

R for Data Science

Center for Health Data Science

June 2025

Who are we?

Center for Health Data Science

- Data Science Research Groups
- HDS Sandbox
- SUND DataLab

<https://heads.ku.dk/>

SUND DataLab

- Courses & Workshops
- Consultations
- Commission Research & Supervision
- Events, Seminars



Thilde Terkelsen, Ph.D.



Henrike Zschach



Diana Andrejeva



Helene Wegener

What will I Learn?



Program

Day 1		Day 2		Day 3	
08:30-09:00	Installation Issues + Coffee	08:30-09:00	Optional Q&A + Coffee	08:30-09:00	Optional Q&A + Coffee
09:00-09:15	Introduction to Course	09:00-10:00	Buffer for E3	09:00-09:15	
09:15-09:30	P0: Intro to R and Quarto	10:00-10:30	P4: Scripting in R		
09:30-10:00	E0: Intro to R and Quarto	10:30-10:45	Break		
10:00-10:30	P1: Data Clean-Up in Base R and Tidyverse	10:45-12:00	E4		
10:30-10:45	Break	12:00-13:00	Lunch	09:15-10:00	
10:45-12:00	E1: Data Clean-Up	13:00-14:00	P5: Modelling in R	10:00-10:45	
11:15-12:00	P2: Advanced Data Wrangling		E5	11:00-12:00	
12:00-13:00	Lunch			12:00-13:00	Lunch
13:00-14:15	E2: Advanced Data Wrangling	13:00-14:30			
14:15-14:30	Break			13:00-16:00	Project work
14:30-15:00	P3: Exploratory Data Analysis	14:45-15:15			
15:00-16:00	E3: Exploratory Data Analysis	15:15-16:00			
16:00	See you tomorrow!	16:00	See you tomorrow!	16:00	Wrap up + Bye Bye!

Program

Day 1		Day 2		Day 3	
08:30-09:00	Installation Issues + Coffee	08:30-09:00	Optional Q&A + Coffee	08:30-09:00	Optional Q&A + Coffee
09:00-09:15	Introduction to Course	09:00-10:00	Buffer for E3	09:00-09:15	
09:15-09:30	P0: Intro to R and Quarto	10:00-10:30	P4: Scripting in R		
09:30-10:00	E0: Intro to R and Quarto	10:30-10:45	Break		
10:00-10:30	P1: Data Clean-Up	10:45-12:00	E4		
10:30-10:45	Break	12:00-13:00	Lunch	09:15-10:00	
10:45-12:00	E1: Data Clean-Up	13:00-14:00	P5: Modelling in R	10:00-10:45	
11:15-12:00	P2: Advanced Data Wrangling		E5	11:00-12:00	
12:00-13:00	Lunch			12:00-13:00	Lunch
13:00-14:15	E2: Advanced Data Wrangling	13:00-14:30			
14:15-14:30	Break			13:00-16:00	Project work
14:30-15:00	P3: Exploratory Data Analysis	14:45-15:15			
15:00-16:00	E3: Exploratory Data Analysis	15:15-16:00			
16:00	See you tomorrow!	16:00	See you tomorrow!	16:00	Wrap up + Bye Bye!

Part 0

R script and Quarto

R script

- Flat script
 - Submit to HPC
- Comment script and build structure using #
- Source

Quarto

- Markdown-based
- Render to get a nice report in html or website
- Headers and text

R script

```
R_script_example.R
1 #####
2 # R for Data Science - How to R script #
3 # Author: DataLab HeaDS #
4 # Date: 8 Novmeber 2024 #
5 #####
6
7 #####
8 ##### Load Packages #####
9 #####
10 library(tidyverse)
11 library(readxl)
12 #####
13 ##### Load Data #####
14 #####
15 #####
16 diabetes <- read_excel('~/Desktop/DataLab/R4DataScience/data/diabetes_toy.xlsx')
17 #####
18 ##### Inspect Data #####
19 #####
20 #####
21
22 # Check dimensions of data
23 dim(diabetes)
24
25 # Check structure of data
26 str(diabetes)
27
28 # Check for NA's in each column
29 colSums(is.na(diabetes))
30
31 #####
32 ##### Exploratory Data Analysis #####
33 #####
34
35 # Plot distribution of BMI
36 diabetes %>%
37   ggplot(aes(x = BMI)) +
38   geom_histogram(bins = 10)
39
40
```

Quarto

```
Quarto_example.qmd
---  
title: "R for Data Science - How to R script"  
format: html  
author: DataLab HeaDS  
editor: visual  
---
```

Load Packages

```
{r}  
library(tidyverse)  
library(readxl)
```

Load Data

```
{r}  
diabetes <- read_excel('~/Desktop/DataLab/R4DataScience/data/diabetes_toy.xlsx')
```

Inspect Data

Check dimensions of data

```
{r}  
dim(diabetes)
```

Check structure of data

```
{r}  
str(diabetes)
```

Check for NA's in each column

```
{r}  
colSums(is.na(diabetes))
```

Exploratory Data Analysis

Plot distribution of BMI

```
{r}  
diabetes %>%
  ggplot(aes(x = BMI)) +
  geom_histogram(bins = 10)
```

Render to html

```
library(tidyverse)  
library(readxl)
```

```
diabetes <- read_excel('~/Desktop/DataLab/R4DataScience/data/diabeti
```

Check dimensions of data

```
dim(diabetes)
```

[1] 100 8

Check structure of data

```
str(diabetes)
```

```
tibble [100 x 8] (S3:tbl_df/tbl/data.frame)  
$ ID : chr [1:100] "ID_035" "ID_020" "ID_090" "ID_146" ...  
$ Fasting_Blood_Sugar : num [1:100] 157 146 128 134 157 101 144 141 ...  
$ Post_Meal_Blood_Sugar: num [1:100] 204 178 206 206 250 204 181 ...  
$ HbA1c : num [1:100] 5.8 6.3 6.6 8 6.9 7.4 6.9 5.9 ...  
$ Age : num [1:100] 36 70 63 49 56 39 59 43 57 56 ...  
$ Sex : chr [1:100] "Male" "Male" "Male" "Male" ...  
$ BMI : num [1:100] 33.7 26.5 26 22.9 25.7 24.4 25 ...  
$ Blood_Pressure : num [1:100] 122 120 137 126 115 121 116 111 ...
```

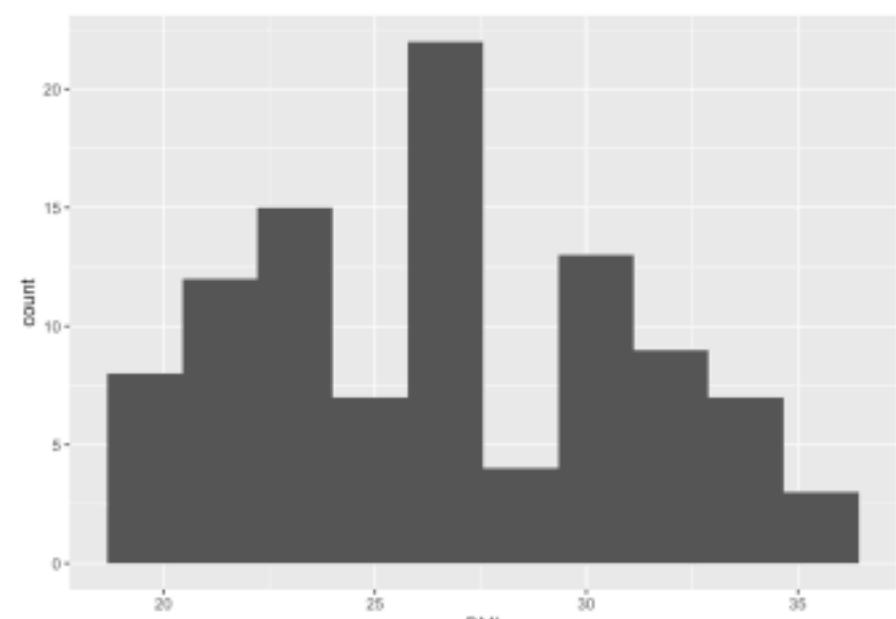
Check for NA's in each column

```
colSums(is.na(diabetes))
```

ID	Fasting_Blood_Sugar	Post_Meal_Blood_Sugar
0	0	0
HbA1c	Age	Sex
0	0	0
BMI	Blood_Pressure	
0	0	

Plot distribution of BMI

```
diabetes %>%
  ggplot(aes(x = BMI)) +
  geom_histogram(bins = 10)
```



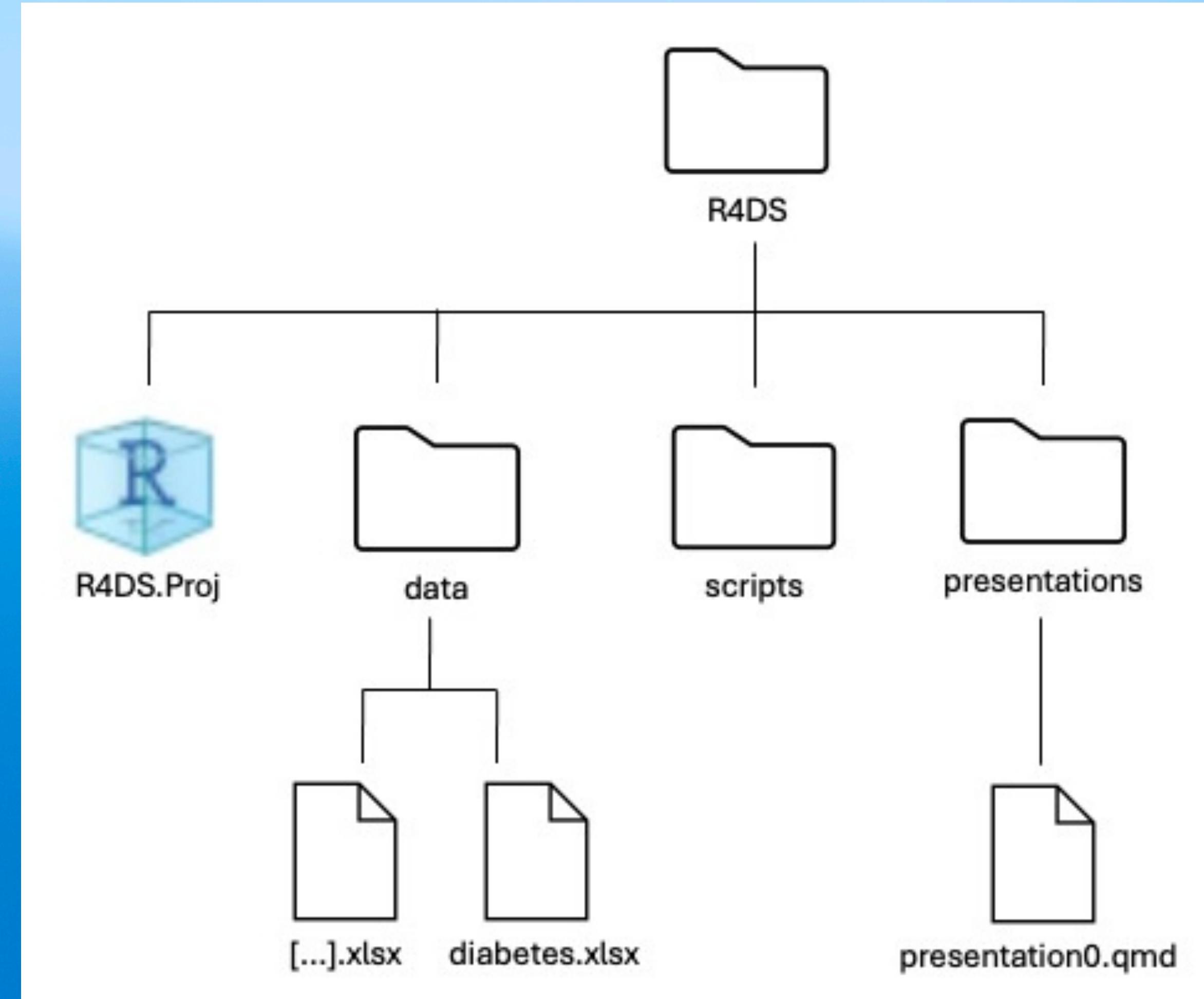
R project

R4DS.Rproj

- Location of files is the root for all files within the directory.

.Rproj.user

- Hidden directory to store of temporary files.



Exercise 0

R script and Quarto

Part 1

Base R and Tidyverse

Base R

- Basic library that is pre-installed in R
- Supports older versions of R
- Efficient for small tasks
- Complex workflows are doable, but syntax can be convoluted
 - Specify data frame multiple times

Tidyverse

- A collection of packages such as
 - dplyr (data manipulation, %>%)
 - ggplot2 (visualization)
 - tidyr (reshaping)
 - readr (reading data)
 - stringr (string manipulation)
- Modern and intuitive syntax, even for complex workflows
- Assumes tidy data (each column is a variable, and each row is an observation)

Cheat Sheet - Base R and Tidyverse

	Base R	Tidyverse
Select columns	<code>df[, c("col1", "col2")]</code>	<code>df %>% select(col1, col2)</code>
Select one column	<code>df["col1"]</code>	<code>df %>% select(col1)</code>
Access one column as vector	<code>df\$col1</code> <code>df[["col1"]]</code>	<code>df %>% pull(col1)</code>
Add new column	<code>df\$col_new <- list_new</code>	<code>df <- df %>% mutate(col_new = list_new)</code>
Filter rows	<code>df[df\$col1 < 10 ,]</code>	<code>df %>% filter(col1 < 10)</code>
Remove rows with NA's	<code>df[complete.cases(df_baseR),]</code>	<code>df %>% drop_na()</code>
Access unique values of column	<code>unique(df\$col1)</code>	
Access count of unique values of column	<code>table(df\$col1)</code>	

Cheat Sheet - String manipulation

	Command
Combines string with a separator	<code>paste(STRING, sep = “ ”)</code>
Combines string without a separator	<code>paste0(STRING)</code>
Splits a string into parts in a list.	<code>str_split(STRING, pattern = “ ”)</code>
Splits a string into parts in a list and retrieves the i'th index.	<code>str_split_i(STRING, pattern = “ ”, i = 1)</code>
Detect substring in main string.	<code>str_detect(STRING, SUBSTRING)</code>
Remove whitespace	<code>str_trim(STRING)</code>

More on stringr: <https://stringr.tidyverse.org/>

Exercise 1

Base R and Tidyverse

Part 2

Advanced Tidyverse

Long and wide format

```
tree_long <- tree_wide %>%
  pivot_longer(cols = starts_with("Site"),
               names_to = "Site",
               values_to = "Average diameter (cm)")
```

Type species	Site A	Site B	Site C	Site D
Acer rubrum	15	8	30	27
Quercus alba	29	17	14	42
Pinus teada	10	19	25	23

```
tree_wide <- tree_long %>%
  pivot_wider(names_from = Site,
              values_from = `Average diameter (cm)`)
```



Tree species	Site	Average diameter (cm)
Acer rubrum	Site A	15
Acer rubrum	Site B	8
Acer rubrum	Site C	30
Acer rubrum	Site D	27
Quercus alba	Site A	29
Quercus alba	Site B	17
Quercus alba	Site C	14
Quercus alba	Site D	42
Pinus teada	Site A	10
Pinus teada	Site B	19
Pinus teada	Site C	25
Pinus teada	Site D	23

Nesting

Tree species	Site	Average diameter (cm)
Acer rubrum	Site A	15
Acer rubrum	Site B	8
Acer rubrum	Site C	30
Acer rubrum	Site D	27
Quercus alba	Site A	29
Quercus alba	Site B	17
Quercus alba	Site C	14
Quercus alba	Site D	42
Pinus teada	Site A	10
Pinus teada	Site B	19
Pinus teada	Site C	25
Pinus teada	Site D	23

```
tree_long_nested <- tree_long %>%
  group_by(`Type species`) %>%
  nest(Data = c(Site, `Average diameter (cm)`)) %>%
  ungroup()
```



Type species	Data
Acer rubrum	<tibble>
Quercus alba	<tibble>
Pinus teada	<tibble>

```
tree_long_nested %>%
  filter(`Type species` == 'Quercus alba') %>%
  pull(Data)
```

Site	Average diameter (cm)
Site A	29
Site B	17
Site C	14
Site D	42

Exercise 2

Advanced Tidyverse

Part 3

Exploratory Data Analysis

Exploratory Data Analysis

- Identify variable type, ranges and distributions
- Identify and remedy quality issues (???)
- Investigate relationships between variables
- Discover patterns, trends, and structures that inform further analysis or modeling

ggplot2 Recap

Syntax

```
df %>%  
  ggplot(aes(x = var1,  
             y = var2,  
             color = var3)) +  
  geom_XXX() +  
  scale_color_manual(values = c("color1", "color2")) +  
  labs(title = "title",  
       x = "x axis title",  
       y = "y axis title",  
       color = "legend title") +  
  theme_XXX()
```

geom's

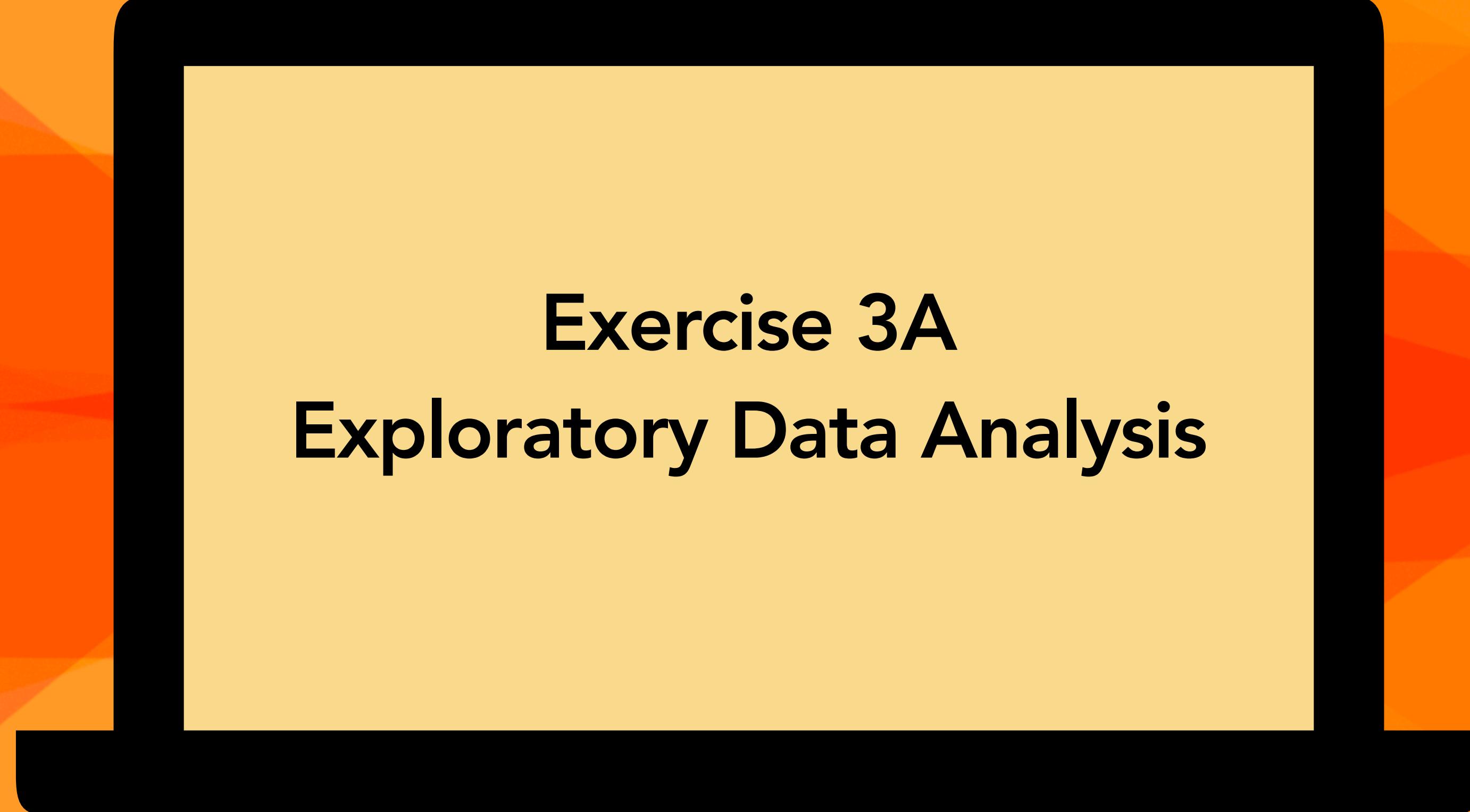
```
geom_point # scatter plot  
geom_line  
geom_boxplot  
geom_violin  
geom_bar  
geom_col
```

Aesthetics (aes)

- x
- y
- color
- fill
- shape
- group
- linetype

Themes

```
theme_grey # default  
theme_classic  
theme_minimal  
theme_bw  
theme_light  
theme_dark
```



Exercise 3A

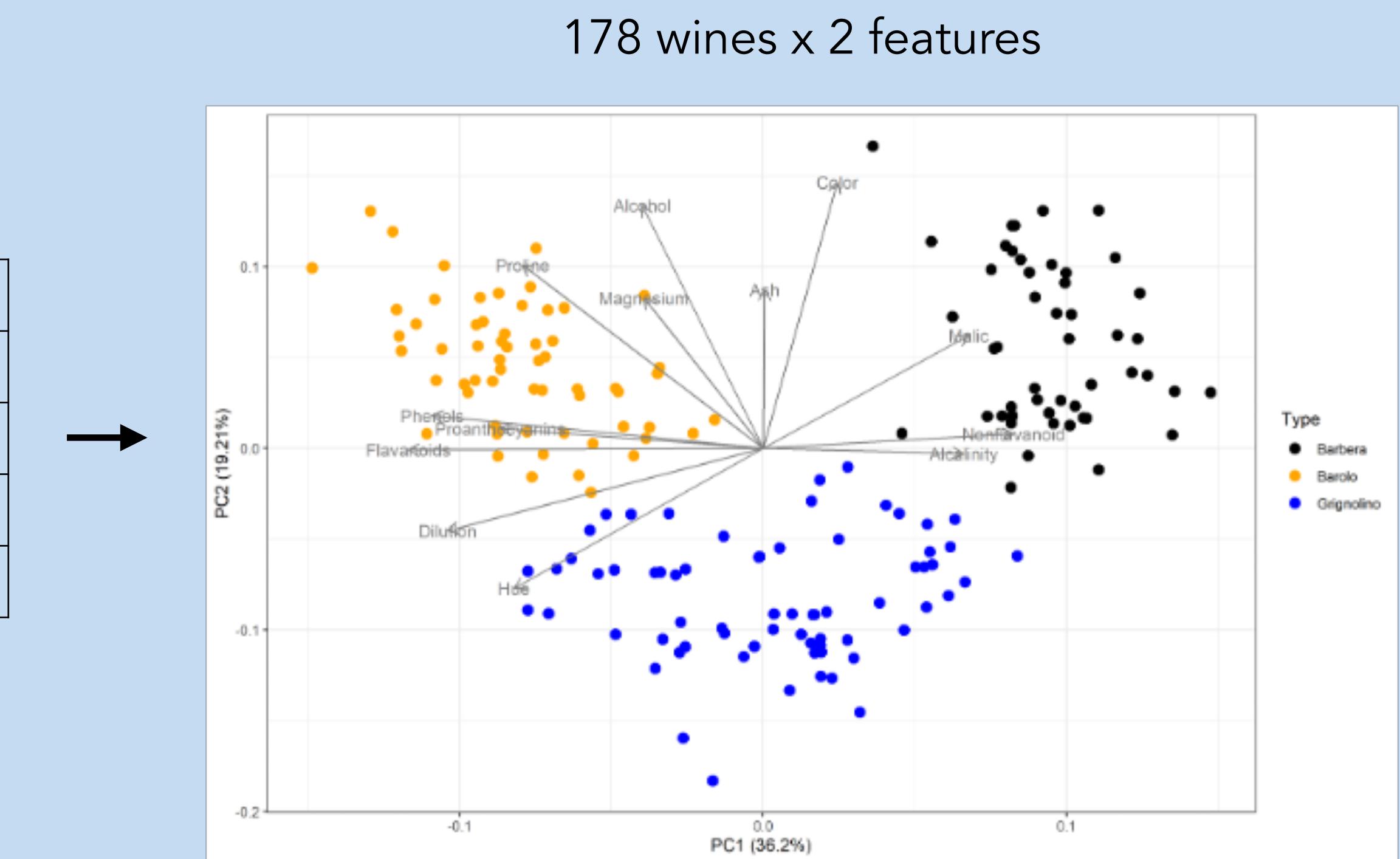
Exploratory Data Analysis

Principal Component Analysis (PCA)

- Linear dimensionality reduction: High dimensional space → Low dimensional space
- Find new dimensions that capture as much variation as possible
- Shows structure of the data across all variables

178 wines x 13 features

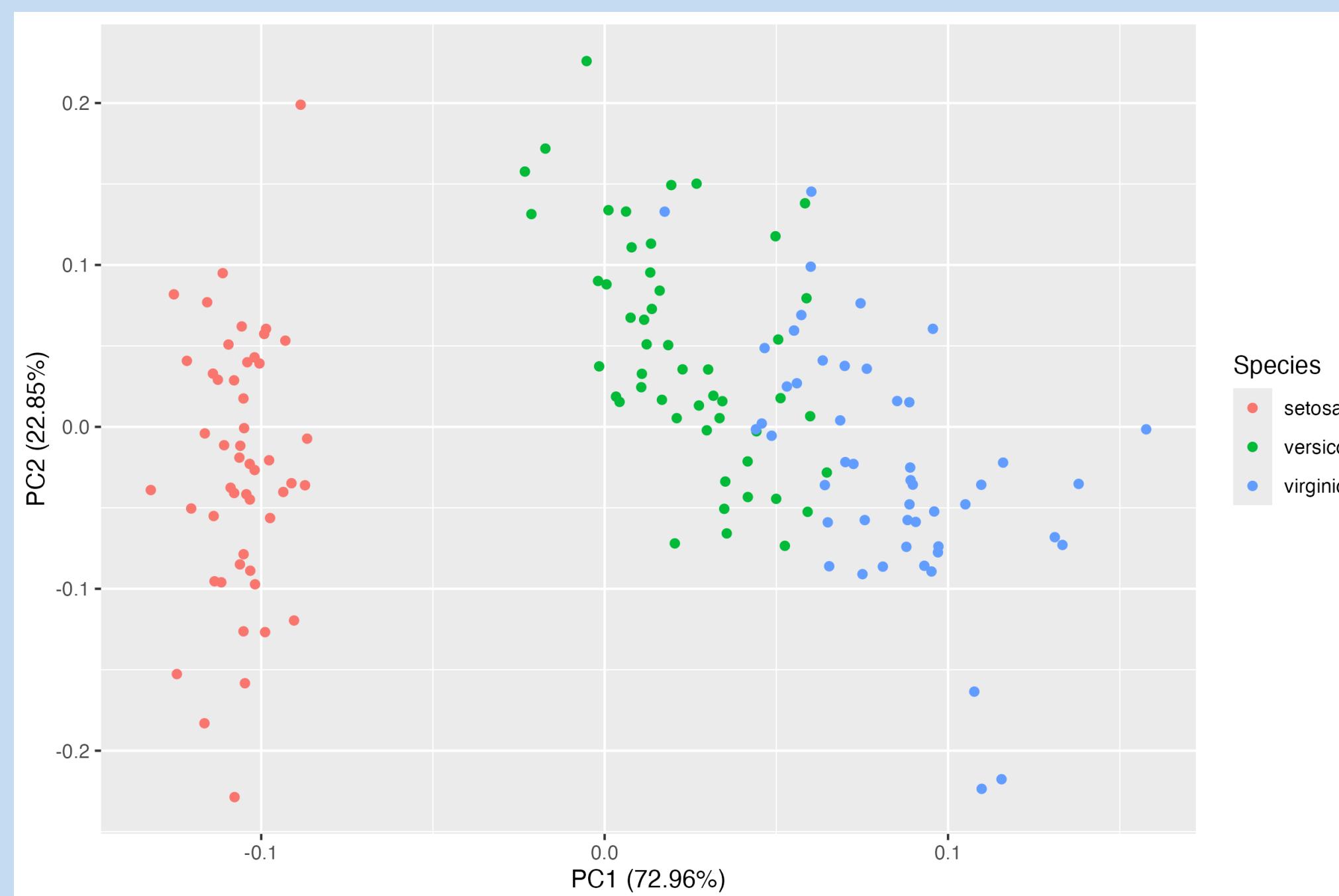
Wine type	Alc.	Alkalinity	Flabonoids	Color	...
Barolo	1.51	-1.17	1.03	0.25	...
Barolo	0.25	-2.48	0.73	-0.29	...
Grignolino	1.50	-0.96	0.97	0.57	...
...



PCA in R

R syntax

```
library(ggfortify)  
df <- iris[1:4] # extract numerical columns  
pca_res <- prcomp(df, scale. = TRUE)  
autoplot(pca_res, data = iris, color = "Species")
```



```
> head(iris)
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa
5	5.0	3.6	1.4	0.2	setosa
6	5.4	3.9	1.7	0.4	setosa

```
> head(pca_res$x)
```

	PC1	PC2	PC3	PC4
[1,]	-2.257141	-0.4784238	0.12727962	0.024087508
[2,]	-2.074013	0.6718827	0.23382552	0.102662845
[3,]	-2.356335	0.3407664	-0.04405390	0.028282305
[4,]	-2.291707	0.5953999	-0.09098530	-0.065735340
[5,]	-2.381863	-0.6446757	-0.01568565	-0.035802870
[6,]	-2.068701	-1.4842053	-0.02687825	0.006586116

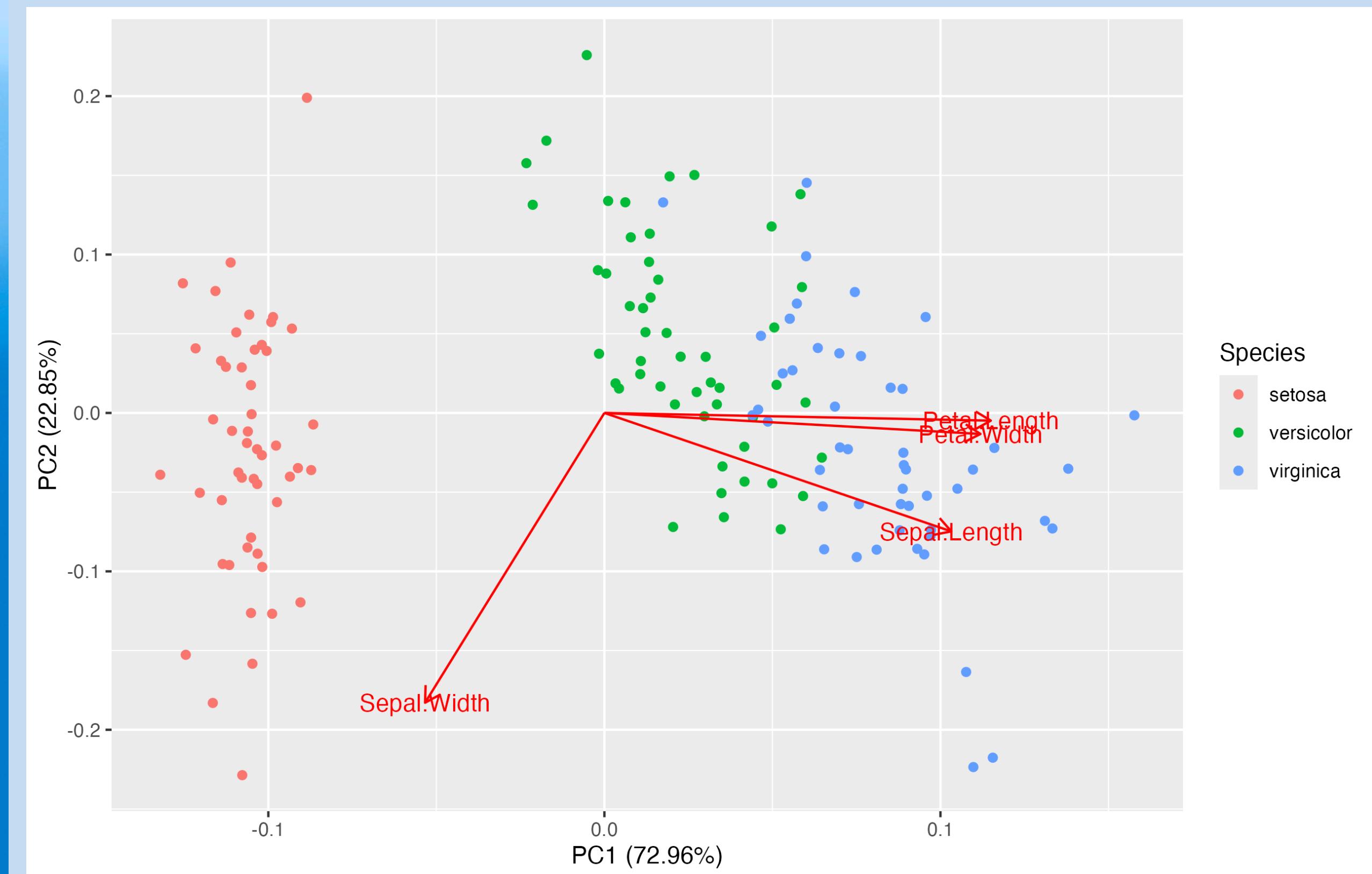
Good and simple guide to PCA in R: https://cran.r-project.org/web/packages/ggfortify/vignettes/plot_pca.html

Biplot: PCA plot with loadings

- Loadings: Arrows represent the direction and magnitude of each variable's contribution to the PC's.
 - The length of the arrow indicates the strength of the variable's contribution.
 - The direction of the arrow shows how the variables correlates with each PC.

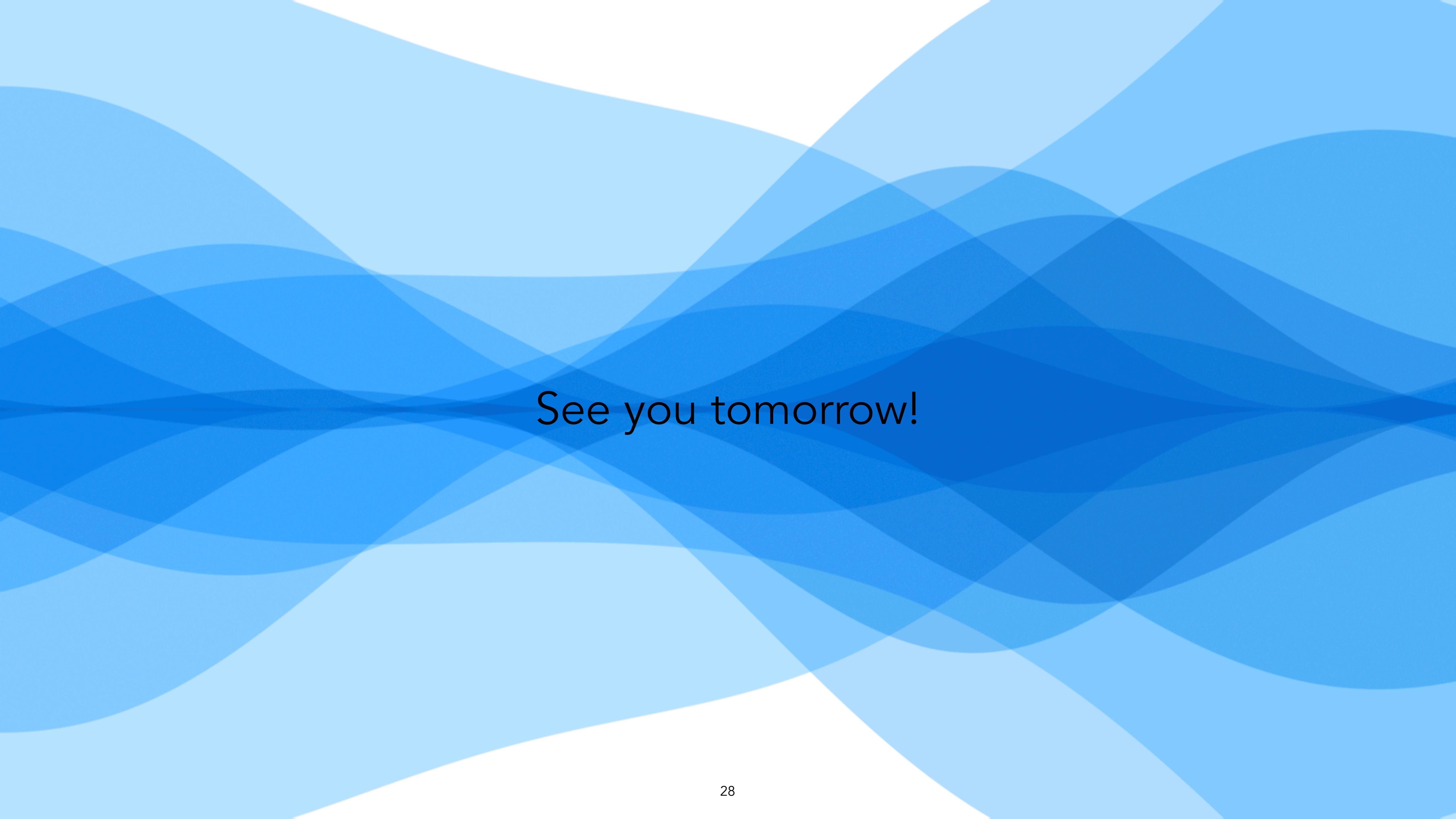
	PC1	PC2
Sepal Length	+	-
Sepal Width	-	-
Petal Length	+	-
Petal Width	+	-

```
autoplot(pca_res, data = iris, color = "Species",  
loadings = TRUE, loadings.label = TRUE)
```



Exercise 3B

Exploratory Data Analysis

The background of the slide features a repeating pattern of stylized, rounded blue shapes that resemble waves or hills. These shapes are rendered in various shades of blue, from light lavender to medium blue, creating a sense of depth and motion. The overall effect is clean and modern, typical of a professional presentation.

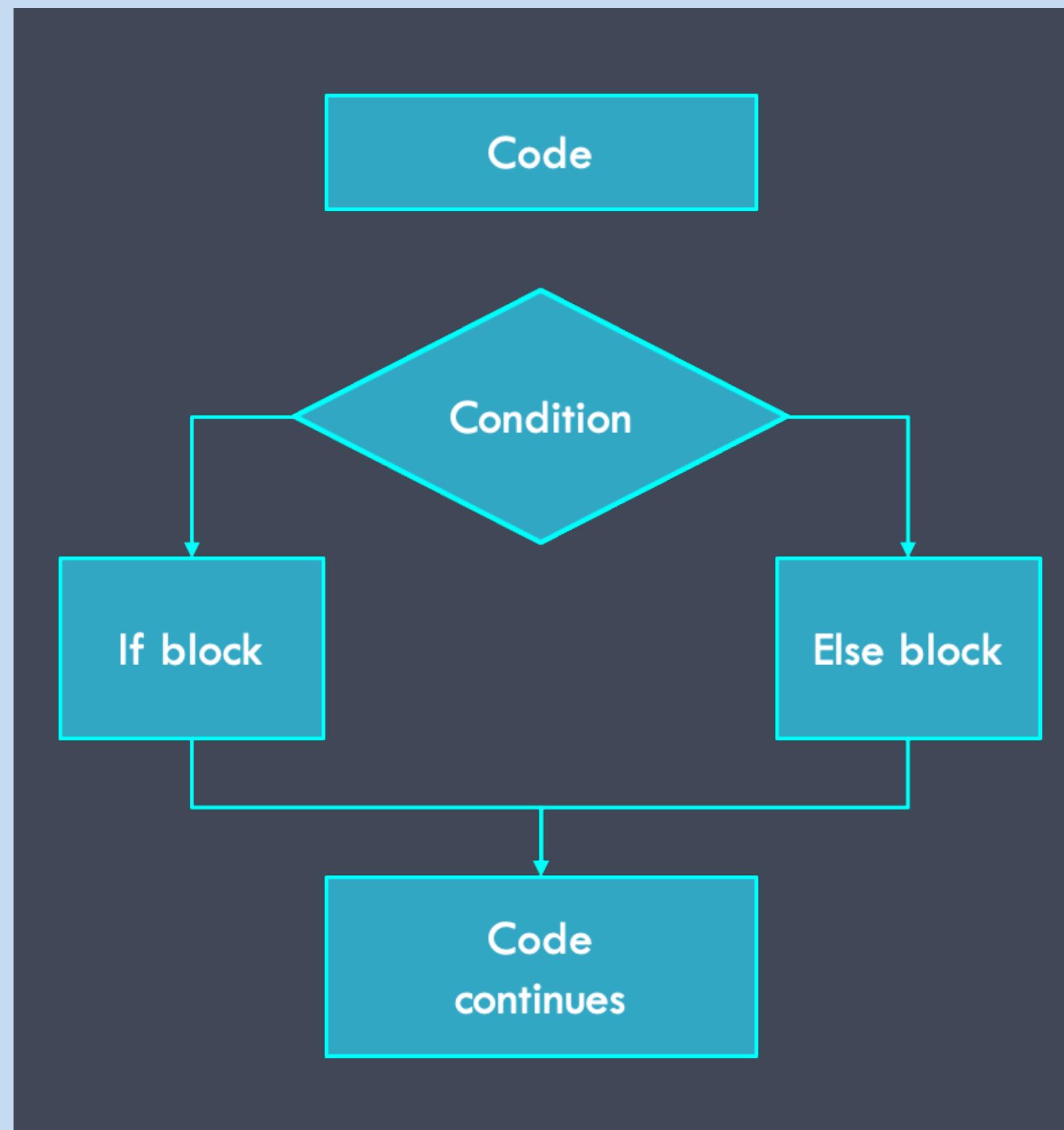
See you tomorrow!

Part 4

Scripting in R

Reusable and functional code

If-else statements



Symbol	Meaning
> , <	Greater, smaller
==	Equal to
!	Not
%in%	Occurs in
is.[type]	Check variable type
&	And
	Or

Conditions can be complex:
`if (age > 30 & !(smoker) | hospital == 'Herlev')`

If-else statements in R

- Conditional logic: Execute code blocks based on whether a condition is TRUE or FALSE.

R syntax

```
if (CONDITION_1) {  
    OPERATION_1  
} else if (CONDITION_2) {  
    OPERATION_2  
} else if (CONDITION_3) {  
    OPERATION_3  
} else {  
    OPERATION_4  
}
```

Common conditions

`var1 > var2`

`var1 < var2`

`var1 == var2`

`var1 != var2 # Not equal to`

`var1 - var2 == var3`

`is.numeric(var1), is.character(var1), ...`

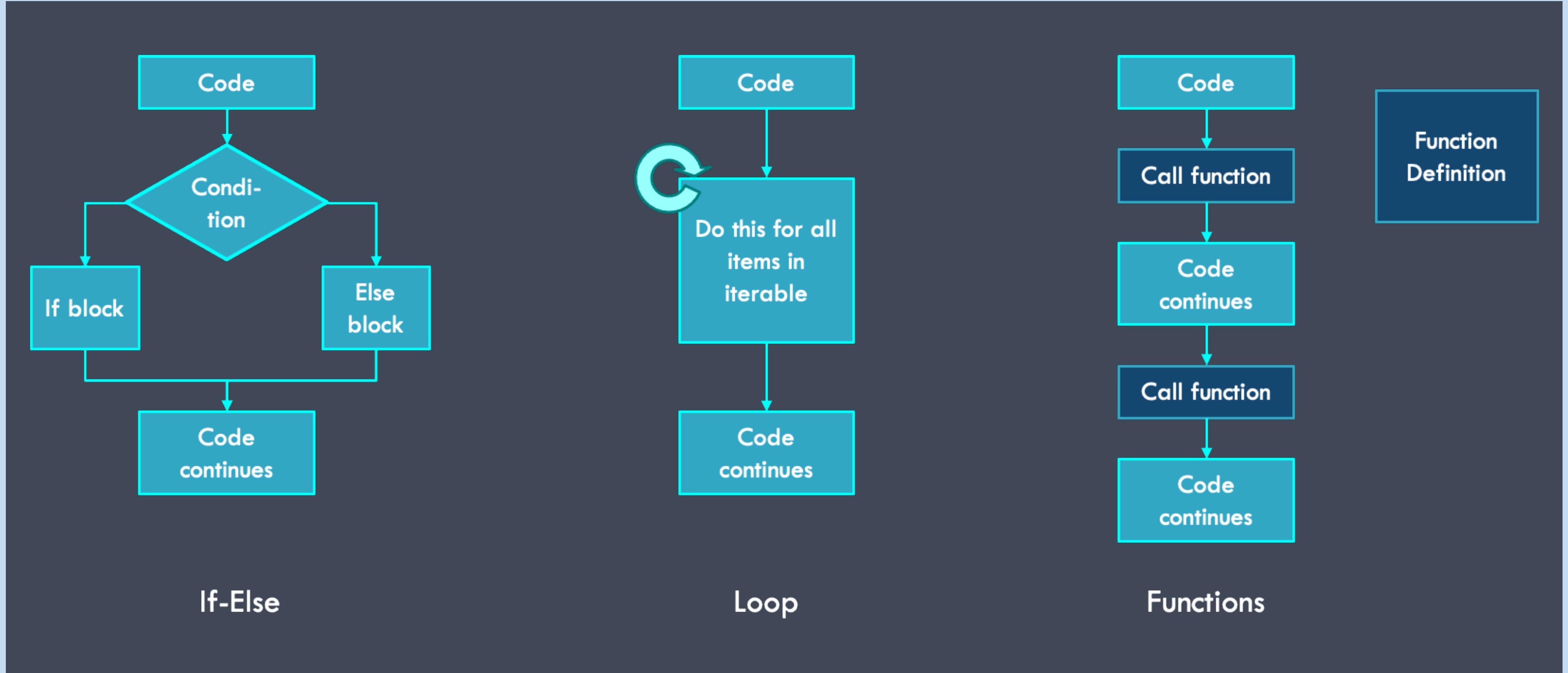
Combine conditions

`var1 > var2 & var1 < var3`

`var1 == var2 | var1 == var3`

`(var1 > var2 & var1 < var3) | (var1 == var2)`

If-else statements



For-loops in R

- Repeat a block of code for each item in a sequence.
- Iterating over objects.

```
```{r}
names <- c('Thilde', 'Diana', 'Henrike', 'Helene', 'Chelsea', 'Signe')

for (name in names){
 print(name)
}
````
```

```
[1] "Thilde"
[1] "Diana"
[1] "Henrike"
[1] "Helene"
[1] "Chelsea"
[1] "Signe"
```

```
```{r}
for (i in seq_along(names)){
 print(names[i])
}
````
```

```
[1] "Thilde"
[1] "Diana"
[1] "Henrike"
[1] "Helene"
[1] "Chelsea"
[1] "Signe"
```

R syntax

```
# var iterates over each item in list.

for (var in list) {

  OPERATION_USING_var

}
```

var name

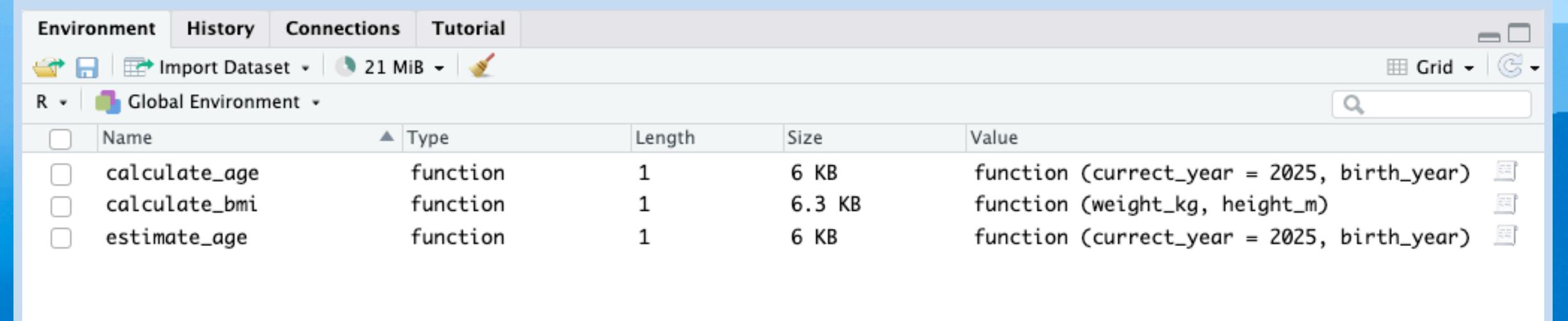
- Give the variable a meaningful name.
- i is often used when looping through indexes.

Functions

- Reusable: Write once, use many times.
- Readable: Makes code cleaner and easier to understand.
- Modular: Breaks down complex tasks into simple steps.
- Consistent: Endures uniform logic across your code.

Globale variables

- Defined outside functions.
- Accessible anywhere in the code.



| Name | Type | Length | Size | Value |
|---------------|----------|--------|--------|--|
| calculate_age | function | 1 | 6 KB | function (current_year = 2025, birth_year) { |
| calculate_bmi | function | 1 | 6.3 KB | function (weight_kg, height_m) { |
| estimate_age | function | 1 | 6 KB | function (current_year = 2025, birth_year) { |

Local variables

- Defined inside functions.
- Only accessible within the specific function.

Functions

Function

```
FUN_1 <- function(var1, var2){  
  result <- OPERATION_WITH_VARIABLES  
  return(result)  
}
```

Function with default variable values that can still be defined when the function is called.

```
FUN_2 <- function(var1 = “Hello”, var2 = 5){  
  result <- OPERATION_WITH_VARIABLES  
  return(result)  
}
```

Source script

```
source(functions.R)
```

Call function (spits out the result)

```
FUN_1(var1 = 2, var2 = 9)
```

```
FUN_2(var1 = “World”, var2 = 3)
```

Save result of function call

```
RESULT_1 <- FUN_1(var1 = 2, var2 = 9)
```

```
RESULT_2 <- FUN_2(var1 = “World”, var2 = 3)
```

Functions

Function

```
FUN_1 <- function(var1, var2){  
  result <- OPERATION_WITH_VARIABLES  
  return(result)  
}
```

Function with default variable values that can still be defined when the function is called.

```
FUN_2 <- function(var1 = "Hello", var2 = 5){  
  result <- OPERATION_WITH_VARIABLES  
  return(result)  
}
```

Source script

```
source(functions.R)
```

Call function (spits out the result)

```
FUN_1(var1 = 2, var2 = 9)
```

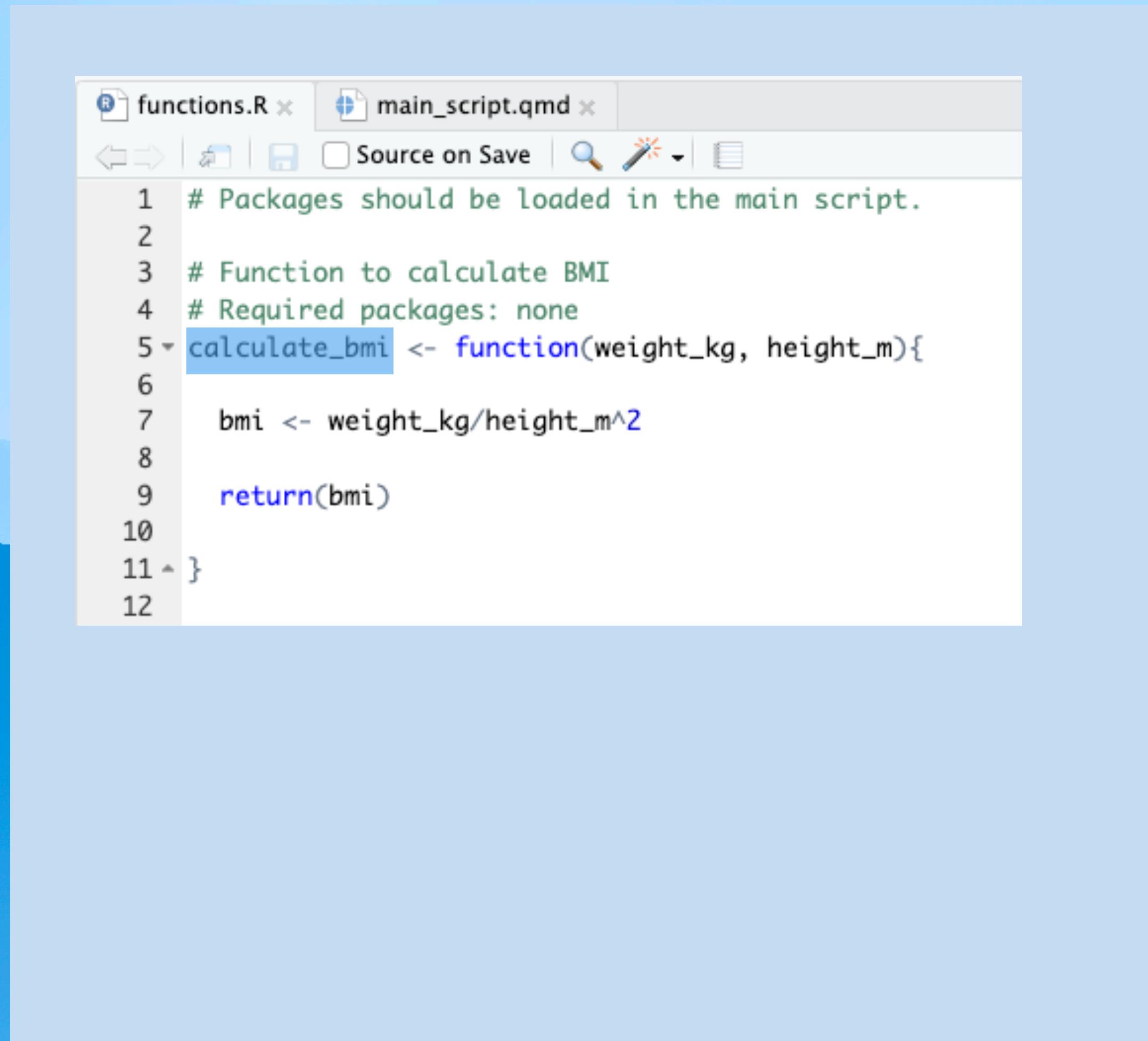
```
FUN_2(var1 = "World", var2 = 3)
```

Save result of function call

```
RESULT_1 <- FUN_1(var1 = 2, var2 = 9)
```

```
RESULT_2 <- FUN_2(var1 = "World", var2 = 3)
```

Functions



The screenshot shows the RStudio interface with two files open:

- functions.R**: Contains the following code:

```
1 # Packages should be loaded in the main script.
2
3 # Function to calculate BMI
4 # Required packages: none
5 calculate_bmi <- function(weight_kg, height_m){
6
7   bmi <- weight_kg/height_m^2
8
9   return(bmi)
10}
11
```
- main_script.qmd**: Contains the following code:

```
1 ---  
2 title: "Main Script"  
3 format: html  
4 ---  
5  
6 ## Load packages  
7  
8 ## Source script  
9 All functions in the functions.R script will be loaded and appear in the  
Global environment in the top left corner.  
10 ``{r}  
11 source('./functions.R')  
12 ``  
13  
14 ## Function calls  
15 Calculate BMI.  
16 ``{r}  
17 calculate_bmi(weight_kg = 70, height_m = 1.80)  
18 ``  
19  
20 ``{r}  
21 calculate_bmi(70, 1.80)  
22 ``  
23  
24 ``{r}  
25 calculate_bmi(1.80, 70)  
26 ``  
27  
28 ``{r}  
29 calculate_bmi(height_m = 1.80, weight_kg = 70)  
30 ``  
31
```



The screenshot shows the RStudio interface with the **main_script.qmd** file open, displaying the following output from the R console:

```
[1] 21.60494
[1] 21.60494
[1] 0.0003673469
[1] 21.60494
```

Functions

The screenshot shows the RStudio interface with two files open:

- functions.R**: Contains R code defining two functions: `calculate_bmi` and `estimate_age`.
- main_script.qmd**: A Quarto document that includes the `functions.R` file and demonstrates its use.

```
functions.R
1 # Packages should be loaded in the main script.
2
3 # Function to calculate BMI
4 # Required packages: none
5 calculate_bmi <- function(weight_kg, height_m){
6
7   bmi <- weight_kg/height_m^2
8
9   return(bmi)
10
11 }
12
13 # Function to estimate age
14 # Required packages: none
15 estimate_age <- function(correct_year = 2025, birth_year){
16
17   age <- correct_year - birth_year
18
19   return(age)
20
21 }
22
```

The screenshot shows the RStudio interface with the `main_script.qmd` file open, displaying the rendered output:

```
main_script.qmd
1 --- title: "Main Script"
2 format: html
3 ---
4 ## Load packages
5
6 ## Source script
7 ````{r}
8 source('./functions.R')
9 ````

12
13 Calculate BMI.
14 ````{r}
15 calculate_bmi(weight_kg = 70, height_m = 1.80)
16 ````

[1] 21.60494

17
18 Estimate age in 2025 when born in 1997.
19 ````{r}
20 estimate_age(birth_year = 1997)
21 ````

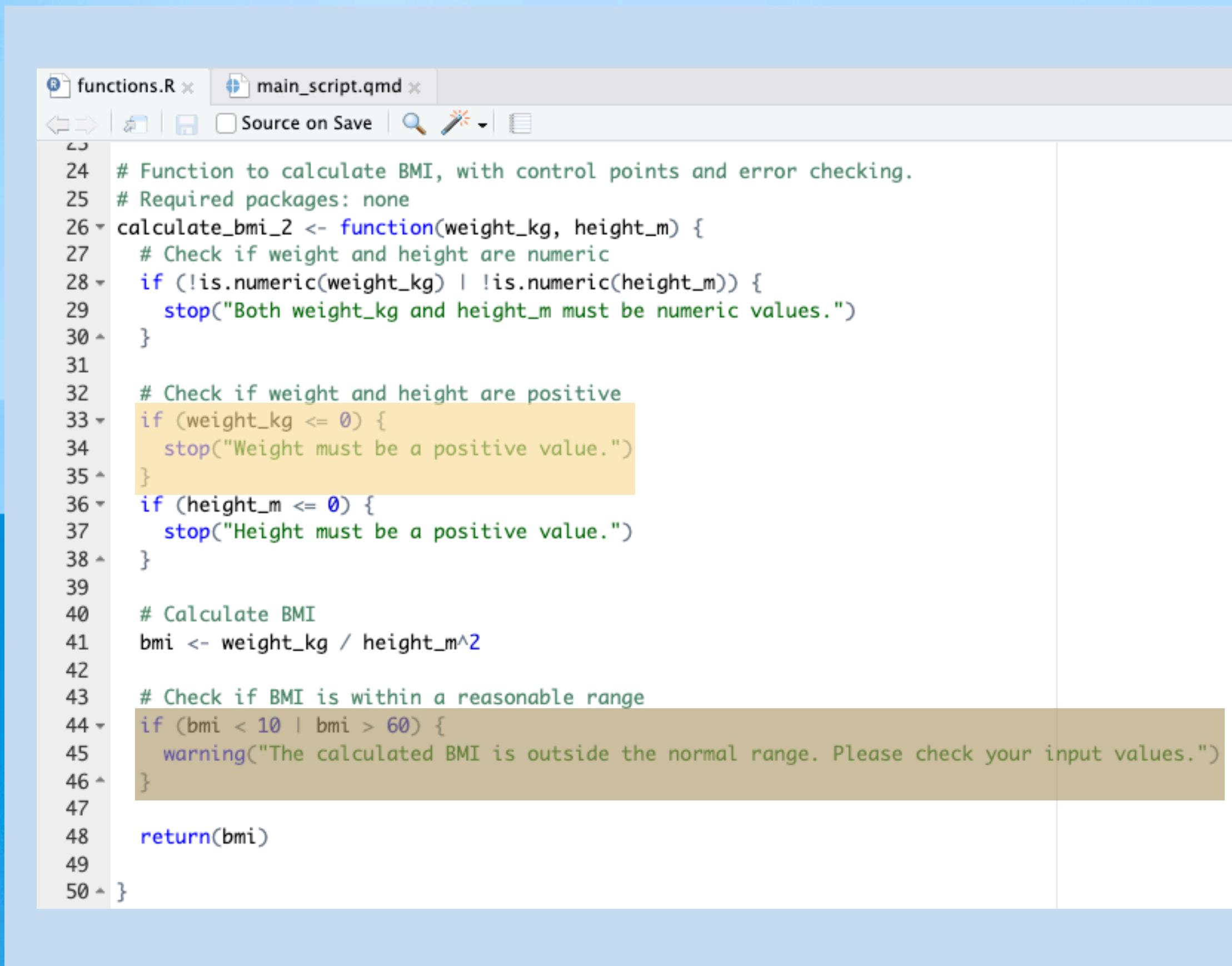
[1] 28

22
23 Estimate age in 2005 when born in 1997.
24 ````{r}
25 estimate_age(correct_year = 2005, birth_year = 1997)
26 ````

[1] 8

27
28
```

Functions and error handling



```
functions.R x main_script.qmd x
Source on Save | Source | ABC | Render on Save | Render | Help

24 # Function to calculate BMI, with control points and error checking.
25 # Required packages: none
26 calculate_bmi_2 <- function(weight_kg, height_m) {
27   # Check if weight and height are numeric
28   if (!is.numeric(weight_kg) | !is.numeric(height_m)) {
29     stop("Both weight_kg and height_m must be numeric values.")
30   }
31
32   # Check if weight and height are positive
33   if (weight_kg <= 0) {
34     stop("Weight must be a positive value.")
35   }
36   if (height_m <= 0) {
37     stop("Height must be a positive value.")
38   }
39
40   # Calculate BMI
41   bmi <- weight_kg / height_m^2
42
43   # Check if BMI is within a reasonable range
44   if (bmi < 10 | bmi > 60) {
45     warning("The calculated BMI is outside the normal range. Please check your input values.")
46   }
47
48   return(bmi)
49
50 }
```



```
functions.R x main_script.qmd x
Source Visual
Render on Save | ABC | Render | Help

127 The BMI function with out error handling returns a meaningless BMI value if given a negative weight.
128 ````{r}
129 calculate_bmi(weight_kg = -50, height_m = 1.80)
130 ````

131 [1] -15.4321

132 The BMI function with error handling returns an error if given a negative weight.
133 ````{r}
134 calculate_bmi_2(weight_kg = -50, height_m = 1.80)
135 ````

136 Error in calculate_bmi_2(weight_kg = -50, height_m = 1.8) :
137   Weight must be a positive value.

138 ````{r}
139 calculate_bmi_2(weight_kg = 25, height_m = 1.80)
140 ````

141 Warning in calculate_bmi_2(weight_kg = 25, height_m = 1.8) :
142   The calculated BMI is outside the normal range. Please check your input values.
[1] 7.716049

143
144
```

Apply functions across multiple arguments

- Running function across multiple arguments.
- Alternative calling function in for-loop.

R syntax

```
# For function with single argument, return list  
lapply(FUN, var_list)  
  
# For function with single argument, return simplified output (vector)  
sapply(FUN, var_list)  
  
# For function with multiple arguments  
mapply(FUN,  
        var1 = var1_list,  
        var2 = var2_list)
```

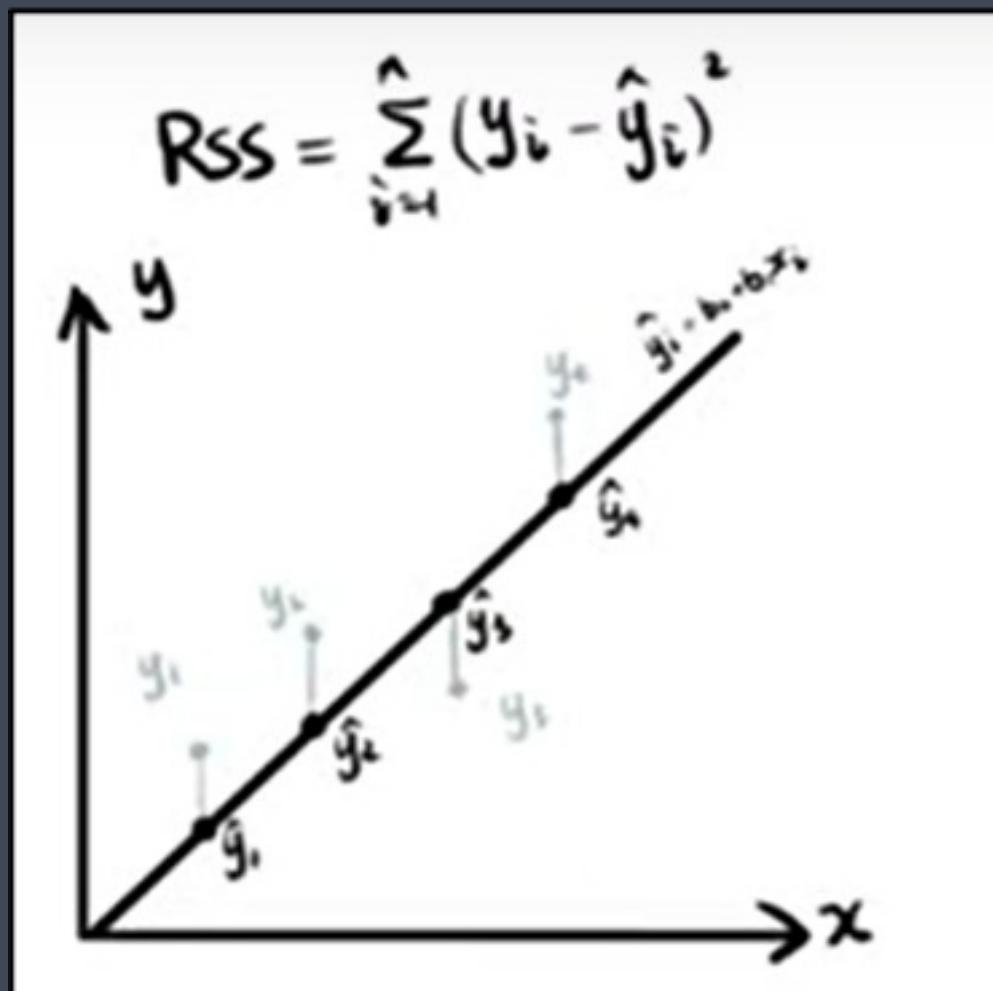
Exercise 4

Scripting in R

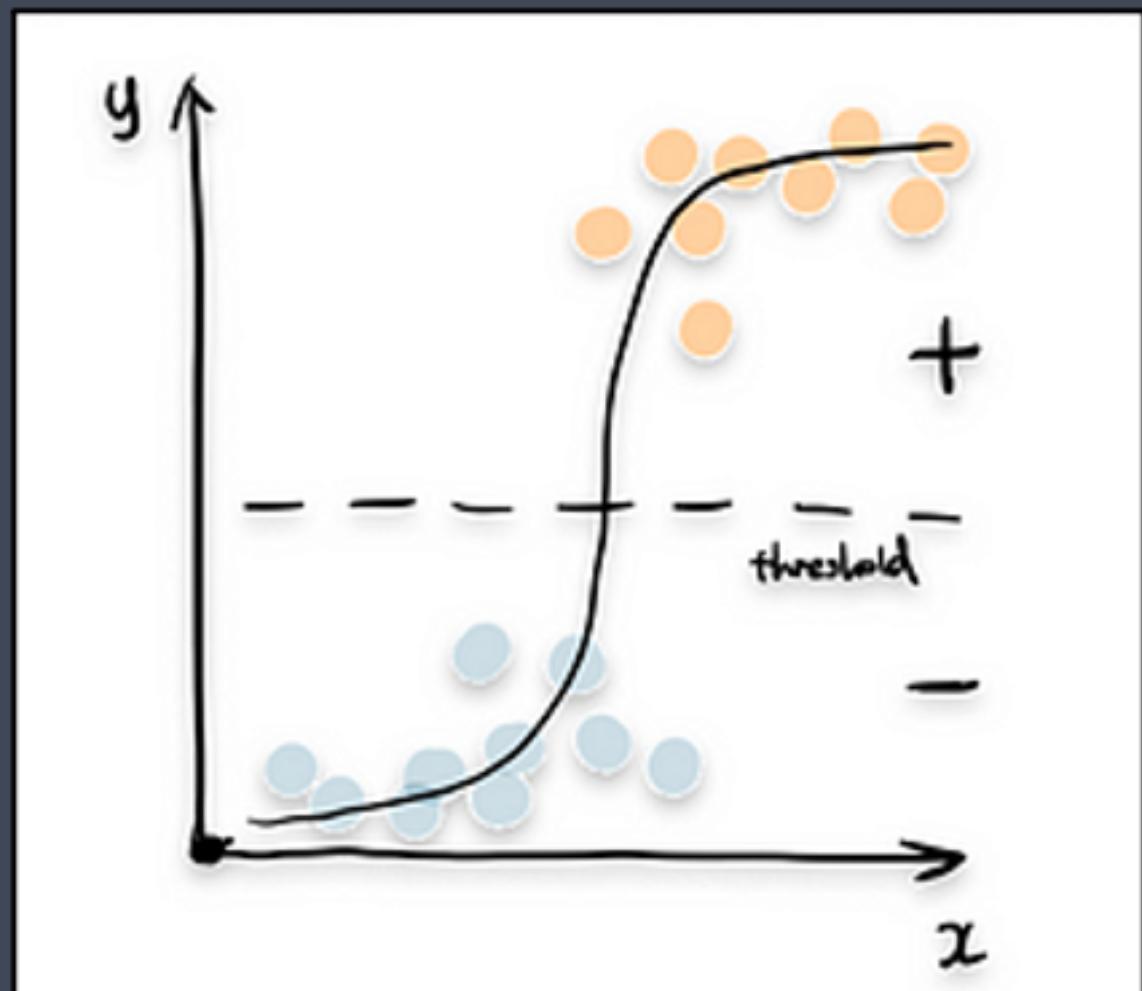
Part 5

Modelling in R

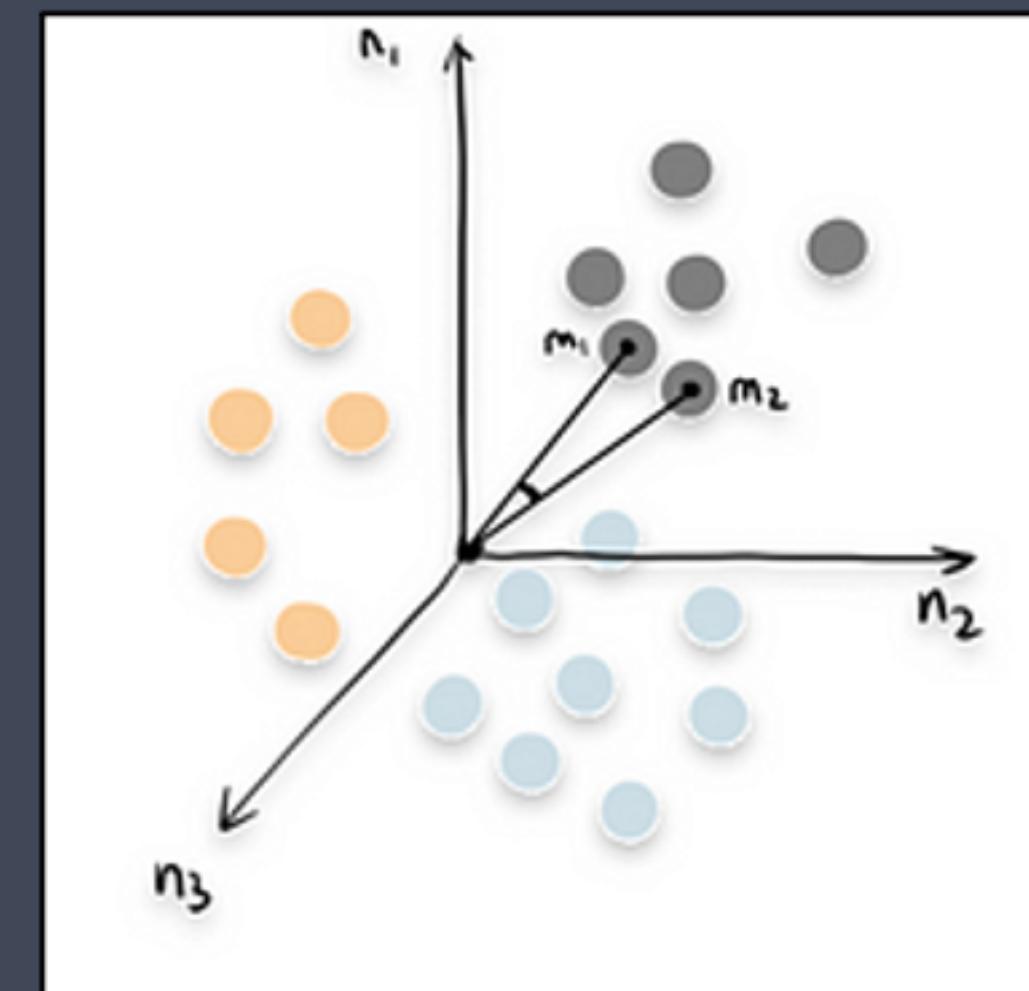
Types of models



Regression



Classification



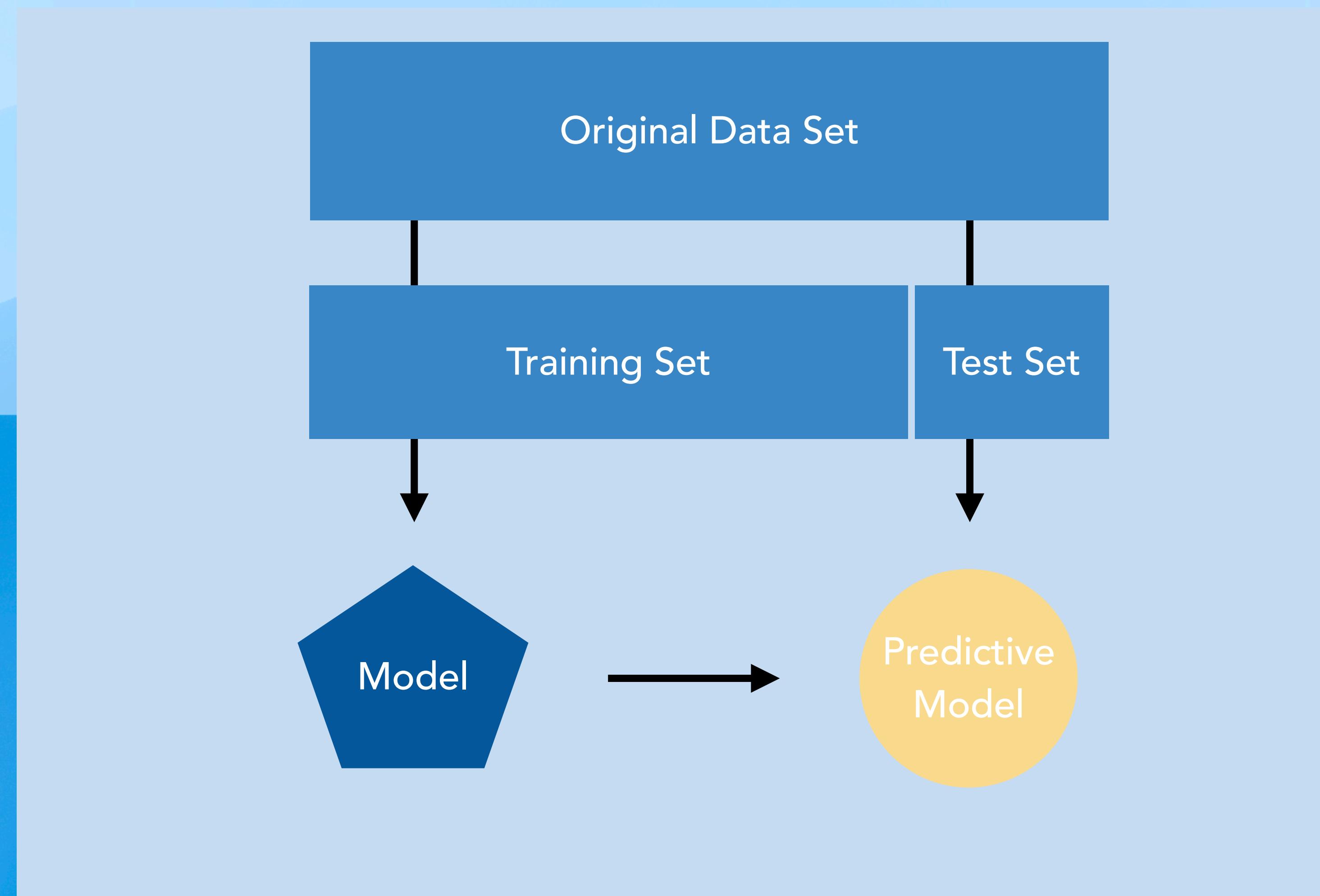
Clustering

Bioconductor

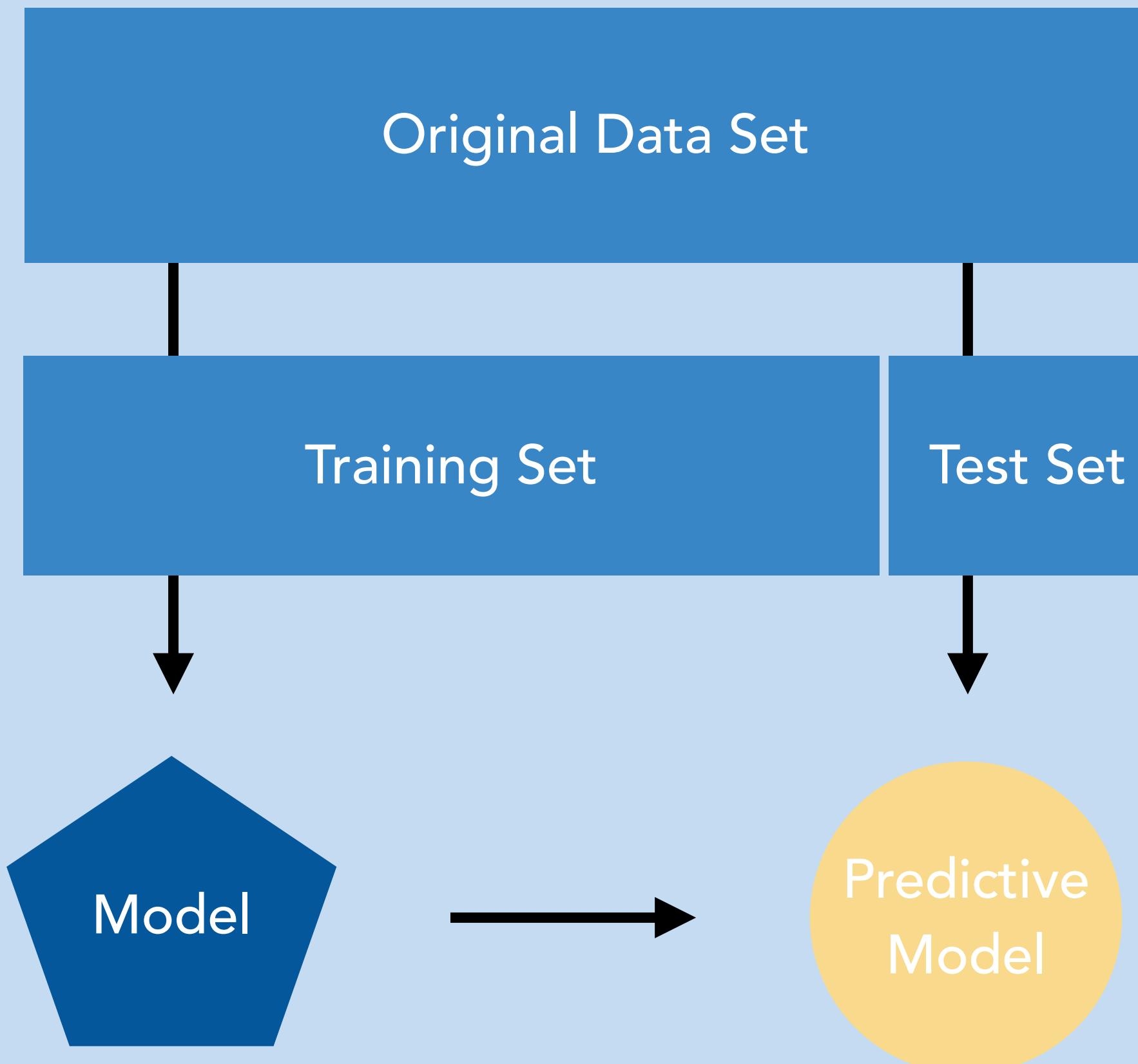
- Community-driven, open-source R packages for bioinformatics
- Well-documented, many tutorials, support forum
- RNA-seq (bulk and single-cell), annotation, epigenetic, proteomics, ect.
- BiocManager in the install wrapper for Bioconductor: `biocmanager::install("package")`



Train and test data set



Train and test data set



```
data <- read_csv("data/data.csv")
```

```
library(caret)
split <- createDataPartition(data$mpg, p = 0.8, list = FALSE)
train_data <- data[split, ]
test_data <- data[-split, ]
```