Module: Summary

Introduction to R

In this session, you'll get started with the most popular programming language among data scientists - R. The R language is widely used for data mining, statistics and data analysis - both by newbie data analysts and top companies and research groups. R is a free open source and it is a language which means that you do data analysis by writing functions and scripts not by pointing and clicking like you do in Excel. It is very easy to pick up for beginners and is sophisticated enough to perform highly advanced data analysis. Through this lecture, you will realise the importance of programming in data analytics. You will also see how and where R is being used by data scientists, as well as understand the reasons for its popularity.

Installing R & R-Studio

You need to install both R and R-Studio on your system before actually getting started with R. In this page, you will be guided through the installation process and get introduced to both of them.

Install R

Step 1: Download the package relevant to your system (Windows or Mac or Linux) from the CRAN website.

Step 2: Install R like you normally install any new software package.

Now, Install **R-Studio**

Step 1: Download the R-Studio Desktop package from the R-Studio website.

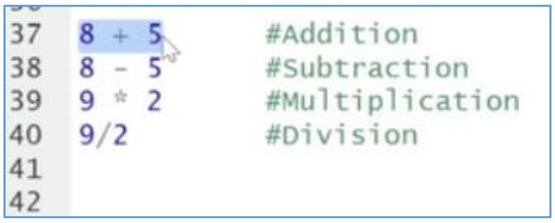
Step 2: Install R-Studio using user's setup process. Install R

Step 1: Download the package relevant to your system (Windows or Mac or Linux) from the CRAN website.

Step 2: Install R like you normally install any new software package.

Basic Operations in R

Basic Arithmetic in R:

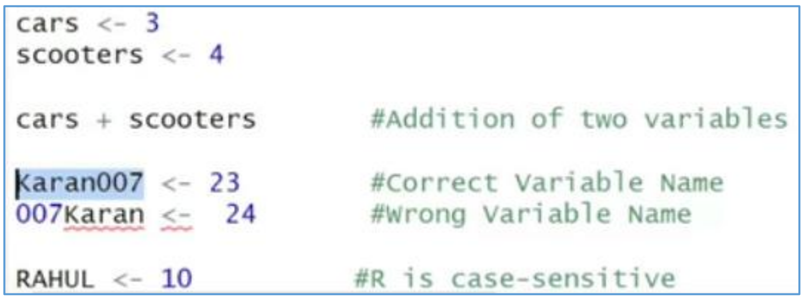


```
8 + 5        #Addition
8 - 5        #Subtraction
9 * 2        #Multiplication
9/2          #Division
```

Figure 1: Operations in R

Assignment Operations

- You can assign value to any variable using "<-" symbol.



```
cars <- 3
scooters <- 4

cars + scooters         #Addition of two variables

Karan007 <- 23          #Correct Variable Name
007Karan <-  24         #Wrong Variable Name

RAHUL <- 10            #R is case-sensitive
```

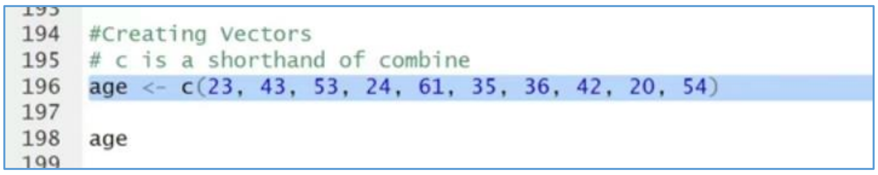
Figure 2: Assignment Operation

Vectors

All well-designed structures have strong building blocks; as buildings have bricks, similarly datasets in R have vectors.

Datasets have rows and columns. In R, they are frequently stored as vectors which are sequences of elements of the same data type (numeric, character etc.).

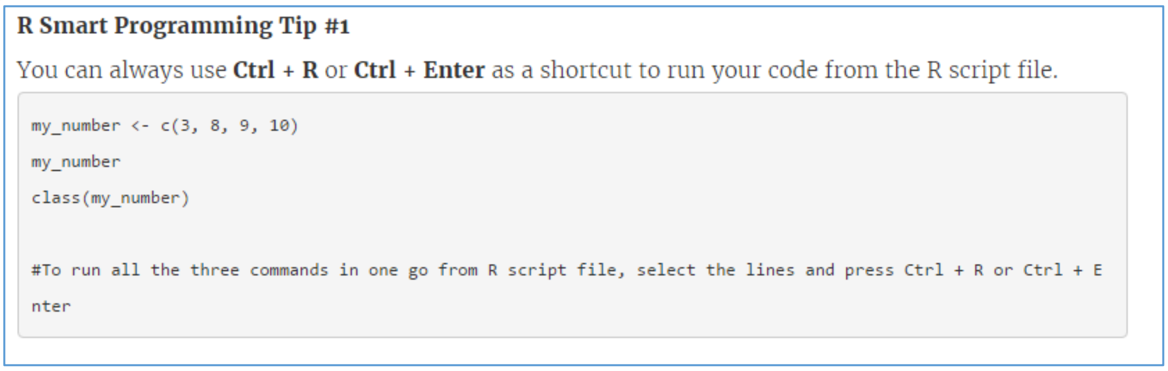
The simplest structure of data is a vector, where data elements are stored one after another. For example, the age of 10 people can be stored as a vector of 10 numbers.



```
#Creating Vectors
# c is a shorthand of combine
age <- c(23, 43, 53, 24, 61, 35, 36, 42, 20, 54)

age
```

Figure 3: Vectors

One important thing to note is that the index starts from 1 in R, unlike other languages where the index may start from 0. The first element in R is accessed by vector[1]; in some other languages, index of the 1st element is 0.



R Smart Programming Tip #1

You can always use **Ctrl + R** or **Ctrl + Enter** as a shortcut to run your code from the R script file.

```
my_number <- c(3, 8, 9, 10)
my_number
class(my_number)

#To run all the three commands in one go from R script file, select the lines and press Ctrl + R or Ctrl + E
nter
```

Figure 4: R Tip #1

Remember that datasets in R are stored as rows and columns of vectors. The more comfortable you are with vectors, the better you would be with datasets in R.

- **Swirl Install**

Install Swirl package in your R console and start playing around with some pre-installed sessions from Swirl course repository. These commands will help you get started.

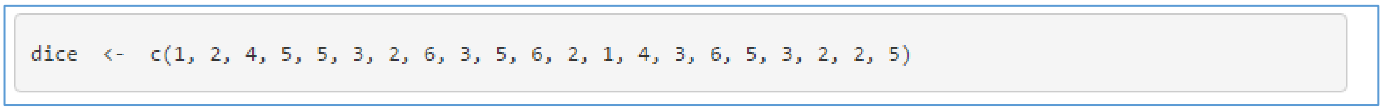
```
install.packages("swirl")
library(swirl)
swirl()
```

You can start with any of the basic R Programming lessons. If you are stuck somewhere, this link will help you get started in SWIRL. Try exploring SWIRL after installing the package because your first SWIRL lesson is coming up in the next segment

Factor

The next thing you should know about is Factors. The factor is a data type in R used to categorise values into discrete categories.

For example, say a numeric vector to store the values of a die roll is created:



```
dice  <-  c(1, 2, 4, 5, 5, 3, 2, 6, 3, 5, 6, 2, 1, 4, 3, 6, 5, 3, 2, 2, 5)
```

Figure 5: dice vector

Since there are only 6 possible values, one can convert the vector into 6-factor levels using a built-in function factor.

```
dice_levels <-  factor(dice)

dice_levels
```

Figure 6:Factor

If you now look at the new vector levels in the console, R will show all elements of dice vector, along with the 6 levels mentioned.

1 2 4 5 5 3 2 6 3 5 6 2 1 4 3 6 5 3 2 2 5

Levels: 1   2   3   4   5   6

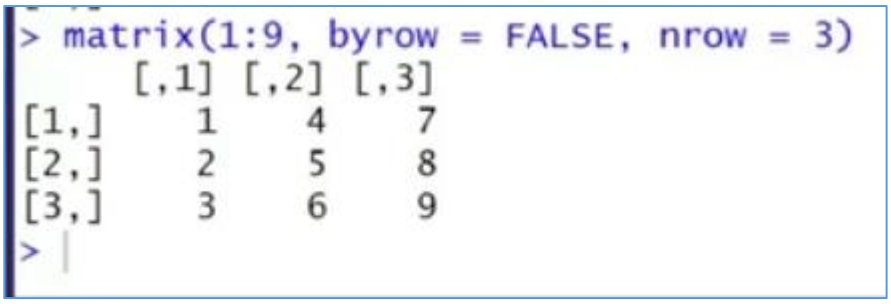
R Smart Programming Tip #2

You can use **Ctrl + L** to clear your console screen.

Figure 7:R tip#2

Matrices

The most basic two-dimensional data structure in R is called a matrix, with all elements in rows and columns having the same data-type as shown in below figure.



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

Figure 8:Matrix

R Smart Programming Tip #3

Always use # to write comments in your R script file – to mention the context about commands and to avoid getting lost in the script file.

```
# You can write whatever you want after putting a hashtag


# R does not take it seriously
```

Figure 9:R tips#3

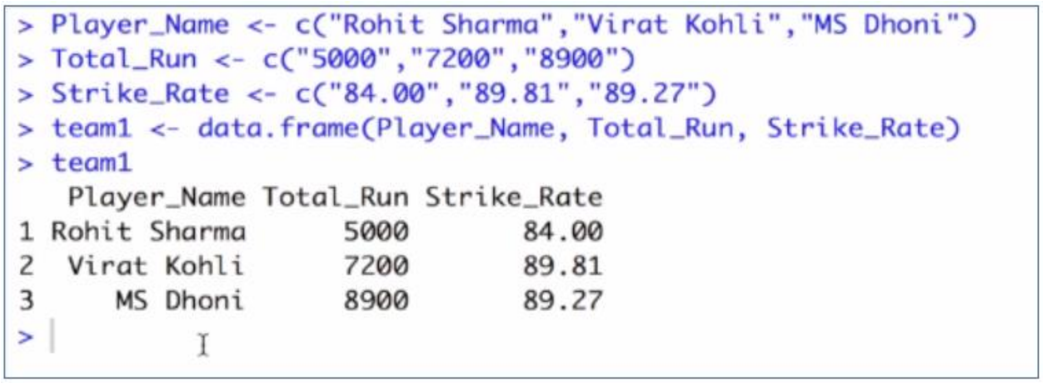
Data Frame

Data frames are the most important data structures in R, and data frames use vectors as their building blocks.

In almost every data analysis project, data frames are used to store data in R.

- **Creating Data frame**

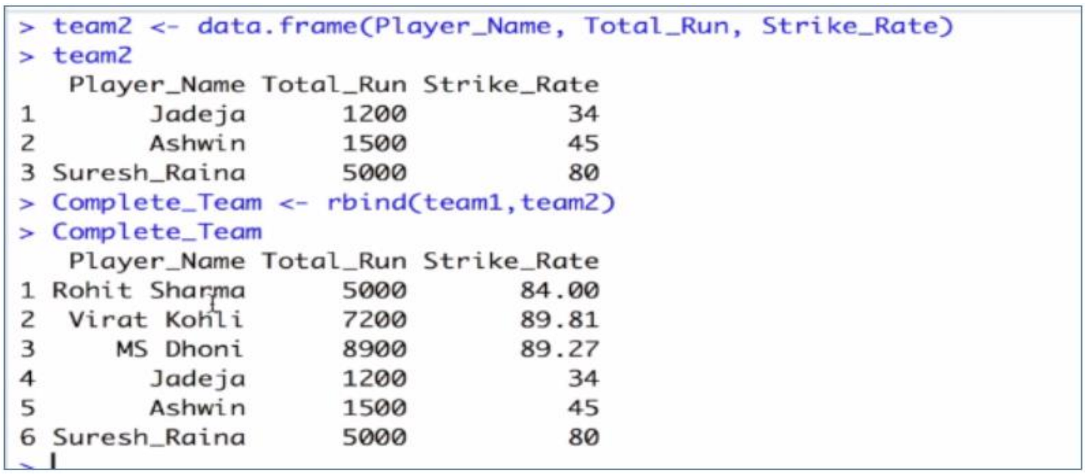
Data frame is created by using a data.frame function to create a data frame containing 3 rows and 3 columns, We will create 3 vectors and put them as columns of the data frame.



```
> Player_Name <- c("Rohit Sharma","Virat Kohli","MS Dhoni")
> Total_Run <- c("5000","7200","8900")
> Strike_Rate <- c("84.00","89.81","89.27")
> team1 <- data.frame(Player_Name, Total_Run, Strike_Rate)
> team1
   Player_Name Total_Run Strike_Rate
1 Rohit Sharma      5000       84.00
2  Virat Kohli      7200       89.81
3     MS Dhoni      8900       89.27
> 
```

Figure 10: team1 dataframe

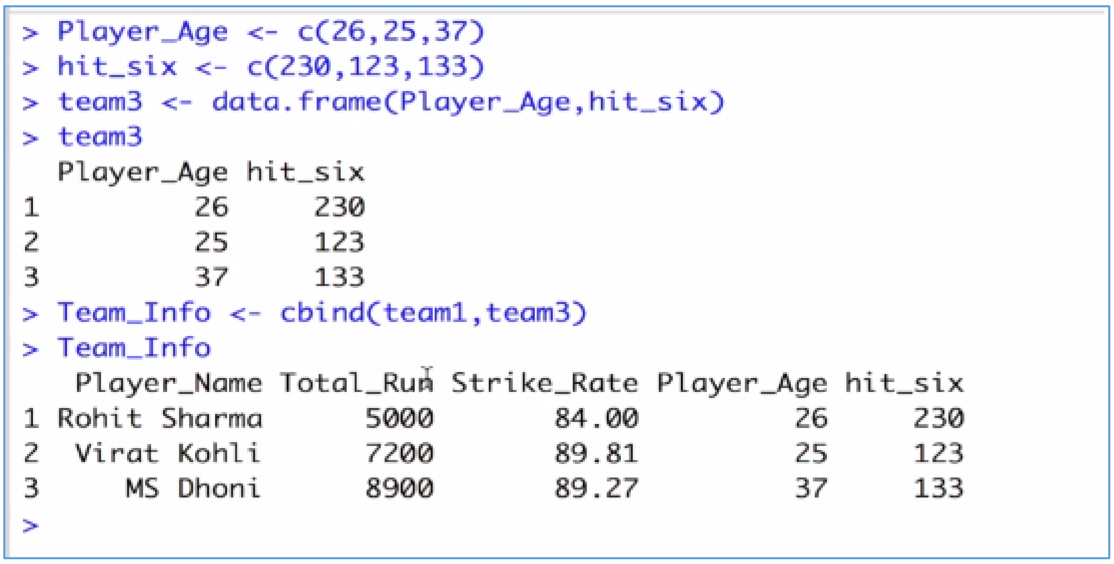
- rbind function



```
> team2 <- data.frame(Player_Name, Total_Run, Strike_Rate)
> team2
   Player_Name Total_Run Strike_Rate
1      Jadeja      1200          34
2      Ashwin      1500          45
3 Suresh_Raina     5000          80
> Complete_Team <- rbind(team1,team2)
> Complete_Team
   Player_Name Total_Run Strike_Rate
1 Rohit Sharma      5000       84.00
2  Virat Kohli      7200       89.81
3     MS Dhoni      8900       89.27
4       Jadeja      1200          34
5       Ashwin      1500          45
6 Suresh_Raina      5000          80
> 
```

Figure 11: rbind function

- cbind function



```
> Player_Age <- c(26,25,37)
> hit_six <- c(230,123,133)
> team3 <- data.frame(Player_Age,hit_six)
> team3
  Player_Age hit_six
1         26     230
2         25     123
3         37     133
> Team_Info <- cbind(team1,team3)
> Team_Info
   Player_Name Total_Run Strike_Rate Player_Age hit_six
1 Rohit Sharma      5000       84.00         26     230
2  Virat Kohli      7200       89.81         25     123
3     MS Dhoni      8900       89.27         37     133
>
```

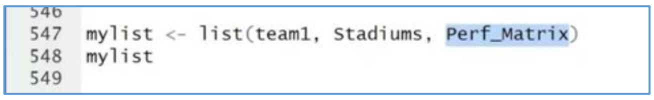
Figure 12:cbind function

** names in the dataframe might be different than used in the video.

Lists

You now have an armoury of data structures and data types to work with - vectors, data frames, matrices, characters, factors etc. In a typical R project, you may need to use multiple data structures.

Naturally, the next problem to solve is to store them together. This is where lists come in. A List is a one-dimensional data structure (as shown in below figure) – similar in appearance to a vector - in each element is itself a data structure. Each element of a list could be a vector, matrix, data frame or even a list itself.



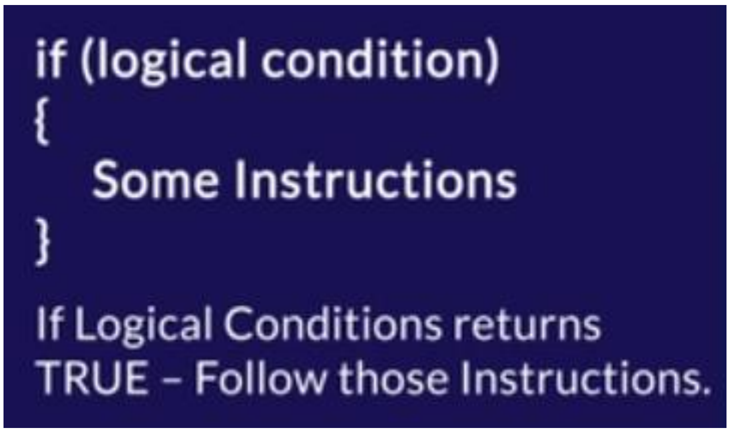
```
mylist <- list(team1, Stadiums, Perf_Matrix)
mylist
```

Figure 13:mylist

This session's objective is to take your skills in R to an intermediate level, where you'll learn some more new tools (or constructs, what Prof RC generally calls it), which are common to most programming languages.

- **Conditional Statements**

The "if" conditional statement. As the name implies, conditional statements are used to implement code if some predefined condition is satisfied.



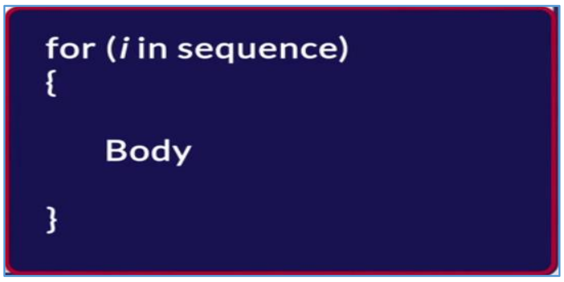
```
if (logical condition)
{
    Some Instructions
}
```

If Logical Conditions returns TRUE – Follow those Instructions.

Figure 14:If condition

- **Loops**

Loops are used when you have to repeat a task multiple times. In R programming and data analysis, loops are mostly used to apply a particular logic over a sequence of data items/objects.



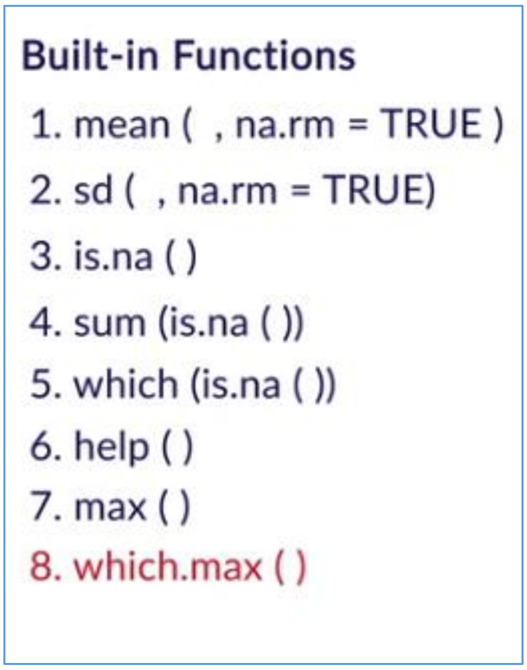
```
for (i in sequence)
{

    Body

}
```

Figure 15: For loop

Apart from the 'for' loop, there are other loops as well, like 'while' and 'repeat'. There also exist statements like break( ) and next( ) which can be used inside loops in certain situations.

Functions

You have already used functions in R - mean(), sqrt(), exp(), etc. These functions are called built-in functions. Study a few more useful functions by applying them on the bank dataset.

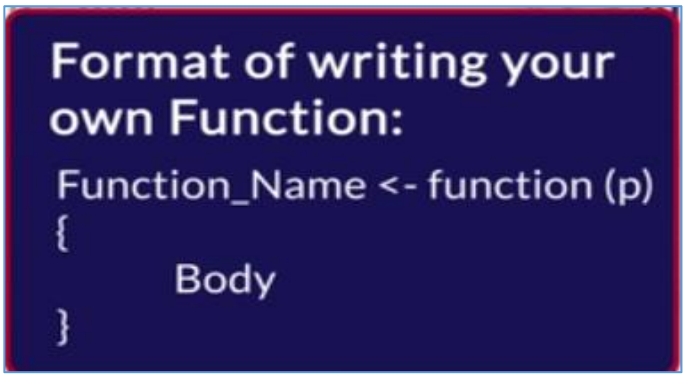


Built-in Functions

1. mean (, na.rm = TRUE)
2. sd (, na.rm = TRUE)
3. is.na ( )
4. sum (is.na ( ))
5. which (is.na ( ))
6. help ( )
7. max ( )
8. which.max ( )

Figure 16:Built-in Functions

Apart from the already built-in functions, you can create your own functions(structure of the function is shown in fig.) and use whenever you want.



Format of writing your own Function:

```
Function_Name <- function (p)
{
        Body
}
```

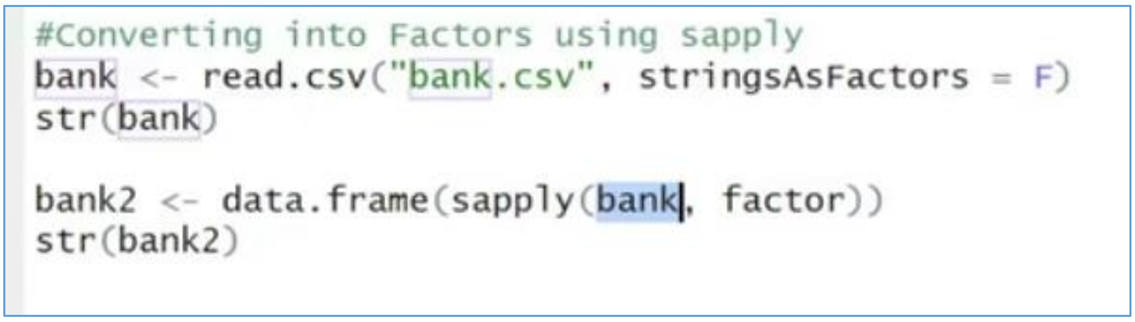
Figure 17:Creating your own Functions

Apply Family

There is an extremely useful set of functions collectively referred to as the Apply family. Some common functions in this family are sapply(), lapply() etc.

These functions are typically used as an alternative to writing a 'for' loop, e.g. to apply a particular function like mean() to a sequence of vectors.

- Sapply function



```
#Converting into Factors using sapply
bank <- read.csv("bank.csv", stringsAsFactors = F)
str(bank)

bank2 <- data.frame(sapply(bank, factor))
str(bank2)
```

Figure 18:sapply function

Summary

Relational and Logical Operators Vectors - Used on different R objects to define your conditions

Conditional Statements - Used to implement any logic for decision-making

Loops - Used when the same logic has to be applied over a sequence of objects

Functions - Used as a black-box, which takes in different R objects and gives output in any required format

APPLY Family - Used when a particular function has to be applied over a sequence of objects.