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**	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		13:00-16:00	**Project work**
14:15-14:30	**Break**				
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

Quarto

Render to html

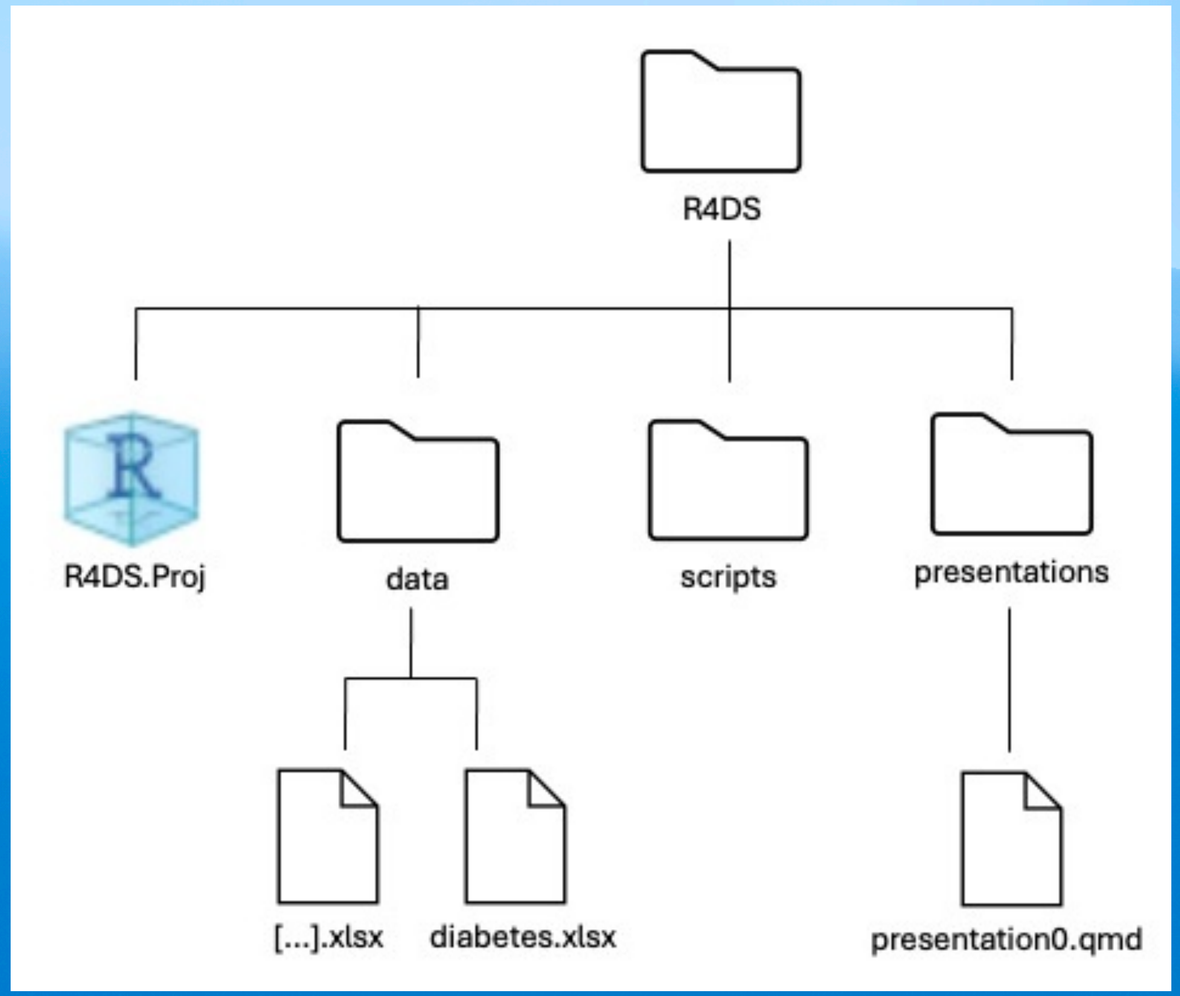
R project

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

Cheat Sheet - String manipulation

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

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

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

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 an 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)

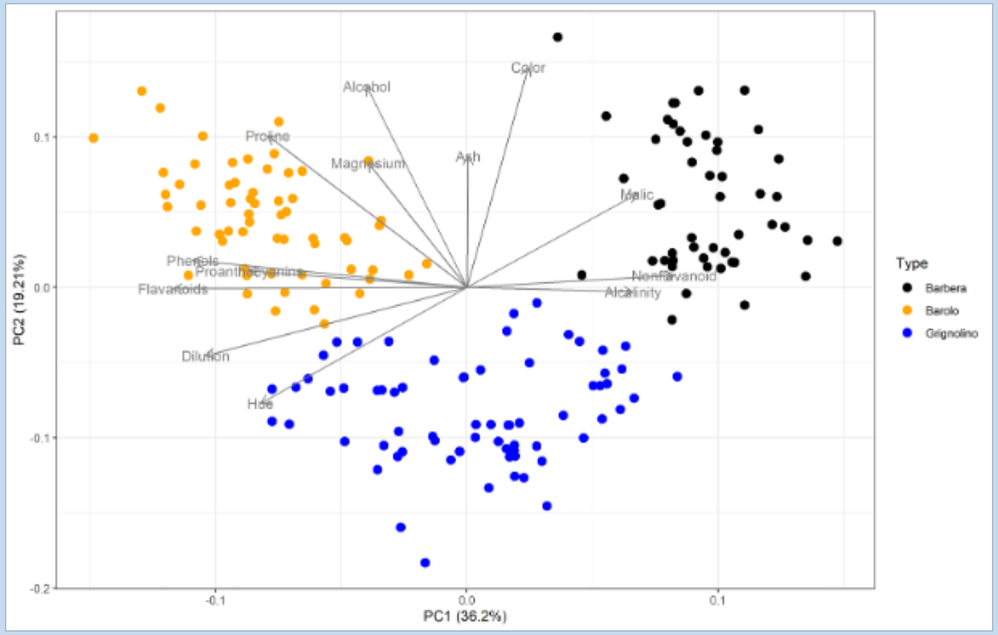
- Linear dimensionality reduction: High dimensional space → Low dimensional space
- Find new dimensions that capture as much variation as possible
- Shows structure of the data across all variables

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	…
…	…	…	…	…	…

→

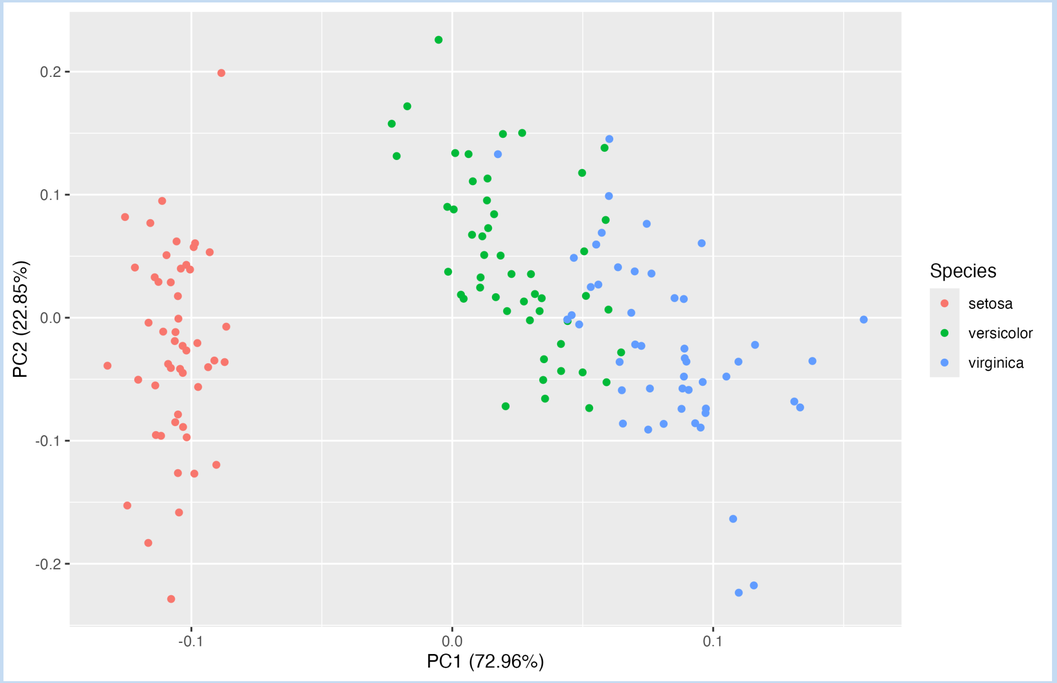
178 wines x 2 features



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
```

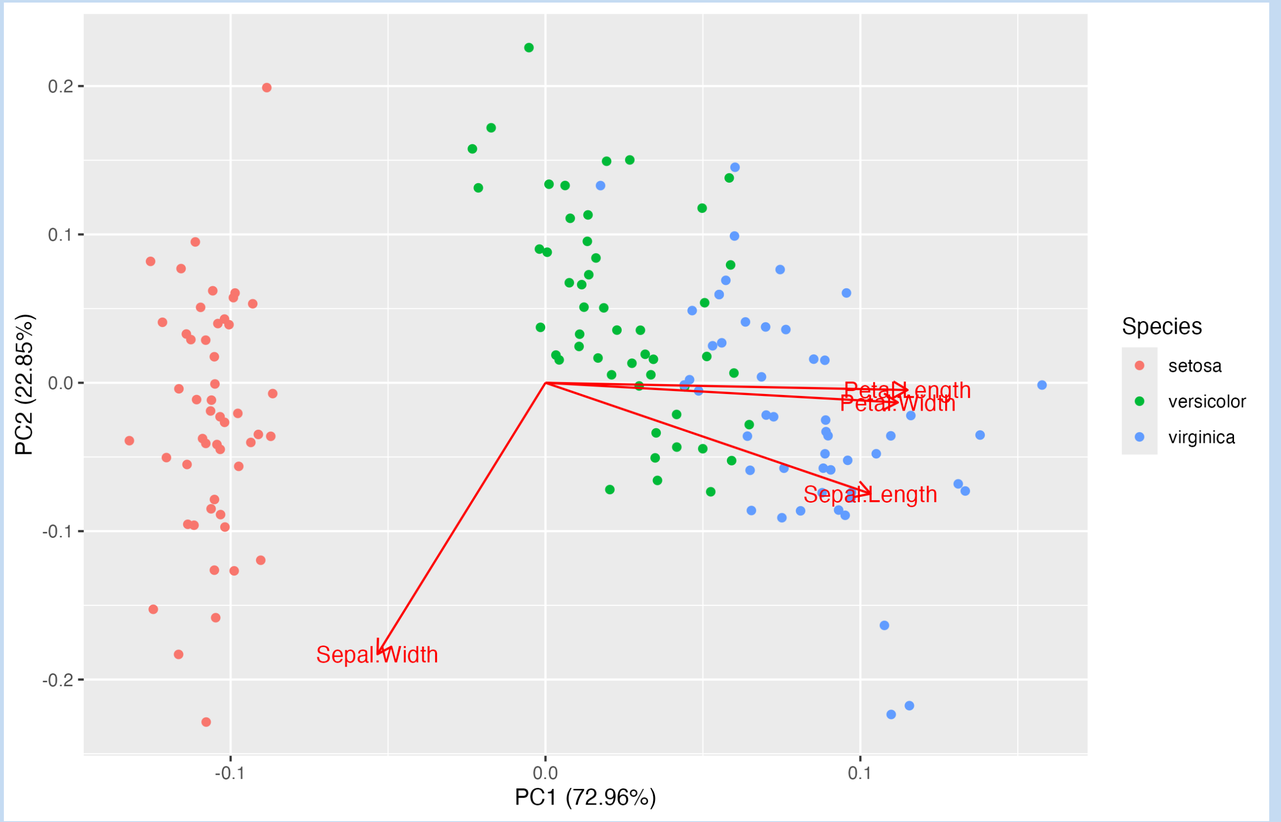
Good and simple guide to PCA in R: https://cran.r-project.org/web/packages/ggfortify/vignettes/plot_pca.html

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

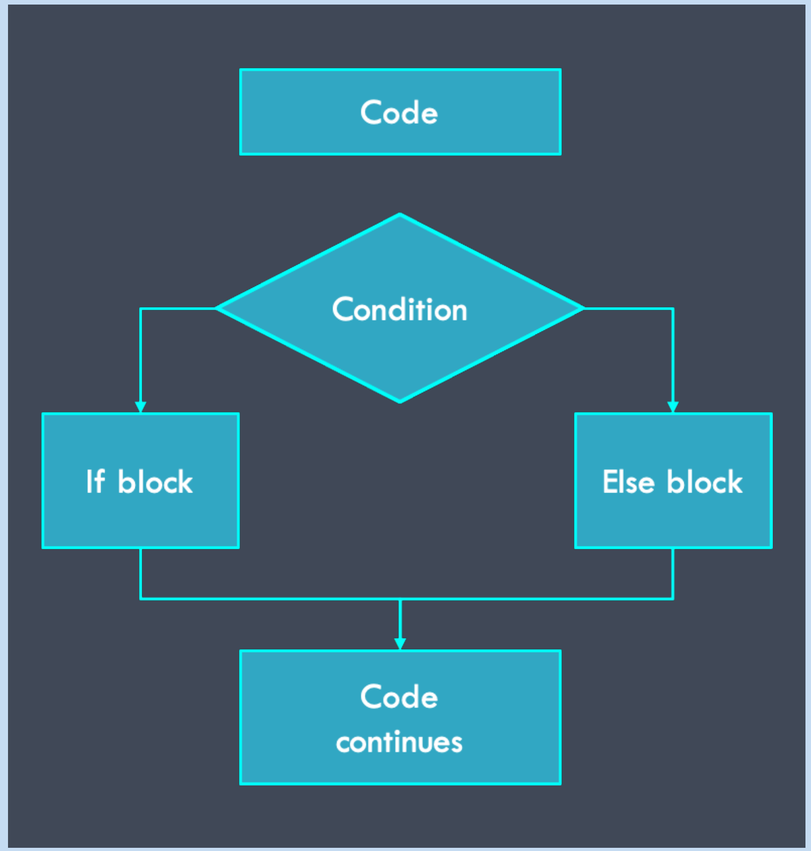
See you tomorrow!

Part 4

Scripting in R

Reuseable 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
}
```

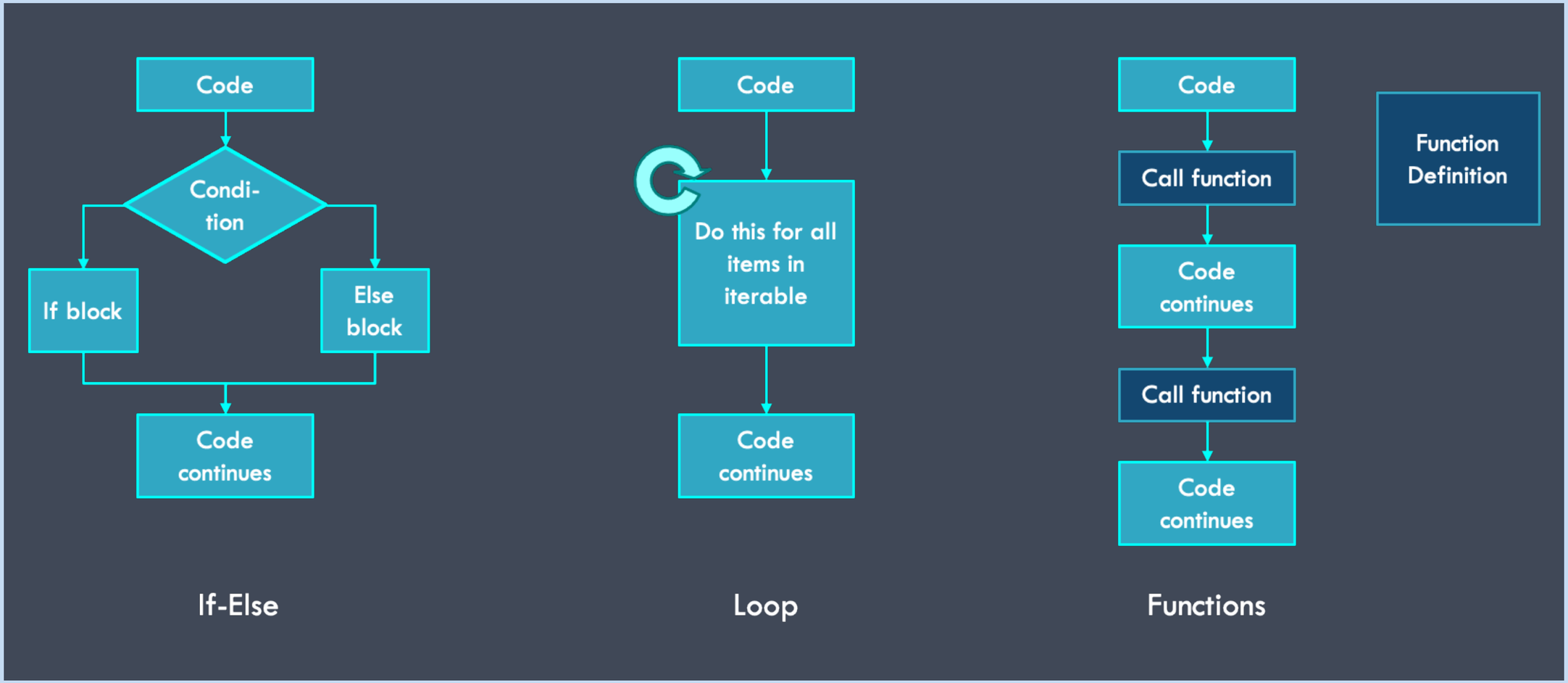
Common conditions

```
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

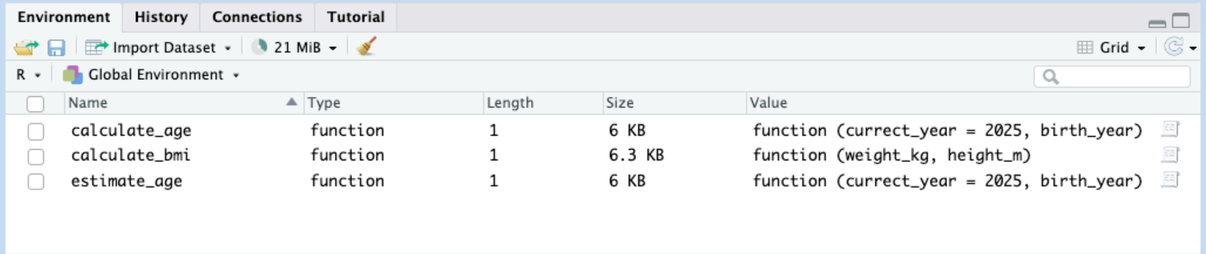
- Give the variable a meaningful name.
- `i` is often used when looping through indexes.

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.



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)
```

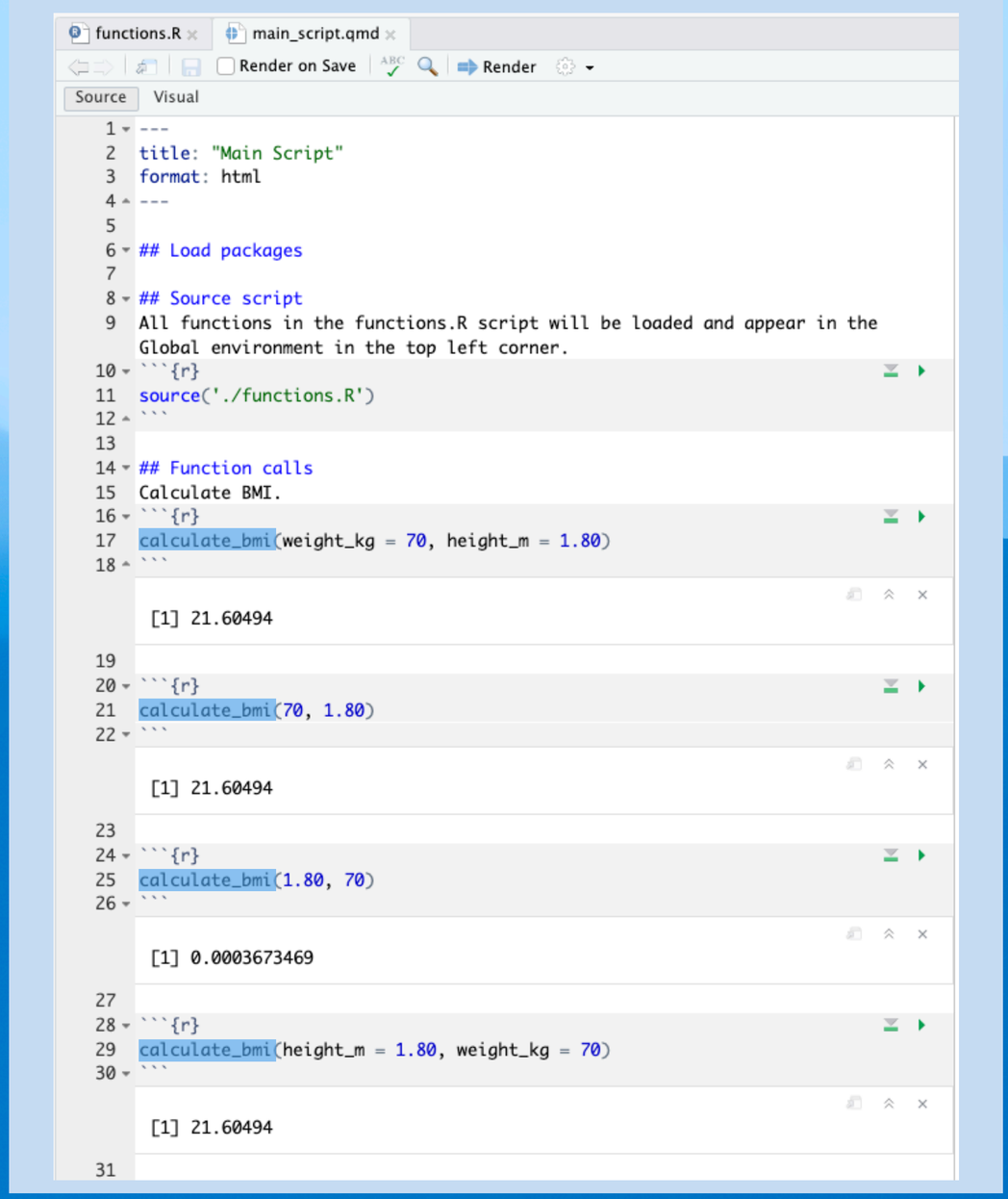
```
RESULT_2 <- FUN_2(var1 = “World”, var2 = 3)
```

Functions

functions.R | main_script.qmd

Source on Save

```r
# Packages should be loaded in the main script.

# Function to calculate BMI
# Required packages: none
calculate_bmi <- function(weight_kg, height_m){

  bmi <- weight_kg/height_m^2

  return(bmi)

}

```



Functions

functions.R | main_script.qmd

Source on Save

```r
# Packages should be loaded in the main script.

# Function to calculate BMI
# Required packages: none
calculate_bmi <- function(weight_kg, height_m){

  bmi <- weight_kg/height_m^2

  return(bmi)

}

# Function to estimate age
# Required packages: none
estimate_age <- function(currect_year = 2025, birth_year){

  age <- currect_year - birth_year

  return(age)

}

```

functions.R | main_script.qmd

Render on Save | Render

Source | Visual

````
---
title: "Main Script"
format: html
---

## Load packages

## Source script
```{r}
source('./functions.R')
```

Calculate BMI.
```{r}
calculate_bmi(weight_kg = 70, height_m = 1.80)
```

[1] 21.60494

Estimate age in 2025 when born in 1997.
```{r}
estimate_age(birth_year = 1997)
```

[1] 28

Estimate age in 2005 when born in 1997.
```{r}
estimate_age(currect_year = 2005, birth_year = 1997)
```

[1] 8
````

Functions and error handling

functions.R | main_script.qmd

Source on Save

```r
# Function to calculate BMI, with control points and error checking.
# Required packages: none
calculate_bmi_2 <- function(weight_kg, height_m) {
  # Check if weight and height are numeric
  if (!is.numeric(weight_kg) | !is.numeric(height_m)) {
    stop("Both weight_kg and height_m must be numeric values.")
  }

  # Check if weight and height are positive
  if (weight_kg <= 0) {
    stop("Weight must be a positive value.")
  }
  if (height_m <= 0) {
    stop("Height must be a positive value.")
  }

  # Calculate BMI
  bmi <- weight_kg / height_m^2

  # Check if BMI is within a reasonable range
  if (bmi < 10 | bmi > 60) {
    warning("The calculated BMI is outside the normal range. Please check your input values.")
  }

  return(bmi)

}
```

functions.R | main_script.qmd

Render on Save | Render

Source | Visual

The BMI function with out error handling returns a meaningless BMI value if given a negative weight.

```r
calculate_bmi(weight_kg = -50, height_m = 1.80)
```

```
[1] -15.4321
```

The BMI function with error handling returns an error if given a negative weight.

```r
calculate_bmi_2(weight_kg = -50, height_m = 1.80)
```

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

The BMI function with error handling returns an warning if a BMI outside the normal range is calculated.

```r
calculate_bmi_2(weight_kg = 25, height_m = 1.80)
```

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

Apply functions across multiple arguments

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

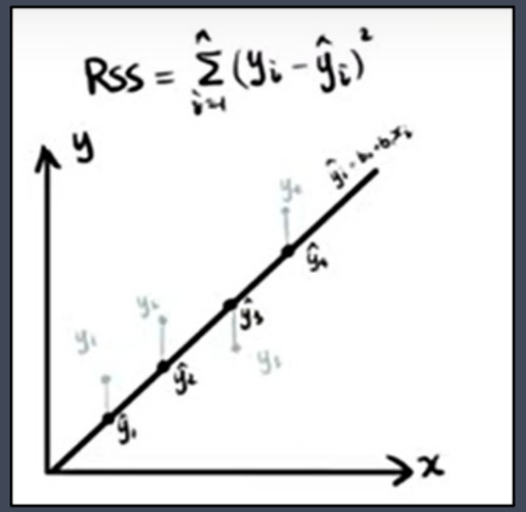
R syntax

```
# 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)
```

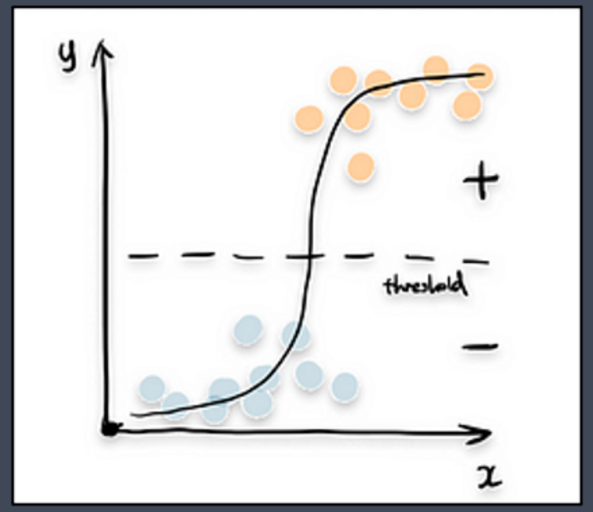
Exercise 4
Scripting in R

Part 5
Modelling in R

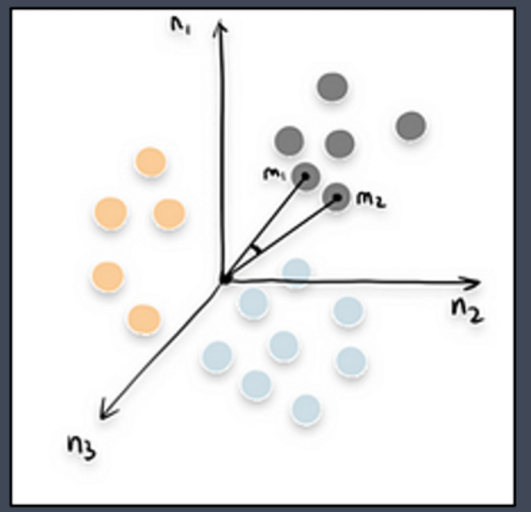
Types of models



Regression



Classification

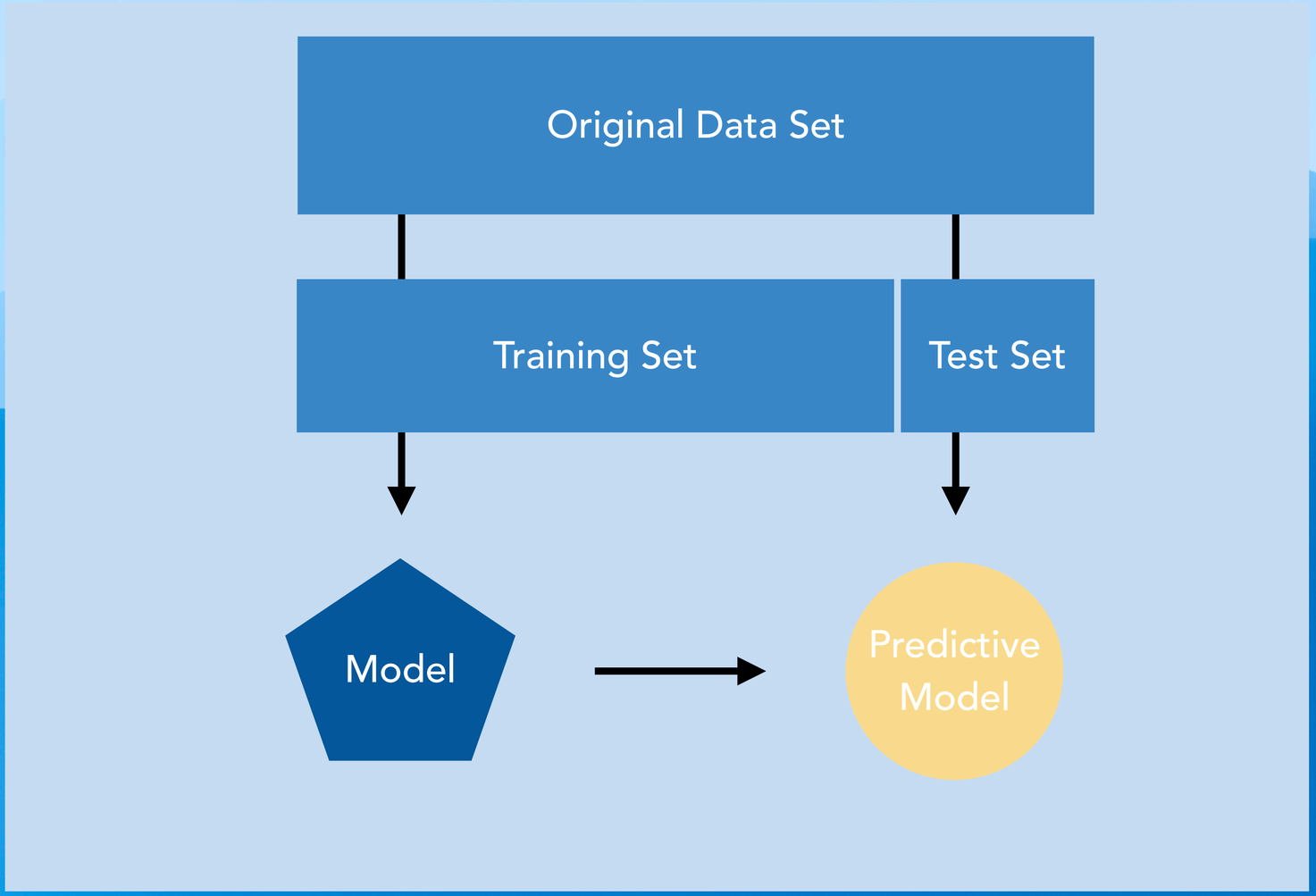


Clustering

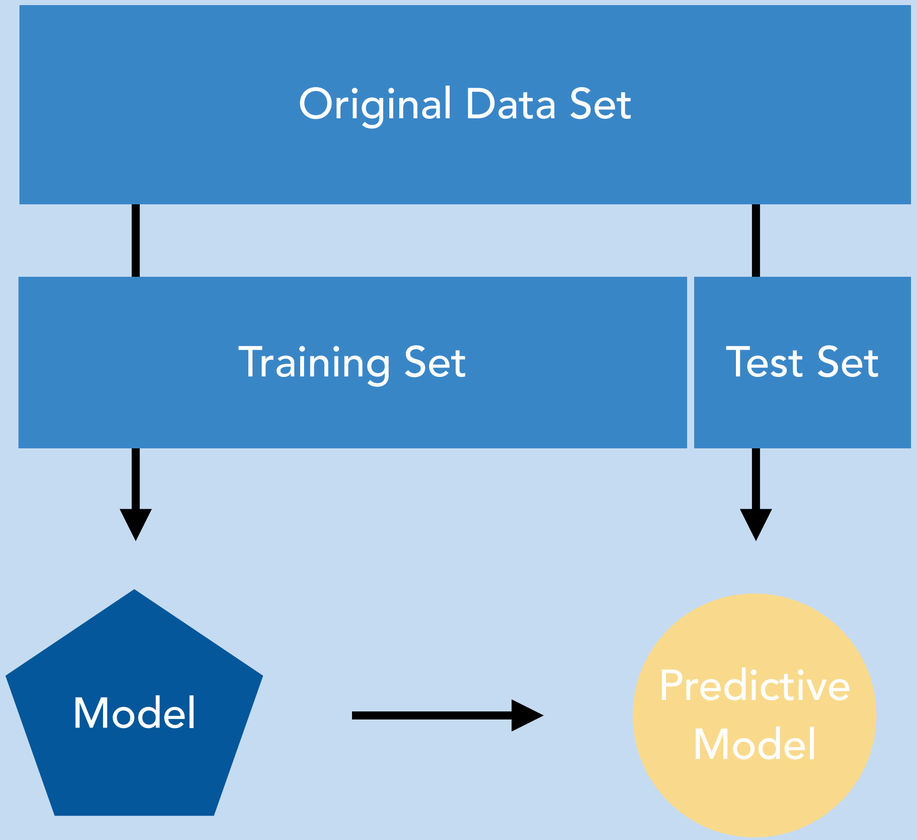
Bioconductor

- 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: `biocmanagaer::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, ]
```