd()**

--You can also change your working directory using the function setwd . Or you can change it through RStudio by clicking on "Session"

--You should be able to see the file in your working directory and can check using **list.files()**

--Reading in csv files using **read.csv()** function
--Analyzing a CSV File: ncol(), nrows(), range(), subset( ) functions
-- Writing to a .csv file: write.csv() function

Exercise: Write a code to import a file "file1.csv" containing following information and perform the following function:

2) **To print number of columns and rows**
3) **To print the range of salary packages**
4) **To print the details of a person with the highest salary**
5) **To print the details of people having salary between 30000 and 40000 and store the results in a new file**
6) **Writing data into a new CSV file**

Sl. No.	empid	empname	empdept	empsalary	empstart_date
1	1	Sam	IT	25000	03-09-2005
2	2	Rob	HR	30000	03-05-2005
3	3	Max	Marketing	29000	05-06-2007
4	4	John	R&D	35000	01-03-1999
5	5	Gary	Finance	32000	05-09-2000
6	6	Alex	Tech	20000	09-05-2005
7	7	Ivar	Sales	36000	04-04-1999
8	8	Robert	Finance	34000	06-08-2008

```
read.data <- read.csv("file1.csv")
print(read.data)

#To print number of columns
print(ncol(read.data))

#To print number of rows
print(nrow(read.data))

#To print the range of salary packages
range.sal <- range(read.data$empsalary)
print(range.sal)

#To print the details of a person with the highest salary, we use the subset() function to
#extract variables and observations
max.sal <- subset(read.data, empsalary == max(empsalary))
print(max.sal)

#To print the details of people having salary between 30000 and 40000 and store the results
#in a new file
per.sal <- subset(read.data, empsalary >= "30000" & empsalary <= "40000")
print(per.sal)

# Writing data into a new CSV file
write.csv(per.sal,"output.csv")
new.data <- read.csv("output.csv")
print(new.data)
```

Output:

Importing a TXT file in R:

We will use read.table's alternative function, `read.delim` to load the text file as an R dataframe**.**

```
data3 <- read.delim('data/drake_lyrics.txt',header = F)
head(data3, 5)
```

Importing data from Excel into R

We will use the readxl package's `read_excel` function to read a single sheet from an Excel file.

```
library(readxl)
data4 <- read_excel("data/Tesla Deaths.xlsx", sheet = 1)
head(data4, 5)
```

Importing data from XML and HTML files

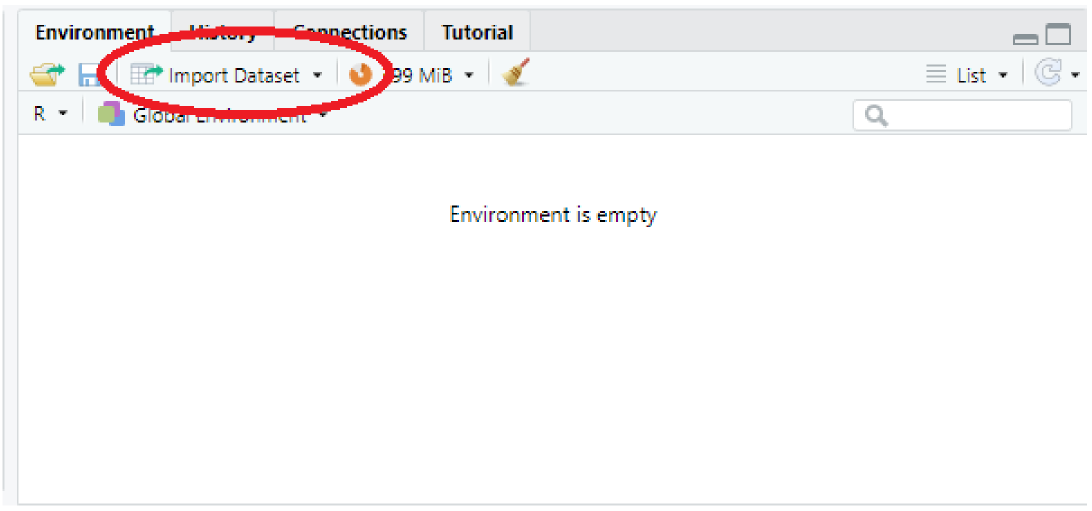
```
library(xml2)
plant_xml <- read_xml('https://www.w3schools.com/xml/plant_catalog.xml')
print(plant_xml)
```

Using R-Studio

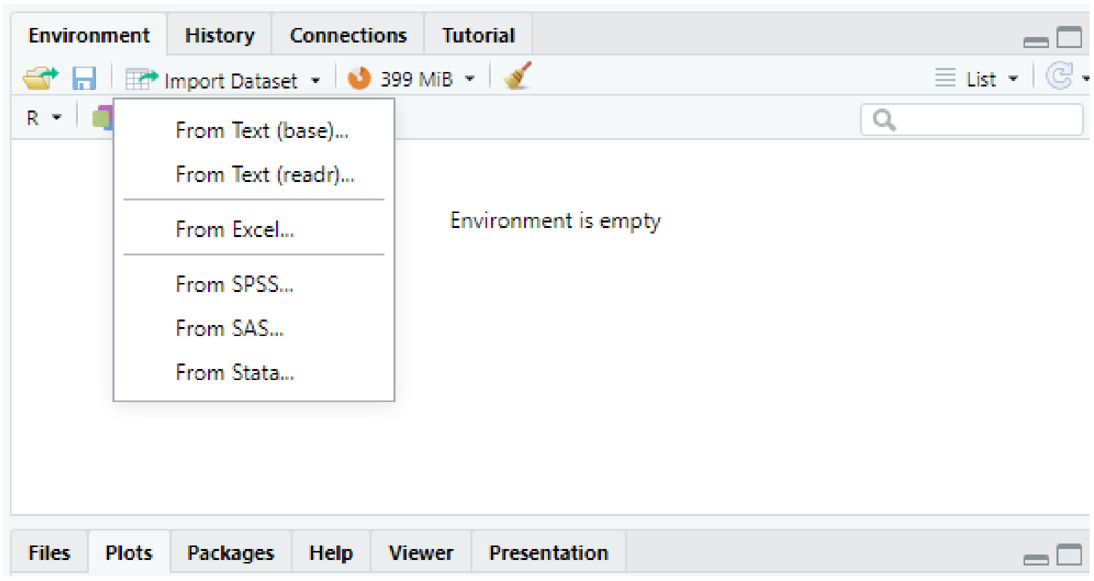
Here we are going to import data through R studio with the following steps

Steps:

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



- Select the file extension from the option.

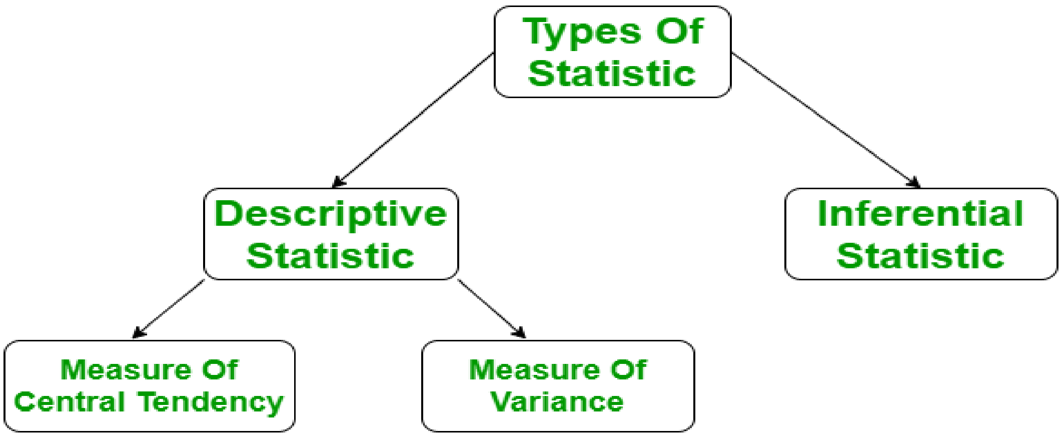


- In the third step, a pop-up box will appear, either enter the file name or browse the desktop.
- The selected file will be displayed on a new window with its dimensions.
- In order to see the output on the console, type the filename.

Output:

Week 4: Write a code for Descriptive Data Analysis in R

In the descriptive analysis, we describe our data in some manner and present it in a meaningful way so that it can be easily understood. Most of the time it is performed on small data sets and this analysis helps us a lot to predict some future trends based on the current findings. Some measures that are used to describe a data set are measures of central tendency and measures of variability or dispersion.

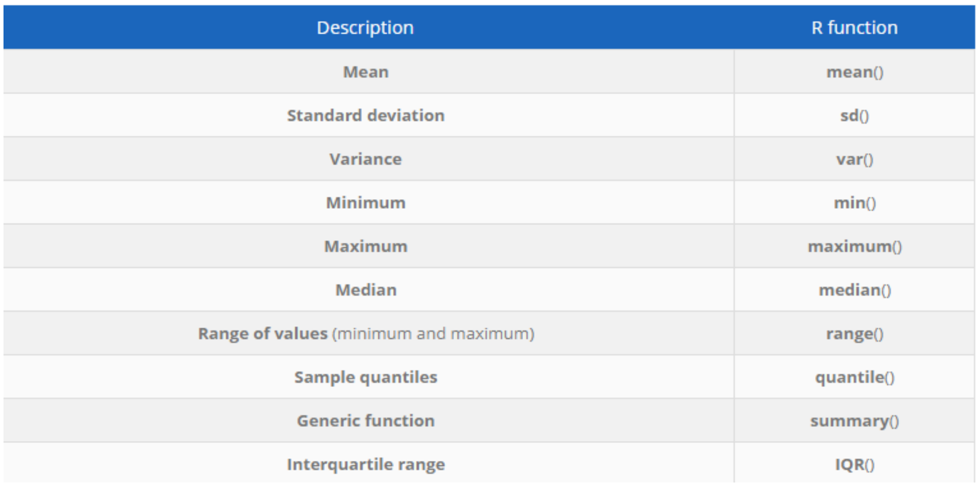


Measure of central tendency

It represents the whole set of data by a single value. It gives us the location of central points. There are three main measures of central tendency:

- Mean
- Mode
- Median

Some R functions for computing descriptive statistics:



Description	R function
Mean	mean()
Standard deviation	sd()
Variance	var()
Minimum	min()
Maximum	maximum()
Median	median()
Range of values (minimum and maximum)	range()
Sample quantiles	quantile()
Generic function	summary()
Interquartile range	IQR()

Exercise: Write R-code to perform Descriptive Data Analysis on ‘iris’ Dataset to compute the following:

1. **Display the structure of dataset and first six observations**
2. **Find minimum , maximum and range of sepal.length**
3. **Find Mean , Median and Mode on sepal.length**
4. **Find First and third quartile and Interquartile range**
5. **Find Standard deviation and variance**
6. **Compute the minimum, 1st quartile, median, mean, 3rd quartile and the maximum for all numeric variables of a dataset**

```
dat <- iris   # load the iris dataset and renamed it dat

head(dat)   # first 6 observations

str(dat)   # structure of dataset

# Minimum and maximum can be found using the min() and max() functions:

min(dat$Sepal.Length)

max(dat$Sepal.Length)

rng <- range(dat$Sepal.Length)

rng

# The range can then be easily computed, as you have guessed, by subtracting the minimum
#from the maximum:

max(dat$Sepal.Length) - min(dat$Sepal.Length)

# The mean and median  can be computed with the mean() and median()function:

mean(dat$Sepal.Length)

median(dat$Sepal.Length)

#  we can easily find mode()  using the functions table() and sort()

tab <- table(dat$Sepal.Length) # number of occurrences for each unique value

sort(tab, decreasing = TRUE) # sort highest to lowest

# The mode of the variable Sepal.Length is thus 5

# the first and third quartiles can be computed thanks to the quantile()

quantile(dat$Sepal.Length, 0.25) # first quartile

quantile(dat$Sepal.Length, 0.75) # third quartile

IQR(dat$Sepal.Length)
```

```
#The standard deviation and the variance is computed with the sd() and var() functions:

sd(dat$Sepal.Length) # standard deviation

var(dat$Sepal.Length) # variance

# compute for all numeric variable in the dataset using summary():

summary(dat)
```

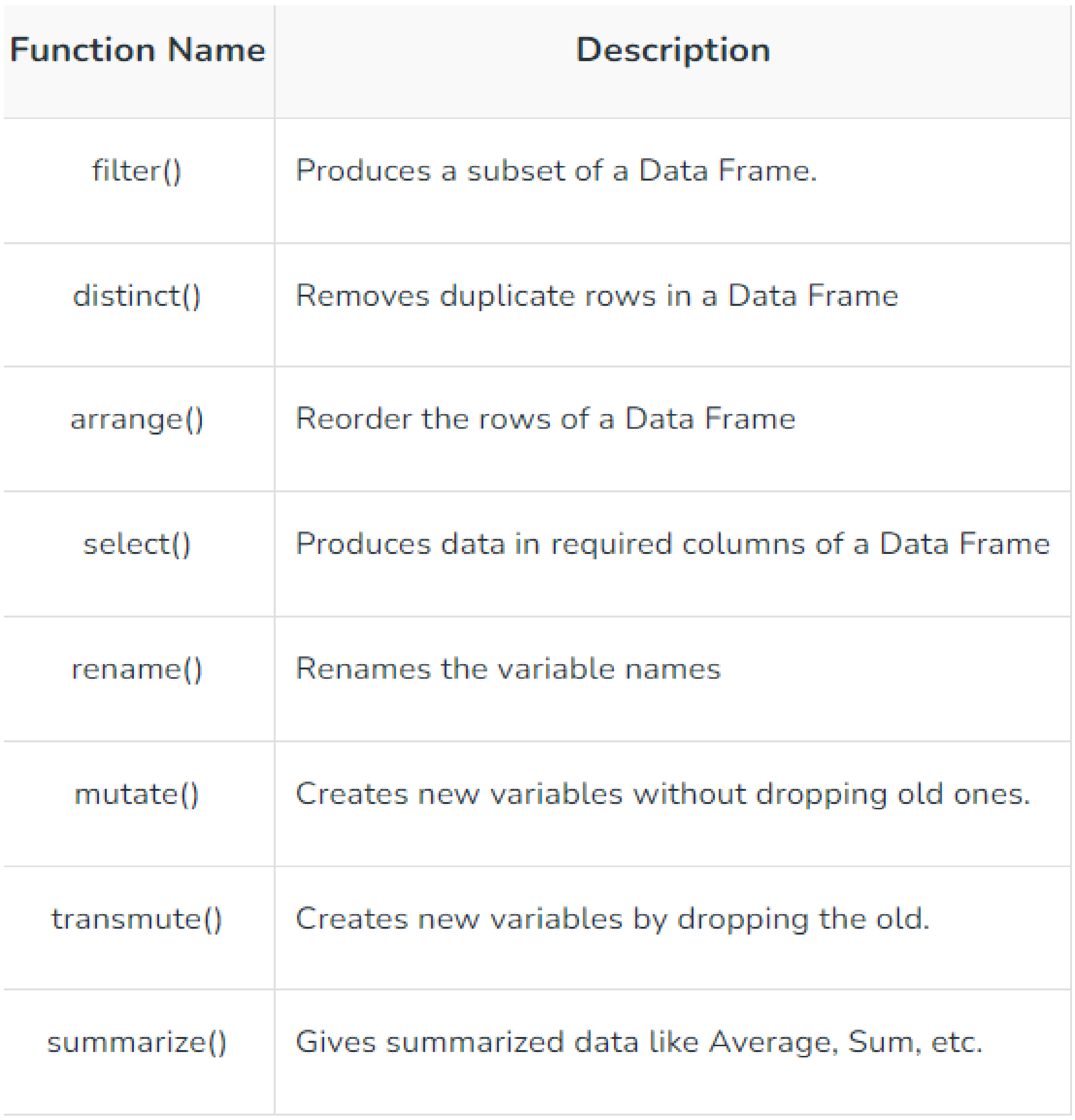
Output:

Week 5: Write a code to load the different packages and use the different functions to perform Data manipulation in R

Data Manipulation

Data manipulation involves modifying data to make it easier to read and to be more organized. We manipulate data for analysis and visualization. It is also used with the term 'data exploration' which involves organizing data using available sets of variables. At times, the data collection process done by machines involves a lot of errors and inaccuracies in reading. Data manipulation is also used to remove these inaccuracies and make data more accurate and precise

Data Manipulation in R with Dplyr Package



Function Name	Description
filter()	Produces a subset of a Data Frame.
distinct()	Removes duplicate rows in a Data Frame
arrange()	Reorder the rows of a Data Frame
select()	Produces data in required columns of a Data Frame
rename()	Renames the variable names
mutate()	Creates new variables without dropping old ones.
transmute()	Creates new variables by dropping the old.
summarize()	Gives summarized data like Average, Sum, etc.

Exercise 1: Create a DataFrame “stats” with following information:

Player = ('A', 'B', 'C', 'D', 'A', 'A')

Runs = (100, 200, 408, 19, 56, 100)

Wickets = (17, 20, NA, 5, 2, 17)

Display the following results:

1. **Fetch the data of players who scored more than 100 runs**
2. **Remove duplicate rows from data frame**
3. **Arrange data based on runs low to high**
4. **Display the wickets taken by each palyer**
5. **Change the column name “runs” to “runs_scored” in stats data frame.**
6. **Create a new column “avg”**
7. **Find total no of runs scored.**

```
library(dplyr)
# create a data frame
stats <- data.frame(player=c('A', 'B', 'C', 'D', 'A', 'A'),
                    runs=c(100, 200, 408, 19, 56, 100),
                    wickets=c(17, 20, NA, 5, 2, 17))
print(stats)

# fetch players who scored more than 100 runs
filter(stats, runs>100)

# removes duplicate rows
distinct(stats)

#remove duplicates based on a column
distinct(stats, player, .keep_all = TRUE)

# ordered data based on runs
arrange(stats, runs)

# fetch required column data: wickets taken by each player
select(stats, player,wickets)
# renaming the column
rename(stats, runs_scored=runs)

#we created a new column avg using mutate() and transmute() methods
mutate(stats, avg=runs/nrow(stats))
transmute(stats, avg=runs/nrow(stats))

# summarize method

summarize(stats, sum(runs), mean(runs))
```

Output:

Exercise 2: Perform the following data manipulation operations on “iris” dataset

1. **Return 5 random rows from the dataset**
2. **Find the frequency distribution of Species in iris table**
3. **Select all columns from Sepal.Length to Petal.Length**
4. **Hide the column Sepal.Length**
5. **Select the first 3 rows with Species as setosa**
6. **Create a column “Greater.Half” which stores TRUE if given condition is true   Sepal.Width > 0.5 * Sepal.Length**

```
#To load datasets package
library("datasets")
#To load iris dataset
data(iris)
summary(iris)
#To return 5 random rows
index<-sample(1:nrow(iris), 5)
index
iris[index,]
#To find the frequency distribution of Species in iris table
data(iris)
freq.table <- table(iris$Species)
head(freq.table)
#To select all columns from Sepal.Length to Petal.Length
selected1 <- select(iris, Sepal.Length:Petal.Length)
head(selected1)
#We use(-)to hide a particular column
selected <- select(iris, -Sepal.Length, -Sepal.Width)
head(selected)
#To select the first 3 rows with Species as setosa
```

```
filtered <- filter(iris, Species == "setosa" )

head(filtered,3)

#To create a column “Greater.Half” which stores TRUE if given condition

#is TRUE

col1 <- mutate(iris, Greater.Half = Sepal.Width > 0.5 * Sepal.Length)

head(col1)

tail(col1)
```

Output:

Week 6: Working with R – Charts and Graphs

R language is mostly used for statistics and data analytics purposes to represent the data graphically in the software. To represent those data graphically, charts and graphs are used in R.

R – Graphs

There are hundreds of charts and graphs present in R. For example, bar plot, box plot, mosaic plot, dot chart, histogram, pie chart, scatter graph, etc.

Types of R – Charts

- Bar Plot or Bar Chart
- Pie Diagram or Pie Chart
- Histogram
- Scatter Plot
- Box Plot

1. Bar Plot or Bar Chart

Bar plot or Bar Chart in R is used to represent the values in data vector as height of the bars. The data vector passed to the function is represented over y-axis of the graph.

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.

Exercise: Table shows the number of students in different department of a university.

Sr. No.	Year	Humanity	Science	Commerce
1	1996	2810	890	540
2	1997	3542	1363	471
3	1998	4301	1662	652
4	1999	5362	2071	895
5	2000	6593	2752	1113

(i) Represent the total number of students for different years by means of a simple bardiagram.
(ii) Represent the data as a subdivided barplot.

```
year.stud<-
matrix(c(2810,890,540,3542,1363,471,4301,1663,652,5362,2071,895,6593,2752,1113)
, byrow=T, ncol=3);
rownames(year.stud)<-c("1996","1997","1998","1999","2000");
colnames(year.stud)<-c("Humanity","Science","Commerce")
total.stud<-margin.table(year.stud,1)
print(total.stud)
# Give the chart file a name
png(file = "marginals.png")
barplot(total.stud, col="white", main="")
barplot(t(year.stud), col=c("white", "gray80", "black"), main="", beside=FALSE);
# Add the legend to the chart
regioan<- c("Humanity","Science","Commerce")
legend("topleft", regions, fill=c("white", "gray80", "black"), horiz=T)

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

Output:

2. **Pie Diagram or Pie Chart**

   Pie chart is a circular chart divided into different segments according to the ratio of data provided. The total value of the pie is 100 and the segments tell the fraction of the whole pie.

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

   where,

- *x is data vector*
- *labels shows names given to slices*
- *col fills the color in the slices as given parameter*
- *main shows title name of the pie chart*
- *radius indicates radius of the pie chart. It can be between -1 to +1*

Exercise: The tax revenue of India (in crores of Rs.), as provided in 1984-85 budget, when broken into various sources are given below. Represent the data by a pie chart.

Sources	**Excise**	**Customs**	**Corporationtax**	**Income tax**	**Other**
Tax revenue	**6526**	**7108**	**2568**	**560**	**763**

```
pie.tax<- c(6526, 7108, 2568, 560, 763);

names(pie.tax) <- c("Excise", "Customs", "Corporation", "Income", "Other");

# Give the chart file a name

png(file = "tax.png")

pie(pie.tax, main = "The Tax Revenue of India (1984-85)", col = c("white", "black",
"gray80", "gray50", "gray70"));

# Save the file

dev.off()
```

Output:

3. **Histogram**

   Histogram is same as bar chart but only difference between them is histogram represents frequency of grouped data rather than data itself.

   Syntax: hist(x, col, border, main, xlab, ylab)

   where:

- *x is data vector*
- *col specifies the color of the bars to be filled*
- *border specifies the color of border of bars*
- *main specifies the title name of histogram*
- *xlab specifies the x-axis label*
- *ylab specifies the y-axis label*

Exercise: The following table shows the projected population (in millions) of a country for the year 2005. The projections are broken down by age groups where grouping follow natural areas of interest such as preschool (below 5 years), education group (divided into 3 intervals, 5-13, 14-17 and 18-24), adult group (covering 25-64 years with 4 intervals of equal widths) and finally senior citizens' group (65 and above). Construct a histogram for the data.

Table: Projected population

Age Group	Projected Population
Below 5	**18**
5-14	**35**
14-18	**16**
18-25	**25**
25-35	**34**
35-45	**41**
45-55	**36**
55-65	**22**
65 and above	**32**

```
#We assume arbitrarily large upper bound, say 100, for the last class, which is open end class.

midx<- seq(12.5, 112.5, 25);

frequency<- c(5, 8, 13, 11, 3);

y <-rep(midx, frequency)

brk<- seq(0, 125, 25)

# output to be present as PNG file

png(file = "hist.png")

hist(y, breaks = brk, xlab = "Sales", main = "", col = "gray70"); midx<-c(2.5, 9.5, 16, 25, 30,
40, 50, 60, 82.5);
```

```
frequency<-c(18, 35, 16, 25, 34, 41, 36, 22, 32);

brk<-c(0.5, 14, 18, 25, 35, 45, 55, 65, 100);

y <-rep(midx, frequency);

hist(y, breaks=brk, xlab="Age Group", ylab="Agewise Projected
        Population", col="gray70");

# saving the file

dev.off()
```

Output:

4. **Scatter Plot**

   A Scatter plot is another type of graphical representation used to plot the points to show relationship between two data vectors.

   Syntax: plot(x, y, type, xlab, ylab, main)

   Where,

- *x is the data vector represented on x-axis*
- *y is the data vector represented on y-axis*
- *type specifies the type of plot to be drawn. For example, "l" for lines, "p" for points, "s" for stair steps, etc.*
- *xlab specifies the label for x-axis*
- *ylab specifies the label for y-axis*
- *main specifies the title name of the graph*

Exercise: Draw the scatter Plot of "Age VS Circumference" from "Orange" dataset.

```
#To load datasets package

library("datasets")

#To load orange dataset

data(Orange)

head(Orange)

# taking input from dataset Orange already present in R

orange <- Orange[, c('age', 'circumference')]

# output to be present as PNG file

png(file = "scatter.png")

# plotting

plot(x = orange$age, y = orange$circumference, xlab = "Age",

    ylab = "Circumference", main = "Age VS Circumference",

    col.lab = "darkgreen", col.main = "darkgreen",

    col.axis = "darkgreen")

# saving the file

dev.off()
```

Output:

5. **Box Plot**
   Box plot shows how the data is distributed in the data vector. It represents five values in the graph i.e., minimum, first quartile, second quartile(median), third quartile, the maximum value of the data vector**.**
   Syntax: boxplot(x, xlab, ylab, notch)
   where,

- *x specifies the data vector*
- *xlab specifies the label for x-axis*
- *ylab specifies the label for y-axis*
- *notch, if TRUE then creates notch on both the sides of* ***the box***

Exercise: Draw the box plot for following data:

Ages of employees: 42, 21, 22, 24, 25, 30, 29, 22, 23, 23, 24, 28, 32, 45, 39, 40

```
# defining vector with ages of employees

x <- c(42, 21, 22, 24, 25, 30, 29, 22, 23, 23, 24, 28, 32, 45, 39, 40)

   # output to be present as PNG file

png(file = "boxplot.png")

# plotting

boxplot(x, xlab = "Box Plot", ylab = "Age",

col.axis = "darkgreen", col.lab = "darkgreen")

# saving the file

dev.off()
```

Output:

Week 7: Write a code to draw graphs like scatter plot, box plots, bar plot, histogram on the different variables in the dataset using ggplot2

ggplot2 is a powerful and a flexible **R package**, implemented by **Hadley Wickham**, for producing elegant graphics. The **gg** in ggplot2 means **Grammar of Graphics**, a graphic concept which describes plots by using a "grammar".

According to ggplot2 concept, a plot can be divided into different fundamental parts:

Plot = data + Aesthetics + Geometry.

The principal components of every plot can be defined as follow:

- **data** is a data frame
- **Aesthetics** is used to indicate x and y variables. It can also be used to control the **color**, the **size** or the **shape** of points, the height of bars, etc…..
- **Geometry** corresponds to the type of graphics (**histogram**, **box plot**, **line plot**, **density plot**, **dot plot**, ….)

Two main functions, for creating plots, are available in **ggplot2** package : a **qplot()** and **ggplot()** functions.

- **qplot()** is a quick plot function which is easy to use for simple plots.
- The **ggplot()** function is more flexible and robust than **qplot** for building a plot piece by piece.

The generated plot can be kept as a variable and then printed at any time using the function **print**().

After creating plots, two other important functions are:

- **last_plot()**, which returns the last plot to be modified
- **ggsave("plot.png", width = 5, height = 5)**, which saves the last plot in the current working directory.

The table below shows common charts along with various important functions used in these charts.

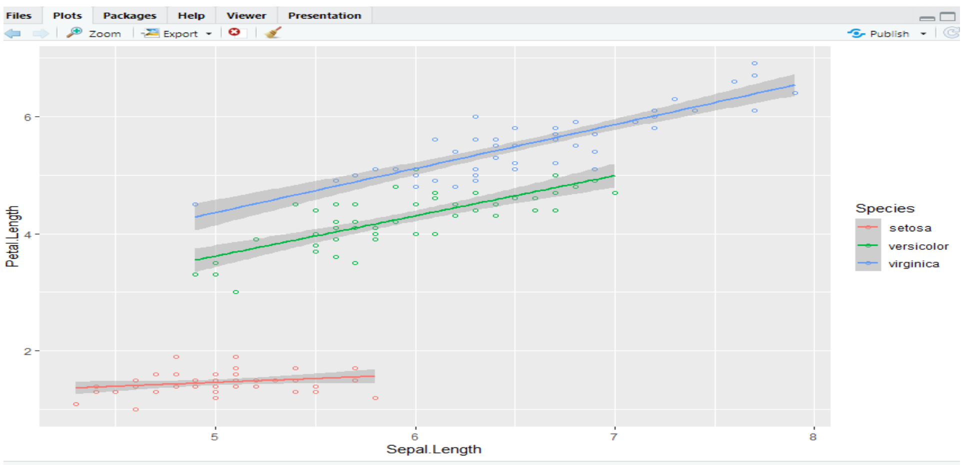
Important Plots	Important Functions
Scatter Plot	geom_point(), geom_smooth(), stat_smooth()
Bar Chart	geom_bar(), geom_errorbar()
Histogram	geom_histogram(), stat_bin(), position_identity(), position_stack(), position_dodge()
Box Plot	geom_boxplot(), stat_boxplot(), stat_summary()
Line Plot	geom_line(), geom_step(), geom_path(), geom_errorbar()
Pie Chart	coord_polar()

Exercise 1: Create a scatterplot on “iris” dataset. And perform the following:

- **Change the shape of points**
- **Add colour to the points**
- **Establish relationship between the variables.**
- **Add regression line.**

```
library(ggplot2)

data(iris)

#a basic scatter plot

ggplot(iris, aes(Sepal.Length, Petal.Length)) + geom_point()

# change the shape of points with a property called shape in geom_point() function.

ggplot(iris, aes(Sepal.Length, Petal.Length)) + geom_point(shape=1)

#Add color to the points

ggplot(iris, aes(Sepal.Length, Petal.Length, colour=Species)) + geom_point(shape=1)

#establishing relationship between the variables.

ggplot(iris, aes(Sepal.Length, Petal.Length, colour=Species)) + geom_point(shape=1)
+geom_smooth(method=lm)

# Add a regression line

ggplot(iris, aes(Sepal.Length, Petal.Length, colour=Species)) +

   geom_point(shape=1) +

   geom_smooth(method=lm)
```

Expected output:

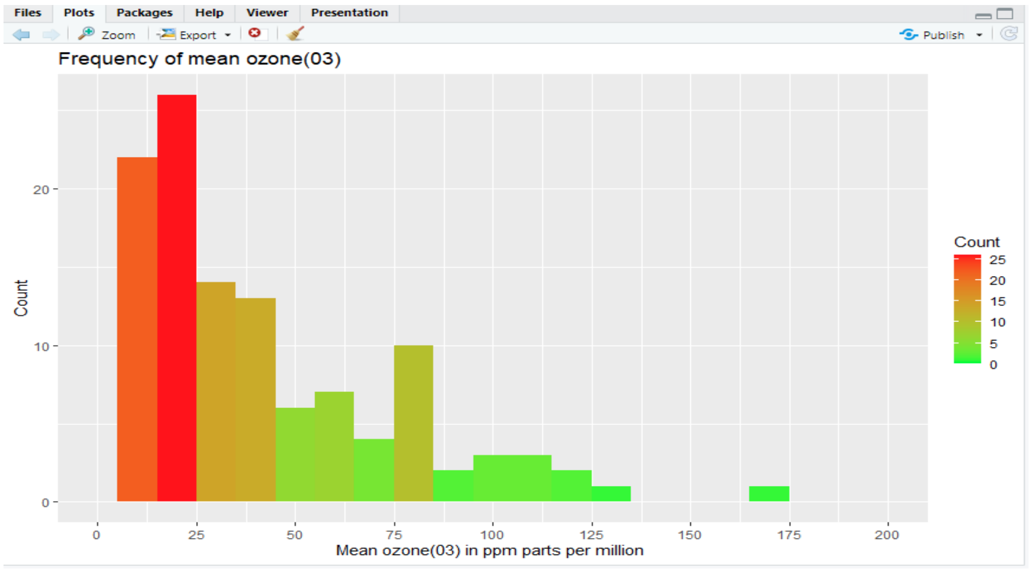


Output:

Exercise 2: Plot Frequency of Mean Ozone (O3) histogram on "airquality" Dataset.

```
plot_hist <- ggplot(airquality, aes(x = Ozone)) +
  # binwidth help to change the thickness (Width) of the bar
  geom_histogram(aes(fill = ..count..), binwidth = 10)+
   # name = "Mean ozone(03) in ppm parts per million "
  # name is used to give name to axis
  scale_x_continuous(name = "Mean ozone(03) in ppm parts per million ",
                     breaks = seq(0, 200, 25),
                     limits=c(0, 200)) +
  scale_y_continuous(name = "Count") +
   # ggtitle is used to give name to a chart
  ggtitle("Frequency of mean ozone(03)") +
  scale_fill_gradient("Count", low = "green", high = "red")
 plot_hist
```

Expected Output:



Output:

Week 8: Write a code to implement Word Clouds, Radar Charts, Waffle Charts in R.

Exercise 1: Create a simple word cloud from a text file "my.txt".

```
library(tm)
library(wordcloud)
#load the data file.
text <- readLines("my.txt")
head(text)
#we need to move the text into a tm element called a Corpus.
#First we need to convert the vector text into a VectorSource.
wc <- VectorSource(text)
wc <- Corpus(wc)
#pre-process the data
#removing punctuation
print(wc)
wc <- tm_map(wc, removePunctuation)
wc <- tm_map(wc, content_transformer(tolower))
wc <- tm_map(wc, removeWords, stopwords("english"))
wc <- tm_map(wc, stripWhitespace)
wordcloud(words = wc, scale=c(4,0.5), max.words=50)
wordcloud(words = wc, scale=c(4,0.5), max.words=50,random.order=FALSE)
```

Output:

Exercise 2: Prepare a Radar Chart for the following information.

	Biology	Physics	Maths	Sport	English	Geography	Art	Programming	music
max	20	20	20	20	20	20	20	20	20
min	0	0	0	0	0	0	0	0	0
Stu1	7.9	10	3.7	8.7	7.9	6.4	2.4	0	20
Stu2	3.9	10	11.5	20	7.2	10.5	0.2	0	20
Stu3	9.4	0	2.5	4	12.4	6.5	9.8	20	20

```
library(fmsb)
exam_scores <- data.frame(
  row.names = c("Student.1", "Student.2", "Student.3"),
  Biology = c(7.9, 3.9, 9.4),
  Physics = c(10, 20, 0),
  Maths = c(3.7, 11.5, 2.5),
  Sport = c(8.7, 20, 4),
  English = c(7.9, 7.2, 12.4),
  Geography = c(6.4, 10.5, 6.5),
  Art = c(2.4, 0.2, 9.8),
  Programming = c(0, 0, 20),
  Music = c(20, 20, 20)
)
exam_scores
# Define the variable ranges: maximum and minimum
max_min <- data.frame(
  Biology = c(20, 0), Physics = c(20, 0), Maths = c(20, 0),
  Sport = c(20, 0), English = c(20, 0), Geography = c(20, 0),
  Art = c(20, 0), Programming = c(20, 0), Music = c(20, 0)
)
rownames(max_min) <- c("Max", "Min")
# Bind the variable ranges to the data
df <- rbind(max_min, exam_scores)
df
student1_data <- df[c("Max", "Min", "Student.1"), ]
radarchart(student1_data, pcol = "Green")
```

Output:

Exercise 3: Prepare a Waffle Charts for the following information.
Dataset of 91822 persons categorized as:
Infants <1 = 16467
Children <11 = 30098
Teens 12-17 = 20354
Adults 18+ = 12456
Elderly 65+ = 12456

```
library(ggplot2)

library(waffle)

Infants<- 16467

Children<-  30098

Teens<- 20354

Adults<- 12456

Elderly <-  12456

expenses <- c(`Infants: <1(16467) `=16467, `Children:  <11(30098) `=30098,

            `Teens: 12-17(20354)`=20354, `Adults:18+(12456) `=12456,

            `Elderly: 65+(12456) `=12456)

waffle(expenses/1000, rows=5, size=0.6,

      colors=c("#44D2AC", "#E48B8B", "#B67093",

              "#3A9ABD", "#CFE252"),

      title="Age Groups bifurcation",

      xlab="1 square = 1000 persons")
```

Output:

Week 9: Write a code to draw maps using different packages in R.

Exercise:

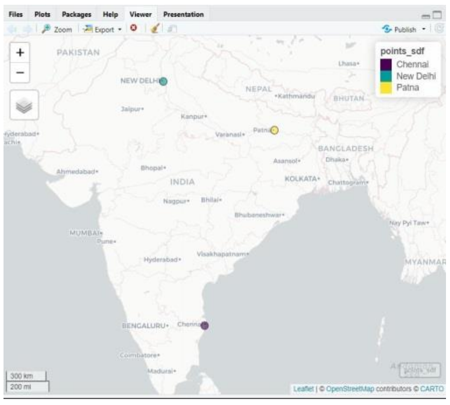
a) Pont the cities “Patna”, “New Delhi”, “Chennai” on India map using mapview.

b) Use the leaflet() function to initialize the map and draw an interactive map that can be zoomed in and out and have popups “Patna City” displayed on markers.

# using mapview

```
library(mapview)

# Create example data of points

lng = c(85.21, 80.23, 77.28)

lat = c(25.59, 12.99, 28.56)

names = c("Patna", "Chennai", "New Delhi")

# Create a data frame with the point data

points_df = data.frame(lng, lat, names)

# Convert the data frame to a spatial points data frame

points_sdf = st_as_sf(points_df,

                     coords = c("lng", "lat"), crs = 4326)

# Plot the points on a map

mapview(points_sdf, label = points_sdf$names)
```

Expected output:

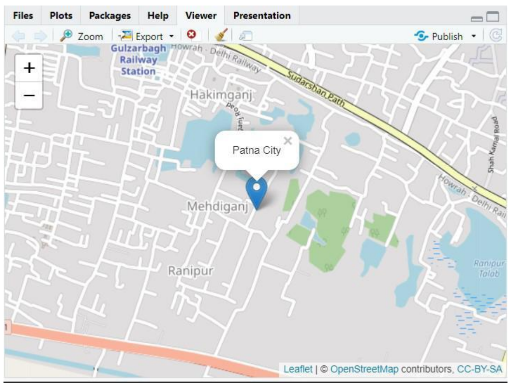


Output:

# using leaflet:

```
# Creating an Interactive Web Map with a leaflet
library(leaflet)
leaflet() %>%
addTiles() %>%
addMarkers(lng = 85.21, lat = 25.59,  popup = "Patna City")
```

Expected output:



Output:

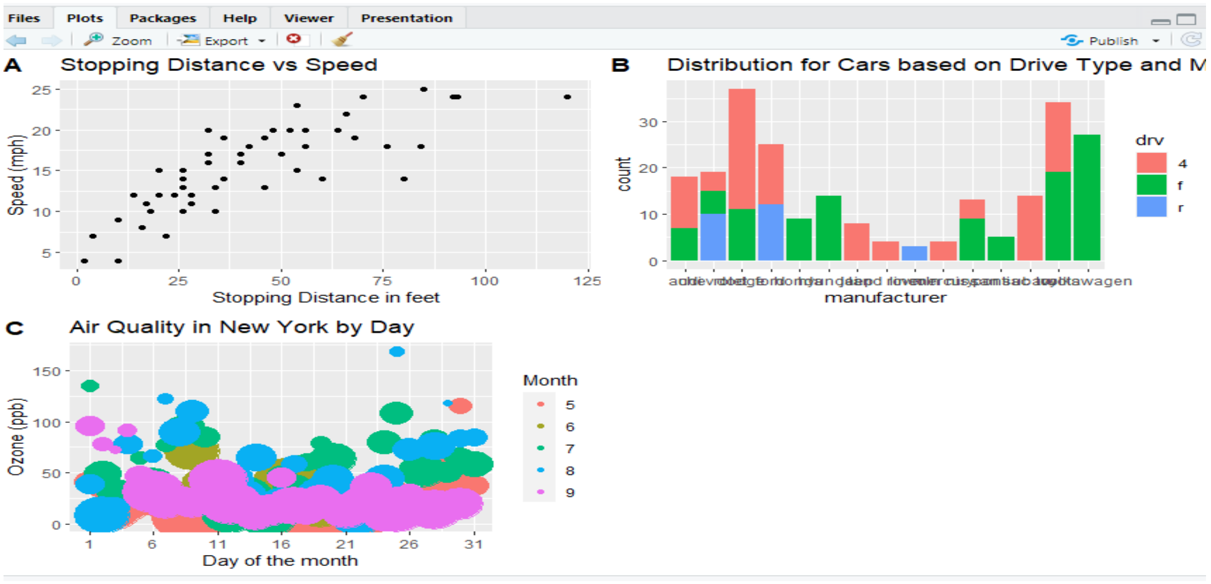
Week 10: Creating Interactive ggplot2 graphs with Plotly in R: Scatter plot, Bar plot, Bubble Plot.

Exercise:

a) **Draw the Distance versus speed scatter plot from the car dataset and add interaction using ggplotly() .**
b) **Create an interactive Bar chart using "mpg" Dataset in R.**
c) **Create an interactive Bubble graph using the "airquality" dataset in R.**
d) **Display all the graphs on single page**

```
# Interactive Scatter plot

install.packages("ggpubr")

install.packages("plotly")

install.packages("ggplot2")

install.packages("dplyr")

install.packages("car")

install.packages("babynames")

install.packages("gapminder")

library(ggpubr)

library(plotly)

library(dplyr)

library(carData)

library(gapminder)

library(babynames)

p1 <- ggplot(data=cars, aes(dist,speed)) + geom_point() +

  ggtitle("Stopping Distance vs Speed") +

  xlab("Stopping Distance in feet") +

  ylab("Speed (mph)")

ggplotly(p1)


#Interactive Bar Chart

p2 <- ggplot(mpg, aes(manufacturer)) + geom_bar(aes(fill = drv))+
ggtitle("Distribution for Cars based on Drive Type and Manufacturers")
ggplotly(p2)

# Interactive Bubble graph

P3 <- ggplot(airquality, aes(x = Day, y = Ozone,

                                   color= as.factor(Month),

                                   text = paste("Month:", Month))) +

  geom_point(size = airquality$Wind) +

  ggtitle("Air Quality in New York by Day") +

  labs(x = "Day of the month", y = "Ozone (ppb)", color="Month") +

  scale_x_continuous(breaks = seq(1, 31, 5))+

  scale_size(range = c(1, 10))

ggplotly(p3, tooltip=c("text","x","y"))

#display Multiple Graphs on The Same Page

ggarrange(p1, p2, p3,

          labels = c("A", "B", "C"),

          ncol = 2, nrow = 2)
```

Expected output:



Output: