Amsterdam University of Applied Sciences | DIGITAL SOCIETY SCHOOL

DATA-DRIVEN TRANSFORMATION

FIRST CONTACT WITH R

EMMA BEAUXIS-AUSSALET

e.m.a.l.beauxis@hva.nl

CONTENT

- BASICS OF PROGRAMMING P.3
- PROGRAMMING WITH R P.11
- BASICS OF VISUALIZATION P.17
- VISUALIZATION WITH R P.17

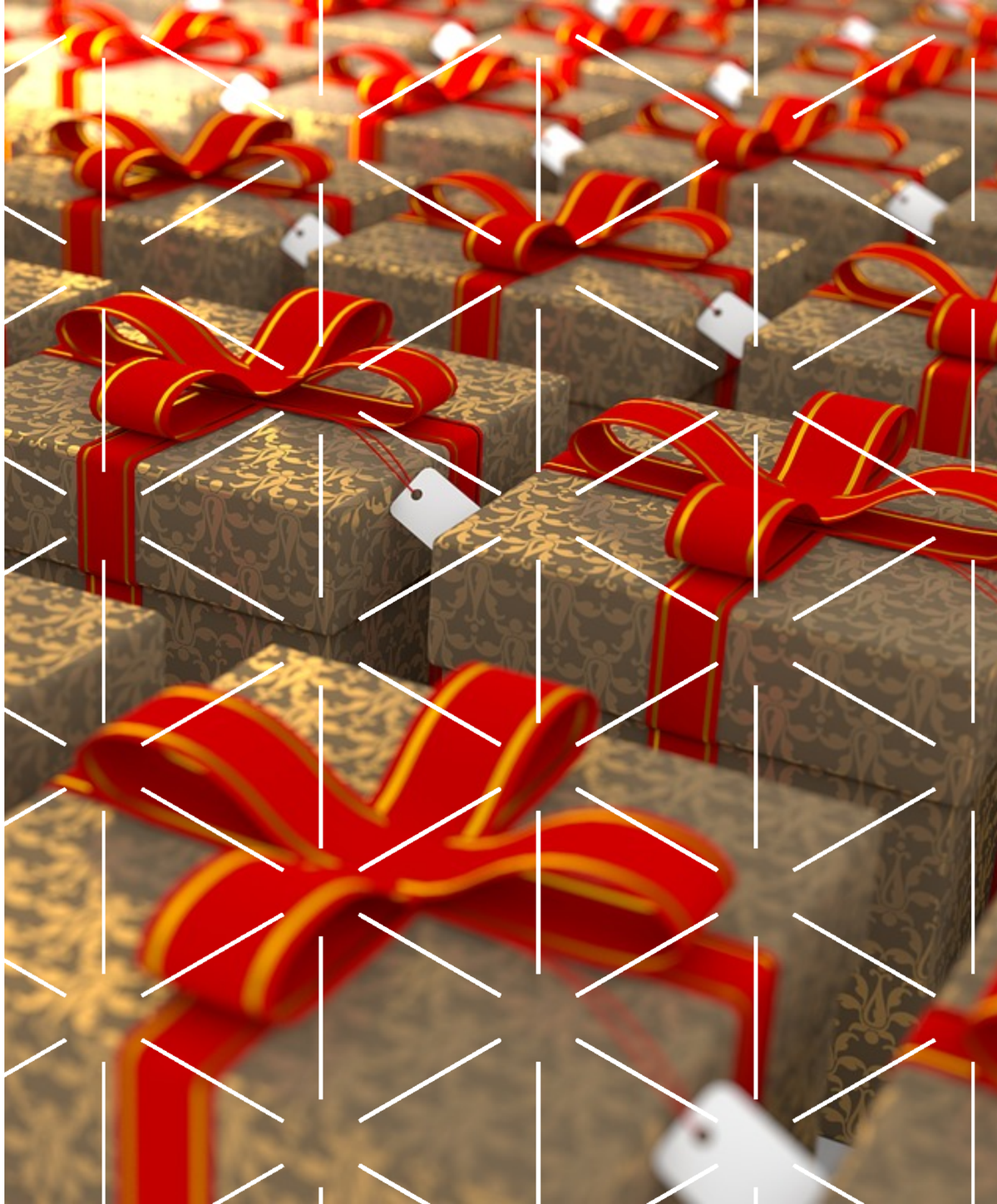
BASICS OF PROGRAMMING

- VARIABLE P.7
- DATA TYPES P.9
- IF - THEN - ELSE P.12
- FUNCTION P.16
- LOOP P.22

VARIABLE

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

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



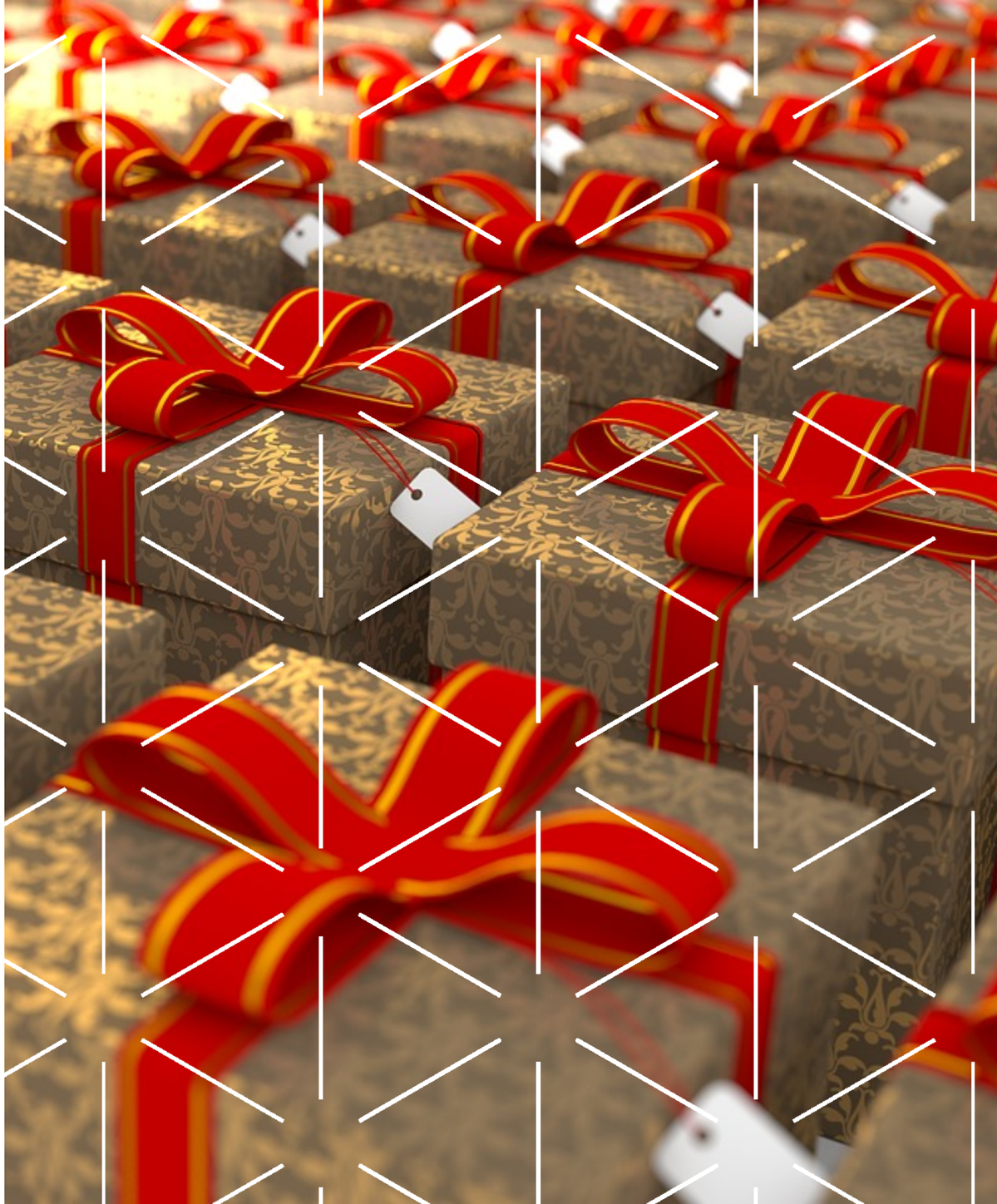
VARIABLE

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
```

Boolean

```
success <- TRUE
```

```
failure <- FALSE
```

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","beta","bravo")
my_table[2,3] <- "BRAVO"
```

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")
}
if(!success){
    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)
    }
}
```

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

EXERCISE

1. Separate into 2 groups: with and without programming experience
2. Make teams (2-3 max) that mix learners from both groups
3. Download the exercises: link
4. Pitch what was tricky or helpful

PROGRAMMING WITH R

- R STUDIO P.4
- LIBRARIES P.25
- DATA TYPES P.9
- PIPE OPERATOR P.19
- TIPS P.27

R STUDIO

R is a **programming language**

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

First install R

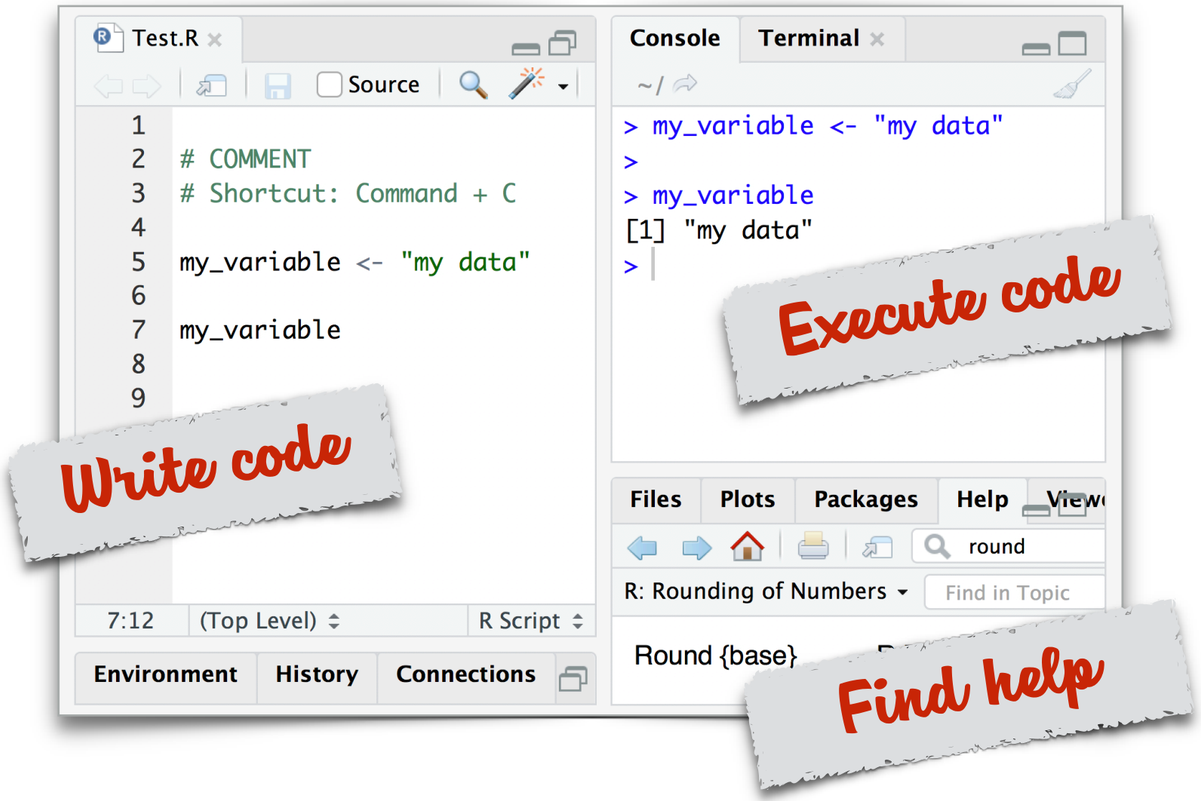
https://cran.r-project.org/

Then install R Studio

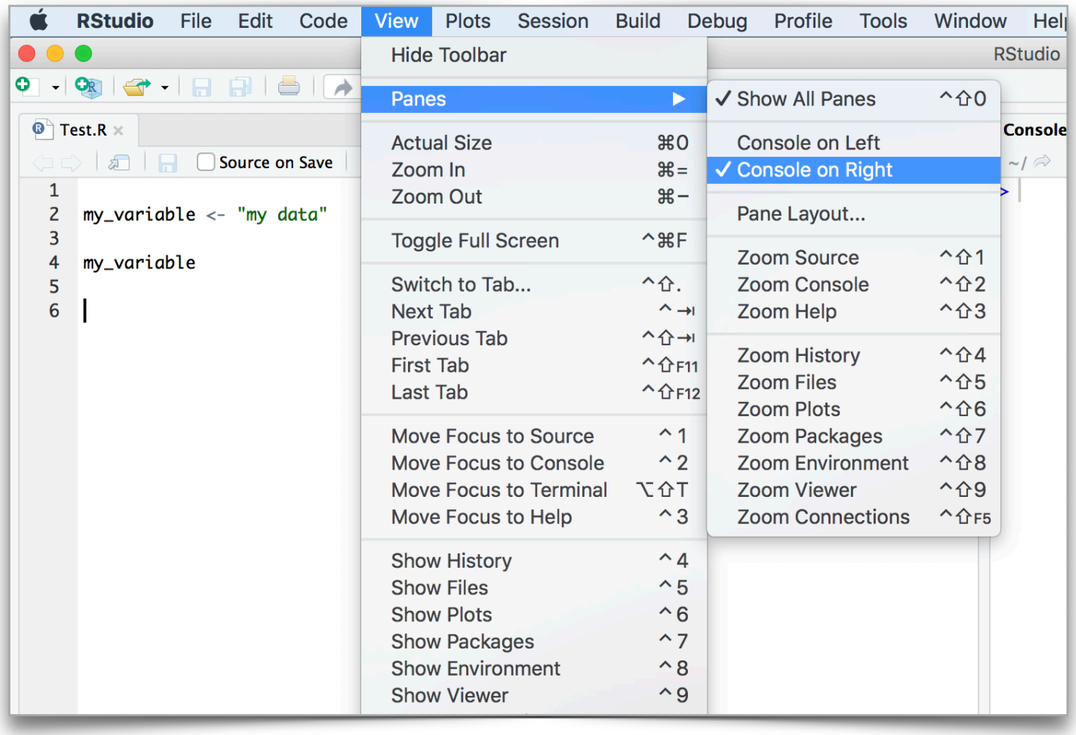
https://www.rstudio.com/products/rstudio/download/

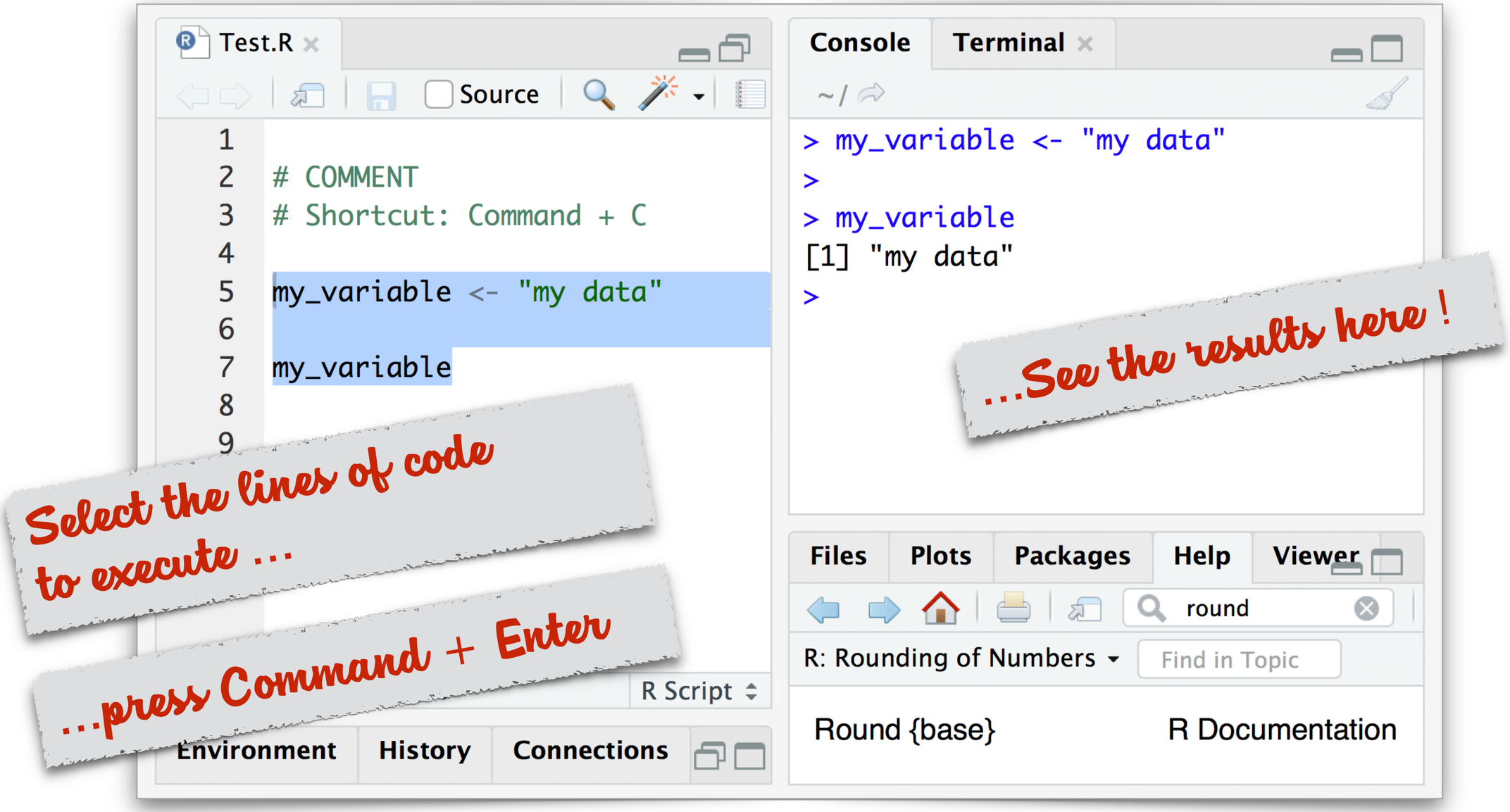
R STUDIO

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 !

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

DATA TYPES

Collections store several elements. Operations can be applied to all elements.

Series of Numbers

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

my_numbers[7] <- 19

my_numbers <- 1:10

my_numbers <- my_numbers / 2
```

Series of Texts

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

my_texts[4] <- "Hallo"

my_texts <- paste(my_texts, "Mary")
```

Series of Booleans

```
my_booleans <- my_texts == "Hi Mary"
```

DATA TYPES

Collections can be searched and ordered.

Check the presence of an element

```
my_boolean <- "Hi Mary" %in% my_texts

if(! "Hi Mary" %in% my_texts ){

   print("'Hi Mary' is missing.")

}
```

Order elements

```
sort(my_texts)

sort(1:10, decreasing = TRUE)
```

Extract unique elements

```
my_data <- c("Red","Red","Blue",
             "Red","Black","Black")

unique(my_data)
```

DATA TYPES

Data types are easily modified

Number to Text

```
my_texts <- as.character(1:10)

my_texts <- paste("Item", 1:10)
```

Text to Number

```
my_number <- as.numeric("1")
```

Series of Texts

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

my_texts[4] <- "Hallo"

my_texts <- paste(my_texts, "Mary")
```

Series of Booleans

```
my_booleans <- my_texts == "Hi Mary"
```

DATA TYPES

Dates and timestamps
are special data types.

Date & Time

```
timestamp <- as.POSIXct(
                  "2019-03-07 12:00:00")
now <- Sys.time()

next_minute <- Sys.time() + 60
```

Date

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

today <- Sys.Date()

tomorrow <- today + 1

format(today, "Year is %Y")

as.numeric(today - tomorrow)
```

https://www.r-bloggers.com/date-formats-in-r/

http://www.noamross.net/blog/2014/2/10/using-times-and-dates-in-r---presentation-code.html

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)

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

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

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]][[1]][5] <- "Hola"
```

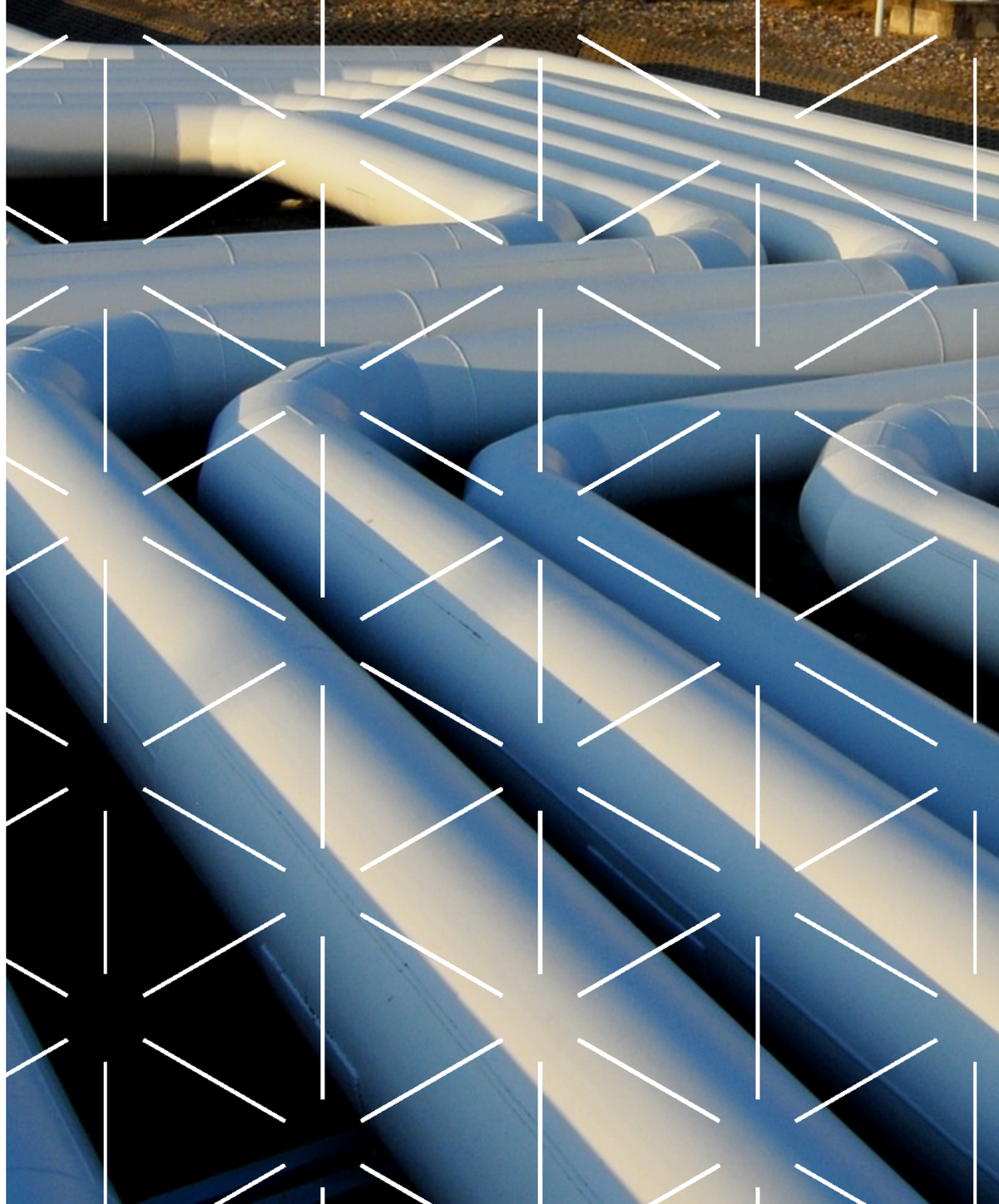
PIPE OPERATOR

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

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

...Same as...

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



PIPE OPERATOR

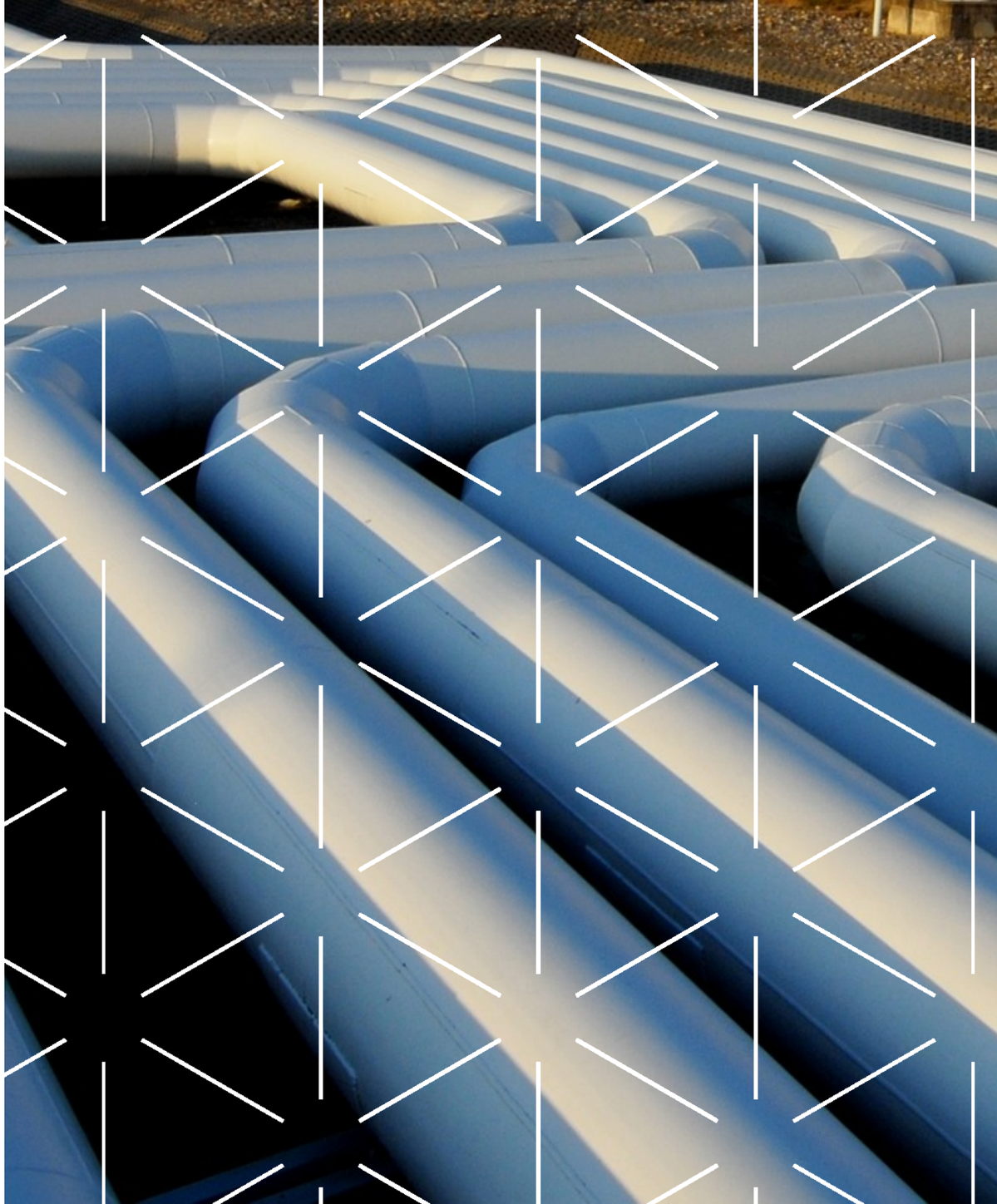
The pipe (**%>%**) makes
our code **easier to read**

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()
   )
```

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 as plain text

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

write("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_data.tsv",
          sep = "\t" )
```

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_n( 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 %>% bind_rows( my_new_rows )
```

MERGE DATA

Add All Columns

```
left_join(

  my_data,
  other_data,
   by=c('id_column'='other_id_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',

)
```

BASICS OF VISUALIZATION

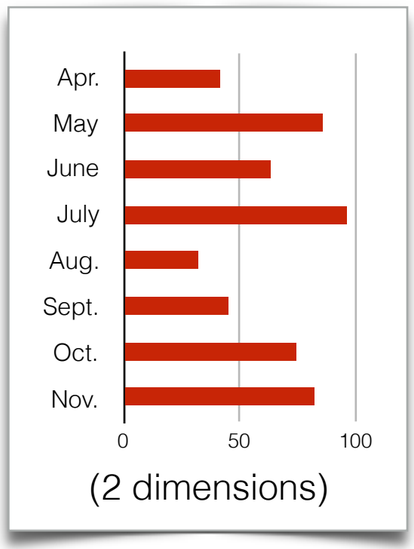
- VISUAL DIMENSIONS P.4
- MULTIPLE DIMENSIONS P.6
- NETWORK & FLOW P.9
- VARIANCE P.11

VISUAL DIMENSIONS

Visual features can be used to represent different variables

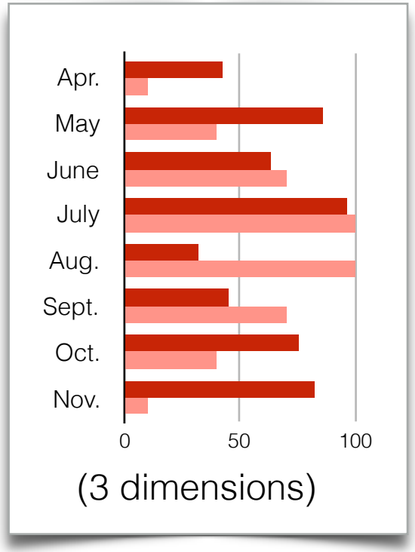
Variables are the data dimensions encoded into visual dimensions

Length



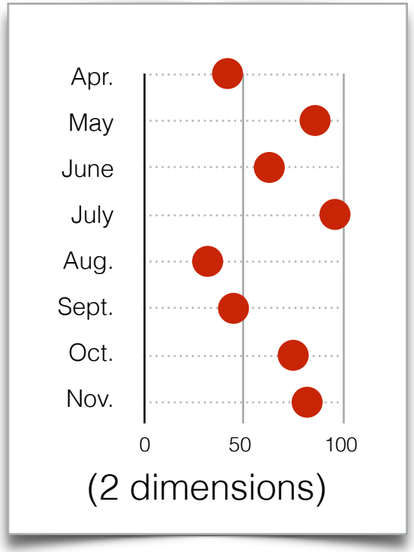
(2 dimensions)

Color



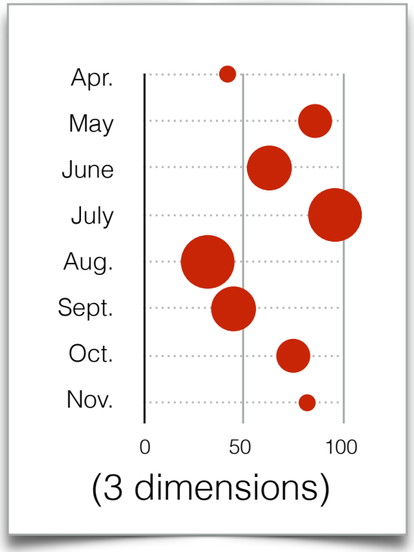
(3 dimensions)

Position



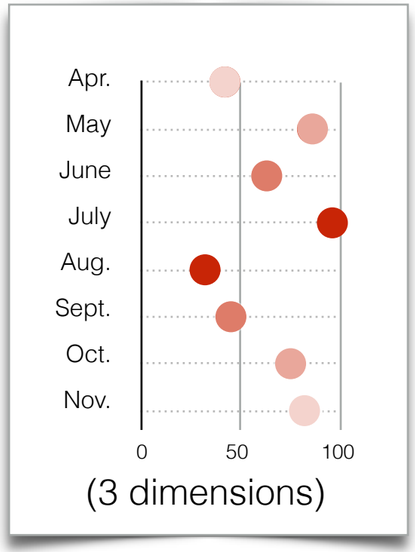
(2 dimensions)

Size



(3 dimensions)

Transparency



(3 dimensions)

VISUAL DIMENSIONS

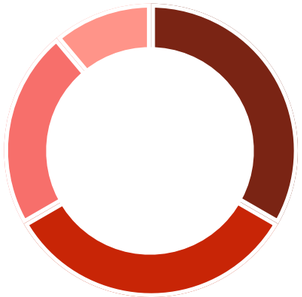
Visual features offer many options to represent the same data…

…but some visual features are harder to compare with human eyes

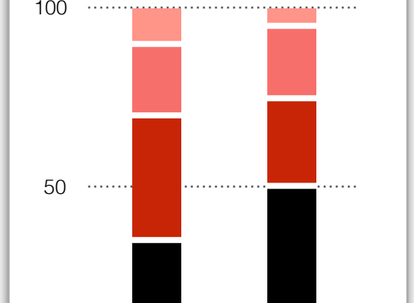
Area

Pie Chart

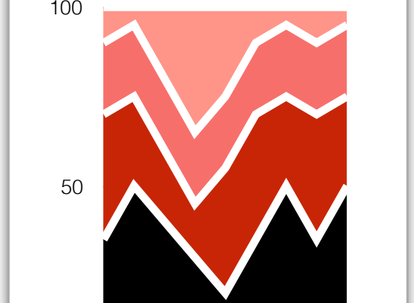
Length



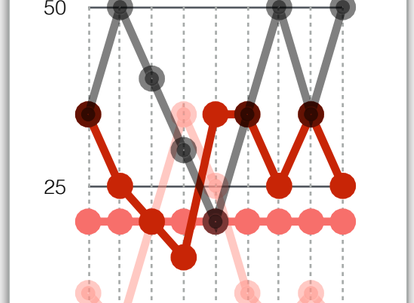
Donut Chart



Area



Position

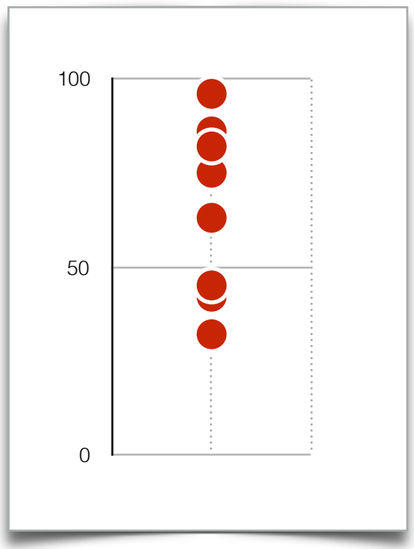


MULTIPLE DIMENSIONS

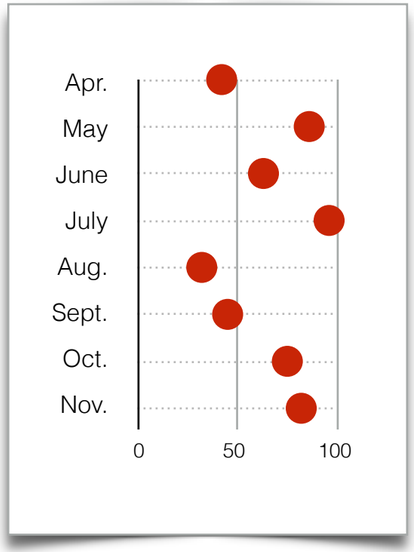
To add a data dimension,
add a visual dimension

…but the graph becomes complex

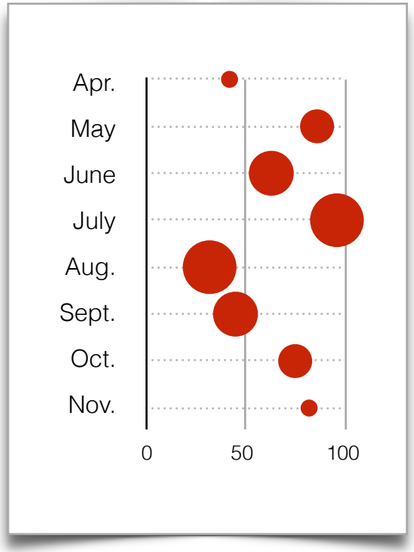
1 Dimension



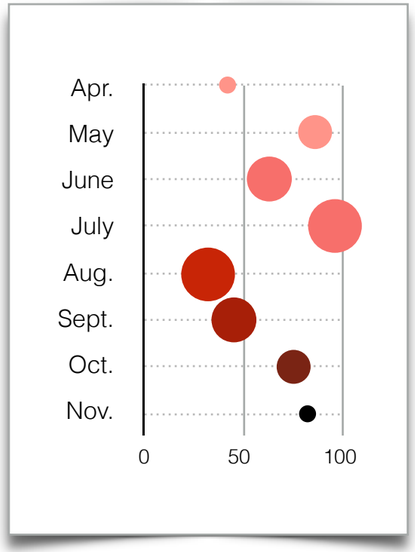
2 Dimensions



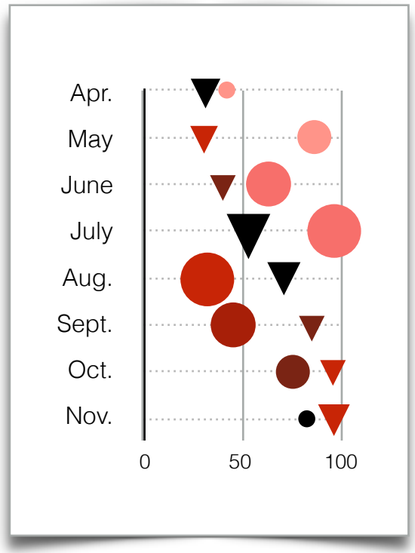
3 Dimensions



4 Dimensions

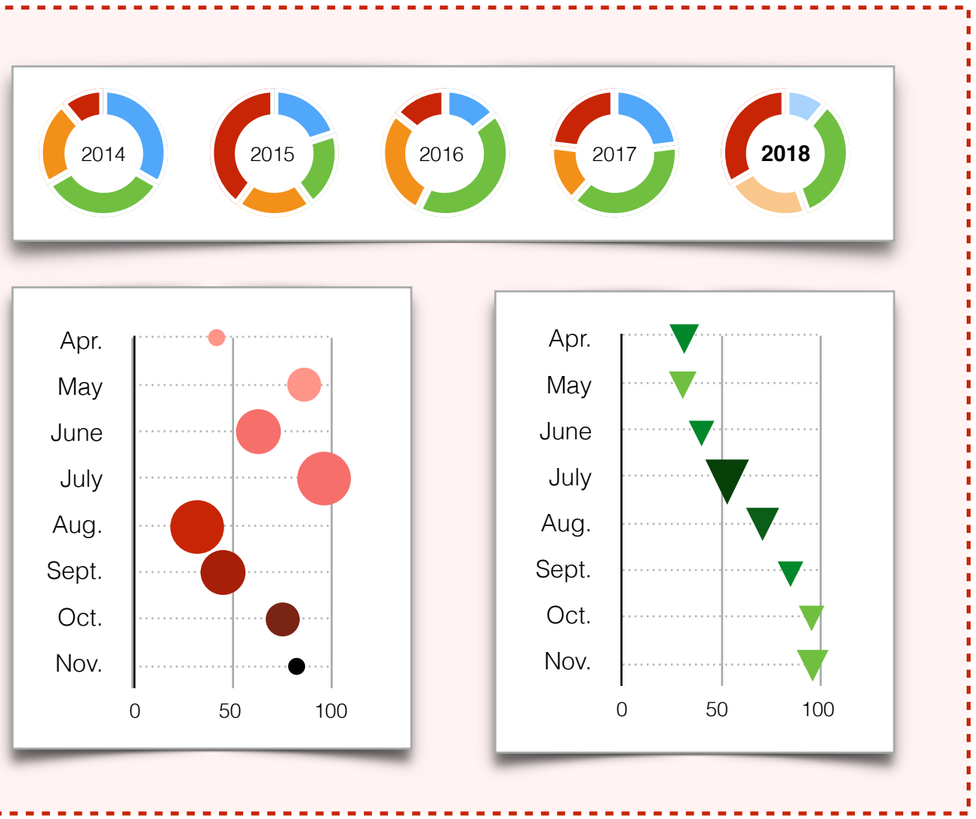


5 Dimensions

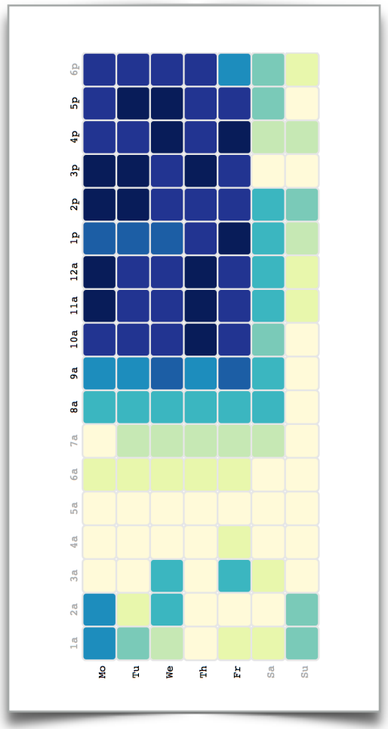


MULTIPLE DIMENSIONS

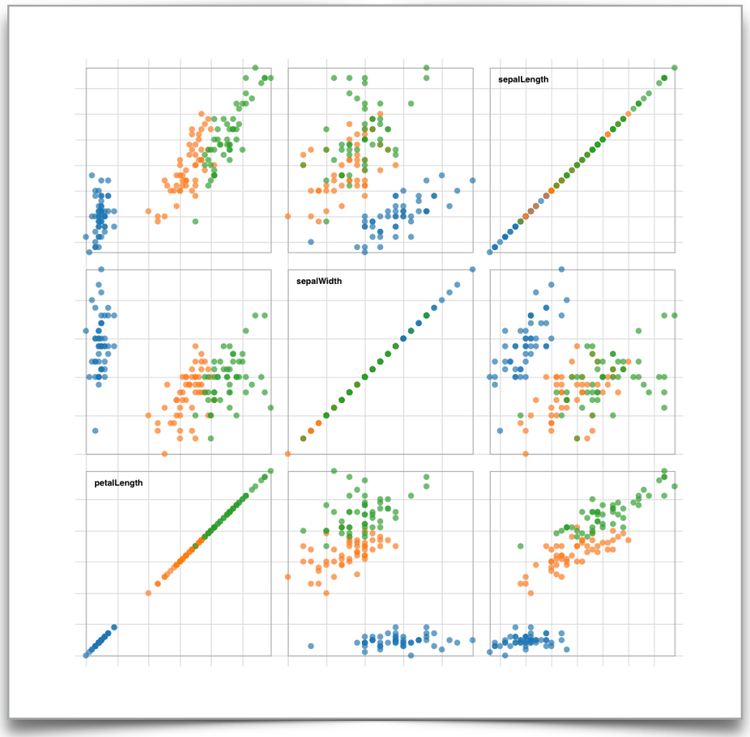
Multiple Views



Heatmap

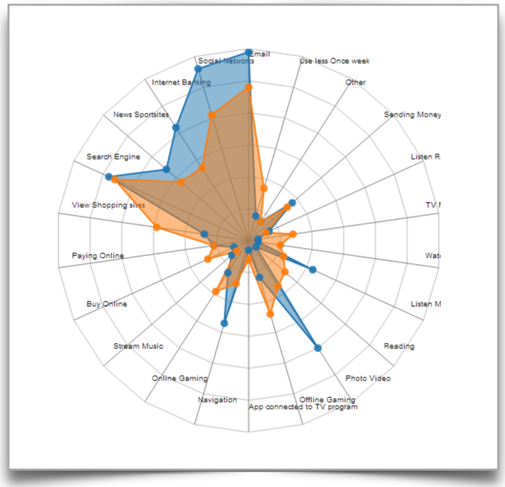


Scatterplot Matrix

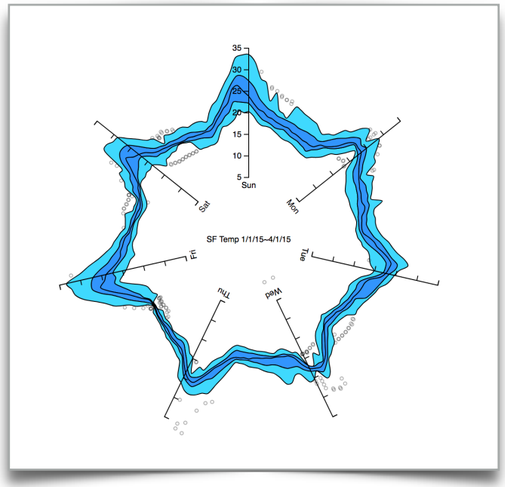


MULTIPLE DIMENSIONS

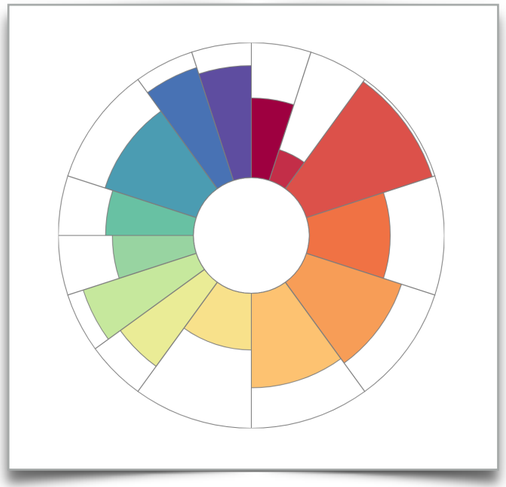
Radar Chart



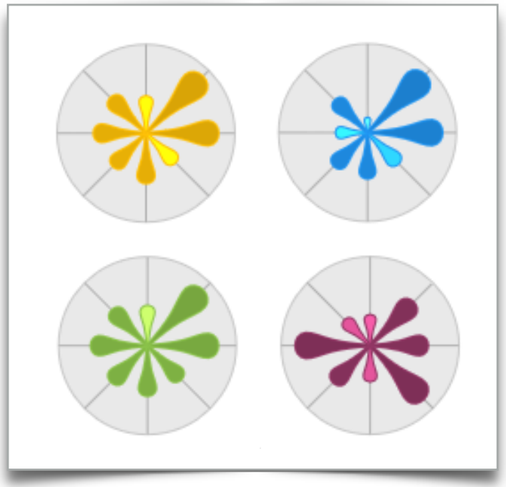
Cyclic Graph



Radial Graph

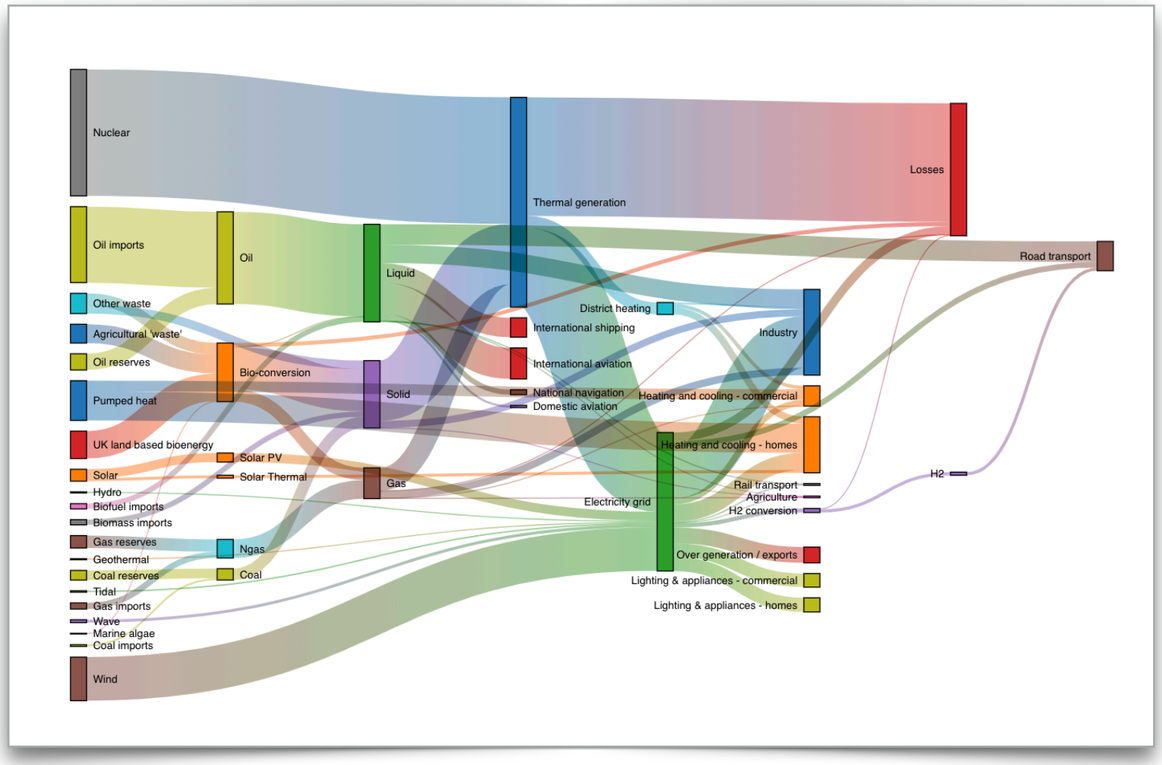


Glyph

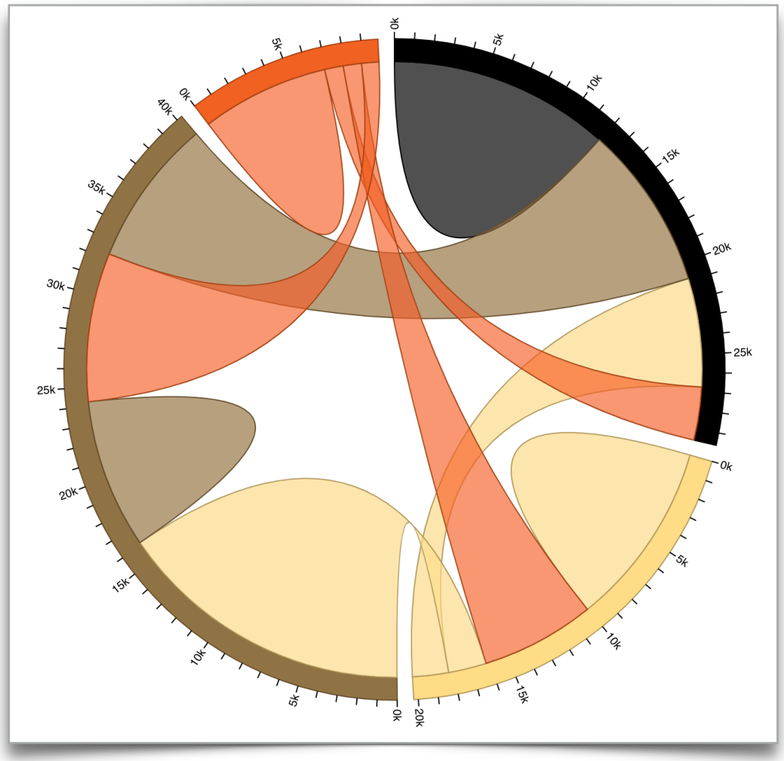


NETWORK & FLOW

Sankey Diagram

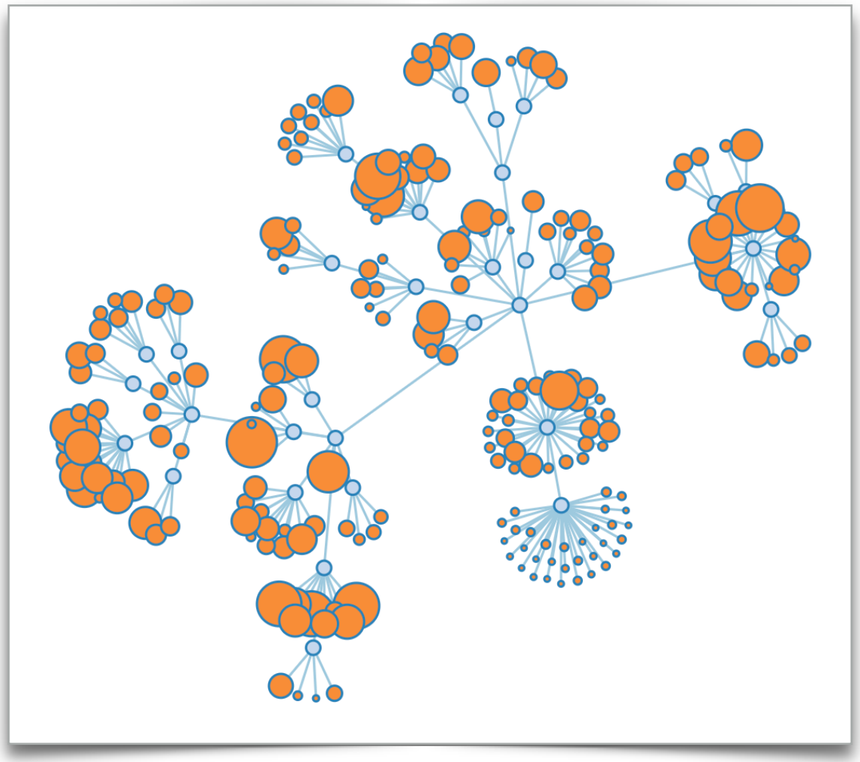


Chord Diagram

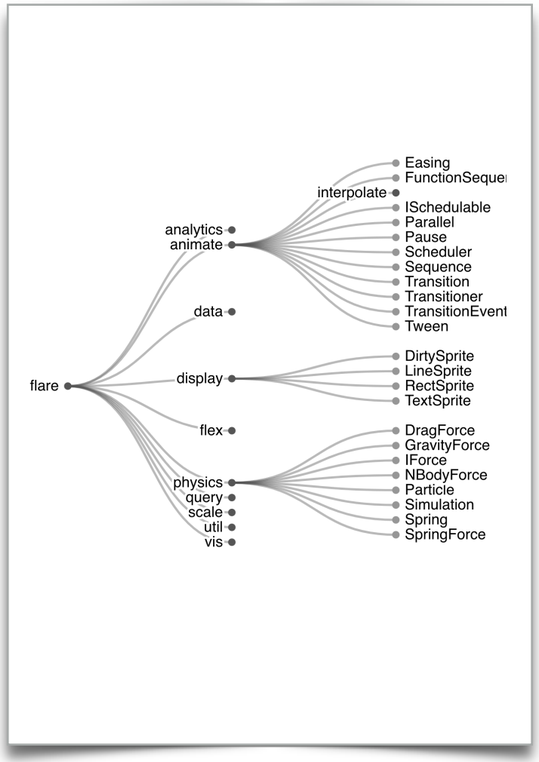


NETWORK & FLOW

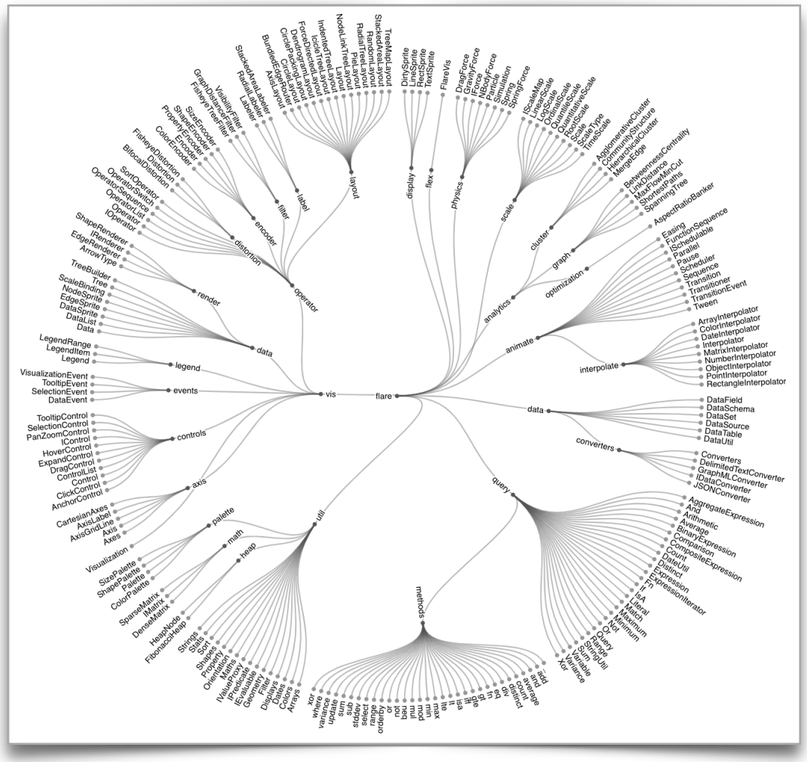
Force-Directed Graph



Tree

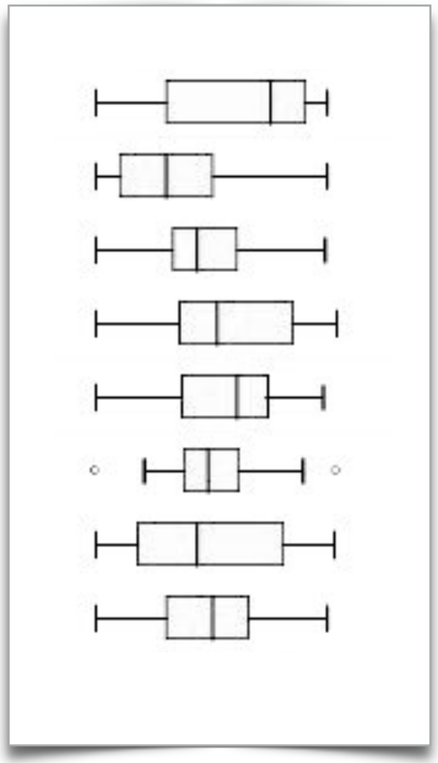


Radial Layout

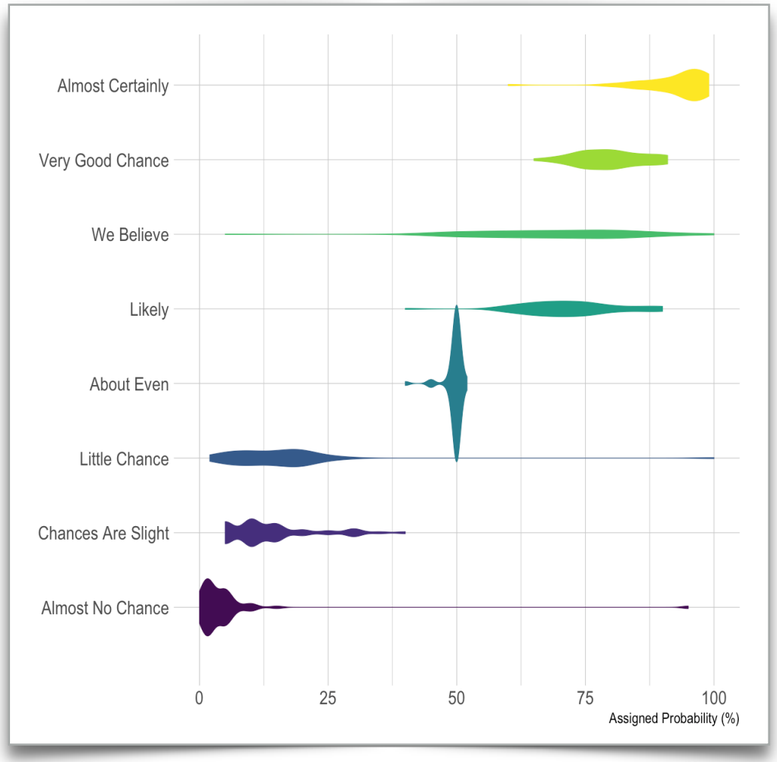


VARIANCE

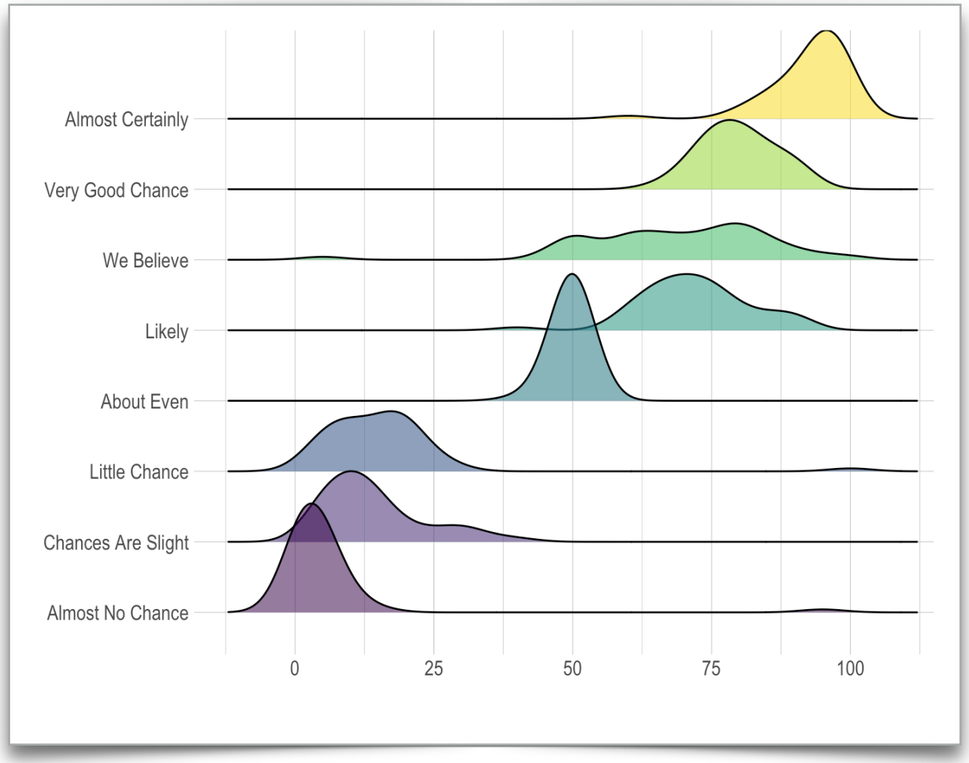
Boxplot



Violin Plot



Ridge Lines



VISUALIZATION WITH R

- BARCHART P.7
- LINE CHART P.9
- DONUT CHART P.12
- SCATTERPLOT P.16
- LAYOUT P.22
- EXPORT P.22


```
library(ggplot2)
```

https://ggplot2.tidyverse.org/
https://github.com/rstudio/cheatsheets/blob/master/data-visualization-2.1.pdf

Complete the template below to build a graph.

```
ggplot (data = <DATA>) +
<GEOM_FUNCTION>(mapping = aes(<MAPPINGS>),
 stat = <STAT> , position = <POSITION> ) +
<COORDINATE_FUNCTION> +
<FACET_FUNCTION> +
<SCALE_FUNCTION> +
<THEME_FUNCTION>
```

required

Not required, sensible defaults supplied

LINE SEGMENTS

common aesthetics: x, y, alpha, color, linetype, size

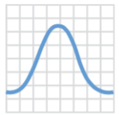
b + geom_abline(aes(intercept=0, slope=1)**)**
b + geom_hline(aes(yintercept = lat)**)**
b + geom_vline(aes(xintercept = long)**)**

b + geom_segment(aes(yend=lat+1, xend=long+1)**)**
b + geom_spoke(aes(angle = 1:1155, radius = 1)**)**

ONE VARIABLE continuous

c <- ggplot(mpg, aes(hwy)); c2 <- ggplot(mpg)

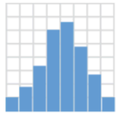
c + geom_area(stat = "bin")
x, y, alpha, color, fill, linetype, size



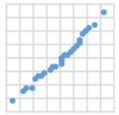
c + geom_density(kernel = "gaussian"**)**
x, y, alpha, color, fill, group, linetype, size, weight

c + geom_dotplot()
x, y, alpha, color, fill

c + geom_freqpoly() x, y, alpha, color, group, linetype, size



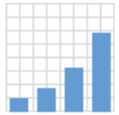
c + geom_histogram(binwidth = 5**)** x, y, alpha, color, fill, linetype, size, weight



c2 + geom_qq(aes(sample = hwy)**)** x, y, alpha, color, fill, linetype, size, weight

discrete

d <- ggplot(mpg, aes(fl))



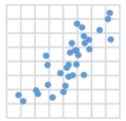
d + geom_bar()
x, alpha, color, fill, linetype, size, weight

TWO VARIABLES

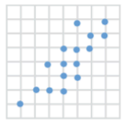
continuous x , continuous y

e <- ggplot(mpg, aes(cty, hwy))

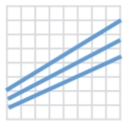
e + geom_label(aes(label = cty), nudge_x = 1, nudge_y = 1, check_overlap = TRUE**)** x, y, label, alpha, angle, color, family, fontface, hjust, lineheight, size, vjust



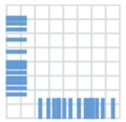
e + geom_jitter(height = 2, width = 2**)** x, y, alpha, color, fill, shape, size



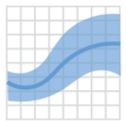
e + geom_point(), x, y, alpha, color, fill, shape, size, stroke



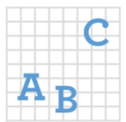
e + geom_quantile(), x, y, alpha, color, group, linetype, size, weight



e + geom_rug(sides = "bl"**)**, x, y, alpha, color, linetype, size



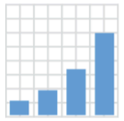
e + geom_smooth(method = lm**)**, x, y, alpha, color, fill, group, linetype, size, weight



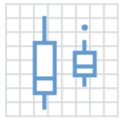
e + geom_text(aes(label = cty), nudge_x = 1, nudge_y = 1, check_overlap = TRUE**)**, x, y, label, alpha, angle, color, family, fontface, hjust, lineheight, size, vjust

discrete x , continuous y

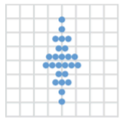
f <- ggplot(mpg, aes(class, hwy))



f + geom_col(), x, y, alpha, color, fill, group, linetype, size



f + geom_boxplot(), x, y, lower, middle, upper, ymax, ymin, alpha, color, fill, group, linetype, shape, size, weight



f + geom_dotplot(binaxis = "y", stackdir = "center"**)**, x, y, alpha, color, fill, group

f + geom_violin(scale = "area"**)**, x, y, alpha, color, fill, group, linetype, size, weight

discrete x , discrete y

g <- ggplot(diamonds, aes(cut, color))

g + geom_count(), x, y, alpha, color, fill, shape, size, stroke

COLOR AND FILL SCALES (DISCRETE)

n <- d + geom_bar(aes(fill = fl)**)**

n + scale_fill_brewer(palette = "Blues"**)**
For palette choices:
RColorBrewer::display.brewer.all()

n + scale_fill_grey(start = 0.2, end = 0.8,
na.value = "red"**)**

COLOR AND FILL SCALES (CONTINUOUS)

o <- c + geom_dotplot(aes(fill = ..x..))

o + scale_fill_distiller(palette = "Blues"**)**

o + scale_fill_gradient(low="red", high="yellow"**)**

o + scale_fill_gradient2(low="red", high="blue",
mid = "white", midpoint = 25)

o + scale_fill_gradientn(colours=topo.colors(6)**)**
Also: rainbow(), heat.colors(), terrain.colors(),
cm.colors(), RColorBrewer::brewer.pal()

SHAPE AND SIZE SCALES

p <- e + geom_point(aes(shape = fl, size = cyl))
p + scale_shape() + scale_size()
p + scale_shape_manual(values = c(3:7)**)**

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25

p + scale_radius(range = c(1,6)**)**
p + scale_size_area(max_size = 6**)**

continuous bivariate distribution

h <- ggplot(diamonds, aes(carat, price))

h + geom_bin2d(binwidth = c(0.25, 500)**)**
x, y, alpha, color, fill, linetype, size, weight

h + geom_density2d()
x, y, alpha, colour, group, linetype, size

h + geom_hex()
x, y, alpha, colour, fill, size

continuous function

i <- ggplot(economics, aes(date, unemploy))

i + geom_area()
x, y, alpha, color, fill, linetype, size

i + geom_line()
x, y, alpha, color, group, linetype, size

i + geom_step(direction = "hv")
x, y, alpha, color, group, linetype, size

Faceting

Facets divide a plot into subplots based on the values of one or more discrete variables.

t <- ggplot(mpg, aes(cty, hwy)) + geom_point()

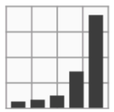
t + facet_grid(. ~ fl)
facet into columns based on fl

t + facet_grid(year ~ .)
facet into rows based on year

t + facet_grid(year ~ fl)
facet into both rows and columns

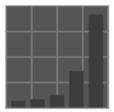
t + facet_wrap(~ fl)
wrap facets into a rectangular layout

Themes

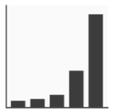


r + theme_bw()
White background with grid lines

r + theme_gray()
Grey background (default theme)

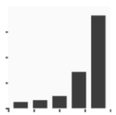


r + theme_dark()
dark for contrast



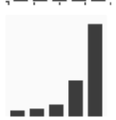
r + theme_classic()

r + theme_light()



r + theme_linedraw()

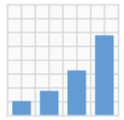
r + theme_minimal()
Minimal themes



r + theme_void()
Empty theme

Coordinate Systems

r <- d + geom_bar()

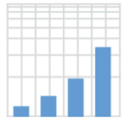


r + coord_cartesian(xlim = c(0, 5)**)**
xlim, ylim
The default cartesian coordinate system

r + coord_fixed(ratio = 1/2**)**
ratio, xlim, ylim
Cartesian coordinates with fixed aspect ratio between x and y units

r + coord_flip()
xlim, ylim
Flipped Cartesian coordinates

r + coord_polar(theta = "x", direction=1 **)**
theta, start, direction
Polar coordinates



r + coord_trans(ytrans = “sqrt"**)**
xtrans, ytrans, limx, limy
Transformed cartesian coordinates. Set xtrans and ytrans to the name of a window function.

π + coord_quickmap()

π + coord_map(projection = "ortho", orientation=c(41, -74, 0)**)**projection, orienztation
xlim, ylim
Map projections from the mapproj package (mercator (default), azequalarea, lagrange, etc.)

SAVING GRAPHS

```
data %>%
   ggplot( aes( x = my_column, y = other_column ) ) +
     geom_point( aes( text = paste("Tip Box:", another_column) ) ) +
     geom_smooth( aes( colour = last_column, fill = last_column ) ) +
     facet_wrap( ~ last_column )

ggsave( "my_visualization.pdf", width=18, height=18)

ggsave( "my_image.png", width=10, height=10, unit="cm")
```