Coding with R

```
print("this is kawsar_sayem")
```

```
?print()
```

This documentation is for beginners to learn R in a simplified manner. In this document I try to merge some basic knowledge about R language.

Table of content:

Installing, updating, loading of packages & data:

For working in R studio we have to make an environment. For that we have to install needed packages. Also, we need data. We can import data or use data that are given by different packages. Remember functions are case sensitive in R.

```
install.packages("tidyverse")      # will install tidyverse package
library("tidyverse")               # will load tidyverse package
data("ToothGrowth")                # load data table
data()                             # this will show all data set for practicing
tidyverse_update()                 # update particular package
update.packages()                  # update all installed packages
browseVignettes("ggplot2")         # this function is used to know details about a package
```

8 core packages of tidyverse & their use of tidyverse

As tidyverse is a collection of packages. That's why we need to know about different packages and their uses.

- ggplot2 # used in data viz.
- tidyr # used in data cleaning
- readr # used in importing data, eg: csv/tsv/delim/table/log
  - readr_example() # will show sample data
  - readxl_example() # will show the spreadsheet sample files.
  - read_csv(readr_example("mtcars.csv")) # will open the "mtcars.csv" file
  - read_excel(readxl_example("type-me.xlsx"), sheet = "numeric_coercion")

- **dplyr** **# used in data manipulation**
- **tibble** **# works with data frame**
- **purrr** **# works with functions & vectors**
- **stringr** **# works with function for string variables**
- **forcats** **# solve problems with factors**

Other non tidyverse packages

- **here** **# referencing file easily**
- **skimr** **# data cleaning & summarizing data**
- **janitor** **# data cleaning**

Data structure & operator in R

We can create data on our own and can store as a variable/table. There some methods of doing that.

1. **Vectors (c)**
   - **Atomic vector** **# used to input one type of data**
     vec_1 <- c(1, 2, 3, 4)
     names(vec_1) <- c("sayem", "kawsar", "ahmed")
   - **List vector** **# multiple types**
     list(1.2, 1L, TRUE, "S") -> say_ka

   ### All about data types
   typeof(vec_1) **# find out the types of data vec_1 hold**
   length(vec_1) **# count the vector**
   is.integer(vec_1) **# will say whether vec_1 is integer or not**
2. **Data frame** **# creates table**
   data.frame(x = c(1, 2, 3) , y = c("a", "b", "c"))
   data.freame(vec_1, names(vec_1))
3. **Matrix**
   matrix(c(3:8), nrow = 3, ncol = 3)
4. **Arrays**

Operators in R

1. **Assignment operator (<-)**
   Use to assign name of variables, vectors, tables.
   number <- c(1, 2, 3, 4, 5)
2. **Mathematical operator (+, -, *, /)**
   Use to perform mathematical calculations.

   x <- 2
   y <- 5

   x + y
   x - y

```
x * y
y %% x        # remainder sign / modulus operator
y / x
y ^ x
```

3. **Logical operator (AND = &/&&, OR = |/||, NOT = !)**
4. **Conditional operator (if, else if, else)**
5. **Relational operator (<, >, ==, !=, <=, >=)**
6. **Pipe operator (%>%)**

Solution of nested coding (pipe operator)

Nested codes are difficult to understand. That's why we can use the pipe operator (%>%) as a solution of nested coding. This operator makes the coding easy to read and understand.

Nested & hard to read coding example:

```
filter_toothgrowt2 <- summarise(group_by(filter(ToothGrowth, dose == 0.5), supp), mean_len = mean(len, na.rm = T), .group = "drop")
```

With pipe operator:

```
filter_toothgrowt <- ToothGrowth %>%
  filter(dose == 0.5) %>%
  group_by(supp) %>%
  summarise(mean_len = mean(len, na.rm = T), .group = "drop")
```

Application of assignment & arithmetical operator in R

```
sales_q1 <- 201
sales_q2 <- 100
sales_half_year <- sales_q1 + sales_q2
sales_year <- sales_half_year * 2
```

Application of logical & relational operator (AND=&, OR=|, NOT=!)

```
x <- 20
x>3 & x<12        # both logical statement has to be TRUE to become the final output as TRUE
x>3 | x<12        # only 1 logical statement has to be TRUE
!x                # 0=False, NOT operator takes the opposite logical value.
```

Application of conditional statement (if, else, else if)

```
x <- 20
if ( x < 0) {
  print("x is a negative number")
} else if ( x == 0) {
  print("x is zero")
} else {
  print("x is a positive number")
}
```

Procedure of viewing data tables

For using different functions, we have to install & load the packages. Then we have to import and load data tables. We already discussed about this. Now, we can apply functions for performing different activities. First of all, we have to familiar with data tables. For that here are some functions that could help.

```
summary(diamonds)      # will give a summary of data tables, we can use this for selected columns
View(ToothGrowth)      # will show the whole data table [capital V]
head(diamonds)         # will show 6 rows with headers of table
glimpse(diamonds)      # will show total row & col. shows the table horizontally.
str(diamonds)          # Gives structure of the table with data types.
colnames(diamonds)     # Gives summary of columns
as_tibble(diamonds)    # works as View function & used for printing
```

Cleaning & organizing data

Before analyzing data, we have to clean nested data. Here are some functions that could help in this task.

```
mutate(diamonds, carat_2 = carat * 100)     # add new column in data frame with calculation.
rename(diamonds, caret_renamed = carat, cut_renamed = cut)    # use to change variable name
summarize(diamonds, mean_carrat = mean(carat), mean_price = mean(price))    # used in analysis
skim_without_charts(penguins)          # gives data summary
select(penguins, species)              # create subset from a large dataset
rename_with(penguins, tolower)         # make the header in lower/upper case
clean_names(penguins)                  # sure that col names are unique & consistent
pivot_longer() / pivot_wider()         # make the data frame into wider to longer or longer to wider
```

Example:

1. ```
   penguins %>%
     select(species)
   ```

2. ```
   penguins %>%
     rename(island_new = island) %>%
     rename_with(penguins, tolower)
   ```

3. ```
   penguins %>%
     group_by(species, island) %>%        # used to create subset
     drop_na() %>%                        # used to eliminate null
     summarise(mean_bl = mean(bill_length_mm), max_bl = max(bill_length_mm))
   ```

4. ```
   filter(penguins, species == "Adelie") %>%          # used for filtration
     drop_na()
   ```

5. ```
   arrange(penguins, -bill_length_mm)        # put minus(-) for descending
   ```
   ```

Visualizing data after completing analysis

Visualization of data helps other people to understand the insight of the analysis easily. People who are not familiar with data can easily understand it. Here is some basic information:

```
ggplot(data = diamonds,
       aes(x = carat, y = price, color = cut)) +          # have to add ( + ) for every layer of the viz
       geom_point() +
       facet_wrap(~cut)
```

 # aes function:

covers the aesthetic part of the visualization. Through this function we can customize color, shape, size of the data points of the viz. We can customize X & Y axis through this function

 # geom_point() function:

indicates the geometric picture of the viz. This means plot will show as scatter graph. Here we can use other functions for other geometric picture.

- **geom_smooth # use for trend line**
- **geom_point # scatter plot**
- **geom_bar # bar chart**
- **geom_jitter # scatter plot with random points**

 # Facet function:

When we need to divide the plot according to subset we can use facet function.

- **facet_wrap(~…….) # works with one variable**
- **facet_grid(……..~…….) # works with two variables**

 # Label/ Annotation

For adding label to the visualization we can use labs or annotation function. These have different usefulness.

- **labs # used for adding chart title, sub-title, caption, name of the axis**
- **annotation # used for adding size, color, adjusting angle, label (inside gridline)**

Example:

```
ggplot(activity_daily, aes(x = totalsteps, y = calories)) +
geom_point() +
geom_smooth() +
labs(title = "Daily Steps VS Calories Burned",
   x = "Total Steps",
   y = "Calories Burned")
```

Data manipulation

Here are some examples of data manipulation.

1. **Separate & merging of column**

```
id <- c(1:10)
first_name <- c("John", "Rob", "Rachel", "Christy", "Johnson", "Candace", "Carlson", "Pansy",
"Darius", "Claudia")
last_name <- c("Mendes", "Stewart", "Abrahamson", "Hickman", "Harper", "Miller", "Landy",
"Jordan", "Berry", "Garcia")
job_title <- c("Professional", "Programmer", "Management", "Clerical", "Developer", "Programmer",
"Management", "Clerical", "Developer", "Programmer")
employee2 <- data.frame(id, first_name, last_name, job_title)
unite(employee2, "name", first_name, last_name, sep = " ")
```

```
id <- c(1:10)
name <- c("John Mendes", "Rob Stewart", "Rachel Abrahamson", "Christy Hickman", "Johnson
Harper", "Candace Miller", "Carlson Landy", "Pansy Jordan", "Darius Berry", "Claudia Garcia")
job_title <- c("Professional", "Programmer", "Management", "Clerical", "Developer", "Programmer",
"Management", "Clerical", "Developer", "Programmer")
employee <- data.frame(id, name, job_title)
print(emolyee)
separate(employee, name, into = c("first_name", "last_name"), sep = " ")
```

2. **Calculating without sorting and grouping**

```
arrange(bookings_df, -lead_time)
max(bookings_df$lead_time)          # $ sign indicates specific column within a table
min(bookings_df$lead_time)
mean(bookings_df$lead_time)
```

3. **Same data different outcomes (statistical measures)**

- ```
  install.packages("Tmisc")
  library("Tmisc")
  data(quartet)
  View(quartet)
  ```

- ```
  quartet %>%
  group_by(set) %>%
  summarize(mean(x), sd(x), mean(y), sd(y), cor(x,y))
  ```

4. **Bias data detection (unbiased data set bring the outcome close to zero)**

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

actual_temp <- c(68, 56, 76, 32, 56)
predicted_temp <- c(65, 70, 72, 64,68)
bias(actual_temp, predicted_temp
```

Date, file, folder

To work with date & file/folder below functions could help.

```
today()                                  # output the date of today
now()                                    # date with time
ymd(today())                             # format as year – month – day
ymd(20121205)
ymd_hms(202310141116)
as_date(now())                           # as like today() function
dir.create("first_folder")               # create folder
file.create("first_file.text")           # create file. Have to cite file extension
file.copy("first_file.text" , "first_folder")   #  copy file from one folder to another
file.remove("first_file.text")           # remove file
```