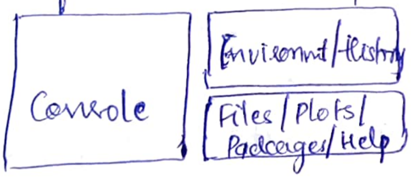
DS [unit-5]

* Introduction to R-studio:

- R studio is an IDE (Integrated Development Environment] for R. IDE is GUI.
- R studio is available as both open source and commercial software
- It is also available as both Desktop and server versions. and various platforms such as windows, Linux and Mac Os.
- It is an open source tool that provides Ide to use R language, and enterprise-ready professional software for DS teams to develop & work.
- R-studio can be downloaded from its official website-://rstudio.com/
- After installation process is over, R studio interface looks like:



- The console pane (left) is the place where R is waiting you to tell what to do & see results generated
- In top right - i) Environment - shows variables generated during the programming in workspace (temporary)

  ii) History - You'll see all the commands used till the start of the R-studio.

- In Right-bottom:

  i) Files - files & directories available by default in R-studio.

  ii) Plots - shows the plots generated during programming

  iii) Packages - helps to see what packages are already installed & install new.

  iv) Help - most important - to get help from R documentation on func in R

* Basics of R-programming:

- R is an open source programming language that is widely used as a statistical software and data analysis tool.
- Generally comes with a command-line interface.
- Used as a leading tool for machine learning, statistics & data analysis. objects, functions and packages can be easily created by R.
- Platform independent language, applied to all the operating system.

Programming features of R:

1. R packages → It has wide availability of libraries. R has CRAN (Comprehensive R Archive Network), which is a repository holding more than 10,000 packages.
2. Distributed Computing → a model where components of a software system are shared among multiple computers.

Advantages:

- most statistical analysis lang
- open source, can run anywhere
- suitable on any operating system
- suitable for Linux OS.
- In R, anyone can provide bug fixes, code enhancements.

Disadvantages :

- standards of some packages is less than perfect
- Can consume all available memory.
- nobody to complain, if something doesn't work.

- Slower than Python or Matlab programming

Applications :

- Data Science → statistical computing and design.
- Many quantitative analysts as its programming tool. → Data import, clean
- Data analysts and Research programmers use it.
- Tech giants like Google, facebook, bing, Twitter, Accenture, Wipro use R

* Math Variables and Strings :-

⇒ Variables : these are containers for storing data values.

- R does not have command declaring a variable, its created the moment you first assign it.
- To assign a variable use the '←' sign.
- To print the variable just write print(var_name). or just the variable name

Example :-

```
a ← "John"
b ← 40

print(a)
print(b)
```

Or

```
a ← "John
b ← 40

a     #print a
b     #print b.
```

Ex:-

```
a ← "John"
class(a) # prints string
```

- To see the datatype of variable use the class() function

⇒ Strings :- Are a bunch of character variables.

- It is 1-D array of characters.
- Can contain numbers, spaces, and special characters.
- Empty string represented using ".
- R strings are always stored as double quoted values.
- A double quoted string can contain single quotes within it.
- strings can be assigned by using the variables

Ex:- valid string → "hello", "hi"
Invalid " → "hello', "hi'bye", etc.

Ex :-

```
e ← "mix"
print(e)   # mix is printed.
```

* Vectors and factors:

⇒ **Vectors**: these are the same as arrays which are used to hold multiple data values of same type.

- But indexing in R starts from 1 and not 0.

Ex:- Vectors:-

Index →	1	2	3	4	5
Values →	10	20	5	7	30

∘ Types of R vectors:

1) Numeric → Int, float, etc
2) Character → alphanumeric, special chars
3) Logical → Boolean values (TRUE, FALSE)

Ex:-

1) Vector of strings:

```
fruits <- c("banana", "apple")
print (fruits)
```

2) Numerical values:

```
numbers <- c(1, 2, 3, 4)
print (numbers)
```

→ To combine list of items to a vector, use the c() functions and separate by comma.

⇒ **Factors**:-

- these are the data structures that are implemented to categorize the data or represent the categorical data and store it on levels.
- they can be stored as integers with a corresponding label to every unique integer.
- the R factor accepts only a restricted number of distinct values.
- the possible cases are known beforehand and are predefined.
- these distinct values are known as levels.
- To create a factor in R, use the factor() command.
- First create a vector and then convert it to factor using function factor()

Ex:-

```
#creating a vector
x <- c("Female", "Male", "Transgender", "Male")
print (x)

# converting vector to factor.

gender <- factor(x)
print (gender).
```

* Vector Operations :-

1) Creating a Vector - using the function c() to combine different elements together.

Ex:-
```
a ← c(1,2,3,4,5)
print(a).
```

2) Accessing vector elements - using the [] subscript operator.

Ex:-
```
a ← c(20,30,10,5,9)
print(a[2])   #prints the value 30.
```

3) Modifying a vector - modified using the operator,

Ex:-
```
a ← c(1,2,3,4,5)
a[2] ← 11
print(a)   # prints - 1,11,3,4,5.
```

4) Deleting a vector - Can be reassigned as NULL.

Ex:-
```
a ← c(1,2,3,4)
a ← NULL
print(a)   # prints NULL.
```

5) Arithmetic Operations - we can perform arithmetic operations on 2 vectors. Here the length of both vectors should be same.

Ex:-
```
a ← c(1,2,3,4)
b ← c(2,3,4,5)

#addition
c ← a+b
print(c)   #prints - 3,5,7,9.

# subtraction
d ← b-a
print(d)  #prints - 1,1,1,1
#multiplication
e ← a*b
print(e)   # prints - 2,6,12,20
f ← a/b  #division.
```

*** Reading CSV file :-** CSV files are text files wherein the values of each row are separated by a comma or tab.

Ex:- sample.csv

```
id ,   name ,   department,   projects
1,     A,       IT,           4
2,     B,       Tech,         5
3,     C,       IT,           7
4,     D,       HR,           2
```

⇒ Reading a CSV file :

- Can be read as a data frame in R using read.csv() funcⁿ.
- the csv should be present in current working directory.
- the csv can also be read from a URL using read.csv().

Example:-

```
csv_data ← read.csv (file = 'sample.csv')
print (csv_data)
```

output:- (same as above)

*** Reading text files (.txt) in R:**

- we can read txt file using the read.table function.
- Importing txt into R rarely needs arguments than specified.
- the basic syntax to almost all txt data files –

syntax:

```
read.table (file,
            header = FALSE,  # display header is True / False.
            sep = " ",       # separate columns of file
            dec = ".")       # separate decimals of numbers
```

Ex:- Consider you have a txt file called my-file.txt and have it in your R working directory.
You can read it by the code given –

```
data ← read.table (file = "my-file.txt", header = TRUE)
head (data)   # display the header (ie, column names)
```

- The output of a txt file with read.table function will be a class of "dataframe".

* Writing text files :-

the function write.table ( ) can be used to export a dataframe or matrix to a file.

Ex:- write.table (x , file , append = FALSE , sep = " " , dec = "." ,
                    row.names = TRUE , col.names = TRUE)

- It is also possible to write csv files using write.csv ( ) funcⁿ.

Ex :- write.csv (my-data , file = "my-data.csv")

=) Writing data to a file -

Below is a code which exports the built-in R mtcars set to a tab-separated (sep = "\t") file called mtcars.txt -

```
#loading mtcars data
data ("mtcars")
#writing mtcars data
write.table (mtcars , file = "mtcars.txt" , sep = "\t"} ,
             row.names = TRUE , col.names = NULL)
```

* String operations :-

1) Concatenation of strings - using the paste ( ) function for larger string.

```
Ex:- str ← paste ("Learn" , "code")
     print (str)        # prints "Learn code".
```

2) Calculate length of string - using length ( ) function.

```
Ex :- print ( length (c ("Learn", "code" ))      # output = 2.
```

3) Case conversion -

i) upper case ⟶ use the function toupper ( )

ii) Lower case ⟶ use tolower ( )

```
Ex :- str ← c (" Hi" , "Morning" )
      print (toupper (str))   # output = "HI" , "MORNING".
      print (tolower (str))   # output = "hi" , "morning".
```

4) character replacement ⟶ using chartr (oldchar , newchar , - - -)

```
Ex :- chartr ("a" , "A" , "An honest man gave me")
   output ⟶ An honest mAn gAve me.
```

5) Splitting the string: using the " " - the default separator.

Ex:- strsplit ("Learn Code!" , " ")

output → "Learn" "Code" "!" .

* Regular Expressions in R:-

- sequence of characters used to search the text.
- Also can search file in a directory using command line.
- We can replace specific text.
- It is a sequence of characters (or one char) that describes a certain pattern found in a text

Some of the character escape sequences in a RE are:-

\b	→	A word boundary
\B	→	A non-word boundary
\n	→	A new line
\t	→	A tab
\v	→	A vertical tab

Some of the quantifiers in RE are:-

*	→	0 or more
+	→	atleast 1
?	→	atmost 1
{n}	→	exactly n
{n,}	→	atleast n
{n,m}	→	atleast n or atmost m.

* Date Format:- the funcn are used to format and convert the dates from one form to another.

- R provides a format function that accepts the date objects & also format parameter that allows us to specify date we needed.

⇒

specifier	Description	specifier	Description
% a	Abbreviated Weekday	%y	year without Century
% A	full Weekday	%Y	year with Century
% b	Abbreviated Month	% d	day of month (0-31)
% B	Full month	% m	month of year (1-12)
% C	Century	% D	date in %m/%d/%y form

* Packages and Libraries :-

- these are a set of R functions, compiled code and sample data.
- these are stored under a directory called "library" within the R environment.
- By default R installs a group of packages during installation.
- Once we start the R console, the default packages are available by default, other packages installed need to load explicitly.
- there are multiple ways to install packages, easiest way is to install from CRAN.

  use the Command : install.packages ("package name")

• Package is a collection of functions bundled together. It is an appropriate way to organise our own work & share others

Library :- It is a command used to load a package. It refers to the place where the package is contained, usually folder in our computer.

- there are many R libraries that contain a host of functions, tools, and methods to manage and analyze data.
- Each of these libraries has a particular focus with some library managing image, textual data, data manipulation, visualization, machine learning, etc
- Examples are;

1) dplyr → data manipulation. $\xrightarrow{\text{Installation}}$ install.packages ("dplyr")
2) ggplot2 → data visualisation (bar charts, pie charts, histograms, API)
3) Shiny → build interactive web applications in R. (without special skills)
4) mlr3 → machine learning, implement ML models like regression, clustering, nearest neighbours, naive bayes, decision trees...
5) lubridate → Focused on making date-time easy to handle. Easy management of date-time data with simple func^n such as second(), minute(), hour(), day()...

* CRAN :- Comprehensive R Archive Network.

- Main repository for R packages.
- It is the easiest way to install packages on CRAN by using command: install.packages( )
- Putting your package on CRAN gave it some exposure
- It is the centralised radio access network.
- Used to also create cloud computing architecture (5G networks)
- It is made up of Base Band Unit [BBU], Remote Radio Unit (RRU) and transport network.
- BBU functions as a cloud or data center.
- RRU connects the wireless devices.

* Downloading and Installing packages from CRAN.

- You just need the name of the package and use the command install.packages ("package").
- CRAN has over 10,000 packages available to choose from.
- the type of package we need to install should be mentioned inside common braces.

  Ex:- dplyr.

  then → install.packages ("dplyr").
- After running this, you will receive some messages on the screen. this will depend on what operating system you are, the dependencies, and if the package was installed successfully.
- If no error messages, your package was successfully installed with all its dependencies.
- You can now give a different folder location using "lib".
- If we execute the command outside R studio (in GUI | ~~comm~~ terminal), we need to choose the CRAN mirror.
- We get some messages of the installation itself, the source code, help, some tests and finally a message that package was successfully installed. Messages can differ on different platforms.
- To install more than one package at a time, just use the command install.packages() with arguments. and vector.

  Ex:- install.packages (c("dplyr", "ggplot2"))