Fundamentals of R

Prepared By Deepali Sonawane, Assistant Professor, SIOM

Introduction of R

- R is a **programming language.**
- R was developed by **Ross Ihaka** and **Robert Gentleman** at the University of Auckland, New Zealand in August 1993.
- R is often used for **statistical computing** and **graphical** presentation to analyse and visualize data.
- R is an **open-source programming language.**
- It is available across widely used **platforms** like Windows, Linux, and MacOS.
- The core of R is an **interpreted** computer language which allows branching and looping as well as modular programming using functions.

Features of R

- R programming is a **great resource** for data analysis, data visualization, data science and machine learning.
- It's a **platform-independent** language.
- It's an **open-source** free language.
- R is a well-developed, simple and effective **programming language** which includes conditionals, loops, user defined recursive functions and input and output facilities.
- R has an effective **data handling** and **storage facility.**
- R provides many **packages** (libraries of functions, objects) for calculations on arrays, lists, vectors and matrices.
- R provides **graphical facilities** for data analysis and it is easy to draw graphs like pie charts, histograms, box plot, scatter plot etc.

Downloading, Installing R and R studio

- **Steps to install R**

1) Use the link https://cran.r-project.org/bin/windows/
2) Click on install R for the first time
3) Click on Download R-4.2.0 for windows
4) Double click on .exe file and run the installer.

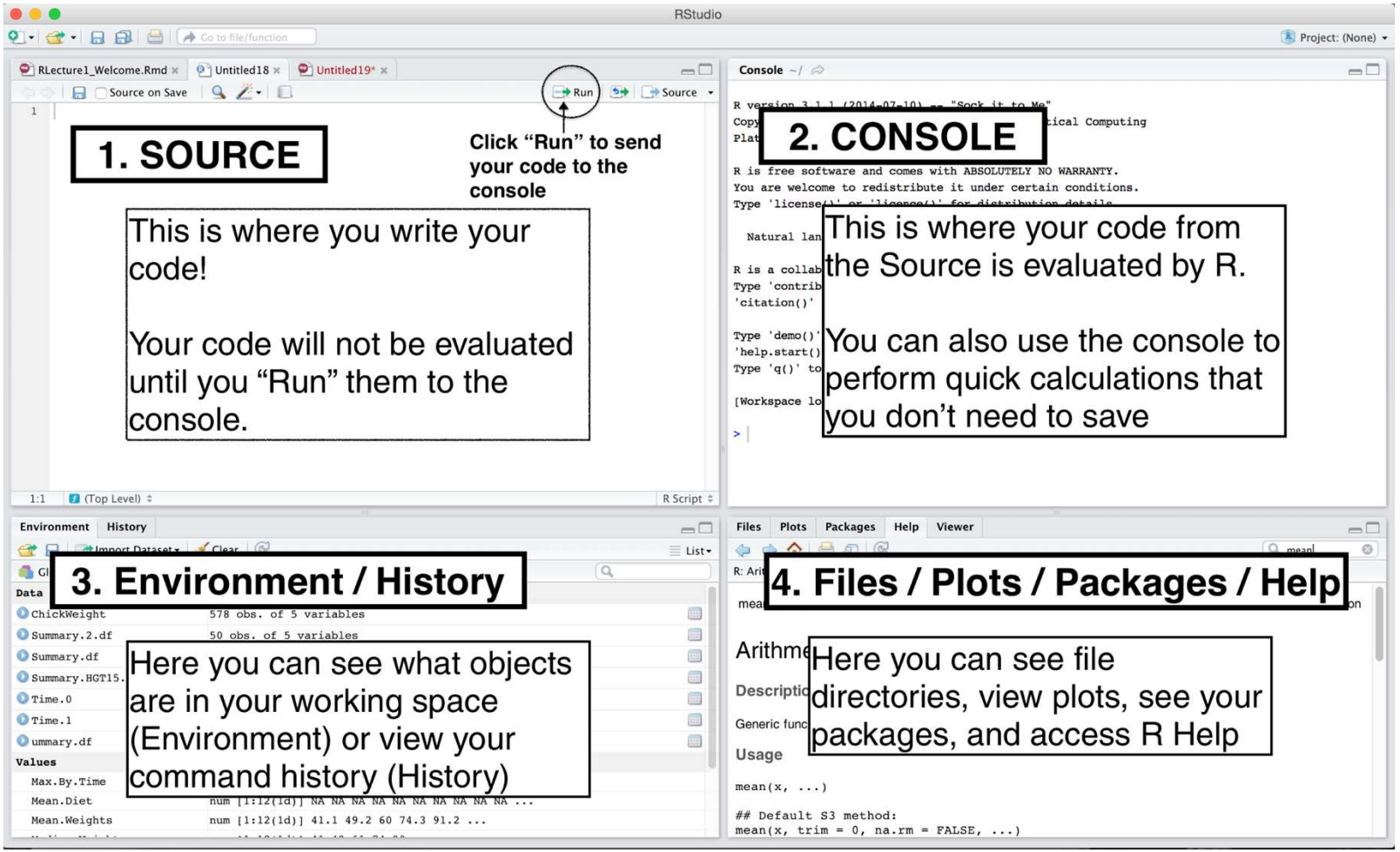
- **Steps to install R studio**

1) Use the link https://www.rstudio.com/products/rstudio/download/
2) Click on download for RStudio Desktop and download RStudio-2022.02.2-485.exe file
3) Double click on .exe file and run the installer accepting the default settings.

Introduction to R Studio

- R Studio is an **Integrated Development Environment** (IDE) for R.
- R studio's primary purpose is to create **free and open source software** for data science, scientific research and technical communication.
- It includes a **console**, **syntax-highlighting editor** that supports direct code execution.
- It is used for **plotting, history, debugging** and **workspace management**.

R Environment



Command line

- R provides two ways to write a program
  1. **R command prompt** : A prompt > where you can start typing your R program.
  2. **R script file :** You can do your programming by writing your programs in script files and then you execute those scripts at your command prompt with the help of Run option which is available in R studio.

Help

- The function **help()** in R provide access to the documentation pages for R functions, data sets and objects,
- **help.start()** starts and displays a hypertext based version of R's **online documentation** in your default browser that provides links to locally installed versions of the R manuals, a listing of your currently installed packages and other documentation resources.
- Example: To access documentation for the standard print function, you can use the command **help('print')** .

Comments

- Comments are like **helping text** in your R program.
- Comments can be used to **explain** R code, and to make it more **readable**.
- It can also be used to **prevent execution** when testing alternative code.
- They are ignored by the interpreter while executing your actual program.
- Single comment is written using # in the beginning of the statement.
- Example : #This is my first R program
- R does not support multi-line comments.
- The easiest way to create a multi-line comment in RStudio is to highlight the text and press **Ctrl + Shift + C**.

Writing your first code

- **Code 1:**

  print("Business Analytics")

- **Code 2:**

  s<-"MBA"

  print(s)

- **Code 3:**

  a<-10

  b<-20

  c=a+b

  print(c)

Concatenation using paste()

- **Example:**

```
name<-"Rohini"
course<-"MBA"
special<-"Business Analytics"
print(paste(name," is a student of ",course," and having ",special," specialization"))
```

Handling Input and Output

- **Example 1:**

```
name<-readline(prompt="Enter student name : ")
course<- readline(prompt="Enter student course : ")
special<- readline(prompt="Enter student specialization : ")
print(paste(name," is a student of ",course," and having ",special," specialization"))
```

- **Example 2:**

```
num1<-as.integer(readline(prompt="Enter first number : "))
num2<-as.integer(readline(prompt="Enter second number : "))
print(paste("Addition is ",num1+num2))
```

File Operations – Working with directories

- **getwd()** : This function returns an absolute file path representing the **current working directory** of the R process. To get the current directory in R, getwd() function is used.

  Syntax : getwd()

  Example : getwd()

- **setwd()** : This function changes the current directory as a working directory. This method takes a **new working directory** as an argument. Use forward slash or two back slashes while specifying the path.

  Syntax : setwd(dir)

  Example : setwd('D:/SIOM/MBA') OR

  setwd('D:\\SIOM\\MBA')

Reading from and writing to text file

- **readLines()** : This function reads **text lines** from an input file. It reads the text line by line and creates **character objects** for each of the lines.

  Syntax : readLines(FilePath)

  Example : d<- readLines("subject.txt")

  d

- **write()** : This function is used to **write the contents** into text file.

  Syntax : write(String,FilePath)

  Example 1: s<-'Business Analytics'

  write(s,'D:\\SIOM\\MBA\\subject.txt')

  Example 2 : s<-'101 \t BA \n 102 \t IB \n 103 HRM'

  write(s,'D:\\SIOM\\MBA\\specialization.txt')

Reading from and writing to csv file

- **read.csv()** : This function reads a **csv file in table format** and creates a data frame from it.

  Syntax : read.csv(FilePath, header=TRUE/FALSE)

  Example : d<- read.csv("students.csv",header=FALSE)

  d

- **write.csv()** : This function is used **to write** the contents into csv file.

  Syntax : write.csv(contents,FilePath)

  Example : write(d,'D:\\SIOM\\MBA\\admissions.csv');

Importing data from spreadsheets

- **read.xlsx()** : This function **reads an excel file**. It is imported from the xlsx library of R language.

  Syntax : read.xlsx(FilePath, sheetIndex)

  Example :

Method 1

```
install.packages("xlsx")
library(xlsx)
d1<-read.xlsx("Marks.xlsx");
d1 #View(d1);
```

Method 2

```
install.packages("openxlsx")
library(openxlsx)
d2<-openxlsx::read.xlsx("Marks.xlsx");
View(d2);
```

Data exploration functions...

- str(object): It is used to display structure of excel file.
- head(object): It is used to display top 6 rows.
- head(object,n): It is used to display top n rows.
- tail(object) : It is used to display last 6 rows.
- tail(object,n) : It is used to display last n rows.
- dim(object) : It is used to display dimension of excel in terms of row and column.
- nrow(object) : It is used to display number of rows of the excel file.
- ncol(object) : It is used to display number of columns of the excel file.

Data exploration functions

- max(object$column_name): It is used to get maximum value of the specified column.
- sum(object$column_name): It is used to get sum of the specified column.
- mean(object$column_name): It is used to get average of the specified column.
- range(object$column_name): It is used to display minimum and maximum value of the specified column.
- summary(object): It is used to display summary of excel file.
- subset(object,object$column_name): It will create subset of excel file.

  Example: subset(d,d$Salary>3000)

Importing data from text file and csv file

- **read.table() :** Reads a **file in table format** and creates a data frame from it, with cases corresponding to lines and variables to fields in the file.

  Syntax : read.table(file, header = FALSE, sep = "", quote = "\"",
  dec = ".".....)

- **read.delim() :** Reads a file in table format and creates a data frame from it, with cases corresponding to lines and variables to fields in the file.

  Syntax : read.delim(file, header = TRUE, sep = "\t", quote = "\"",
  dec = ".".....)

Importing data from SAS file

- SAS stands for **Statistical Analysis Software** is a widely used statistical software system. When a SAS Statistics data file is saved, the file extension . sas7bdat is used.
- **read_sas() :** This function **reads a sas file**. It is imported from the heaven library of R language.

  Syntax : read_sas(FilePath)

  Example:

```
install.packages("haven")
library(haven)
read_sas('employee.sas7bdat')
```

Importing data from SPSS file

- SPSS stands for **Statistical Package for the Social Sciences** is a widely used statistical software system. When an SPSS Statistics data file is saved from SPSS, the file extension . sav is used
- **read_spss() :** This function **reads a spss file**. It is imported from the heaven library of R language.

  Syntax : read_spss(FilePath)

  Example:

```
install.packages("haven")
library(haven)
read_spss('employee.sav')
```

Connect to RDBMS from R using ODBC

- **RDBMS** : Relational Database Management System
- **ODBC** : Open Database Connectivity
- **DSN** : Data Source Name
- Link to install SQL Server express : https://www.microsoft.com/en-in/sql-server/sql-server-downloads
- Link to install SQL Server Management Studio : https://docs.microsoft.com/en-us/sql/ssms/download-sql-server-management-studio-ssms?view=sql-server-ver16

Connect to RDBMS from R using ODBC

- Code to connect to RDBMS using ODBC

```
install.packages("RODBC")
library(RODBC)
conn<-odbcConnect("DSN_Name") #Conn of SQL server with ODBC DSN
sqlTables(conn) #Show all tables of database
rec<-sqlQuery(conn,"Select * from tablename") #select records from table
rec      OR
rec<-sqlFetch(conn,"tablename") #Fetch records from specified table
rec
```

Basic SQL queries in R

SQL stands for Structure query language.

Types of SQL statements are follows:

1. Data Definition Language statements (DDL) : Create, alter, drop, truncate
2. Data Manipulation Language statements (DML) : Insert, update, delete
3. Data Query Language statements (DQL) : Select

DQL : Select queries

- Select * from tablename : Returns entire table
- Select column1,column2…from tablename : Returns specific columns
- Select * from tablename where columnname=value : Returns specific rows
- Select * from tablename order by columnname : Returns table in ascending order as per columnname
- Select column1, groupfunction(column2) from tablename group by columnname : Returns group function columnname wise.

Example:

```
SELECT  Country, COUNT(CustomerID)
FROM Customers
GROUP BY Country;
```

DDL statements

- create: CREATE TABLE table_name (column1 datatype, column2 datatype, column3 datatype, …);
- alter: ALTER TABLE *table_name* ADD *column_name datatype*;

  ALTER TABLE *table_name* DROP COLUMN *column_name*;

  ALTER TABLE *table_name* MODIFY COLUMN *column_name datatype*;
- drop: DROP TABLE *table_name*;
- truncate: TRUNCATE TABLE *table_name*;

DML statements

- insert: INSERT INTO *table_name* (*column1*, *column2*, *column3*, …)
  VALUES (*value1*, *value2*, *value3*, …); AND
  INSERT INTO *table_name* VALUES (*value1*, *value2*, *value3*, …);
- update: UPDATE *table_name* SET *column1* = *value1*, *column2* = *value2*, … WHERE *condition*;
- delete: DELETE FROM *table_name* WHERE *condition*;

sqldf Package

- sqldf is a R package for running SQL statements on R data frames.
- The user simply specifies an SQL statement in R using data frame names in place of table names and the data frames are automatically loaded.
- The result is read back into R.
- Example:

```
install.packages("sqldf")
library(sqldf)
data() #display all data sets of R environment
df<-sqldf("Select * from tablename")
View(df)
```

RMySQL Package

- RMySQL is an R package for running SQL statements with MySQL database using R program.
- RMySQL is a database interface and MySQL driver for R.
- Example 1:

```
install.packages("RMySQL")
library(RMySQL)
#establish Connection with MySQL
conn=dbConnect(MySQL(),user="root",password="admin",dbname="mysql",host="localhost")
dbListTables(conn) #display all tables of MySql database
query<-"Select * from emp" #select query to retrieve emp table
result<-dbSendQuery(conn,query)
fetch(result)
```

SQL queries with RMySQL Package

- Example 2:

```
install.packages("RMySQL")
library(RMySQL)
conn=dbConnect(MySQL(),user="root",password="admin",dbname="mysql",host="localhost") #establish Connection with MySQL
query1<-"update emp set salary=4000 where ename='Mahesh'"
dbSendQuery(conn,query1)
query2<-"select * from emp"
result<-dbSendQuery(conn,query2)
fetch(result)
```

SQL queries with RMySQL Package

- Example 3:

```
install.packages("RMySQL")
library(RMySQL)
conn=dbConnect(MySQL(),user="root",password="admin",dbname="mysql",host="localhost") #establish Connection with MySQL
query1<-"insert into emp(empid,ename,salary) values(3,'Neha',5000)"
dbSendQuery(conn,query1)
query2<-"select * from emp"
result<-dbSendQuery(conn,query2)
fetch(result)
```

SQL queries with RMySQL Package

- Example 4:

```
install.packages("RMySQL")
library(RMySQL)
conn=dbConnect(MySQL(),user="root",password="admin",dbname="mysql",host="localhost") #establish Connection with MySQL
query1<-"delete from emp where ename='Neha'"
dbSendQuery(conn,query1)
query2<-"select * from emp"
result<-dbSendQuery(conn,query2)
fetch(result)
```

Web Scraping

- Web scraping is a technique to **fetch data from websites.**
- While surfing on the web, many websites **don't allow** the user to save data.
- One way is to **manually** copy-paste the data, which both tedious and time-consuming.
- Web Scraping is the automatic process of **data extraction from websites.**
- R is considered as one of the programming languages for **Web Scraping** because of features like – a rich library, easy to use, dynamically typed, etc.
- The commonly used web scraping tools for R is **rvest**.
- Example: install.packages("rvest")

  library(rvest)

Example of Web Scraping in R

```
install.packages("rvest")
library(rvest)
# Reading the HTML code from the website
webpage =
read_html("https://www.geeksforgeeks.org//data-structures-in-r-programming")

# Using CSS selectors  to scrape the heading section
heading = html_node(webpage, 'title')

# Converting the heading data to text
text = html_text(heading)
print(text)
```

```
 # Using CSS selectors to scrape
# all the paragraph section
# Note that we use html_nodes() here
paragraph = html_nodes(webpage, 'p')

# Converting the heading data to text
pText = html_text(paragraph)

# Print the top 6 data
print(head(pText))
```

Thank you