7. Data Analysis with R Programming

R is a programming language used for statistical analysis, visualization, and other data analysis.

R makes it easier to present your data with a beautiful, artistic style. A few other advantages of R include its:

- Popularity: R is frequently used for data analysis
- Tools: R has a convenient library of ready-to-use tools for data cleaning and analysis
- Focus: R was created with statistics in mind; data analysts can conveniently use a rich library of statistical routines
- Adaptability: R adapts well for use in both machine learning and data analysis projects
- Availability: R is an open source programming language

RStudio - allows you to create and manage projects using R more efficiently.

- R is case sensitive.
- In the R q() command use to quit from the console.
- Programs like **RStudio**, an interactive development environment (IDE) for programming in R, use the R Console and other tools to make it easier to write and execute R code. In RStudio, the R Console is often referred to as the **console pane**

- In R console

message starts with R version and your version number, and ends with Type 'q()' to quit R. Above the message, you will find a menu with icons that represent the functions of the console and graphical user interface (known in the program as **RGui**).

- IDE(Integrated development environment) : Rstudio
  A software application that brings together all the tools you may want to use in a single place.
- Rstudio cloud and Rstudio Desktop this are two things
- Rstudio cloud run directly on browser
- And if you want to use Rstudio offline then Rstudio desktop will be the option.

- In the Google course we are using the Rstudio cloud.
- To change the path in R studio go to the file pane and then setting symbol set working dir.
- Or in the session menu and then select working dir.
- Installing and loading packages one by one in code chunks all at one time does not work.

R and RStudio are designed to handle large data sets, which spreadsheets might not be able to handle as well. RStudio also makes it easy to reproduce your work on different datasets. When you input your code, it's simple to just load a new dataset and run your scripts again. You can also create more detailed visualizations using RStudio.

Packages :

Packages are units of reproducible R code.Members of the R community create packages to keep track of the R functions that they write and reuse. Packages offer a helpful combination of code, reusable R functions, descriptive documentation, tests for checking your code, and sample data sets.

Packages include :

1. Reusable R functions
2. Documentation about the functions
3. Sample dataset
4. Tests for checking your code.

- Use installed.packages() to see which packages are installed.

When you run this function you will get the list of packages and there is a column priority in that if it is base then that package is already loaded but if priority is recommended then it is not loaded.

- If you need help then type help() in console and topic in parenthesis
  e.g: help("base")

- CRAN(Comprehensive R Archive Network) : an online archive with online packages , source code , manuals, and documentation.

- Packages can be found in repositories, which are collections of useful packages that are ready to install. You can find repositories on Bioconductor, R-Forge, rOpenSci, or GitHub, but the most commonly used repository is the Comprehensive R Archive Network or CRAN. CRAN stores code and documentation so that you can install packages into your own RStudio space.

- **Tidyverse**: the tidyverse is a collection of R packages specifically designed for working with data. It's a standard library for most data analysts, but you can also download the packages individually.

- **Quick list of useful R packages**: this is RStudio Support's list of useful packages with installation instructions and functionality descriptions.

- **CRAN Task Views**: this is an index of CRAN packages sorted by task. You can search for the type of task you need to perform and it will pull up a page with packages related to that task for you to explore.

- CRAN makes sure that any content of R open to the public meets the required quality standard.

Tidyverse package : a system of packages in R with common design philosophy for data manipulation , exploration and visualization.

To install tidyverse package type in console
Install.packages("tidyverse")

Then load the package tidyverse inside tidyverse type :
Library("tidyverse")

8 core tidyverse packages

1. Ggplot2
2. Tibble
3. Tidyr
4. Readr
5. Purrr
6. Dplyr

7. Stringr
8. Forcats

- A **vignette** is documentation that acts as a guide to an R package. A vignette shares details about the problem that the package is designed to solve and how the included functions can help you solve it. The **browseVignettes** function allows you to read through vignettes of a loaded package. V should be capital here.

To check out vignettes for one specific package, type **browseVignettes(“packagename”)**

E.g: - browseVignettes(“ggplot2”)

- **Ggplot2 :** create a variety of data Viz by applying different visual properties to the data variables in R.
- **Tidyr** : is a package used for data cleaning to make tidy data.
- **Readr** : used for importing data.
- **Dplyr** : offers a consistent set of functions that help you complete some common data manipulation tasks.
- **Tibble** : - works with data frames. Never change the data type of the input. Never change the names of your variables. Never create row names. Make printing easier.
- **Purrr** : - works with functions and vectors helping your code easier to write and more expressive.
- **Stringr** : - include function that makes it easier to work with strings.
- **Forcats** : - provide tools that solve common problems with factors.

- **Factor** : - store categorical data in R where the data values are limited and usually based on a finite group like country or year.

Functions:

- In console print("hello") it will print hello in console.
- If you want information about any function in R then write ? And then the function name.
  E.g: ?print()

- **Variable** : - to assign value to the variables in R there are different symbols, one is <-

E.g:-
first_var <- "dipali"
second_var <- 12.45

- **Vector :** - a group of data elements of same type stored in a sequence in R. to create vector add letter c before parenthesis
  One way to create a vector is by using the c() function (called the "combine" function)

  E.g :
  > vec_1 <- c(23,12,3.4)
  If we type vec_1 this vector name in console and press enter then we get the values in vector
  Output
  > vec_1
  [1] 23.0 12.0 3.4

To create a vector of integers using the c() function, you must place the letter "L" directly after each number.

c(1L, 5L, 15L)

You can also create a vector containing characters or logicals.

c("Sara" , "Lisa" , "Anna")

c(TRUE, FALSE, TRUE)

There are two types of vectors

1.atomic vector 2. list

- **Atomic Vector** : there are 6 types of atomic primary vectors.
  Logical, integer, double, character, complex and raw. Last two aren't as common.
- Every vector you create will have two key properties: type and length.
- **typeof() :** determine what type of vector typeof() function.

E.g:-

typeof(c(“a” , “b”))

#> [1] "character"

E.g:2

typeof(c(1L , 3L))

#> [1] "integer"

- **Length**() : - to find how many elements in a vector.

  E.g:

  x <- c(2.4,5.6,7.6)
  > length(x)
  [1] 3

- You can also check if a vector is a specific type by using anis function: is.logical(), is.double(), is.integer(), is.character()

  x <- c(2L, 5L, 11L)

  is.integer(x)

  #> [1] TRUE

- **Names**() : -

  name the elements of a vector with the names() function
  E.g:
  > x <- c(2.4,5.6,7.6)
  > names(x) <- c("a","b","c")
  > x
  a b c
  2.4 5.6 7.6

- Remember that an atomic vector can only contain elements of the same type. If you want to store elements of different types in the same data structure, you can use a list.

- **List**() : - List are different from atomic vectors because their elements can be of any type—like dates, data frames, vectors, matrices, and more. Lists can even contain other lists.

e.g:-

```
list("a", 1L, 1.5, TRUE)
list(list(list(1 , 3, 5)))
```

- **Str**() : - find out what types of elements a list contains, you can use the str() function

  E.g:-

  ```
  > str(list("a",2,34,4.5))
  List of 4
   $ : chr "a"
   $ : num 2
   $ : num 34
   $ : num 4.5
  ```

  E.g: 2 --

```
> z <- list(list(list(1,2,3)))
> str(z)
List of 1
 $ :List of 1
  ..$ :List of 3
  .. ..$ : num 1
  .. ..$ : num 2
  .. ..$ : num 3
```

  - Here $ symbol shows the nested structure list with lists.

- Naming list: you can name list when you create it.

  E.g: -

  ```
  > list('nashik' = 1, 'pune' = 2, 'mumbai' = 3)
  $nashik
  [1] 1
  ```

$pune
[1] 2
$mumbai
[1] 3

- pipe : - a tool in R for expressing sequence of multiple operations, represented with "%>%"

min() and max()

You can also find out the maximum and minimum lead times without sorting the whole dataset using the `arrange()` function. Try it out using the max() and min() functions below:
E.g:

max(hotel_bookings$lead_time)
min(hotel_bookings$lead_time)
Remember, in this case, you need to specify which dataset and which column using the $ symbol between their names.

you just want to know what the average lead time for booking is because your boss asks you how early you should run promotions for hotel rooms. You can use the `mean()` function to answer that question since the average of a set of number is also the mean of the set of numbers:

```{r mean}
mean(hotel_bookings$lead_time)
```

You should get the same answer even if you use the v2 dataset that included the `arrange()` function. This is because the `arrange()` function doesn't change the values in the dataset; it just re-arranges them.

```{r mean part two}
mean(hotel_bookings_v2$lead_time)
```

- Now, your boss wants to know a lot more information about city hotels, including the maximum and minimum lead time. They are also interested in

how they are different from resort hotels. You don't want to run each line of code over and over again, so you decide to use the `group_by()`and`summarize()` functions. You can also use the pipe operator to make your code easier to follow. You will store the new dataset in a data frame named 'hotel_summary':

```{r group and summarize}
hotel_summary <-
  hotel_bookings %>%
  group_by(hotel) %>%
  summarise(average_lead_time=mean(lead_time),
        min_lead_time=min(lead_time),
        max_lead_time=max(lead_time))
```

Check out your new dataset using head() again:

```{r}
head(hotel_summary)
```

Date and Time :

- To use date and time functions in R we will have first install tidyverse package.
  Install.packages("tidyverse")
- Lubridate is the part of tidyverse.
- Now load packages tidyverse and lubridate using library function.
  Library(tidyverse)
  Library(lubridate)
- Today() : - you will get the current date.
  > today()
  [1] "2022-09-14"

- Now() : get the current date-time.
  > now()
  [1] "2022-09-14 16:42:54 UTC"

R creates dates in the standard yyyy-mm-dd format by default.

You can convert strings into dates and date-times using the tools provided by lubridate. First, identify the order in which the year, month, and day appear in your dates. Then, arrange the letters *y*, *m*, and *d* in the same order For example, for the date *2021-01-20,* you use the order *ymd*:

```
> ymd("2021-03-23")
[1] "2021-03-23"
> mdy("january 20th,2021")
[1] "2021-01-20"
> dmy("20-jan-2023")
[1] "2023-01-20"
```

```
ymd_hms("2021-01-20 20:11:59")

#> [1] "2021-01-20 20:11:59 UTC"

mdy_hm("01/20/2021 08:01")

#> [1] "2021-01-20 08:01:00 UTC"
```

- As_date() : to convert date-time to date

  E.g:

  ```
  as_date(now())

  #> [1] "2021-01-20"
  ```

Data Frame :

- Data frame is a collection of columns - similar to SQL tables or spreadsheets.
- To manually create data frame use function data.frame()

  E.g:

  ```
  data.frame(x = c(1, 2, 3) , y = c(1.5, 5.5, 7.5))
  ```

- **Dir.create() :**

  dir.create function to create a new folder, or directory, to hold your files.

file.create (“new_text_file.txt”)

file.create (“new_word_file.docx”)

file.create (“new_csv_file.csv”)

- **File.copy() :**
  file.copy (“new_text_file.txt” , “destination_folder”)

- **Unlink() :** - to delete R file
  unlink (“some_.file.csv”)

- **Matrix** : is a two dimensional collection of data elements.
  It has both rows and columns. By contrast the vector is one dimensional. But like vectors, the matrix contains only a single data type.
  E.g: if you want to create matrix 2 x 3 containing values 3:8
  1…. matrix(c(3:8), nrow = 2)
  2… matrix(c(3:8), ncol = 2)
- E.g : we want to use diamonds dataset and it is in tidyverse package so install tidyverse first

```
install.packages("tidyverse")
Library(ggploat2)
Data(diamonds)
View(diamonds)
Head(diamonds)
Str(diamonds)
Colnames(diamonds)
```

- **Head**() : - (h small) it will give the first 6 rows . In case you just want to review data then use this function. View function will return all the rows.
- **Str**() :- this function will return the structure of the dataset.
- **Colnames**() : - this function will just the column names of the dataset.
- **Mutate**() : - it makes changes to our dataframe. It is part of the dplyr package which is in the tidyverse package so we will have to load the tidyverse package. (m small). Mutate() can also be used for column calculations.

E.g: mutate(diamonds,carat_2=carat*100)
This will create a new column carat_2.

- **Glimpse**() : - it will also return the summary of the dataframe like str() function. Str() and glimpse() and head() are the summary functions to give a preview of the dataframe.
- To create data frames first create 2 vectors.
  E.g: names <- c("a", "b", "c", "d")
  age <- c(10,12 ,14 , 15)
  With these two vectors, you can create a new data frame called `people`:
  people <- data.frame(names, age)

  To add another column use mutate()
  mutate(people, age_in_20 = age + 20)

- Create a dataframe manually, first make an id vector of values 1 to 10 then name a vector of 10 names , then job_title vector with 10 titles and finally create the dataframe using these 3 vectors.

```
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)
```

- **Separate**(): - this function separates columns.
  E.g: separate(employee, name, into= c("first_name","last_name"),sep= ' ')

- **Unite**() : - it combines 2 columns
  unite(employee, 'name', first_name, last_name, sep = ' ')

There are 3 logical operators in R

1. AND (sometimes presented as & or && in R)
2. OR (sometimes presented as | or || in R)
3. Not (!)

Conditional statements in R

4. If()
5. Else()
6. Else if()

E.g:

```
x <- -1

if (x < 0) {

  print("x is a negative number")

} else if (x == 0) {

  print("x is zero")

} else {

  print("x is a positive number")

}
```

- While writing the else statement it should be at the same line where the if statement ends.

Arithmetic operators :

Operator	Description	Example Code	Result/ Output
+	Addition	x + y	[1] 7
-	Subtraction	x - y	[1] -3

*	Multiplication	x * y	[1] 10
/	Division	x / y	[1] 0.4
%%	Modulus (returns the remainder after division)	y %% x	[1] 1
%/%	Integer division (returns an integer value after division)	y%/% x	[1] 2
^	Exponent	y ^ x	[1]25

Logical operators :

Operator	Description
&	Element-wise logical AND
&&	Logical AND
|	Element-wise logical OR
||	Logical OR
!	Logical NOT

The main difference between element-wise logical operators (&, |) and logical operators (&&, ||) is the way they apply to operations with vectors. The operations with double signs, AND (&&) and logical OR (||), only examine the *first* element of each vector. The operations with single signs, AND (&) and OR (|), examine all the elements of each vector.

First, create two variables, *x* and *y*, to store the two vectors:

```
x <- c(3, 5, 7)
```

```
y <- c(2, 4, 6)
```

Then run the code with a single ampersand (&). The output is boolean (TRUE or FALSE).

```
x < 5 & y < 5
```

```
[1]  TRUE FALSE FALSE
```

When you compare each element of the two vectors, the output is TRUE, FALSE, FALSE. The first element of both *x* (3) and *y* (2) is less than 5, so this is TRUE. The second element of x is *not* less than 5 (it's equal to 5) but the second element of y is less than 5, so this is FALSE (because you used AND). The third element of both x and y is not less than 5, so this is also FALSE.

Now, run the same operation using the double ampersand (&&):

```
x < 5 && y < 5
```

```
[1] TRUE
```

In this case, R only compares the *first* elements of each vector: 3 and 2. So, the output is TRUE because 3 and 2 are both less than 5.

Logical NOT (!)

The NOT operator simply negates the logical value, and evaluates to its opposite. In R, zero is considered FALSE and all non-zero numbers are considered TRUE.

For example, apply the NOT operator to your variable (x <- 10):

```
!(x < 15)
```

```
[1] FALSE
```

Assignment operators : -

Operator	Description	Example Code (after the sample code below, typing x will generate the output in the next column)	Result/ Output
<-	Leftwards assignment	x <- 2	[1] 2
<<-	Leftwards assignment	x <<- 7	[1] 7
=	Leftwards assignment	x = 9	[1] 9
->	Rightwards assignment	11 -> x	[1] 11
->>	Rightwards assignment	21 ->> x	[1] 21

Relational Operators :

Operator	Description	Example Code	Result/Output
<	Less than	x < y	[1] TRUE
>	Greater than	x > y	[1] FALSE
<=	Less than or equal to	x < = 2	[1] TRUE
>=	Greater than or equal to	y >= 10	[1] FALSE
==	Equal to	y == 5	[1] TRUE
!=	Not equal to	x != 2	[1] FALSE

Pipes :

- a tool in R for expressing a sequence of multiple operations , represented with "%>%"
- To insert symbol of pipe in windows keyboard shortcut is ctrl+shift+m

- It takes the output of one statement and makes the input for the next statement.

E.g : Toothgrowth dataset is already present in R. to Load this dataset use this command

data("ToothGrowth")

To view the dataset type in the script editor and then select that part and run it.

View("ToothGrowth") --- here V should be capital

- Then install package dplyr and load it from library function

Install.packages("dplyr")
Library(dpllyr)

Example from Rstudio

```
data("ToothGrowth")
view(ToothGrowth)

# first method to filter data
filt_tg <- filter(ToothGrowth,dose==0.5)
view(filt_tg)

arrange(filt_tg,len)   #arrange filtered dataset with len of tooth

# another method to get the same results

arrange(filter(ToothGrowth,dose==0.5),len)  # nested functions used here fun within fun

#now we will use pipe for same sorting and filtering

filt_tg <- ToothGrowth %>%
  filter(dose==0.5) %>%
  arrange(len)

# to view filtered dataset it is stored in filt_tg
view(filt_tg)
```

E.g:2 :- pipe

```
filt_tg <- ToothGrowth %>%
```

```
filter(dose==0.5) %>%
group_by(supp) %>%
arrange(len)
```

Tibbles :

Tibbles are like streamlined data frames that are automatically set to pull up only the first 10 rows of a dataset, and only as many columns as can fit on the screen. This is really useful when you're working with large sets of data. Unlike data frames, Tibbles never change the names of your variables, or the data types of your inputs. Overall, you can make more changes to data frames, but tibbles are easier to use. The tibble package is part of the core tidyverse.

E.g: - load the package and the dataset

```
Library(tidyverse)
Data(diamonds)
View(diamonds)
```

- Now the view function will show all the columns and all rows of the dataset.
- To create tibble use function as_tibble()
  E.g: - as_tibble(diamonds)
- Now the as_tibble() function will display only 10 rows in a neatly organized table.

Tidy Data : A way of standardizing the organization of data within R.

Import Data :

- **Data**() : if you run the data() function without argument then R will display a list of all available datasets.
- When you want to load a specific dataset then specify its name in the parenthesis. E.g: data(diamonds)
- Now if you just type diamonds on the console it will display a dataset because it is now loaded.
- If you click on the dataset name in the environment pane then also it will be displayed on the script editor.
- The readr package in R is a great tool for reading rectangular data. Rectangular data is data that fits nicely inside a rectangle of rows and columns, with each column referring to a single variable and each row referring to a single observation.

- Here are some examples of file types that store rectangular data:

1. .csv (comma separated values): a .csv file is a plain text file that contains a list of data. They mostly use commas to separate (or delimit) data, but sometimes they use other characters, like semicolons.

1. .tsv (tab separated values): a .tsv file stores a data table in which the columns of data are separated by tabs. For example, a database table or spreadsheet data.

1. .fwf (fixed width files): a .fwf file has a specific format that allows for the saving of textual data in an organized fashion.

1. .log: a .log file is a computer-generated file that records events from operating systems and other software programs.

- Base R also has functions for reading files, but the equivalent functions in readr are typically much faster. They also produce tibbles, which are easy to use and read.

- **Readr** : -The readr package is part of the core tidyverse. So, if you've already installed the tidyverse, you have what you need to start working with readr. If not, you can install the tidyverse now.

readr functions

- The goal of readr is to provide a fast and friendly way to read rectangular data. readr supports several read_ functions. Each function refers to a specific file format.
- **read_csv()**: comma-separated values (.csv) files
- **read_tsv()**: tab-separated values files
- **read_delim()**: general delimited files
- **read_fwf()**: fixed-width files
- **read_table()**: tabular files where columns are separated by white-space
- **read_log()**: web log files
- **Readr_example() :** The readr package comes with some sample files from built-in datasets that you can use for example code. To list the sample files, you can run the readr_example() function with no arguments.

- **Lets use read_csv() function**

  First run the readr_example() function to see what files are there. Then select one csv file, and provide the path of that file to read_csv() function.

  E.g: read_csv(readr_example("mtcars.csv")

  When you run the function, R prints out a column specification that gives the name and type of each column, R will also print tibble.
- **Readxl : -**To import spreadsheet data into R, you can use the readxl package. The readxl package makes it easy to transfer data from Excel into R. Readxl supports both the legacy .xls file format and the modern xml-based .xlsx file format.

  The readxl package is part of the tidyverse but is not a *core* tidyverse package, so you need to load readxl in R by using the library() function.

  library(readxl)

  Like the readr package, readxl comes with some sample files from built-in datasets that you can use for practice. You can run the code **readxl_example()** to see the list.

  You can use the **read_excel()** function to read a spreadsheet file just like you used read_csv() function to read a .csv file. The code for reading the example file **"type-me.xlsx"** includes the path to the file in the parentheses of the function.

  read_excel(readxl_example("type-me.xlsx"))

  You can use the excel_sheets() function to list the names of the individual sheets.

  excel_sheets(readxl_example("type-me.xlsx"))

  [1] "logical_coercion" "numeric_coercion" "date_coercion" "text_coercion"

  You can also specify a sheet by name or number. Just type **"sheet ="** followed by the name or number of the sheet. For example, you can use the sheet named **"numeric_coercion"** from the list above.

read_excel(readxl_example("type-me.xlsx"), sheet = "numeric_coercion")

When you run the function, R returns a tibble of the sheet.

Data Cleaning Packages :

- Here
- Skimr
- Janitor
- Install and load all these packages
  Install.packages("here")
  Library("here")
  Install.packages(skimr)
  Library(skimr)
  Install.packages("janitor")
  Library(janitor)
- Also install and load dplyr package because we are going to use some of its features
  Install.packages("dplyr")
  Library(dplyr)
- Functions to get summary of dataframes
  1. Skim_without_charts()
  2. Glimpse()
  3. Head()
  4. Select()

- E.g:- display only species column from penguin dataset
  Penguins %>%
  select(species)

  It will display only species columns.
  Penguins %>%
  select(-species)

  It will display all the columns except species
- Rename() - this function is used to change the column name.
  Penguins %>%
  rename(island_new=island)

- Rename_with() : change to lower or upper case. In R variables are mostly in lower case.
  Rename_with(penguins, tolower)
- Clean_names(penguin) : -this function will make sure there are only characters , numbers, and underscores in names of the penguin dataset

Organizing Data :

- Arrange()
- Group_by()
- Filter()

- All packages are in core package tidyverse so load that
  Install.packages("tidyverse")
  Library(tidyverse")
  install.packages("palmerpenguins")
  library("palmerpenguins")

```
penguins %>% arrange(bill_length_mm)
```

This will arrange data in ascending order of bill length, if we want it in decending order then use - (minus) sign before the column name

```
penguins %>% arrange(-bill_length_mm)
```

To save this results as a dataframe use following code.

```
peng_2 <- penguins %>% arrange(-bill_length_mm)
> view(peng_2)
```

if we want to get the summary

```
> penguins %>% group_by(island) %>% drop_na() %>%
summarise(mean_bill_len = mean(bill_length_mm))
# A tibble: 3 × 2
  island    mean_bill_len
  <fct>             <dbl>
1 Biscoe             45.2
2 Dream              44.2
3 Torgersen          39.0
```

Here drop_na() function will drop all the values with NA.

- E.g: find out penguin with longest bill lives on which planet

- penguins %>% group_by(island) %>% drop_na() %>% summarise(max_bill_len = max(bill_length_mm))

  E.g: group by island and species of penguins and then find max and mean

  penguins %>% group_by(island,species) %>% drop_na %>% summarize(max_bill = max(bill_length_mm), mean_bill = mean(bill_length_mm))

```
# A tibble: 5 × 4
# Groups:   island [3]
  island    species   max_bill mean_bill
  <fct>     <fct>        <dbl>     <dbl>
1 Biscoe    Adelie        45.6      39.0
2 Biscoe    Gentoo        59.6      47.6
3 Dream     Adelie        44.1      38.5
4 Dream     Chinstrap     58        48.8
5 Torgersen Adelie        46        39.0
```

E.g: - unite data

- example_df <- bookings_df %>%

  select(arrival_date_year, arrival_date_month) %>%

  unite(arrival_month_year, c("arrival_date_month",

"arrival_date_year"), sep = " ")

arrange function descending order :

- arrange(hotel_bookings, desc(lead_time))

E.g: -

You can also use the`mutate()` function to make changes to your columns. Let's say you wanted to create a new column that summed up all the adults, children, and babies on a reservation for the total number of people. Modify the code chunk below to create that new column:

example_df <- bookings_df %>%

mutate(guests = adults+children+babies)

head(example_df)

E.g2 :

calculate some summary statistics! Calculate the total number of canceled bookings and the average lead time for booking - you'll want to start your code after the %>% symbol. Make a column called 'number_canceled' to represent the total number of canceled bookings. Then, make a column called 'average_lead_time' to represent the average lead time. Use the `summarize()` function to do this in the code chunk below:

To find average use mean() function.

example_df <- bookings_df %>% summarise(num_canceled = sum(is_canceled), avg_lead_time = mean(lead_time))

head(example_df)

Use RStudio. Set the working directory at (Menu) > Session > Set Working Directory > Choose Directory.

Pivot_longer() and pivot_wider()

- As part of the tidyr package, you can use pivot_longer() to lengthen the data in a data frame by increasing the number of rows and decreasing the number of columns. Similarly, if you want to convert your data to have more columns and fewer rows, you would use the pivot_wider() function.

Visualization in R :

- Popular packages in visualization
    - Ggplot2 : here gg stands for grammer of graphics
    - Plotly
    - Lattice
    - RGL : focuses on 3D visuals
    - Dygraphs
    - Leaflets
    - Highcharter
    - Patchwork
    - Gganimate
    - Ggridges
- Benefits of ggplot2
    - Creates different types of plots
    - Customize the look and feel of plots
    - Create high quality visuals
    - Combine data manipulation and data visualization
- Core concepts in ggplot2
    - Aesthetics : visual property of an object in your plot.
      E.g:- size, shape, color
- Geoms : geometric object used to represent your data. E.g: points to create scatter plot, bars to create bar chart, lines to create line diagrams.
- Facets : let you display smaller groups , or subset of your data.
- Labels and annotations : customize your plot ,title,subtile,

Ggplot in R :

```
install.packages("tidyverse")
install.packages("ggplot2")
```

```
install.packages("palmerpenguins")
library(ggplot2)
library(palmerpenguins)
data("penguins")
head(penguins)
#creating plot
ggplot(data=penguins)+geom_point(mapping=aes(x=flipper_length_mm, y=
body_mass_g))

ggplot(data = penguins) + geom_point(mapping = aes(x =
flipper_length_mm, y = body_mass_g))
```

ggplot(data = penguins): In ggplot2, you begin a plot with the ggplot() function. The ggplot() function creates a coordinate system that you can add layers to. The first argument of the ggplot() function is the dataset to use in the plot. In this case, it's "penguins."

+: Then, you add a "+" symbol to add a new layer to your plot. You complete your plot by adding one or more layers to ggplot().

geom_point(): Next, you choose a geom by adding a geom function. The geom_point() function uses points to create scatterplots, the geom_bar function uses bars to create bar charts, and so on. In this case, choose the geom_point function to create a scatter plot of points. The ggplot2 package comes with many different geom functions. You'll learn more about geoms later in this course.

(mapping = aes(x = flipper_length_mm, y = body_mass_g)): Each geom function in ggplot2 takes a mapping argument. This defines how variables in your dataset are mapped to visual properties. The mapping argument is always paired with the aes() function. The x and y arguments of the aes() function specify which variables to map to the x-axis and the y-axis of the coordinate system. In this case, you want to map the variable "flipper_length_mm" to the x-axis, and the variable "body_mass_g" to the y-axis.

Pro-Tip: You can write the same section of code above using a different syntax with the mapping argument inside the ggplot() call: ggplot(data =

penguins, mapping = aes(x = flipper_length_mm, y = body_mass_g)) +
geom_point()

E.g:-
A stakeholder tells you, "I want to target people who book early, and I have a hypothesis that people with children have to book in advance."

When you start to explore the data, it doesn't show what you would expect. That is why you decide to create a visualization to see how true that statement is-- or isn't.

```
ggplot(data = hotel_bookings) +
  geom_point(mapping = aes(x = lead_time, y = children))
```

E.g:2
Your stakeholder says that she wants to increase weekend bookings, an important source of revenue for the hotel. Your stakeholder wants to know what group of guests book the most weekend nights in order to target that group in a new marketing campaign. She suggests that guests without children book the most weekend nights. Is this true?

Try mapping 'stays_in_weekend_nights' on the x-axis and 'children' on the y-axis by filling out the remainder of the code below.

```
ggplot(data = hotel_bookings) +
 geom_point(mapping = aes(x = stays_in_weekend_nights , y = children ))
```

Aesthetics :

- Visual property of object in your plot.
- Like how it looks
- Aesthetics for points
    1. X
    2. y
    3. Size
    4. Shape
    5. Color
    6. Alpha

```
install.packages("tidyverse")
install.packages("ggplot2")
install.packages("palmerpenguins")
library(ggplot2)
library(palmerpenguins)
data("penguins")
head(penguins)
#creating plot
ggplot(data=penguins)+geom_point(mapping=aes(x=flipper_length_mm, y= body_mass_g,color=species))
```

- Here we have added color aesthetics to species so different species will have different colors in plot.

```
ggplot(data=penguins)+geom_point(mapping=aes(x=flipper_length_mm, y= body_mass_g,shape=species))
```

- Here we have added different sizes for different species

```
ggplot(data=penguins)+geom_point(mapping=aes(x=flipper_length_mm, y= body_mass_g,shape=species,color = species))
```

- How we will get the plot with different shapes and sizes for different species.

```
ggplot(data=penguins)+geom_point(mapping=aes(x=flipper_length_mm, y= body_mass_g,shape=species, alpha = species))
```

- Here we used alpha aes to make points transparent; it is useful in dense data.
- ggplot(data=penguins)+geom_point(mapping=aes(x=flipper_length_mm, y= body_mass_g), color = "purple")
- Here we gave color purple to each point by adding color outside of aes function.

Geom :

- Geoms : geometric object used to represent your data. E.g: points to create scatter plots, bars to create bar charts, lines to create line diagrams.
- Geom functions
    1. Geom_point()
    2. Geom_bar()

3. Geom_line()
4. Geom_smooth() : - will create line plot

```
ggplot(data=penguins)+geom_smooth(mapping=
aes(x=flipper_length_mm, y=body_mass_g) )
```

- We can also and more than one plot e.g:

```
ggplot(data=penguins)+geom_smooth(mapping=
aes(x=flipper_length_mm, y=body_mass_g) ) +
  geom_point(mapping = aes(x= flipper_length_mm, y= body_mass_g))
```

- These example will show different types of lines for different species

```
ggplot(data=penguins)+geom_smooth(mapping =
aes(x=flipper_length_mm ,y =body_mass_g, linetype = species))
```

- Geom_bar

  This will create bar chart with x axis and R by default provides y-axis here number of diamonds

```
ggplot(data=diamonds) + geom_bar(mapping = aes(x=cut))
```

- This will the color

```
ggplot(data=diamonds) + geom_bar(mapping = aes(x=cut,
color=cut))
```

- To fill the bars with color

```
ggplot(data=diamonds) + geom_bar(mapping = aes(x=cut,
fill=cut))
```

- If we add clarity as a fill then we will get stacked bar chart

```
ggplot(data=diamonds) + geom_bar(mapping = aes(x=cut,
fill=clarity))
```

- Types of Smoothening

Type of smoothing	Description	Example code
Loess smoothing	The loess smoothing process is best for smoothing plots with less than 1000 points.	`ggplot(data, aes(x=, y=))+ geom_point() + geom_smooth(method="loess")`
Gam smoothing	Gam smoothing, or generalized additive model smoothing, is useful for smoothing plots with a large number of points.	`ggplot(data, aes(x=, y=)) + geom_point() + geom_smooth(method="gam", formula = y ~s(x))`

Facets :

- Facets : let you display smaller groups , or subsets of your data.
- It is mostly used to compare data.
- Facet functions
    - Facet_wrap()
    - Facet_grid()
- E.g: this will create plot for different species 3 plots
  ggplot(data=penguins,aes(x=flipper_length_mm,y= body_mass_g))+geom_point(aes(color=species))+facet_wrap(~species)

- To facet your plot with 2 variables use facet_grid()
  E.g: -ggplot(data=penguins)+geom_point(mapping= aes(x=flipper_length_mm,y=body_mass_g, color=species))+facet_grid(sex~species)
  Here we can remove sex or species and then observe data.

```
ggplot(data = hotel_bookings) +
  geom_bar(mapping = aes(x = distribution_channel)) +
  facet_wrap(~deposit_type) +
  theme(axis.text.x = element_text(angle = 45))
```

```{r filtering a dataset with the pipe}
onlineta_city_hotels_v2 <- hotel_bookings %>%
  filter(hotel=="City Hotel") %>%
  filter(market_segment=="Online TA")
```

E.g: hotel booking
Now, you will add in a subtitle using `subtitle=` in the `labs()` function. Then, you can use the `paste0()` function to use your newly-created variables in your labels. This is really handy, because if the data gets updated and there is more recent data added, you don't have to change the code below because the variables are dynamic:

```
ggplot(data = hotel_bookings) +
  geom_bar(mapping = aes(x = market_segment)) +
  facet_wrap(~hotel) +
  theme(axis.text.x = element_text(angle = 45)) +
  labs(title="Comparison of market segments by hotel type for hotel bookings",
           subtitle=paste0("Data from: ", mindate, " to ", maxdate))
```

- Now you want to clean up the x and y axis labels to make sure they are really clear. To do that, you can add to the `labs()` function and use `x=` and `y=`. Feel free to change the text of the label and play around with it:

```{r city bar chart with x and y axis}
ggplot(data = hotel_bookings) +
  geom_bar(mapping = aes(x = market_segment)) +
  facet_wrap(~hotel) +
```

```
  theme(axis.text.x = element_text(angle = 45)) +
  labs(title="Comparison of market segments by hotel type for hotel
bookings",
            caption=paste0("Data from: ", mindate, " to ", maxdate),
            x="Market Segment",
            y="Number of Bookings")
```

- Then run the following code chunk to save that plot as a .png file named `city_payment_chart`, which makes it clear to your stakeholders what the .png file contains. Now you should be able to find this file in your 'Files' tab in the bottom right of your screen. Check it out!

```{r save your plot}
ggsave('hotel_booking_chart.png')
```

- default dimensions of this ggsave() image are 7x7. You can see these dimensions listed after you run the code chunk.

If you wanted to make your chart bigger and more rectangular to fit the slide show presentation, you could specify the height and width of your .png in the `ggsave()` command. Edit the code chunk below to create a 16x8 .png image:

```{r save your plot}
ggsave('hotel_booking_chart.png',
            width=16,
            height=8)
```

Annotation :

- To add notes to a document or diagram to explain or comment upon it.

```
ggplot(data=penguins)+geom_point(mapping=
aes(x=flipper_length_mm, y= body_mass_g, color= species))+
```

labs(title = "palmer penguins: body mass vs flipper length", subtitle = "3 penguins dataset", caption= "data collected by DR. Gorman")
Here the caption will be displayed at bottom.

- To save the plot use the export tab near plot and save with format you would like, name it. If you want to open it then click the file pane tab and open the image.
- Ggsave() : - it is a useful function to save plots. It default saves the last plot you created. And uses the size of the current graphic device.

E.g: -

```
install.packages("tidyverse")
library(tidyverse)
install.packages("ggplot2")
library(ggplot2)
install.packages("palmerpenguins")
library(palmerpenguins)

ggplot(data=penguins)+geom_point(mapping = aes(x= flipper_length_mm, y = body_mass_g, color = species))

ggsave("three penguins.png")
```

Example of using png()	Example of using pdf()
`png(file = "exampleplot.png", bg = "transparent") plot(1:10) rect(1, 5, 3, 7, col = "white") dev.off()`	`pdf(file = "/Users/username/Desktop/example.pdf", width = 4, height = 4) plot(x = 1:10, y = 1:10) abline(v = 0) text(x = 0, y = 1, labels = "Random text") dev.off()`

Bias function :

- To use bias function we will need to install package "SimDesign"

Install.packages("SimDesign")
Library(SimDesign)

E.g: check local weather forcast is biased or not

```
install.packages("SimDesign")
library(SimDesign)
actual_temp <- c(68.3,70,72.4,71,67,70)
predicted_temp <- c(67.9,69,71.5,70,67,69)
bias(actual_temp,predicted_temp)
```

Output - 0.7166667

Which is close to zero but still it's biased.

- sample() function allows you to take a random sample of elements from a data set

R markdown :

- A file format for making dynamic documents with R.
- Markdown : - a syntax for formatting plain text files.
- R Notebook : - lets you run your code and show the charts and graphs that visualize the code.
- To use markdown notebook in R , first install package install.packages("rmarkdown")
- Hashtag(#) used in R markdown for headers
- To create reports in markdown click knit button.
- To create new markdown file go into file menu then click new -> new markdown file , give notebook name and author name then filename.rmd
- Other notebooks
    1. Kaggle
    2. Jupiter

- YAML stands for yet another markup language.

  The first part of your notebook is the YAML header section. YAML is a language used in data files to improve human readability, and the YAML header section exists to provide information about a document to the humans reading it. RStudio automatically populates this section

with the information you provide and other general information, such as the date you create the file.

```
---
title: "Penguins Plots"
author: "DA Cert"
date: "2/26/2021"
output: html_document
---
```

again, RStudio automatically populates the notebook with this formatted default code chunk. This chunk basically means that your code will be shown in your final report when you're ready to render it.

All code chunks begin and end with delimiters. To start a code chunk, you can type three tick marks followed by a lowercase "r" in curly brackets: ```{r}

To end it, type just the three tick marks: ```

- To add the bullet points in R markdown notebook use * at start only
- To include link put the link into < > symbols
- If you want to embed like like e.g: click here [link] (put actual link here)
- To include images start with ! Mark e.g: ![display text](image link)
- In a code chunk after r in the braces add the heading you want to add for the code chunk.

There are two shortcuts to adding code. On your keyboard, you can press Ctrl + Alt + I (PC)

Click the end of the last line of your Rmd file. Use either of the previously-mentioned shortcuts to create a code chunk.

2. Press Enter (Windows) two or three times after the default code chunk to create space between the existing code chunk and the next code chunk you will add.

3. Copy the code from the analysis file you opened earlier and paste it in the gray area between the beginning and ending delimiters.

4. Select the rest of the template content in the file and delete it. This gives you a blank space to work in to help avoid potential errors from mixing your own comments and code with the pre-existing ones in the template.

The white background is where you will type plain text with markdown syntax. As you learned earlier in this course, markdown is a syntax for formatting plain text files. Using markdown makes it easier to write and format text in your notebook.

here are some basic formatting options:

- To start a new paragraph, end a line with two spaces
- To apply italics to a word or phrase, place an asterisk at the beginning and at the end of the word or phrase, for example, *italics works*
- To apply bold to a word or phrase, place two asterisks at the beginning and at the end of the word or phrase, for example, **bold is useful**
- To create a header, type a hashtag (#) followed by a space and your text for example: # Getting Started with R Markdown

When creating headers keep the following in mind:

- Headers will appear in blue
- A single hashtag is the largest header
- The more hashtags you add (up to six), the smaller the header

To format comments in your notebook, follow these steps:

1. Click in a line above the code chunk you added but below the YAML section.

2. Type a main header for your report using a single hashtag. You might want to restate the title in the YAML in a different way or add to it with a short description.

3. Add a smaller header below that to label the first part of your programming. Follow that with a description of the code chunk that you added.

- <<Rmarkdown_output_format.pdf>>

  Other packages provide even more output formats:

  The bookdown package is helpful for writing books and long-form articles.

  The prettydoc package provides a range of attractive themes for R Markdown documents.

  The rticles package provides templates for various journals and publishers.

  popular packages with templates for R Markdown include the following:

- The vitae package contains templates for creating and maintaining a résumé or curriculum vitae (CV)
- The rticles package provides templates for various journals and publishers
- The learnr package makes it easy to turn any R Markdown document into an interactive tutorial
- The bookdown package facilitates writing books and long-form articles
- The flexdashboard package lets you publish a group of related data visualizations as a dashboard