R-Programming

(case sensitive)

* multiplication	/ division
^ or ** exponential	log(x, base) logarithm
π → pi	%% modulus

R Studio

(shows suggestions in commands unlike R)

- assigning : =, <-, 5 -> x
- erase value : rm(x)
- save file : File > Save or ctrl + S (as file.R)
- run : ctrl + enter

R DATA TYPES

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

- Logical (True, False)
- Numeric (12.3, 5, 999)
- Integer (2L, 34L, 0L)
- Complex (3+2i)
- Character ('a', "good", "TRUE", '23.4')
- Raw (charToRaw ("Hello"))
  ↳ "Hello" is stored as 48 65 6c 6c 6f

1. VECTORS

- creating: <obj> <- c('_', "_")
- the very basic data types are the R-objects called VECTORS which holds elements of diffrent classes (logical, numeric..)

2. LISTS

- an R-obj which can contain many diff^{n} types of elements inside like vectors, function or another list inside it.
- creating: list1 <- list (<values>)

3. MATRICES

- a 2D rectangular data set.
- can be created using a vector input to the matrix function.
- creating:

```
M = matrix ( c("a", 'a', 'b', 'c', 'b', 'a'), nrow=2,
             ncol=3, byrow = TRUE)
```

```
     [,1] [,2] [,3]
[1,] "a"  "a"  "b"     ← output
[2,] "c"  "b"  "a"
```

4. ARRAYS

- can be any number of dimensions.
- creates the reqd. no. of dimension.
- creating : a<- array(c('a','b'), dim = c(3,3,2))

output →

, , 1

	[,1]	[,2]	[,3]
[1,]	'a'	'b'	'a'
[2,]	'b'	'a'	'b'
[3,]	'a'	'b'	'a'

, , 2

	[,1]	[,2]	[,3]
[1,]	'b'	'a'	'b'
[2,]	'a'	'b'	'a'
[3,]	'b'	'a'	'b'

5. FACTORS

- R-objects created using a vector.
- stores the vector along with distinct values of elements in vector as labels.

↓

always character irresp. of number or boolean.

- creating : factor(), nlevels funct^n
  - gives count of levels.

```
apple <- c('green', 'red', 'yellow')
fac_apple <- factor(apple)
print(fac_apple)
```

Output → [1] green red yellow
Levels : green red yellow

```
print(nlevels(fac_apple))
```

Output → [1] 3

6. DATA FRAMES

- tabular data objects
- each column can contain diff^n modes of data.
- it IS A LIST OF VECTORS OF EQUAL LENGTH
- creating : data.frame()

```
BMI <- data.frame ( Person = c ('Jake', 'Julia'),
                    height = c(174, 165),
                    weight = c(60, 55),
                    age = c(23, 20))
print (BMI)
```

Output →

	Person	height	weight	age
1.	Jake	174	60	23
2.	Julia	165	55	20

R - Variables

- consists of letters, numbers, dot, underscore
- starts with letter or dot
  - ↳ should not be followed by a no.
- stores values
- assigned: <-, ->, =
- printing: print(), cat()
- variables currently available: ls()
  ls(pattern = 'abc')

Variables starting with dot (.) are hidden, they can be listed using "all.names = TRUE" in ls() funcn

- deleting var: rm()
- delete all variables: rm(list = ls())

R - Operators

- symbol that tells the compiler to perform specific mathematical or logical manipulatn.

- Arithmetic (^ or **, +, -, *, /, %%, %/%)
- Relational (>, <, ==, <=, >=, !=)
- Logical (& (and), | (OR), ! (not))
- Assignment (<-, =, <<-, ->, ->>)
- Miscellaneous (: , %in% , %*%)

: creates sequence or series

%in% identify if element belongs to a vector

%*% multiply a matrix

the logical operator && and || considers only the first element of the vectors and give a vector of single element as output.

R- Decision making

- if statement : Boolean exp. followed by 1 or more statements.
- if... else : if statement can be followed by an optn else statmt
- switch statement : allows a var to be tested for equality against a list of values

```
if (<expresn>) {
    <statement>
} else if (<expn>) {
} else {
    <statement>
} else {
    <statement>
}
```

```
if (<expression>) {
    <statement>
} else {
    <statement>
}
```

switch (expression, case 1, case 2 ...)

Rules to switch statement:

- if value of exp. is not a character string, it is coerced to integer.
- any no. of case statements
- if value of int is b/w 1 & nargs()-1

UNIT 1:

An overview of R language

Basic fundamentals of R

- Open-source language primarily used for statistical computing, data analysis, and graphics.
- Interpreted language: R is not compiled; instead commands are interpreted line-by-line.
- Case-sensitive: R distinguishes b/w uppercase and lowercase letters.
- Vectorized: Operations are often applied to entire vectors, making the language efficient for numerical tasks.

Installation and use of Software

- Installing R
- using R

R console : directly execute commands

RStudio : Integrated Development Environment (IDE), offers a user-friendly interface, including an editor, debugging tools, and visualizatn windows.

Terminal : one can run R directly from a terminal by typing R.

Data editing in R

- edit() or fix() functn to manually edit data in spread sheet like interface.
- data.frame(), read.csv() and read.table() are commonly used to load and manipulate datasets programmatically

R as a calculator

- Basic arithmetic : +, -, *, /, ^, %%(modulus), %/%(integer division)
- Mathematical Functions :

sqrt(), log(), log10(), sin(pi/2)

↳ natural log ↳ radian

Functions and Assignments

- Assignments : <-, ->, =
- Functions : function()

→ functions allow you to encapsulate reusable code.

Getting R and Running R

- Download :
- Running R : R can be run in the console, through an IDE like RStudio.
- RStudio interface : it is popular for script writing, plotting, and accessing various panes like the environment, console etc

R packages

- R packages are collections of functions, data, and compiled code that extend the base R functionality
- Scope of R, such as for statistical analysis, data manipulation, visualization, machine learning, & even web development.

components of an R packages

R functions: The core functionality of the package. These functions provide users with tools to perform specific tasks.

Data sets: may include data sets for users to practice and analyze (eg. include mtcars and iris datasets)

Documentation: help files for each function or dataset, which explain their usage, arguments, & examples.

Vignettes: long-form documentation (articles) that provide in-depth examples of how to use the package

Compiled code: some packages include C, C++, or Fortran code for performance-intensive tasks.

Types of R packages

- CRAN Packages: available through the comprehensive R Archive Network.
- Bioconductor Packages: designed for bioinformatics and genomic analysis
- Github Packages: many developers host experimental or cutting-edge packages on GitHub.

Installing and loading

install: install.packages("ggplot2") ← name of package

load: library(ggplot2)

- install from Github:

```
install.packages("devtools")
devtools::install_github("username/packages")
```

- Biconductor packages

```
if (!requireNamespace("BiocManager",
    quietly = TRUE))
    install.packages("BiocManager")
BiocManager::install("GenomicFeatures")
```

Popular R Packages

- DATA MANIPULATION
  - → dplyr : fast, intuitive data manipⁿ
    filter(), mutate(), select()
  - → tidyr : reshaping & tidying data
    spread(), gather()
  - → data.table : data manipⁿ & aggregatⁿ
    for large datasets.
- DATA VISUALIZATION
  - → ggplot2 : creating static graphics
    using the grammar of graphics

→ shiny : interactive web applicat^n.

→ plotly : interactive web-based graphs & plots.

- STATISTICAL MODELING

→ lme4 : for fitting linear and generalized linear mixed models.

→ glmnet : for fitting generalized linear models via penalized max^ likelihd

- MACHINE LEARNING

→ caret : package that provides a uniform interface to various machine learning algorithms.

→ randomForest : for building random forest models for classification and regression.

- BIOINFORMATICS

→ bioconductor : a suite of tools for bioinformatics, including packages like edgeR and DESeq2 for diff^n expression analysis of RNA-seq data.

Managing Installed Packages

list installed packages : installed.packages()

update : update.packages()

remove : remove.packages(<name>)

Package Documentation

accessing help : help(package = "<name>")

function help : ?ggplot

vignettes : vignette(package = "ggplot2")

vignette("ggplot2-specs")

(extra)

Creating your own package

1: Package Directory Structure

my package/

- DESCRIPTION
- NAMESPACE
- R/
- data/
- man/

2. Description File : contains metadata about the package, including name, version, author, dependencies, and description.

3. Namespace File : declared which functn are available to users and which functn are internal.

4. R directory : contains R script files that define the functn of your package.

5. Documentatn : use roxygen2 to automatically generate documentatn for your functn based on comments in the code.

we can build packages using devtools :

```
library (devtools)
devtools :: create ("mypackage")
devtools :: document()
devtools :: install()
```

Package Dependencies

Depends : R (>= 3.5.0)

Imports : dplyr , ggplot2

Suggests : testthat

- Depends : specifies the min version of R needed and packages that must be loaded when your package is loaded.
- Imports : packages that are necessary for your package's internal workings but are not loaded by the user.
- Suggests : Optional packages used for testing , vignettes , or enhanced functionality

Benefits of using R packages

- Modularity : modulise code into reusable component
- Community-driven : many packages ava
- Extensibility : easily extended by loading
- Open-source : one can inspect , modify or contribute them .

Expressions

- An expression is a combinat^{n} of values, variables, operators, and functn calls.
- every expression returns a value.

Objects

- In R, everything is an object (eg variables, data structures, functions)
- Objects are created via assignment

x <- 5 , y = c(3, 2, 1)

→ vectors → matrices → data frames → lists

Symbols

- symbol is a name that refers to an object
- ls() → list all symbols in environment

Functions

- first-class objects my_function (<arg>)

Special values

NA: not available

NaN: not a number

Inf, -Inf: ∞, $-\infty$

↳ resulting by zero division

UNIT 2:

constants

- fixed values that do not change during the execution of a program.
  1. Numeric constants (5, 10.5 ...)
  2. Character constant ("Hello")
  3. Logical (TRUE, FALSE)
  4. Special constants (NA, Inf, -Inf, NaN)

Numeric vectors

- Vectors are the simplest type of data structure in R, used to hold a seqn of data elements of the same type
  - Numeric vectors contain numbers.
  - syntax: c(<nos>)
  - operatn is performed element-wise

Character vectors

- char vectors contains char string
- syntax : c(<strings>)
- string manipulation using funct^n like paste(), substr()

Operations in R

- Arithmetic opr^n : +, −, *, /, ^, %%, %/%, ! (modulus, integer division)
- Relational : >, <, >=, <=, ==, !=
- Logical operat^n : &, |, ! (and, or, not)
- Assignment : <−, →, =
- Miscellaneous : %in%, : (sequence)
  - ↳ (membership)

R syntax

- case sensitive
- assignment : x <- 10, 10 -> x, x = 10
- comments : #
- funct^n call : function_name (arguments)
- semicolons (;) : end statement
  - ↳ optional

Data structures in R

- MATRICES
  - 2D arrays (rows and columns)
  - syntax: matrix(data, nrow, ncol)
  - operations are performed element-wise or matrix-wise.
- ARRAYS
  - similar to matrices but has more than 2 dimensions
  - syntax: array(data, dim = c(1, 2))
- FACTORS
  - used to handle categorical data
  - syntax: factor(c("__", "__"))
- DATA FRAMES
  - structure for tabular data
  - syntax: data.frame(col1, col2)

Attributes

- Attributes are metadata about objects, such as names, dimensn, class
- accessed: attributes()
  names()
  dim()

example: names(x) = c("A", "B", "C")

Symbols and Environments

- SYMBOLS: objects in R are stored using symbol (names), which point to data in the environment.
- ENVIRONMENT: the collection of symbol-value pairs. Every R session has an environment (global environment by default)
  → functions can create their own environments.

Functions

- block of codes that perform a specific task.
- syntax: function_name = function(argument){
  <block code>
  return()
  }
- functn can return() a value using return() or automatically return the last evaluated expressions

Loading, Saving, & Editing Data

- LOADING: from various file formats
  read.csv(), read.table(), readRDS()
- SAVING DATA:
  write.csv(), save(), saveRDS()
- EDITING: basic editing can be done using indexing (data[rows, colounns]), or more complex operatn can be done by dplyr or data.tables packages.

Combining Datasets

- ROW-WISE : rbind()

→ combining datasets with same columns

- COLOUMN-WISE : cbind()

→ combining datasets with same rows

- MERGING : merge()

→ combine data frames by matching columns

Transformations

- transformations are operations that change the structure or content of the data

- common transformations include log transformatn, scaling normalizatn and applying functn like log(), scale()

- Vectorized operations allow you to apply transformatn across vectors or matrices.

Binning data

- binning refers to dividing data into intervals (bins).
- use cut() to divide continuous data into bins

binned data = cut(x, breaks = 3)

- useful for discretizing continuous data for analysis.

syntax: cut(x, breaks, labels = NULL,
include.lowest = FALSE,
right = TRUE, dig.lab = 3)

x = numeric vector to be binned

breaks = no. of intervals or a vector specifying the exact breakpoints.

labels = if not provided, default will be interval ranges value

include.lowest: if TRUE, includes the lowest

right: if TRUE, include the right-most edge

dig.lab: no. of digits used when formatting bin labels.

DATA FRAME

2-D table-like structure in R

→ storing and manipulating tabular data.

Columns, Rows } indexing is supported

```
df = data.frame (
        Employee = c("Jake", "Alex", "Ryan"),
        Age = c(23, 22, 25),
        Salary = c(50000, 60000, 70000)
)
print(df)
```

	Employee	Age	Salary
1	Jake	23	50000
2	Alex	22	60000
3	Ryan	25	70000

df $ Employee (accessing column Employee)

df [, 1] first column (Employee)

df [1,] first row (Jake, 23, 50000)

df [2, 3] 2nd row 3rd column (Employee, Salary)
2nd row 3rd col.

df $ Department = c("HR", "IT", "Finance")
(new column)

new_row = data.frame(
Employee = "Alice", Age = 26,
Salary = 80000, Department = "IT"
)

df = rbind (df, new_row) (new row)

index
df $ Age[2] = 25 (updating value)
column new value

```
subset_df = subset(df, Department = "IT")
print(subset_df)
```

	Employee	Age	Salary
1	Alex	25	60000
2	Alice	26	80000

```
selected_columns = subset(df[, c("Employee", "Age")])
print(selected_columns)
```

	Employee	Age
1	Jake	23
2	Alex	25
3	Ryan	25
4	Alice	26

```
selected_element = df[2,3]
print(selected_element)
```

60000

```
subset_rows = df[1:3, "Employee", drop=FALSE)
print(subset_rows)
```

	Employee
1	Jake
2	Alex
3	Ryan

```
exclude_columns = df[, -3]
print(excluded_columns)
```

	Employee	Age
1	Jake	23
2	Alex	25
3	Ryan	25
4	Alice	26

```
exclude_rows = df[-1:-3]
print(excluded rows)
```

	Employee	Age
2	Alex	25
3	Ryan	25
4	Alice	26

filtered_df = subset(df, Age <= 26 & Salary >= 55000)

print(filtered_df)

	Employee	Age	Salary
2	Alex	25	60000
3	Ryan	25	70000

indices = which(df$Salary > 65000)

print(indices)

3 4

returns indices of rows which meet the reqr. conditions.

str(df) — check structure of data frame.

'data.frame' : 4 obs. of 3 variables

$ Employee : chr "Jake" "Alex" "Ryan" "Alice"

$ Age : num 23 25 25 26

$ Salary : num 50000 60000 70000 80000

summary(df) → get a summary of each column

Employee	Age	Salary
length : 4	Min : —	Min : —
Class : character	1st Qu. : —	1st Qu. : —
Mode : character	Median : —	Median : —
	Mean : —	Mean : —
	3rd Qu. : —	3rd Qu. : —
	Max. : —	Max. : —

nrow(df) → 4 (no. of rows)

ncol(df) → 3 (no. of columns)

dim(df) → 4 3 (dimension)

colnames(df) (gets column names)

"Employee" "Age" "Salary"

colnames(df) = c("EmployeeName",
"EmployeeAge",
"Employee Salary")

(to change name of columns)

head (df, 2)

tail (df, 2)

	Employee	Age	Salary
1	Jake	23	50000
2	Alex	25	60000

} first 2 rows

	Employee	Age	Salary
3	Ryan	25	70000
4	Alice	26	80000

} last 2 rows

apply (df [, 2:3], 2, mean) → columnwise mean

↳ columns 2 and 3

Age	Salary
49.5	65000

sorted_df = df [order (-df $ Salary),]

print (sorted_df)

↳ sorts in descending

	Employee	Age	Salary
	Alice	26	80000
	Ryan	25	70000
	Alex	25	60000
	Jake	23	50000

```
df2 = data.frame (Employee = c ("Alex", "Alic"),
                  Bonus = c (2000, 1000))
merged_df = merge (df, df2, by = "Employee")
print (merged_df)
```

	Employee	Age	Salary	Bonus
1	Alex	25	50000	2000
2	Alice	26	80000	10000

```
df $ Bonus = c(NA, 3000, NA, 1000)
```

is.na (df) → (checks if null values present)

	Employee	Age	Salary	Bonus
1	FALSE	FALSE	FALSE	TRUE
2				FALSE
3				TRUE
4				FALSE

```
df $ Bonus [is.na (df $ Bonus)] = 0
print (df)
```

↓

(replaces NA values with 0)

write.csv (df, "employee_data.csv"), row.names=FALSE

↳ writes df into file

imported_df = read.csv ("employee_data.csv")

↳ reads from file

new_row = data.frame (<values>)

df = rbind (df, new_row)

↳ adds new row

~~new_column = data.~~

df $ <colname> = c (<new values)

print (df) ↳ adds new column & values

UNIT 3:

Data Manipulation and Analysis in R

Subsets

- involves extracting parts of data based on certain conditions or indices.
- Techniques
  - → USING INDICES: `data[1:5, ]`
  - → LOGICAL CONDITN: `data[`
    `data[data$column > 10, ]`
  - → FUNCTIONS: subset() used for complex subsetting.

Summarising Functions

- summary(): provides statistical summary (min, max, mean, median
- mean(), median(), sd()
- table(): counts occurances of each unique value in a column.
- aggregate(): summarises by group.

Data cleaning

- ensures data quality and involves
  - → Handling missing values : na.omit() , is.na()
  - → Outliner treatment : identifying and managing outliers based on domain knowledge
  - → Data transformation : using mutate() to create or modify cols
  - → Removing duplicates : distinct() from dplyr or unique()

Analysing Data

- EDA : exploratory data analysis : summary() , plot()
- grouped analysis : dplyr functn group_by() and summarize()

Probability Distribution

- NORMAL DISTRIBUTION: dnorm(), pnorm(), qnorm(), rnorm()
- BINOMIAL DISTRIBUTION: dbinom(), pbinom(), qbinom(), rbinom()
- POISSON DISTRIBUTION: dpois(), ppois(), qpois(), rpois()

Continuous and Discrete Data

- CONTINUOUS DATA: data that can take any value within a range.
- DISCRETE DATA: data can only take specific values.

T-Test Design

- One sample t-test: t.test(data, mu = value)
- Two-sample t-test: t.test(data1, data2)
- Paired t-test: t.test(data1, data2, paired = TRUE)

ANOVA Test Design

- ANOVA (Analysis of Variance) is used to compare means across multiple groups.
- One-way ANOVA

```
anova_result = aov(response ~
                   factor, data = dataset)
summary(anova_result)
```

- Two-way ANOVA

```
anova_result = aov(response ~
                   factor1 * factor2, data=dataset)
summary(anova_result)
```

Introduction to Regression

- Regression is used to model the relationship b/w variables
- SIMPLE REGRESSION LINEAR: Models the relatn b/w one independent & one dependent var.
- MULTIPLE LINEAR REGRESSION: includes multiple independent variables.

Linear Model

- lm() function is used to fit linear models

```
model = lm(y~x, data = dataset)
summary(model)
```

Smoothing

- Smoothing reduces noise in data for better trend visualization.
- METHODS
  - → MOVING AVERAGE : commonly used for time series data.
  - → LOESS (LOCALLY WEIGHTED SCATTERPLOT SMOOTHING : available through geom_smooth() in ggplot2.

UNIT 4 :

Graphics and Plots in R

Scatter Plots

- used to display relationship b/w 2 continuous variable

```
plot (x, y)
```

- ggplot2 :

```
ggplot ( data , aes ( x = var1 , y = var2 )
    + geom_point ( ).
```

Bar charts

- used to display categorical data.

```
barplot (height , names.arg = labels)
```

- ggplot2 :

```
ggplot ( data , aes ( x = category ,
    y = value )) + geom_bar (start =
    "identity"
```

Pie Charts

- useful for showing proportions.

pie (values, labels = categories)

3D data

- 3D plots can be created with packages like - plot 3D, scatterplot 3d, and rgl.

library (plot 3D)

scatter (x, y, z)

Plotting distribution

- HISTOGRAM: shows distn of ~~continuous data~~ continuous data.

hist (data)

- DENSITY PLOT: smooth distribution of visualization.

plot (density (data))

Customizing Charts

- customize charts with titles, axis labels, colors, and themes

```
plot(x, y, main = "Title", xlab = "xlab",
     ylab = "ylab", col = "blue")
```

Basic Graphic Functions

- plot() : diff^{n} plot types
- hist() : histograms
- barplot() : bar charts
- pie() : pie charts

Common Arguments for chart Functions.

- main : title of the chart
- xlab | ylab : labels for x and y axis
- col : color of plot elements
- xlim | ylim : limits for x & y axes.