Amsterdam University of Applied Sciences | DIGITAL SOCIETY SCHOOL

DATA-DRIVEN TRANSFORMATION

R

FIRST CONTACT

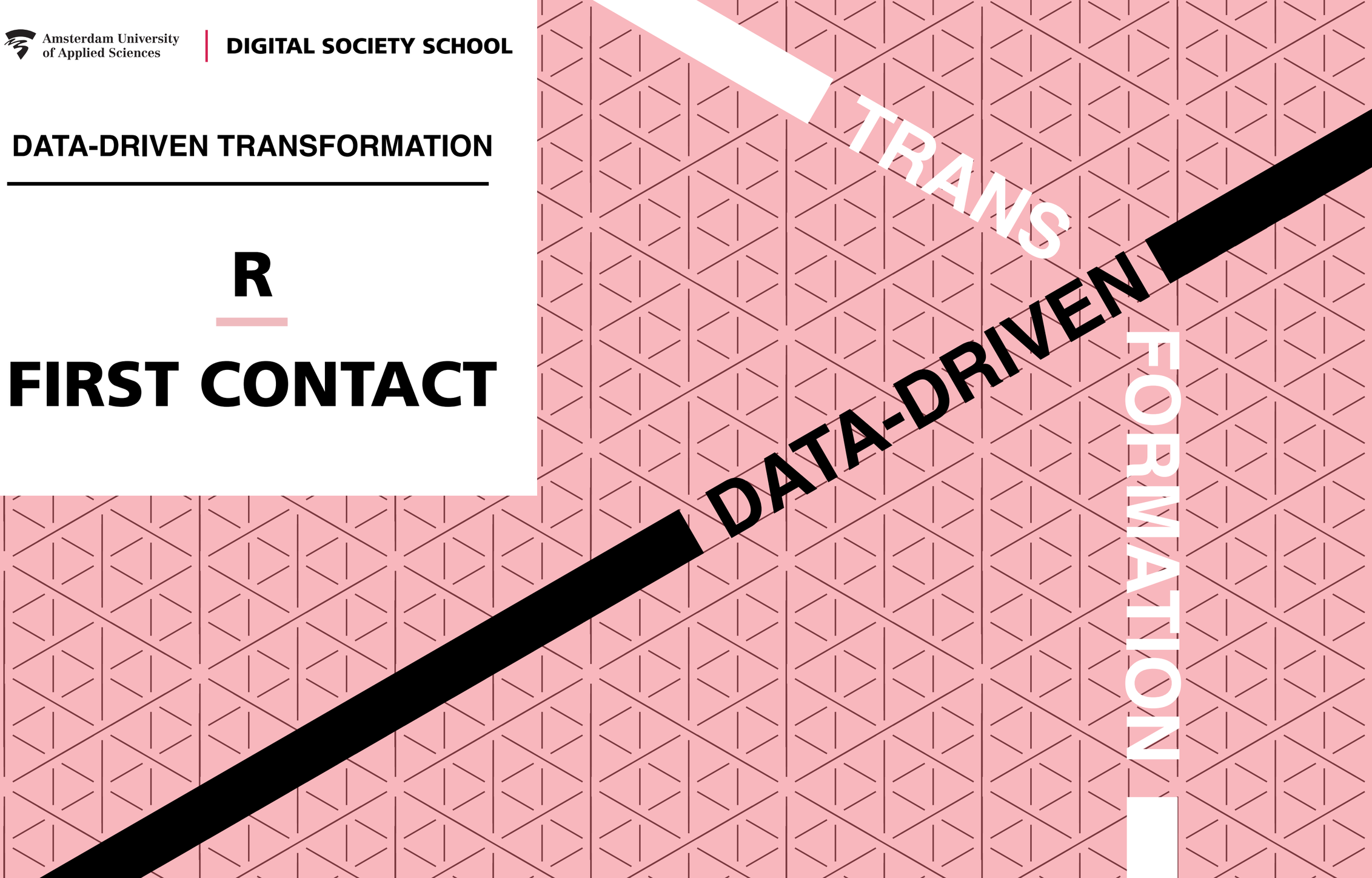


TABLE OF CONTENT

- **PROGRAMMING WITH R P.3**
- **READING & WRITING DATA P.28**
- **MANIPULATING DATA P.34**

PROGRAMMING WITH R

- R STUDIO P.4
- VARIABLE P.7
- DATA TYPES P.9
- IF - THEN - ELSE P.12
- FUNCTION P.16
- PIPE OPERATOR P.19
- LOOP P.22
- LIBRARIES P.25
- TIPS P.27

R STUDIO

R is a **programming language**

R Studio is an **environment** for programming, debugging, and **executing R**

First install R

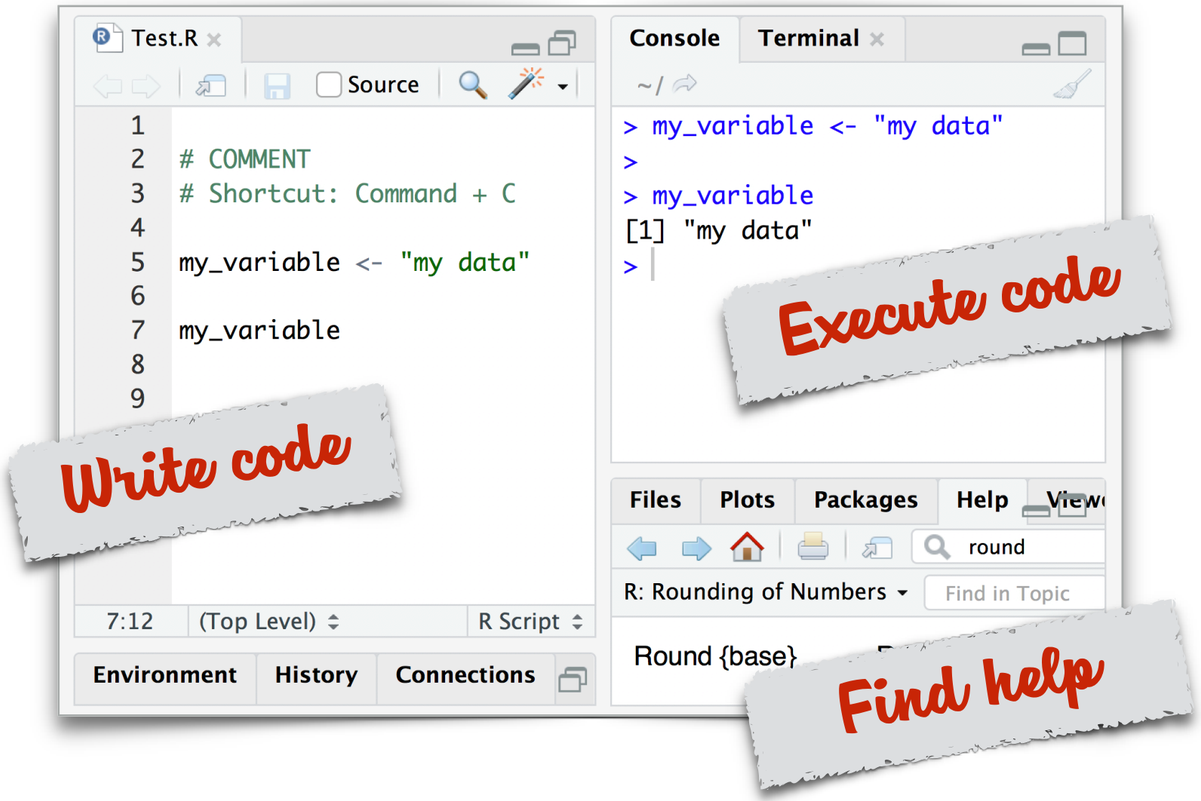
https://cran.r-project.org/

Then install R Studio

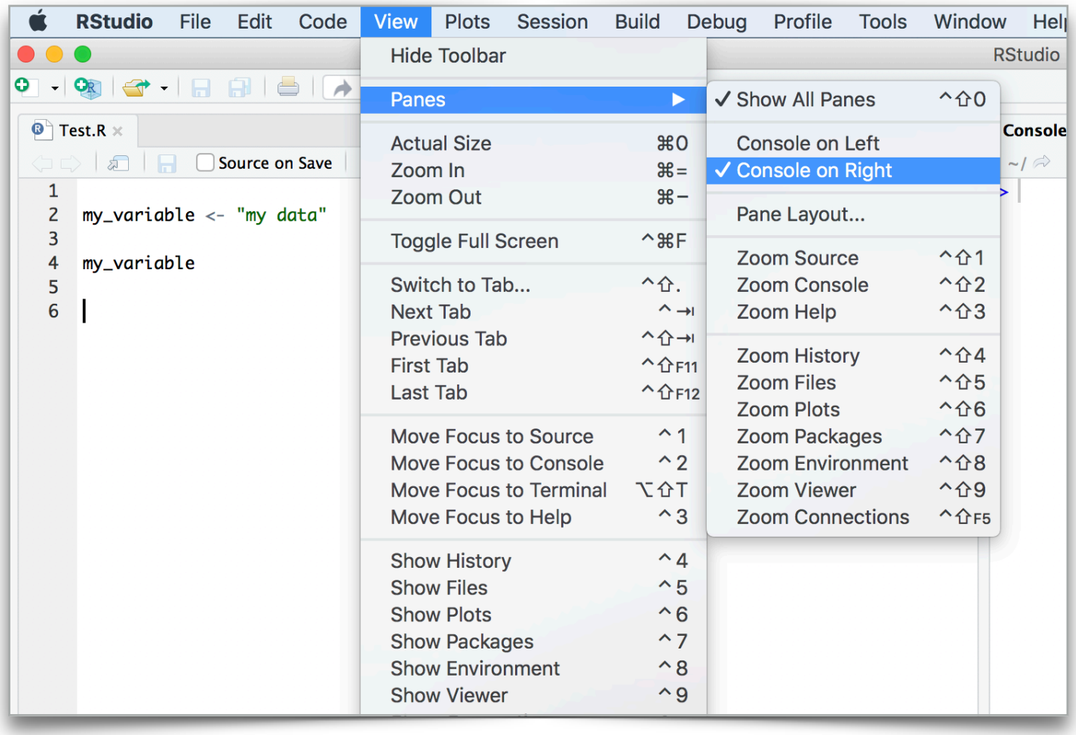
https://www.rstudio.com/products/rstudio/download/

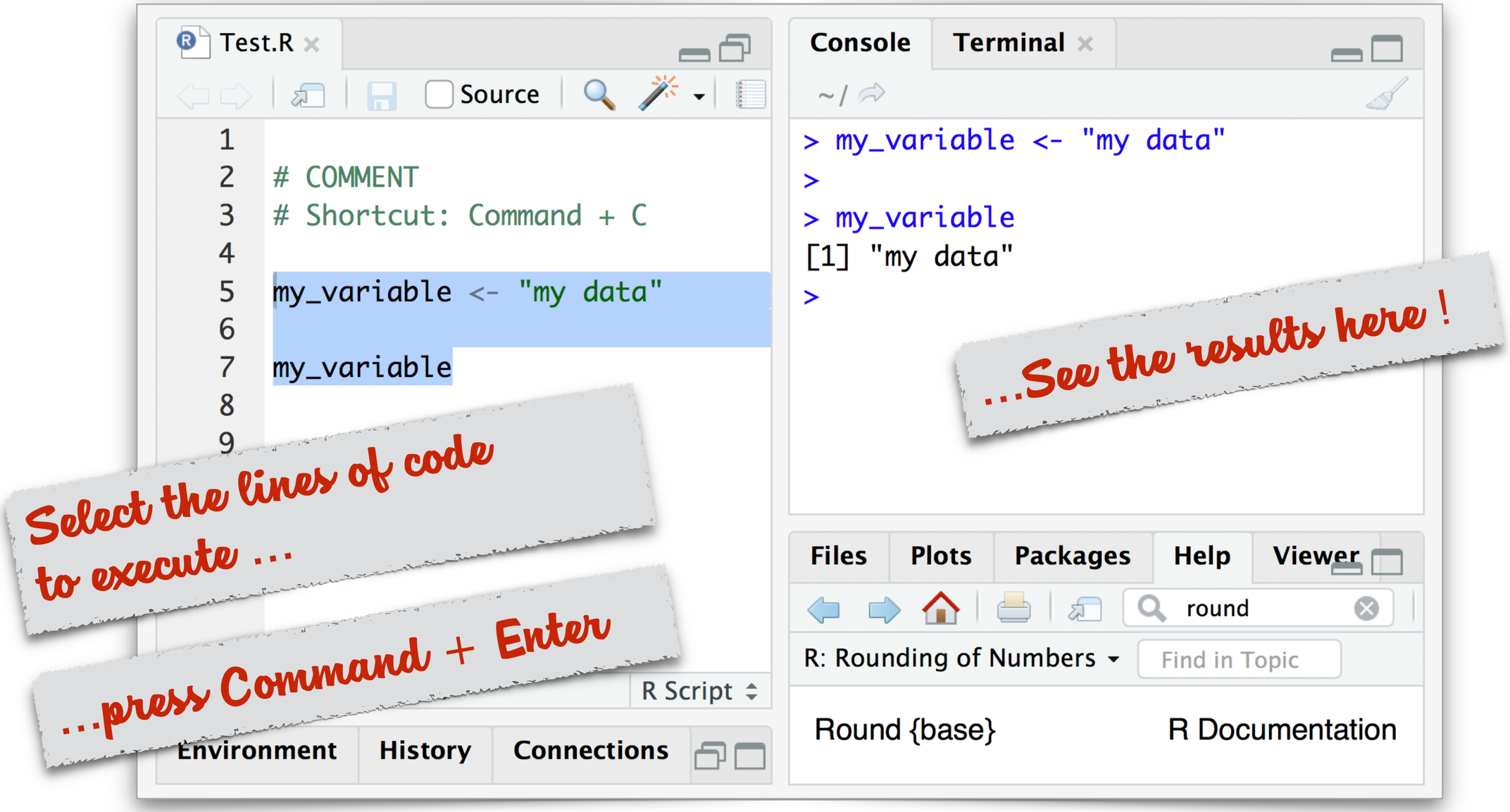
R STUDIO

R Studio has **3 main panels**



Using the **console** on the **right side** is recommended

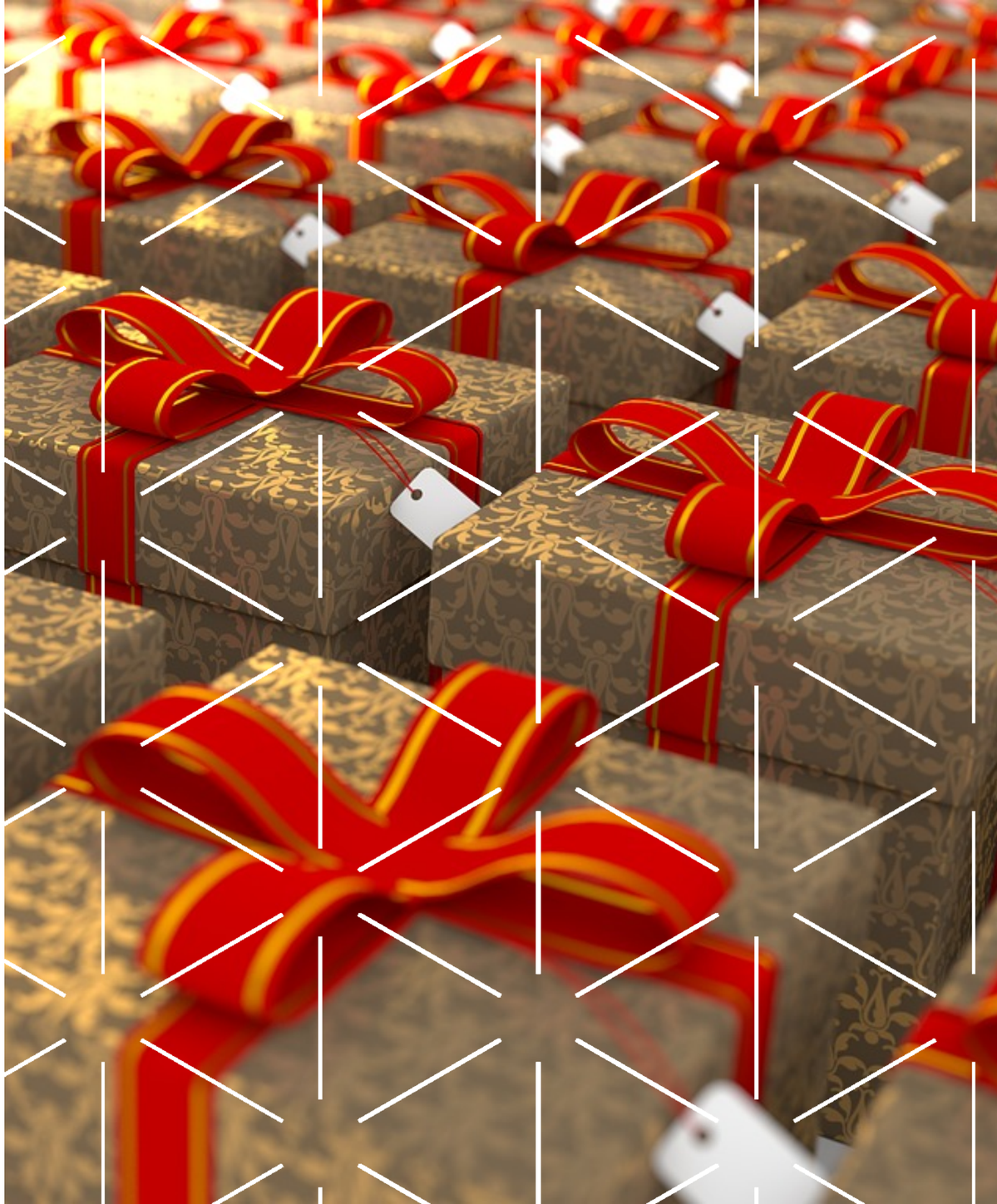



Test.R
Source
1
2 # COMMENT
3 # Shortcut: Command + C
4
5 my_variable <- "my data"
6
7 my_variable
8
9
R Script
Environment
History
Connections
Console
Terminal
~/
> my_variable <- "my data"
>
> my_variable
[1] "my data"
>
Files
Plots
Packages
Help
Viewer
round
R: Rounding of Numbers
Find in Topic
Round {base}
R Documentation
Select the lines of code to execute ...
...press Command + Enter
...See the results here !

VARIABLE

A **variable** is like a **box**
in which we can **store data**

```
my_variable <- "my data”
```



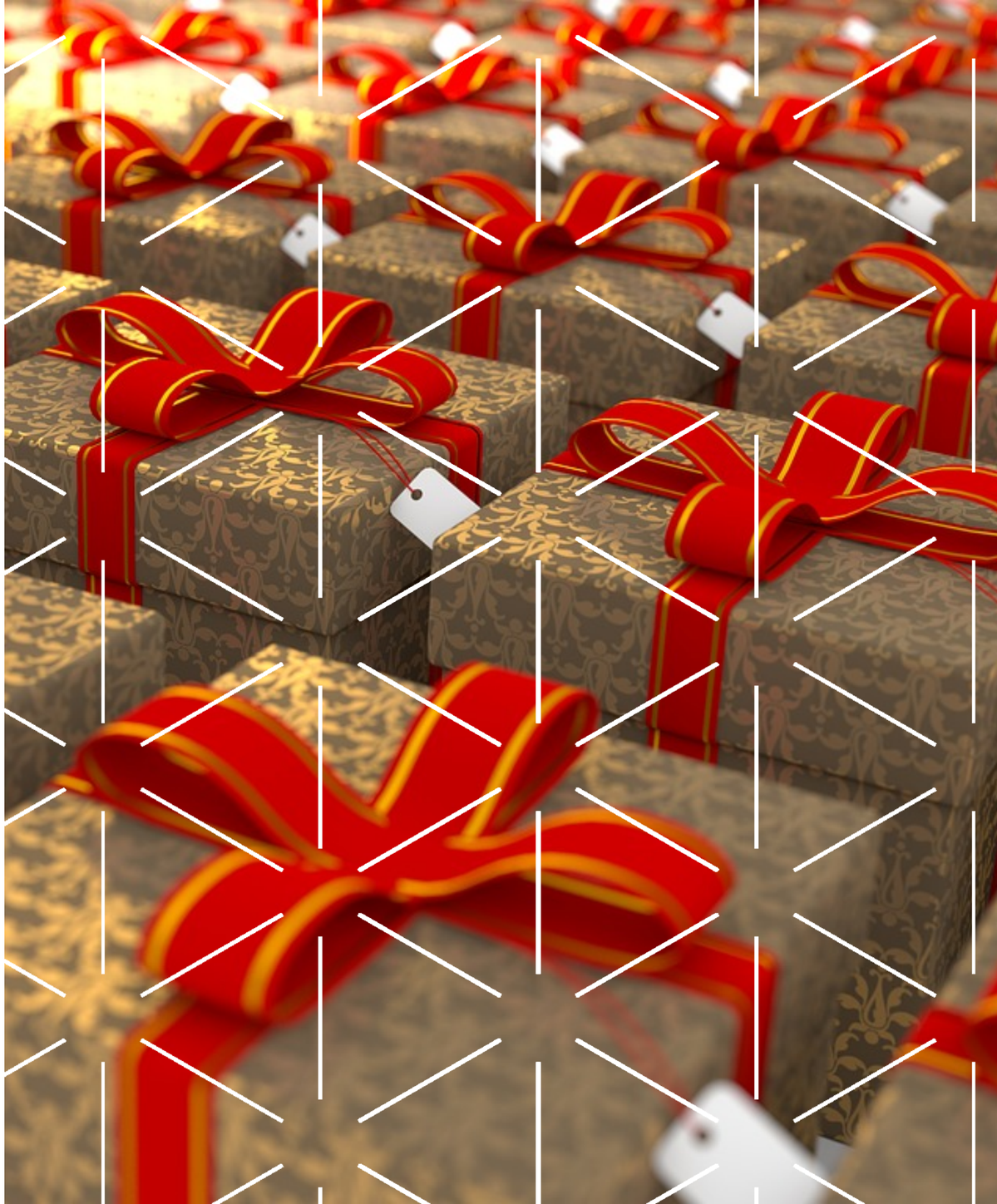
VARIABLE

A **variable** is like a **box**
in which we can **store data**

name the variable as you please

```
my_variable <- "my data"
```

but it must start with a letter



DATA TYPES

Variables can store different kinds of data

Text

```
my_text_variable <- "Hello!"
```

Number

```
my_number_variable <- 2019
```

Variables can store several elements

Series of Texts

```
my_texts <- c("Hi", "Salut", “Ciao")

my_texts[4] <- “Hallo"
```

Series of Numbers

```
my_numbers <- c(3, 5, 7, 11, 13, 17)

my_numbers[7] <- 19

my_months <- 1:12
```

DATA TYPES

Boolean

```
success <- TRUE
```

```
failure <- FALSE
```

Date & Time

```
intro_R_date <- as.Date("2019-03-07")

today <- Sys.Date()

tomorrow <- today + 1
```

Table

```
my_table <- rbind(c("A","alpha"),
                  c("B","beta"))

my_table <- cbind(my_table,
                  c("alpha","bravo"))


my_table[ ,3] <- c("ALPHA","BRAVO")

my_table[2,] <- c("B","Bravo")

my_table[2,3] <- "BRAVO"


colnames(my_table) <- c("letter",
                        "greek","code")

my_table$code <- c("alpha","bravo")
```

DATA TYPES

Variables can store
complex data structures

List

```
my_list <- list(my_table, my_texts,
                my_numbers, "etc")

my_list[[4]] <- "anything"

my_list[[1]][,3] <- c("Alpha","Bravo")
```

Data Frame

```
my_data_frame <- data.frame(my_table)

my_data_frame[,4] <- 1:2
```

Tibble

```
my_tibble <- tibble(1:2, c("a","b"),
                    list(my_texts, my_numbers),
                    list(my_table, my_list))

my_tibble[ ,2] <- c("A","B")

my_tibble[1,3][[1]][5] <- "Hola"
```

IF - THEN - ELSE

Computations can be **executed** only under certain **conditions**

```
if(age < 18) {
    print("Warning!")
} else {
    print("Welcome")
}
```

IF - THEN - ELSE

Computations can be **executed** only under certain **conditions**

```
if(age < 18) {
    print("Warning!")
} else {
    print("Welcome")
}
```

test the condition

done if test is passed

done if test is failed

IF - THEN - ELSE

The **condition** is a **boolean**

```
success <- TRUE
if(success) {
    print("This is executed")
} else {
    print("Not happening!")
}
```

Conditions can be **tested one after the other**

```
if(grade == 0){
   print("Terrible Fail!")
} else if(grade < 5){
   print("Fail")
} else if(grade < 8){
   print("Pass")
} else {
   print("Awesome!")
}
```

IF - THEN - ELSE

Conditions can be **tested together**

All conditions

```
if(grade > 4 & grade < 5) {
    print("Almost there!")
}
```

At least one condition

```
if(grade < 2 | grade > 8) {
    print("That's extreme!")
}
```

Conditions can be **complex** and **nested**

```
if( (dish == "Soup" & wine == "None")|
    (dish == "Fish" & wine != "Red") |
    (dish == "Duck" & wine == "Red") )
{
   print("I want to eat this.")
}
```

```
if( (dish == "Fish" & wine != "Red") &
    (wine == "Rosé" &
        (hour > 23 | temperature > 25)
    ) )
{
   print("This is an exception.")
}
```

FUNCTION

A **function** is a **series of operations** that can be **repeated**

```
my_function <- function(grade){

    if(grade < 5) {
        print("Fail")
    } else {
        print("Pass")
    }
}
```

```
my_function(0)
```

```
my_function(8)
```

FUNCTION

A **function** is a **series of operations** that can be **repeated**

```
my_function <- function(grade){

    if(grade < 5) {
        print("Fail")
    } else {
        print("Pass")
    }
}
```

Results depend on the input variable

```
my_function(0)
```

```
my_function(8)
```

FUNCTION

Functions can take
several **variables** as **input**

```
give_grade <- function(Q1, Q2){
    grade <- 0
    if(Q1 == "France") {
        grade <- grade + 1
    }
    if(Q2 == “Louis XIV") {
        grade <- grade + 1
    }
    print(grade)
}
```

Functions can **return**
a single **result**

```
test_Q1 <- function(Q1){
    if(Q1 == "France") {
        return(TRUE)
    }
}
```

Input variables can have
default values

```
grade_Q1 <- function(Q1, grade = 0){
    if( test(Q1) ) {
        return(grade + 1)
    }
}
```

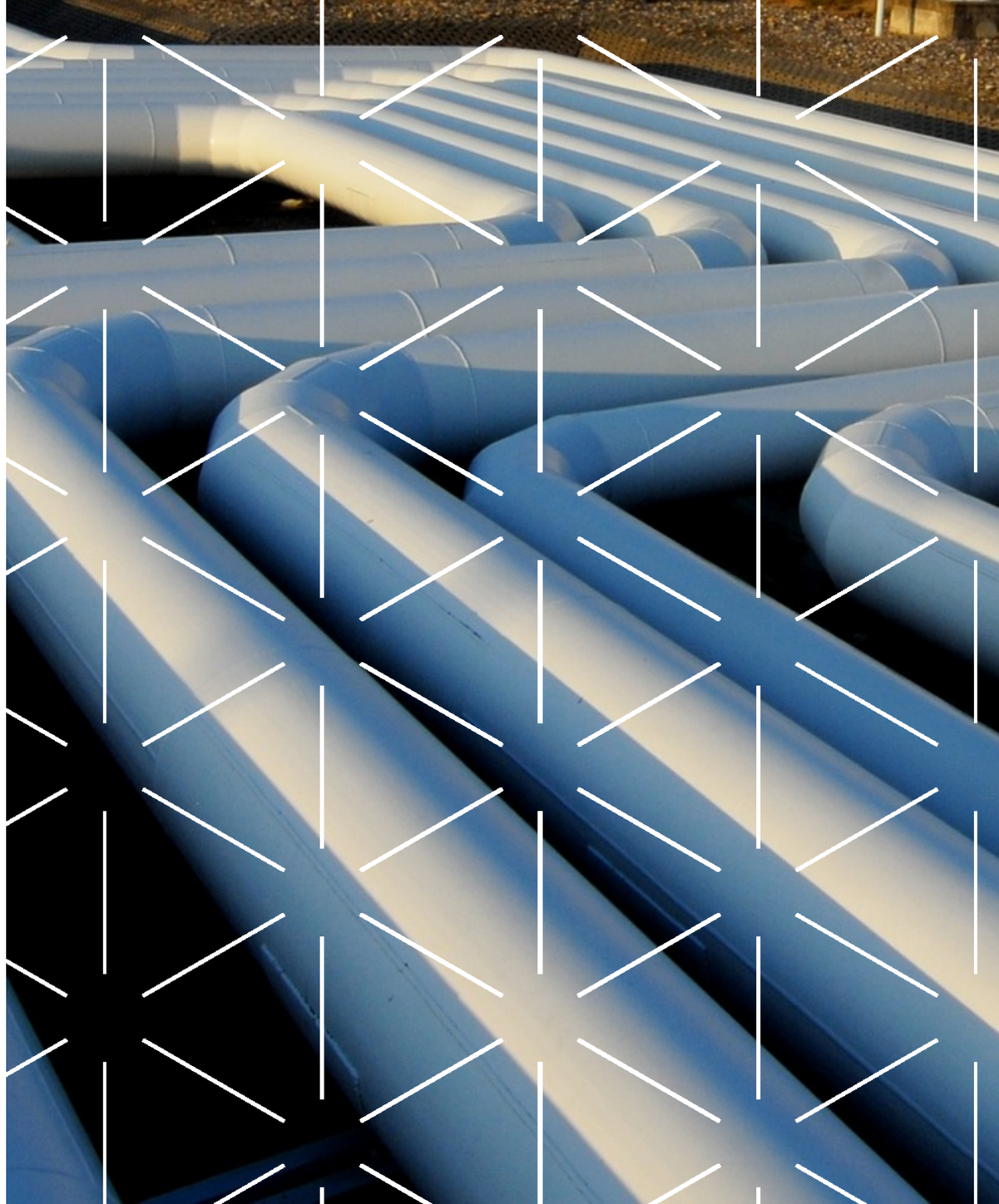
PIPE OPERATOR

The pipe (**%>%**) makes
our code **easier to read**

```
second_function( first_function(data) )
```

...Same as...

```
data %>% first_function() %>%
         second_function()
```



PIPE OPERATOR

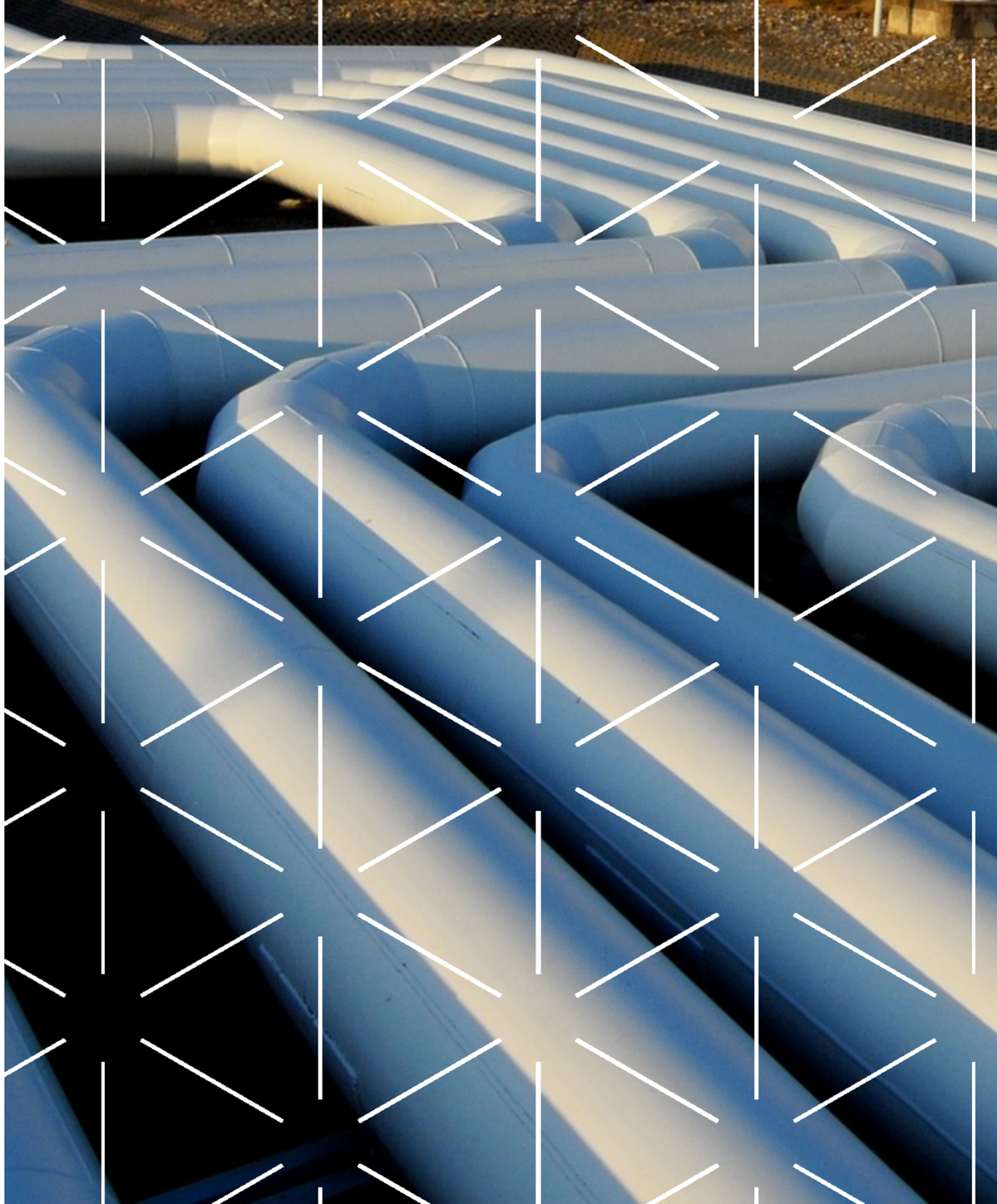
The pipe (**%>%**) makes
our code **easier to read**

This is executed first...

```
second_function(first_function(data))
```

...But we read it last !

```
data %>% first_function() %>%
         second_function()
```



PIPE OPERATOR

Functions take the **piped variable** as their **first input** variable

```
my_function(my_data, my_variable)

my_data %>% my_function(my_variable)
```

```
grade %>% round(digits = 1)

grade %>% round(digits = my_data)
```

The pipe can be used **anywhere**

```
if( grade %>% my_function() < 5 ) {
        print("Pass")
}
```

```
grade %>% round() %>%
   my_function(
       my_variable = time %>% round()
   )
```

LOOP

Operations can be **repeated**
a certain **number of times**

```
for( i in 1:10 ){
    print(i)
}
```

```
for( my_letter in my_table$letter ){
    print(my_letter)
}
```

LOOP

Loops can **repeat operations until** certain **conditions change**

While Loop

```
i <- 5

while( i > 0 ){
  print( my_table$letter[i] )
  i <- i - 1
}
```

Loops can be **interrupted**

```
for( my_letter in my_table$letter ){
    print(my_letter)
    if( my_letter == "B") {
       break
    }
}
```

```
for( i in 1:10 ){

    if( i %in% c(2,4,6,8,10) ) {
       next
    }
    print(i)
}
```

LOOP

Loops can **repeat operations until** certain **conditions change**

While Loop

```
i <- 5

while( i > 0 ){
  print( my_table$letter[i] )
  i <- i - 1
}
```

Loops can be **interrupted**

```
for( my_letter in my_table$letter ){
    print(my_letter)
    if( my_letter == "B") {
      break
    }
}
```

Terminate the loop

```
for( i in 1:10 ){

    if( i %in% c(2,4,6,8,10) ) {
      next
    }
    print(i)
}
```

Skip to the next item

LIBRARIES

A **library** is a collection
of useful **pre-made functions**

```
install.packages("tidyverse")

library(tidyverse)
```

LIBRARIES

A **library** is a collection
of useful **pre-made functions**

Download a library

...needs internet
...to do only once for the whole computer

```
install.packages("tidyverse")

library(tidyverse)
```

Load a library

...needs no internet
...to do each time you open R Studio

TIPS

Cheatsheet

https://www.rstudio.com/resources/cheatsheets/

https://github.com/DigitalSocietySchool/R_FirstContact/tree/master/Cheatsheet

Tutorial

https://r4ds.had.co.nz/

https://www.datacamp.com/tracks/tidyverse-fundamentals

https://www.tidyverse.org/learn/

Start your scripts with nice **settings**

```
# Set the working environment
if(!require('rstudioapi') ){
   install.packages('rstudioapi')
}
library(rstudioapi)
setwd(
   dirname(
      getSourceEditorContext()$path
))

# Disable scientific notation
options(scipen = 999)

# Disable text encoding as 'factors'
options(stringsAsFactors = FALSE)
```

READING & WRITING DATA

https://www.datacamp.com/community/tutorials/r-data-import-tutorial

- TEXT FILE P.29
- CSV FILE P.30
- EXCEL FILE P.31
- XML DATA P.32
- HTML DATA P.33

TEXT FILE

Write

```
write("My text", "my_text.txt")

write("Next Line", "my_text.txt",
       append = TRUE)
```

Read

```
read_file("my_text.txt")
```

Format the content of tables

```
for( i in 1:nrow(my_table) ){
    paste( my_table$name[i],
           "has grade",
           my_table$grade[i]
    ) %>%
    write( "class_grade.txt",
           append = TRUE )
}
```

Use the raw console output

```
my_table %>%
    print() %>%
    capture.output() %>%
    write("my_text.txt")
```

CSV FILE

Write

```
write.csv( "A, alpha\n
       "B, beta", “my_data.csv")

write.csv(“C, gamma", "my_data.csv",
       append = TRUE)
```

Read

```
read.csv("my_data.csv")
```

Write tables as they are

```
write.csv( my_table, "my_data.csv",
                row.names = FALSE)
```

Use other separator

```
read.csv( my_table, "my_data.csv",
                sep = "," )
```

EXCEL FILE

Main Libraries

https://www.datacamp.com/community/tutorials/r-tutorial-read-excel-into-r

```
library(readxl)

library(gdata)

library(xlsReadWrite)

library(XLConnect)

library(xlsx)
```

Excel files have **different formats** depending on the version of Excel

The **R libraries** for reading Excel files **may not work with all formats**

The right library can be found with **trials and errors**

XML DATA

Main Libraries

https://www.datacamp.com/community/tutorials/r-data-import-tutorial#xml

https://github.com/r-lib/xml2/blob/master/README.md

```
library(xml2)
read_xml("<client>
             <name> Emma </name>
          </client>")

library(XML)
xmlTreeParse('<client name="Emma">')
```

XML data has a **standard format** that also applies to HTML data

The **R libraries** for reading XML files **may work slightly differently**

The first step to begin with is to **follow a tutorial** (https://www.w3schools.com/xml/)

HTML DATA

Access Webpages

```
library(RCurl)

getURL("https://digitalsocietyschool.org")
```

Read HTML Content

```
library(rvest)

"https://digitalsocietyschool.org" %>%
    read_html()
```

Browse Links

https://www.datacamp.com/community/tutorials/r-web-scraping-rvest

```
"https://digitalsocietyschool.org" %>%
    read_html() %>%
    html_nodes( "a" ) %>%
    html_attr( "href" )
```

Read HTML Table

```
read_html(
"http://w3schools.com/html/html_tables.asp"
) %>% html_table()
```

MANIPULATING DATA

https://dplyr.tidyverse.org/

```
library(tidyverse)
```


- OVERVIEW DATA P.35
- ORDER DATA P.36
- EXTRACT DATA P.37
- ADD DATA P.38
- MERGE DATA P.39

OVERVIEW DATA

Simple functions for
quickly checking the data

https://dplyr.tidyverse.org/reference/summarise.html

Overview

```
# Get row names & first values
data %>% glimpse()

# Get first & last rows
data %>% head()
data %>% tail()
```

Summary

```
# Get mean & sum of each column
data %>% colMeans()
data %>% colSums()

# Get summary statistics of each column
data %>% summary()

# Get specific statistics of any column
data %>% summarise( min(my_column),
                    max(my_column) )

# Get statistics for groups of row
data %>% group_by( other_column ) %>%
         summarise( min(my_column),
                    max(my_column) )
```

ORDER DATA

Order by column value

```
# From lowest to highest column value
data %>% arrange( my_column )

# From highest to lowest column value
data %>% arrange( desc(my_column) )

# Order by one column then another
data %>% arrange( my_column,
                  other_column )
```

Group by column value

```
# Group rows with the same column value
data %>% group_by( my_column )

data %>% group_by( my_column,
                   other_column )

# Best value per group
data %>% group_by( my_column ) %>%
           top_n( 1, another_column )
```

EXTRACT DATA

https://dplyr.tidyverse.org/reference/select.html

https://dplyr.tidyverse.org/reference/filter.html

Extract Column

```
# Get certain columns
data %>% select( my_column,
                 other_column )

data %>% select( starts_with("my_") )

data %>% select( contains("other") )
```

Extract Row

```
# Get rows with certain column values
data %>% filter( my_column > 5 )

data %>% filter( my_column > 5 &
                 my_column < 8 )

data %>% filter( my_column > 5,
                 other_column == 0 )

# Random sample 10 rows
data %>% sample( 10 )

# Get rows with the 10 highest values
data %>% top_n( 10, my_column )

# Get rows with the 10 lowest values
data %>% top_n( -10, my_column )
```

ADD DATA

https://dplyr.tidyverse.org/reference/mutate.html

Add Column

```
# Add a new column
data %>%
   mutate( new_column = my_numbers )

# Make a column from another column
data %>%
   mutate( bonus = my_column + 2 )
```

Add Row

```
data %>% add_row( my_new_row )
```

MERGE DATA

Add All Rows

```
left_join(

   my_data,
   other_data,
   by = c('my_column'='other_column')

)
```

Equivalent

```
right_join(

   other_data,
   my_data,
   by = c('other_column'='my_column')

)
```

MERGE DATA

Keep Only Matching Rows

```
inner_join(

   my_data,
   other_data,
   by = c('my_column'='other_column')

)
```

Match on several columns

```
inner_join(

   my_data,
   other_data,
   by = c('my_column'='other_column',
          'my_extra'='other_extra',

)
```