**R Programming Language**

Introduction

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

**R** offers convenient statistical features for data analysis and is useful for creating advanced data visualizations. Check out these resources to learn more about R:

* [The R Project for Statistical Computing](https://www.r-project.org/" \o "This link takes you to the R Project home page." \t "_blank): a website for downloading R, documentation, and help
* [R Manuals](https://cran.r-project.org/manuals.html): links to manuals from the R core team, including introduction, administration, and help
* [Coding Club R Tutorials](https://ourcodingclub.github.io/tutorials.html): a collection of coding tutorials for R
* [R for Beginners](https://cran.r-project.org/doc/contrib/Paradis-rdebuts_en.pdf): a starting guide to help you work with data, graphics, and statistics in R

RStudio Cloud (now Posit Cloud)

**Source editor** – save scripts

**Console** – run code line by line, commands will be forgotten once session is closed

Packages

Units of reproducible R code. Packages offer a helpful combination of code, reusable R functions, descriptive documentation, tests for checking your code, and sample datasets.

Packages in R include reusable R functions and documentation about the functions, including how to use them. They also contain sample datasets and tests for checking your code to make sure it does what you want it to do.

Check installed packages

> installed.packages()

**Base package** – installed and loaded

**Recommended package** – installed, but not loaded

CRAN – Comprehensive R Archive Network

It's an online archive with R packages, source code, manuals and documentation.

Installing a package

> install.packages("tidyverse")

Loading a library/package

> library(tidyverse)

> library(lubridate)

**INITIAL DATA EXPLORATION**

Get summary of data

> summary(penguins)

View data

> View(penguins)

Scatterplot

> ggplot(data=penguins, aes(x=flipper\_length\_mm, y=body\_mass\_g)) + geom\_point(aes(color=species))

Help for any function

> ?print()

Details will appear in the help window

**BASICS**

Variable

A variable name can contain numbers and underscores, but it should begin with a letter.

Shortcut for Multi-line Comment in R

Ctrl + Shift + C

Cmd + Shift + C

Assignment operator

variable <- value

first\_variable <- “This is my variable”

second\_variable <- 12.5

Modulus operator (returns the remainder after division)

%%

y %% x

Integer division operator (returns an integer value after division)

%/%

y %/% x

Exponent operator

^

y^x

Vector

A vector is a group of data elements of the same type stored in a sequence in R.

You can make a vector using the combined function.

variable <- c(x, y, z, …)

Atomic Vector

There are six primary types of atomic vectors: logical, integer, double, character (which contains strings), complex, and raw. **All elements are of same type.**

**Numeric:** c(2.5, 48.5, 101.5)

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

c(1L, 5L, 15L)

c(“Sara” , “Lisa” , “Anna”)

c(TRUE, FALSE, TRUE)

Type of vector

typeof(c(“a”, “b”, “c”))

Length of vector

length(c(“a”, “b”, “c”))

is operator

is.logical(), is.double(), is.integer(), is.character()

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

is.integer(x)

#> [1] TRUE

y <- c(TRUE, TRUE, FALSE)

is.character(y)

#> [1] FALSE

Naming elements of Vector

x <- c(1, 3, 5)

names(x) <- c("a", "b", "c")

x

#> a b c

#> 1 3 5

Lists

**Multiple data types are allowed in same list.**

Structure of lists

list("a", 1L, 1.5, TRUE)

str(list("a", 1L, 1.5, TRUE))

List of 4

$ : chr "a"

$ : int 1

$ : num 1.5

$ : logi TRUE

z <- list(list(list(1 , 3, 5)))

str(z)

List of 1

$ :List of 1

..$ :List of 3

.. ..$ : num 1

.. ..$ : num 3

.. ..$ : num 5

The indentation of the $ symbols reflect the nested structure of this list. Here, there are three levels (so there is a list within a list within a list)

Naming lists

Lists, like vectors, can be named. You can name the elements of a list when you first create it with the list() function:

list('Chicago' = 1, 'New York' = 2, 'Los Angeles' = 3)

$`Chicago`

[1] 1

$`New York`

[1] 2

$`Los Angeles`

[1] 3

**LOGICAL OPERATORS**

* AND (sometimes represented as & or && in R)
* OR (sometimes represented as | or || in R)
* NOT (!)

Element-wise logical AND (&) and OR (|)

x <- 10

The AND operator returns TRUE only if both individual values are TRUE.

x > 2 & x < 12

[1] TRUE

Just one of the values of the OR operation needs to be TRUE for the entire OR operation to evaluate to TRUE. Only if both values are FALSE will the entire OR operation evaluate to FALSE.

x > 2 | x < 8

[1] TRUE

Logical AND (&&) and OR (||)

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.

For example, imagine you are working with two vectors that each contain three elements: c(3, 5, 7) and c(2, 4, 6). The element-wise logical AND (&) will compare the first element of the first vector with the first element of the second vector (3&2), the second element with the second element (5&4), and the third element with the third element (7&6).

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.

Date & Date Time

Using package tidyverse and library lubridate.

Today’s date

today()

Current date time

now()

Converting strings/numerals to date

ymd("2021-01-20")

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

mdy("January 20th, 2021")

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

dmy("20-Jan-2021")

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

ymd(20210120)

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

Converting strings/numerals to date-time

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"

Switching b/w data-time and date

as\_date(now())

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

**CONDITIONAL STATEMENTS**

* if()
* else()
* else if()

x <- -1

# run the code

if (x < 0) {

print("x is a negative number")

} else if (x == 0) {

print("x is zero")

} else {

print("x is a positive number")}

**FILE HANDLING**

Files: Create a new folder

dir.create ("destination\_folder")

Files: Create a blank file

file.create (“new\_text\_file.txt”)

file.create (“new\_word\_file.docx”)

file.create (“new\_csv\_file.csv”)

Files: Copy Paste file

file.copy (“new\_text\_file.txt” , “destination\_folder”)

Files: Delete file

unlink (“some\_.file.csv”)

**DATAFRAME**

The data.frame() function takes vectors as input.

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

x y

1 1 1.5

2 2 5.5

3 3 7.5

Create dataframe from scratch using vectors

names <- c("A", "B", "C", "D")

age <- c(20, 30, 40, 50)

people <- data.frame(names, age)

head(people)

str(people)

glimpse(people)

colnames(people)

mutate(people, age\_in\_20=age+20)

Matrices

**All elements are of same type.**

matrix(c(3:8), nrow = 2)

[,1] [,2] [,3]

[1,] 3 5 7

[2,] 4 6 8

matrix(c(3:8), ncol = 2)

[,1] [,2]

[1,] 3 6

[2,] 4 7

[3,] 5 8

Display the columns and the first several rows of data

head(diamonds)

Summary of each column in your data arranged horizontally

str(diamonds) 🡪 Structure function to check structure of dataframe

glimpse(diamonds)

List of column names in dataset

colnames(diamonds)

Rename column

rename(diamonds, carat\_new = carat, cut\_new = cut)

Summary statistics

summarize(diamonds, mean\_carat = mean(carat))

Make changes to dataframe

mutate() function is within dlpyr package.

Add new column to dataframe

library(tidyverse)

mutate(diamonds, carat\_2=carat\*100)

Visualization with ggplot2()

To build a visualization with `ggplot2` you layer plot elements together with a `+` symbol.

**Scatterplot**

ggplot(data = diamonds, aes(x = carat, y = price)) + **geom\_point()**

**Show data categories with colors**

ggplot(data = diamonds, aes(x = carat, y = price, **color = cut**)) + geom\_point()

**Separate out data for each category**

ggplot(data = diamonds, aes(x = carat, y = price, color = cut)) + geom\_point() + **facet\_wrap(~cut)**

tidyverse

Collection of packages in R with a common design philosophy for data manipulation, exploration, and visualization.

Update tidyverse packages

tidyverse\_update()

Update.packages() – updates all packages

Install.packages(“package name”) – update only required packages

Vignette

A vignette is a 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.

browseVignettes(“packagename”)

browseVignettes(“ggplot2”)

8 core tidyverse packages

* ggplot2
* tibble
* tidyr
* readr
* purr
* dplyr
* stringr
* forcats

4 tidyverse packages at the centre of workflow for data analysis:

* **ggplot2** – create a variety of data viz by applying different visual properties to the data variables.

ggplot(data=penguins, aes(x=flipper\_length\_mm, y=body\_mass\_g)) + geom\_point()

* **tidyr** – package used for data cleaning to make tidy data
* **readr** – used for importing data.

The most common function from readr is read\_csv().

To accurately read a dataset with readr, you combine the function with a column specification. The column specification describes how each column should be converted to the most appropriate data type. This isn't usually necessary because readr will figure it out for you automatically.

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
* **dplyr** – offers a consistent set of functions that help you complete some common data manipulation tasks.

select() function

filter() function

Factors

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

Pipe

A pipe is a tool in R for expressing a sequence of multiple operations. A pipe is represented by a % sign, followed by a > sign, and another % sign. It's used to apply the output of one function or statement into another function or statement. Pipes can make your code easier to read and understand.

ToothGrowth %>% filter(dose == 0.5) %>% arrange(len)

Pipe is a way to code. Instead of typing out functions contained inside other functions (nested functions), you could use the pipe operator to do the same work.

Load the already installed dataset

data("ToothGrowth")

View data

View(ToothGrowth)

Write a pipe

filtered\_toothgrowth <- ToothGrowth %>%

filter(dose==0.5) %>%

arrange(len)

* Add the pipe operator at the end of each line of the piped operation, except the last one.
* RStudio automatically indents lines of code that are part of a pipe.
* If a line in code isn’t indented, it probably hasn’t been added to the pipe

Filter data

filtered\_tg <- filter(ToothGrowth, dose==0.5)

hotel\_bookings\_city <- filter(hotel\_bookings, hotel\_bookings$hotel == "City Hotel")

Filter data using pipe

penguins %>% filter(species == "Adelie")

Sort/Arrange data

arrange(filtered\_tg, len)

Arrange in ascending order

penguins %>% arrange(bill\_length\_mm)

Arrange in descending order (just add a minus sign before column name)

penguins %>% arrange(-bill\_length\_mm)

Doing filter and sort using nested function

arrange(filter(ToothGrowth, dose==0.5), len)

Shortcut for inserting pipe operator “%>%”

Ctrl+Shift+M

Cmd+Shift+M

Group by and summarize data using pipe – find average tooth length for each supplement category

filtered\_toothgrowth <- ToothGrowth %>%

filter(dose==0.5) %>%

group\_by(supp) %>%

summarize(mean\_len = mean(len, na.rm = T), .group="drop")

na.rm = T 🡪 remove null values = True

Group by categories and summarize by mean

penguins %>%

group\_by(island) %>%

drop\_na() %>%

summarize(mean\_bill\_length\_mm = mean(bill\_length\_mm))

Group by categories and summarize by max

penguins %>%

group\_by(island) %>%

drop\_na() %>%

summarize(max\_bill\_length\_mm = max(bill\_length\_mm))

Group by multiple categories and summarize by multiple aggregate functions

penguins %>%

group\_by(species, island) %>%

drop\_na() %>%

summarize(max\_bl = max(bill\_length\_mm), mean\_bl = mean(bill\_length\_mm))

Find max, min, mean of a column directly

max(hotel\_bookings$lead\_time)

min(hotel\_bookings$lead\_time)

mean(hotel\_bookings$lead\_time)

Specify which dataset and which column using the $ symbol between their names.

Tibble

Streamlined dataframe.

* **Never change the data type of inputs**. Saves time because you won't have to do as much cleaning or changing data types in tibbles.
* **Never change the names of variables**.
* **Never create row names**.
* **Make printing in R easier**. Tibbles don’t accidently load the console because they **automatically set to pull up only the first 10 rows**, and as many columns as fit on screen.

Create tibble from an existing dataframe

tibble(diamonds)

Tidy data

Way of standardizing the organization of data. Variables in columns, observations in rows, each value has its own cell.

Read excel file

library(readxl)

read\_excel(readxl\_example("type-me.xlsx"))

List the names of the individual sheets

excel\_sheets(readxl\_example("type-me.xlsx"))

Read a csv file,

bookings\_df <- read\_csv("hotel\_bookings.csv")

head(bookings\_df)

str(bookings\_df)

glimpse(bookings\_df)

colnames(bookings\_df)

Select specific data from existing dataframe

new\_df <- select(bookings\_df, `adr`, adults)

Add new column to the selected data without and with pipe

mutate(new\_df, total = adr / adults)

pg <- penguins %>% mutate(body\_mass\_kg = body\_mass\_g/1000, flipper\_length\_m = flipper\_length\_mm/1000)

**DATA CLEANING**

here package

Makes referencing easier

install.packages("here")

library(here)

skimr package

Makes summarizing data really easy and let's you skim through it more quickly.

install.packages("skimr")

library(skimr)

janitor package

Has functions for cleaning data.

install.packages("janitor")

library(janitor)

Comprehensive summary of dataset

skim\_without\_charts(penguins)

Select one column

penguins %>%

select(species)

Select all columns except that one column

penguins %>%

select(-species)

Rename all columns to upper/lower case

rename\_with(penguins, toupper)

rename\_with(penguins, tolower)

clean\_names() function in janitor package

Automatically make sure that the column names are unique and consistent. This ensures that there's only characters, numbers, and underscores in the names.

clean\_names(penguins)

**Example:** The different columns will be changed or not changed as follows:

column 2 🡪 Yes, only characters, numbers, and underscores are allowed

column4 🡪 No, it already has only characters, numbers.

column.1 🡪 Yes, only characters, numbers, and underscores are allowed

column\_3 🡪 No, it already has only characters, numbers, and underscores

**DATA TRANSFORMATION**

Combine 2 or more columns into 1 column

example\_df <- bookings\_df %>%

select(arrival\_date\_year, arrival\_date\_month) %>%

unite(arrival\_month\_year, c("arrival\_date\_month", "arrival\_date\_year"), sep = " ")

Split a column into 2 or more separate columns

Syntax: separate(<dataframe>, <column-to-split>, into=<vector-of-new-column-names> c("col\_new\_1", "col\_new\_2"), sep=<separator> " ")

separate(employee, name, into=c("first\_name", "last\_name"), sep=" ")

Join 2 or more columns

Syntax: unite(<dataframe>, "<new-column-name>", <col1-to-join>, <col2-to-join>, sep=" ")

unite(employee2, "name", f\_name, l\_name, sep=" ")

Wide and Long data

**Wide data** – No. of columns >> No. of rows. Wide data has observations across several columns. Each column contains data from a different condition of the variable.

**Long data** – No. of rows >> No. of columns. Long data has all the observations in a single column, and the variable conditions are placed into separate rows.

To lengthen the data in a data frame by increasing the number of rows and decreasing the number of columns

pivot\_longer()

To convert your data to have more columns and fewer rows

pivot\_wider()

Anscombe's Quartet

Anscombe's quartet has four datasets that have nearly identical summary statistics. But those summary statistics might be misleading. Data visualizations, especially for datasets like these, are so important. They help us discover things in our data that would otherwise remain hidden.

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))

# A tibble: 4 × 6

set `mean(x)` `sd(x)` `mean(y)` `sd(y)` `cor(x, y)`

<fct> <dbl> <dbl> <dbl> <dbl> <dbl>

1 I 9 3.32 7.50 2.03 0.816

2 II 9 3.32 7.50 2.03 0.816

3 III 9 3.32 7.5 2.03 0.816

4 IV 9 3.32 7.50 2.03 0.817

Based on the summaries created with statistical measures, these datasets are identical, **but sometimes just looking at the summarized data can be misleading**. Let's put together some simple graphs to help us visualize this data and check if the datasets are actually identical.

ggplot(quartet, aes(x,y)) + geom\_point() + geom\_smooth(method=lm, se=FALSE) + facet\_wrap(~set)

# datasauRus package

install.packages("datasauRus")

library("datasauRus")

ggplot(datasaurus\_dozen, aes(x=x, y=y, color=dataset)) + geom\_point() + theme\_void() + theme(legend.position = "none") + facet\_wrap(~dataset,ncol=3)

Bias function

bias() function compares the actual outcome and the predicted outcome of the data to determine the average amount the actual outcome is greater than the predicted outcome. **An unbiased model should be close to zero**.

**Example 1:**

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)

[1] 0.7166667 🡪 Actual is close to Predicted

**Example 2:**

> actual\_sales <- c(150, 203, 137, 247, 116, 287)

> predicted\_sales <- c(200, 300, 150, 250, 150, 300)

> bias(actual\_sales, predicted\_sales)

[1] -35 🡪 Actual is much smaller than Predicted

Take a random sample of elements from a dataset

sample()

**DATA VISUALIZATION**

Common visualization packages

* ggplot2 – most popular, gg stands for **grammer of graphics**
* Plotly
* Lattice
* RGL
* Dygraphs
* Leaflet
* Highcharter
* Patchwork
* gganimate
* ggridgers

Grammar of Graphics

A system for describing and building data visualizations. The essential idea behind the grammer of graphics is that we can build any plot from the same basic components, like building blocks.

These building blocks include:

* A **dataset**
* A **set of geoms**: A geom refers to the geometric object used to represent your data. For example, you can use points to create scatterplot, bars to create bar chart, lines to create line diagram, etc.
* A **set of aesthetic attributes**: An aesthetic is a visual property of an object in your plot. You can think of an aesthetic as a connection, or mapping, between a visual feature in your plot and a variable in your data.

For example, in a scatterplot, aesthetics include things like size, shape, colour, or location (x-axis, y-axis) of your data points.

To create a plot with ggplot2, you first choose a dataset. Then, you determine how to visually organize your data on a coordinate system by choosing a geom to represent your data points and aesthetics to map your variables.

ggplot2 benefits

* Create different types of plots – scatter plots, bar charts, line diagrams, etc.
* Customize the look and feel of plots – change colours, layout and dimensions, add text elements like titles, captions and labels.
* Create high quality visuals
* Combine data manipulation and visualization – using the pipe operator

ggplot2 core concepts

* **Aesthetics** – Visual property of an object in plot. For example, in a scatter plot aesthetics include things like the size, shape or color of your data points.
* **Geoms** – Geometric object used to represent data. For example, you can use points to create a scatter plot, bars to create a bar chart, or lines to create a line diagram. You can choose a geom to fit the type of data you have. Points show the relationship between two quantitative variables. Bars show one quantitative variable varies across different categories.
* **Facets** – Let you display smaller groups or subsets of the data. With facets, you can create separate plots for all the variables in your dataset.
* **Labels and annotations** – Let you customize your plot. You can add text like titles, subtitles and captions to communicate the purpose of your plot or highlight important data.

ggplot2 Components

Template

ggplot(data = <DATA>) + <GEOM\_FUNCTION>(mapping = aes(<AESTHETIC\_MAPPINGS>))

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 ggplot() function. The ggplot() function creates a coordinate system that you can add layers to. The first argument of ggplot() 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.

* geom\_point() – create scatterplots
* geom\_bar() – create bar charts

(mapping = aes(x = flipper\_length\_mm, y = body\_mass\_g))

Each geom function in ggplot2 takes a mapping argument. This defines how variables in dataset are mapped to visual properties.

aes()

The mapping argument is always paired with aes() function. It tells what aesthetics to use for the plot. The x and y arguments of aes() function specify which variables to map to x-axis and y-axis. In this case, you want to map “flipper\_length\_mm” to x-axis and “body\_mass\_g” to y-axis.

3 Steps to create a plot

1. Start with the ggplot() function and choose a dataset to work with.
2. Add a geom\_function() to display your data
3. Map the variables you want to plot in the argument of the aes() function.

Represent different categories by **colour**

ggplot(data=penguins) +

geom\_point(mapping = aes(x = flipper\_length\_mm, y = body\_mass\_g, **color = species**))

Represent different categories by colour and **shape**

ggplot(data=penguins) +

geom\_point(mapping = aes(x = flipper\_length\_mm, y = body\_mass\_g, color = species, **shape = species**))

Represent different categories by colour, shape and **size**

ggplot(data=penguins) +

geom\_point(mapping = aes(x = flipper\_length\_mm, y = body\_mass\_g, color = species, shape = species, **size = species**))

Represent different categories by density or transparency

alpha is a good plot when we have a dense plot with lots of data points

ggplot(data=penguins) +

geom\_point(mapping = aes(x = flipper\_length\_mm, y = body\_mass\_g, **alpha = species**))

Represent whole data by different colour

color parameter will sit outside aes() function and inside geom function

ggplot(data=penguins) +

geom\_point(mapping = aes(x = flipper\_length\_mm, y = body\_mass\_g), color = "purple")

Smooth Line Trend

ggplot(data=penguins) +

**geom\_smooth**(mapping = aes(x = flipper\_length\_mm, y = body\_mass\_g))

Combine Trend Line and Scatter Point

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))

Represent different categories by different Smooth Lines

ggplot(data=penguins) +

geom\_smooth(mapping = aes(x = flipper\_length\_mm, y = body\_mass\_g, **linetype = species**))

**geom\_jitter() function** – create a scatter plot and then add a small amount of random noise to each point in the plot.

Jittering helps us deal with over-plotting, which happens when the data points in a plot overlap with each other. Jittering makes the points easier to find.

ggplot(data=penguins) +

geom\_jitter(mapping = aes(x = flipper\_length\_mm, y = body\_mass\_g))

Bar Chart

ggplot(data=diamonds) +

**geom\_bar**(mapping = aes(x = cut))

We didn't supply a variable for the y-axis. When you use geom\_bar(), R automatically counts how many times each x-value appears in the data, and then shows the counts on the y-axis. The default for geom\_bar() is to count rows.

Represent different categories in Bar Chart by adding colour to boundary of bars

ggplot(data=diamonds) +

geom\_bar(mapping = aes(x = cut, **color=cut**))

Represent different categories in Bar Chart by filling colour into bars

ggplot(data=diamonds) +

geom\_bar(mapping = aes(x = cut, **fill=cut**))

Stacked Bar Chart – Represent 2 categories in Bar Chart by filling colour into bars with 2nd category

ggplot(data=diamonds) +

geom\_bar(mapping = aes(**x = cut, fill=clarity**))

Smoothing

Sometimes it can be hard to understand trends in your data from scatter plots alone.

**Smoothing enables the detection of a data trend even when you can't easily notice a trend from the plotted data points**.

ggplot2’s smoothing functionality is helpful because it adds a smoothing line as another layer to a plot; the smoothing line helps the data to make sense to a casual observer.

ggplot(data, aes(x=distance, y= dep\_delay)) + geom\_point() + **geom\_smooth()**

Two types of Smoothing

| **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))** |

Facet Functions

A facet is a side or section of an object. Facets show different sides of data by placing each subset on its own plot. Faceting can help discover new patterns in data and focus on relationships between different variables.

**facet\_wrap()** – Facet plot by a single variable

# penguins dataset

ggplot(data=penguins) +

geom\_point(mapping = aes(x = flipper\_length\_mm, y = body\_mass\_g, color = species))+

**facet\_wrap**(~species)

# diamonds dataset

View(diamonds)

ggplot(data=diamonds) +

geom\_bar(mapping = aes(x = color, fill = cut)) +

facet\_wrap(~cut)

**facet\_grid()** – Facet plot by 2 variables

facet\_grid() splits the plot into facets vertically by the values of the first variable and horizontally by the values of the second variable.

ggplot(data=penguins) +

geom\_point(mapping = aes(x = flipper\_length\_mm, y = body\_mass\_g, color = species))+

**facet\_grid**(sex~species)

facet\_grid() works with 1 variable as well, to allow focus on a single variable’s categories.

Then what is the difference between using facet\_wrap() and facet\_grid() for 1 variable?

**facet\_grid() it will include plots even if they are empty.**

ggplot(data=penguins) +

geom\_point(mapping = aes(x = flipper\_length\_mm, y = body\_mass\_g, color = species))+

facet\_grid(~species)

ggplot(data=penguins) +

geom\_point(mapping = aes(x = flipper\_length\_mm, y = body\_mass\_g, color = species))+

facet\_grid(~sex)

Filtering data for plotting

Filtering data before you plot it allows you to focus on specific subsets of your data and gain more targeted insights. Just include the dplyr filter() function in your ggplot syntax.

data %>%

filter(variable1 == "DS") %>%

ggplot(aes(x = weight, y = variable2, colour = variable1)) +

geom\_point(alpha = 0.3, position = position\_jitter()) + stat\_smooth(method = "lm")

**Example**:

# create a plot that shows the relationship between lead time and guests traveling with children for online bookings at city hotels

library(tidyverse)

library(dplyr)

onlineta\_cityhotels <- hotel\_bookings %>%

filter(hotel=="City Hotel" & market\_segment=="Online TA")

View(onlineta\_cityhotels)

ggplot(onlineta\_cityhotels) +

geom\_point(aes(x=lead\_time, y=children))

1. Based on your previous filter, this scatterplot shows data for online bookings for city hotels.
2. The plot reveals that bookings with children tend to have a shorter lead time, and bookings with 3 children have a significantly shorter lead time (<200 days).
3. So, promotions targeting families can be made closer to the valid booking dates.

**# Shortcut - using pipe for filtering and plotting**

hotel\_bookings %>%

filter(hotel=="City Hotel" & market\_segment=="Online TA") %>%

ggplot() + geom\_point(aes(x=lead\_time, y=children))

Label and Annotate

**Annotate** – means to add notes to a document or diagram to explain or comment upon it.

Add title, subtitle, caption, annotation

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="Sample of Three Penguin Species", caption = "Data collected by Dr. Kristen Gorman")** +

**annotate("text", x=220, y=3500, label="Gentoos are the largest")**

Change color and style of annotation

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="Sample of Three Penguin Species", caption = "Data collected by Dr. Kristen Gorman") +

annotate("text", x=220, y=3500, label="Gentoos are the largest", **color="purple", fontface="bold", size=4.5, angle=25**)

Storing graph as variable and adding annotation to it

**p <-** 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="Sample of Three Penguin Species", caption = "Data collected by Dr. Kristen Gorman")

p + annotate("text", x=220, y=3500, label="Gentoos are the largest", color="purple", fontface="bold", size=4.5, angle=25)

Saving a plot

* Export in Plots tab.
* Use ggsave() function from ggplot2

ggsave("<file-name-with-extension>", width = <custom-width>, height = <custom-height>)

* Send to png graphics device

png(file = "exampleplot.png", bg = "transparent")

plot(1:10)

rect(1, 5, 3, 7, col = "white")

dev.off()

* Send to pdf graphics device

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()

**R MARKDOWN (.Rmd files)**

File format for making dynamic documents in R.

Markdown

A syntax for formatting plain text files.

R Notebook

Lets users run your code and show the graphs and charts that visualize the code.

R Markdown package

install.packages("rmarkdown")

YAML – Yet Another Markup Language

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.

**YAML header section:**

---

title: "R Markdown Intro"

author: "Nilesh"

date: "2023-07-06"

output: html\_document

---

This YAML header section contains metadata, i.e., data about the data.

Headers in Markdown

# Header 1 style heading

## Header 2 style heading

### … so on

Italics

\_word\_ OR \*word\*

Bold

\*\*word\*\*

Code in-between Mardown Text

Text text `<code>` text text

Tick marks format the text to appear as code even though the text is not in a code chunk.

Code Chunk in Markdown

```{r <optional-label-for-code>}

<code>

```

OR

Go to Code > **+C**Insert Chunk

Example:

```{r loading packages}

install.packages("ggplot2")

install.packages("palmerpenguins")

```

Shortcut to add Code Chunk in Markdown

Ctrl + Alt + I

Cmd + Option + I

Delimiter

A delimiter is a character that indicates the beginning or end of a data item.

Start a new paragraph

End a line with two spaces.

Create/Render HTML Report

Use Knit tool in the source code pane.

Add Clickable Link

Use angular brackets

<http://rmarkdown.rstudio.com>

Embed link within text

For example, to create something like “click here”

text text [<word-to-embed-link-in>](<url>)

click [here](http://rmarkdown.rstudio.com)

Embed image

![<caption>](<url>)

![Plot this way](https://cdn.pixabay.com/photo/2013/07/12/13/52/arrow-147464\_\_340.png)

Bullet points

\* followed by space

Markdown is a simple formatting syntax for authoring:

\* HTML

\* PDF

\* MS Word documents

Convert Rmd file to other formats

Just replace the default html\_document in YAML header section to pdf or required file format.

---

title: "R Markdown Intro"

author: "Nilesh"

date: "2023-07-06"

output: **pdf**\_document

---

Available document outputs

* **pdf\_document** – This creates a PDF file with LaTeX (an open source document layout system). If you don’t already have LaTeX, RStudio will automatically prompt you to install it.
* **word\_document** – This creates a Microsoft Word document (.docx).
* **odt\_document** – This creates an OpenDocument Text document (.odt).
* **rtf\_document** – This creates a Rich Text Format document (.rtf).
* **md\_document** – This creates a Markdown document (which strictly conforms to the original Markdown specification)
* **github\_document** – This creates a GitHub document which is a customized version of a Markdown document designed for sharing on GitHub.

Presentations

You can also use R Markdown to produce presentations. Automatically inserting the results of your R code into a presentation can save you lots of time.

* **beamer\_presentation** – for PDF presentations with beamer
* **ioslides\_presentation** – for HTML presentations with ioslides
* **slidy\_presentation** – for HTML presentations with Slidy
* **powerpoint\_presentation** – for PowerPoint presentations
* **revealjs : : revealjs\_presentation** – for HTML presentations with reveal.js (a framework for creating HTML presentations that requires the reveal.js package)

Shiny

**Shiny** is an R package that lets you build interactive web apps using R code. You can embed your apps in R Markdown documents or host them on a webpage.

To call Shiny code from an R Markdown document, add runtime: shiny to the YAML header:

---

title: "Shiny Web App"

output: html\_document

runtime: shiny

---

To learn more about Shiny and how to use R code to add interactive components to an R Markdown document, check out the [Shiny](https://shiny.rstudio.com/tutorial/) tutorial from RStudio.

Template

Creating a template for reports allows to run one line of code to update data without having to recreate the report from scratch.

Templates can also help customize the appearance of final report.