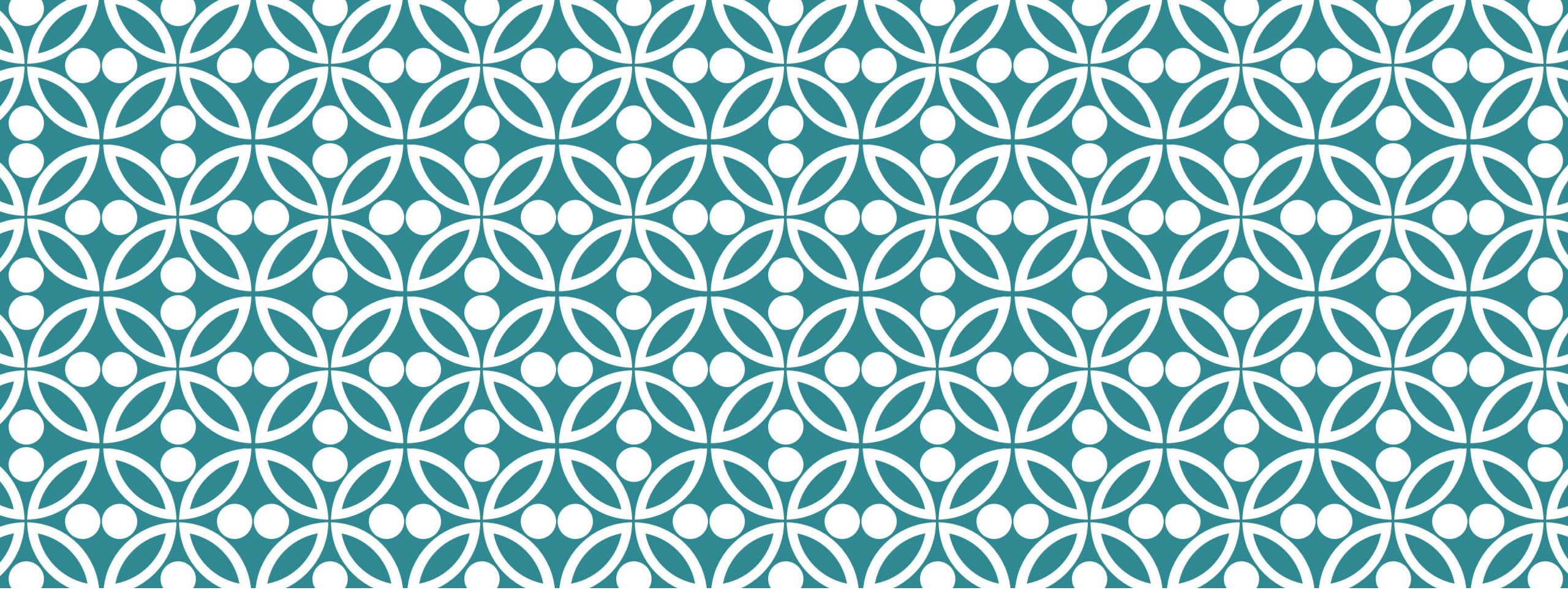


SESSION 2: DATA VISUALIZATION

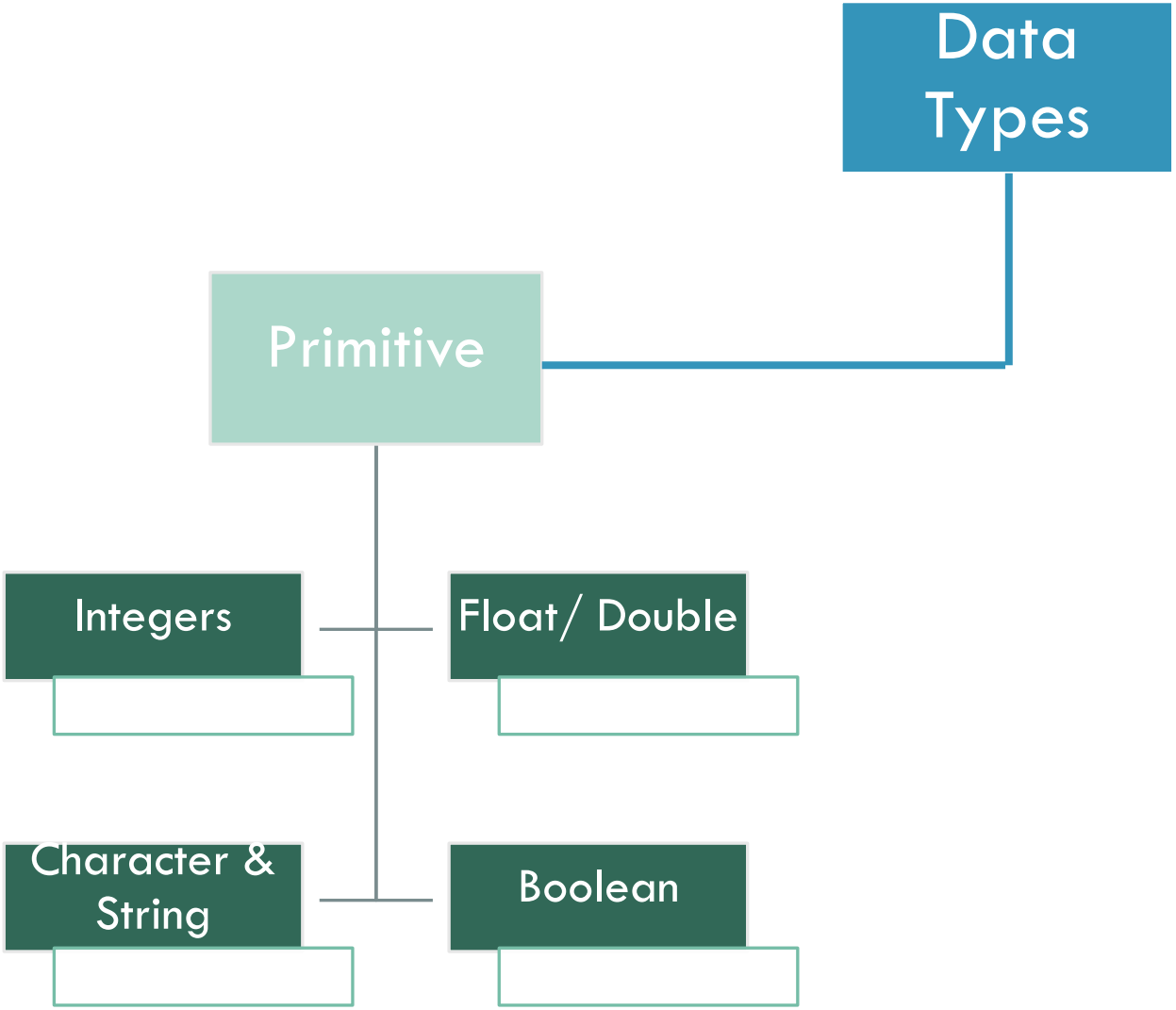
DR. SOFIA GIL-CLAVEL

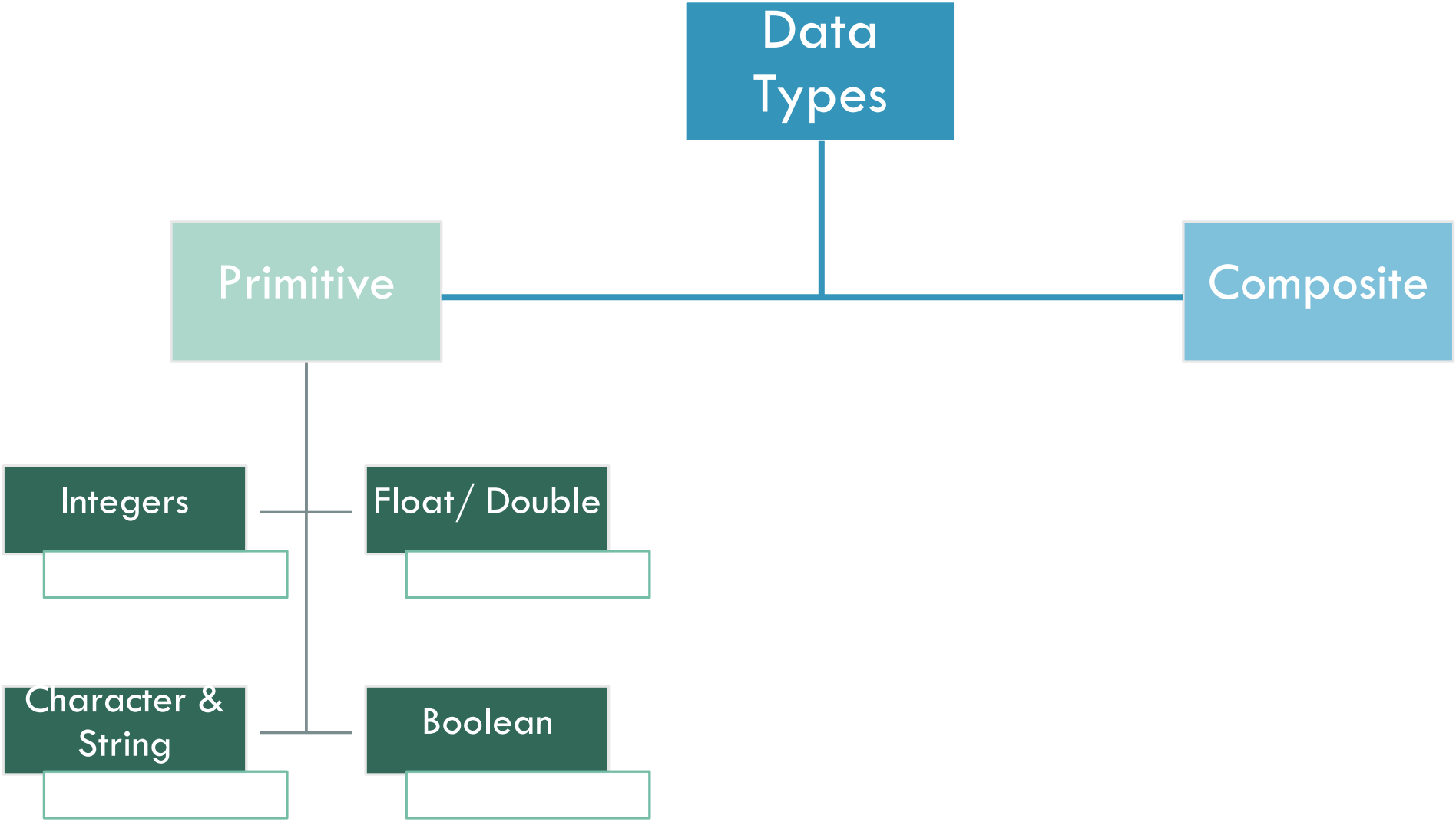
- Session 1: Recap
- Ggplot2 & the Grammar of Graphs

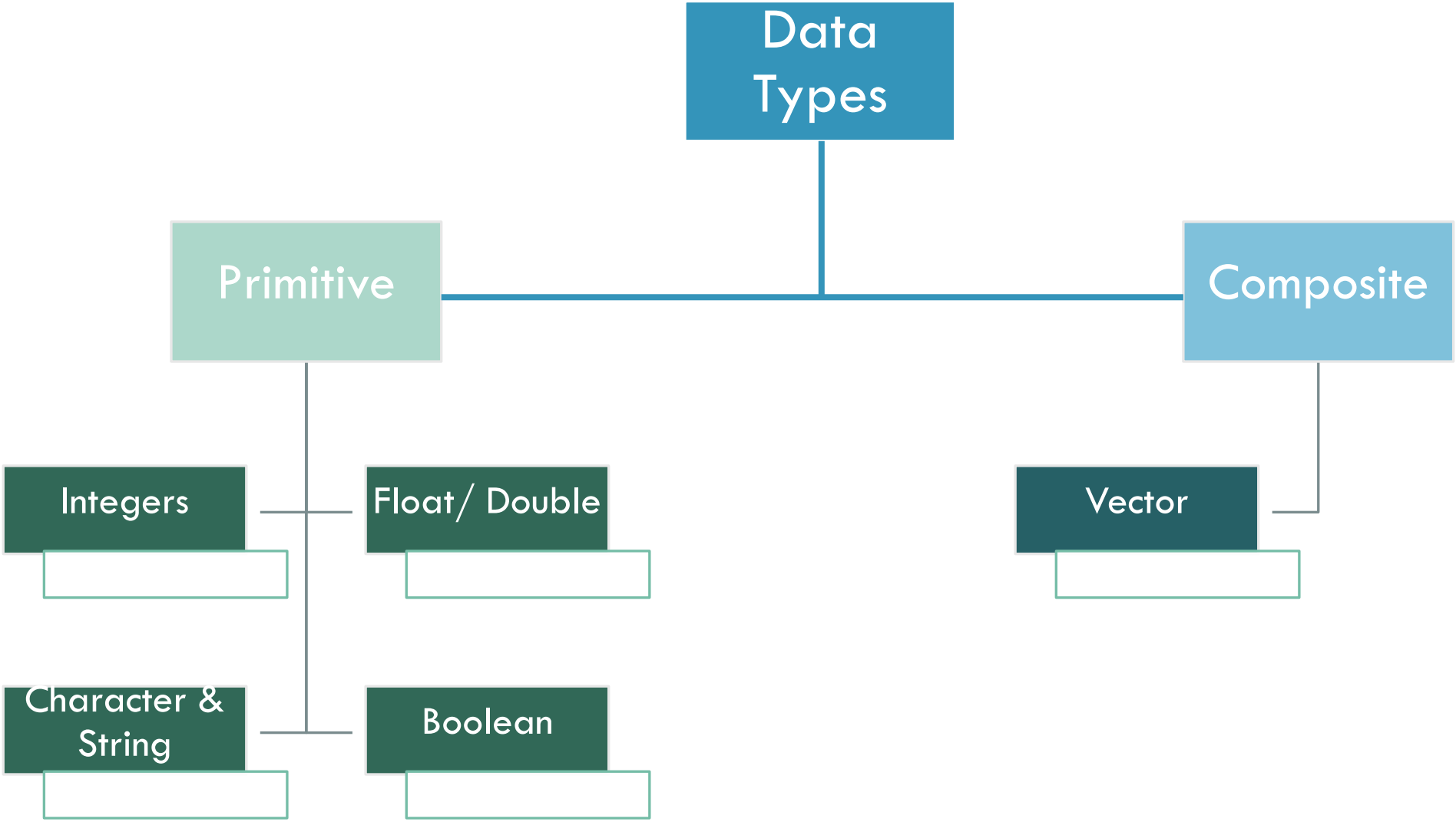
1. SESSION 1: RECAP

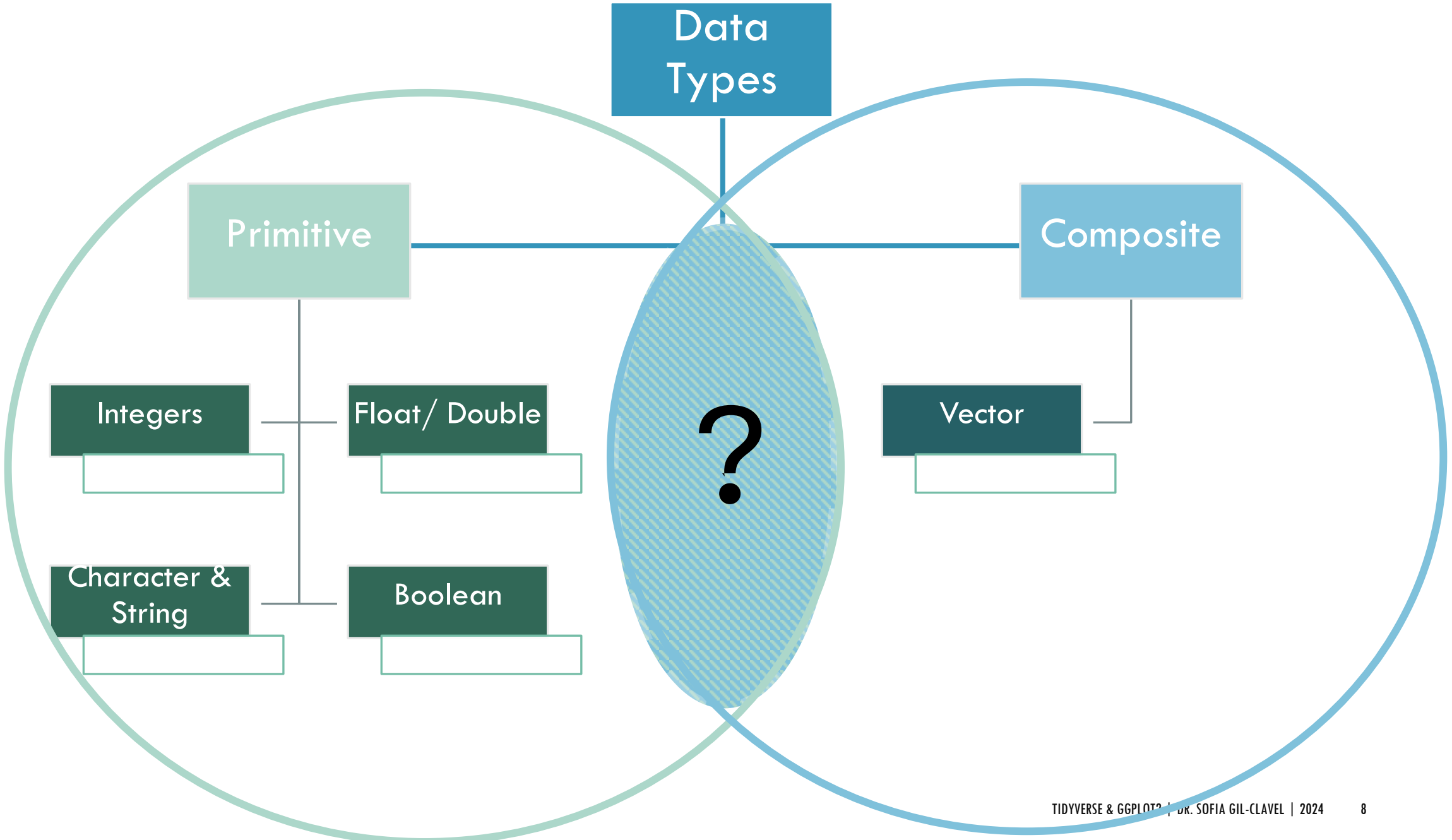
Data Types

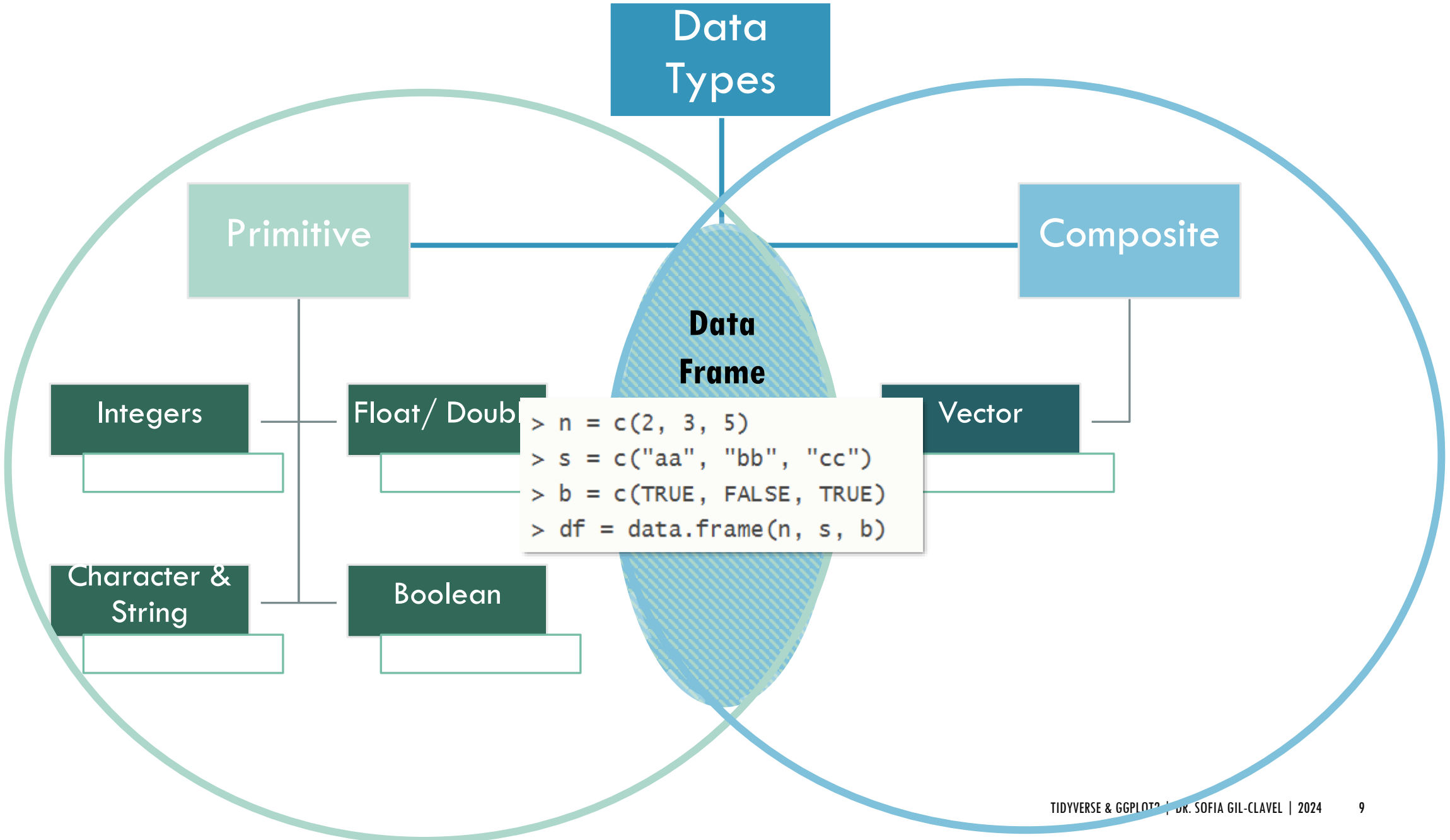

Data Types
Primitive


Data Types
Primitive
Integers
Float/ Double
Character & String
Boolean


Data Types
Primitive
Composite
Integers
Float/ Double
Character & String
Boolean


Data Types
Primitive
Composite
Integers
Float/ Double
Character & String
Boolean
Vector


Data Types
Primitive
Composite
Integers
Float/ Double
?
Vector
Character & String
Boolean


Data Types
Primitive
Composite
Data Frame
Integers
Float/ Doubl
Vector
Character & String
Boolean
> n = c(2, 3, 5)
> s = c("aa", "bb", "cc")
> b = c(TRUE, FALSE, TRUE)
> df = data.frame(n, s, b)

PRIMITIVE DATA AND ITS OPERATORS

Operators					
Arithmetic		**Comparative**		**Boolean**	
+	addition	<	less than	! x	logic NO
-	subtraction	>	more than	x & y	element-wise AND
*	multiplication	<=	less or equal than	x && y	single comparison AND
/	division	>=	more or equal than	x | y	element-wise OR
^	power	==	equal than	x || y	single comparison OR
%/%	integer division	!=	different than	xor(x,y)	exclusive OR

DATA FRAMES: OPEN AND SAVE

With the *foreign* library you can manipulate databases other than ".csv"

Open

```
DIR="WRITE HERE THE PATH TO THE DATA\\"

CredHist<-read.csv(paste0(DIR,"CredHist.csv"))
```

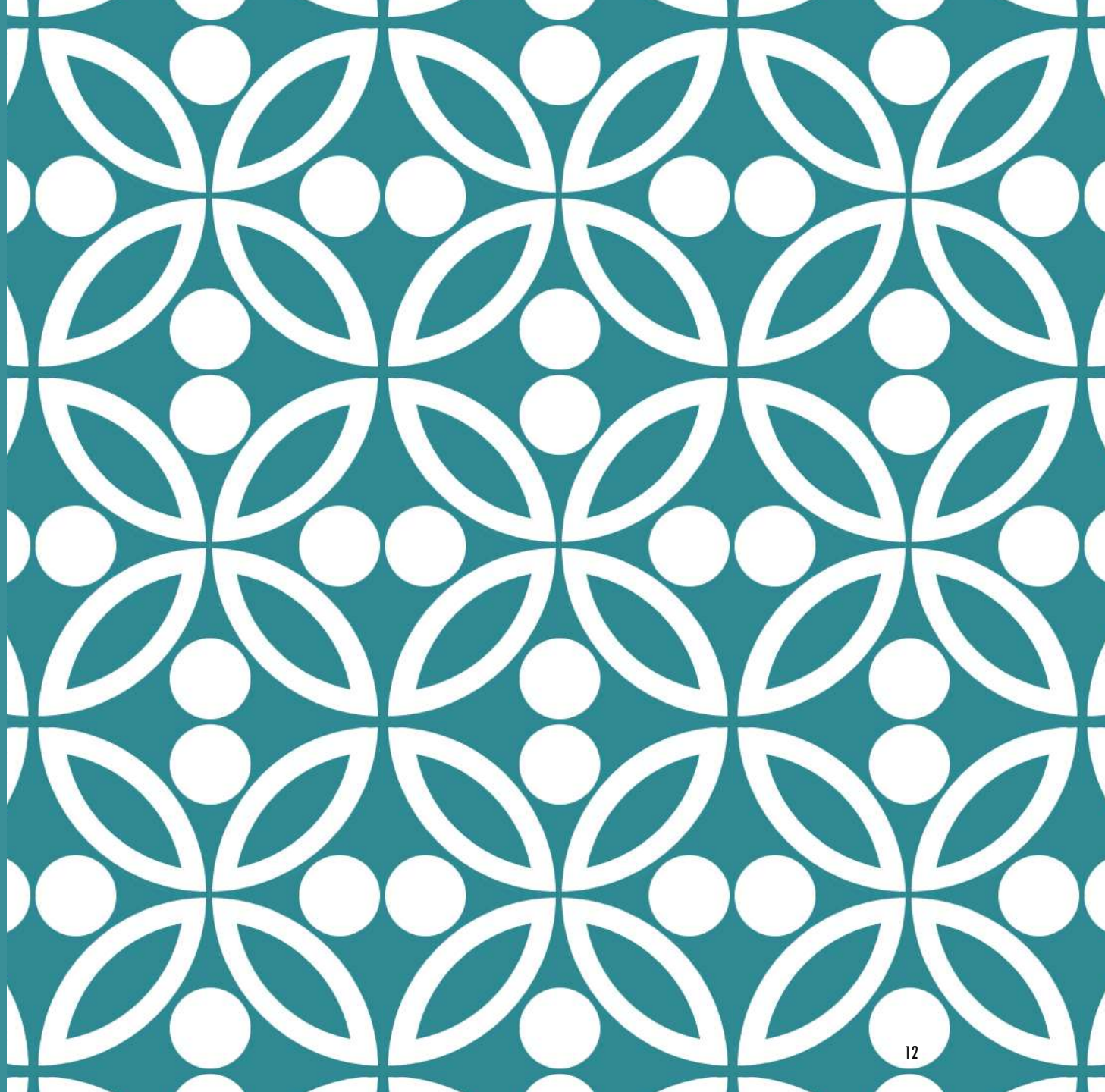
Modify

```
CredHist$HISTORY[CredHist$HISTORY=="bad"]=FALSE
```

Save

```
write.csv(CredHist,paste0(DIR,"CredHist2.csv"),row.names = FALSE)
```

2. TIDYVERSE



DATA PIPELINES

1.


```
> iris
# A tibble: 150 × 5
   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
          <dbl>       <dbl>        <dbl>       <dbl> <fct>
 1          5.1         3.5          1.4         0.2 setosa
 2          4.9         3            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           3.6          1.4         0.2 setosa
 6          5.4         3.9          1.7         0.4 setosa
 7          4.6         3.4          1.4         0.3 setosa
 8          5           3.4          1.5         0.2 setosa
 9          4.4         2.9          1.4         0.2 setosa
10          4.9         3.1          1.5         0.1 setosa
# … with 140 more rows
# ℹ Use `print(n = ...)` to see more rows
```

2.

Basic piping

- `x %>% f` is equivalent to `f(x)`
- `x %>% f(y)` is equivalent to `f(x, y)`
- `x %>% f %>% g %>% h` is equivalent to `h(g(f(x)))`

3.

dplyr is a grammar of data manipulation, providing a consistent set of verbs that help you solve the most common data manipulation challenges:

- `mutate()` adds new variables that are functions of existing variables
- `select()` picks variables based on their names.
- `filter()` picks cases based on their values.
- `summarise()` reduces multiple values down to a single summary.
- `arrange()` changes the ordering of the rows.

EXERCISE 2.1

What do the following functions do?

pipe (%>%)

```
iris%>%
    function()
```

select

```
iris%>%
  select(Sepal.Length,Species)
```

mutate

```
iris%>%
  mutate(SL_=
    Sepal.Length<=median(Sepal.Length))
```

arrange

```
iris%>%
  arrange(Sepal.Length)
```

filter

```
iris%>%
  filter(Species=="setosa")
```

summarise

```
iris%>%
  summarise(mean =
    mean(Sepal.Length), n = n())
```

EXERCISE 2.2

In the GitHub repository, you will find CredHist.csv, for this exercise you will have to create the code in R to be able to **open and find the number of people who can have a loan with the bank according to the following restrictions**:

a) The person does not have a bad credit history.

b) The person has a monthly income greater than 1000EUR.

c) The person is not older than 65 years old.

The bank applies the following exceptions:

- If you are an employee of the bank, you are granted.
- If you are over the age of 65, but your monthly income exceeds 5000EUR, you are awarded.

Once the people have been found, the subbase must be saved in a new ".csv". What are the dimensions of the resulting base? What kind of data did you have to work with?

Group by one or more variables

Source: `R/group-by.R`

Most data operations are done on groups defined by variables. `group_by()` takes an existing tbl and converts it into a grouped tbl where operations are performed "by group". `ungroup()` removes grouping.

```
group_by(.data, ..., .add = FALSE, .drop = group_by_drop_default(.data))

ungroup(x, ...)
```

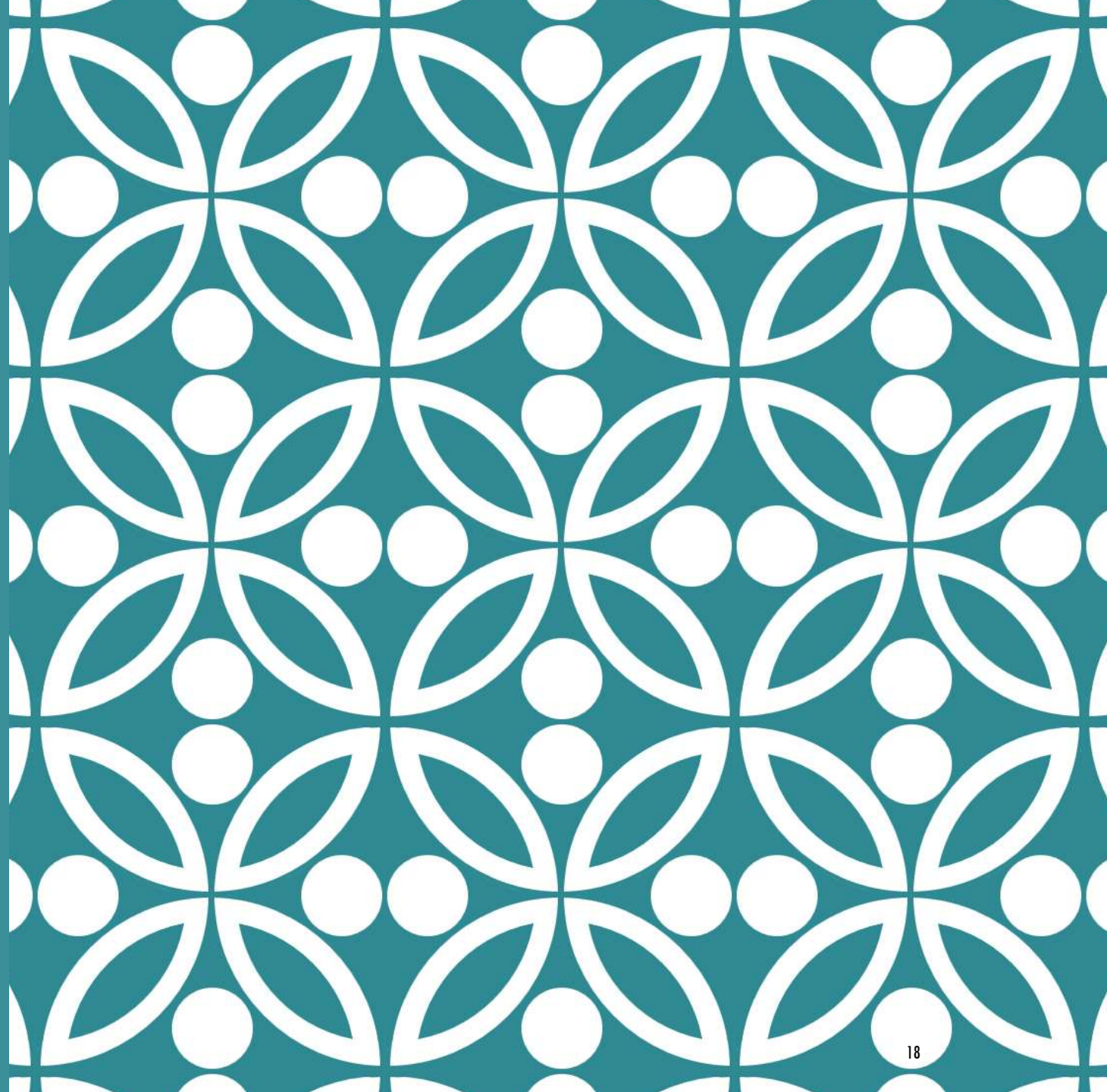
Source: https://dplyr.tidyverse.org/reference/group_by.html

EXERCISE 2.3

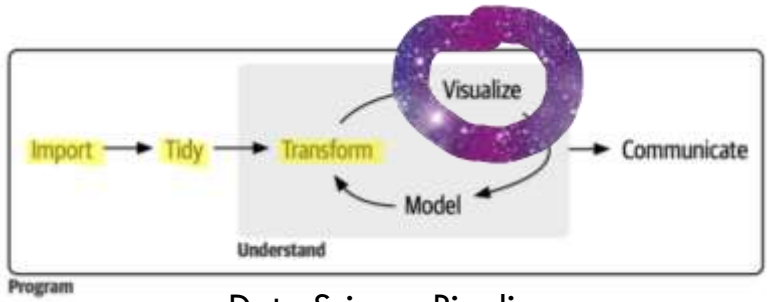
With the subbase that was found in "Exercise 2.2" find the following data:

1. Number of Men and Women Who Can Have the Credit.
2. Average Age of Men and Women.
3. Median monthly Income of Women and Men Over 25.
4. Number of Bank Female and Male employees in the database.
5. Average monthly Income of Female and Male Bank Employees.

3. GGPLOT2 & THE GRAMMAR OF GRAPHS



❖ GGPLOT: HTTPS://WWW.TIDYVERSE.ORG/



Data Science Pipeline

ggplot2 is a system for declaratively creating graphics, based on The Grammar of Graphics. You provide the data, tell ggplot2 how to map variables to aesthetics, what graphical primitives to use, and it takes care of the details.

Source: https://ggplot2.tidyverse.org/

❖ GGPLOT2

ggplot2 is a package for creating graphs, which follows the grammar of graphs.

What is the grammar of graphs?

The grammar of graphs tells us that "... Every statistical graph is a mapping of data to a set of geometric objects (points, lines, bars) that contain aesthetic attributes (color, shape, size). The graph can even have statistical transformations of the data, and these are drawn on a specific Cartesian plane."

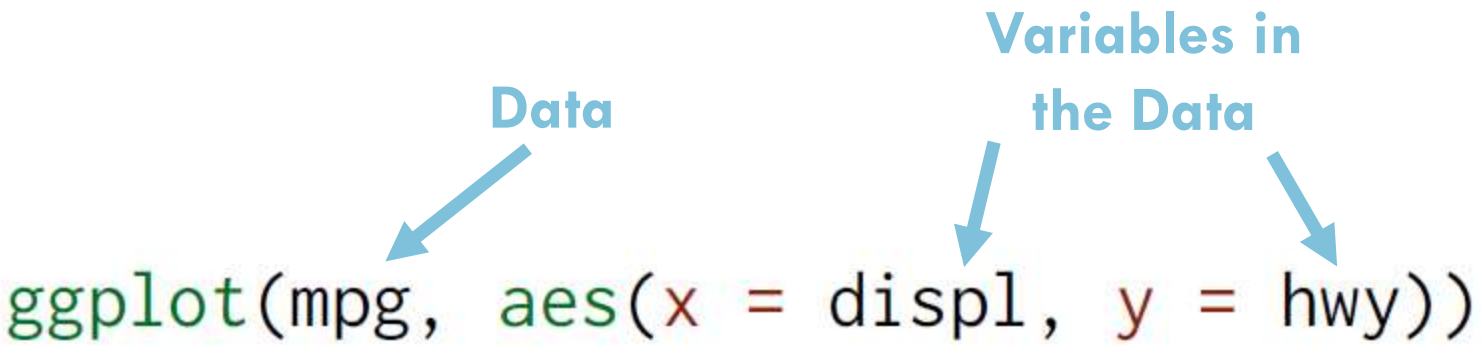
THE COMPONENTS OF A GRAPH

1) The **data** we want to visualize and a set of aesthetic transformations that describe how the variables in the data behave.
2) **Layers** made of geometric objects (which we'll call **geoms**) and statistical transformations (which we'll call **stats**). Geoms represent what you see on the graph: points, lines, polygons, etc. Stats are a summary of the data being observed.
3) **Scales** maps the values of the data to values in an aesthetic space, such as color, size, or shape. Scales draws the legends or axes, which provides reverse mapping that makes it possible to read the original data from the graph.
4) The facet specification (**facet**) describes how to divide data into subsets and how to display it in subgraphs within the same graph. This is also called a conditioner.
5) A **theme** controls the points that are displayed, such as font size and color.

A quick guide is here: https://github.com/rstudio/cheatsheets/blob/main/data-visualization.pdf

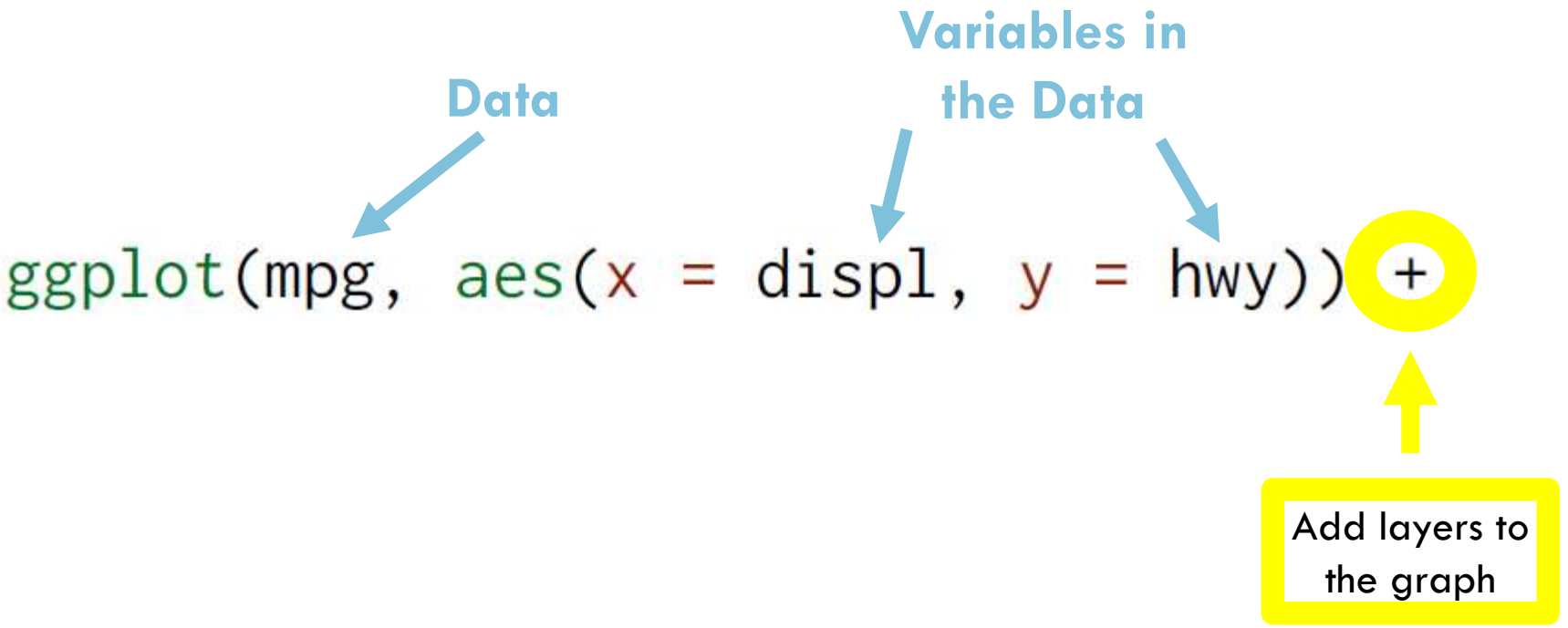
1) THE DATA

```
library(ggplot2)
mpg
```



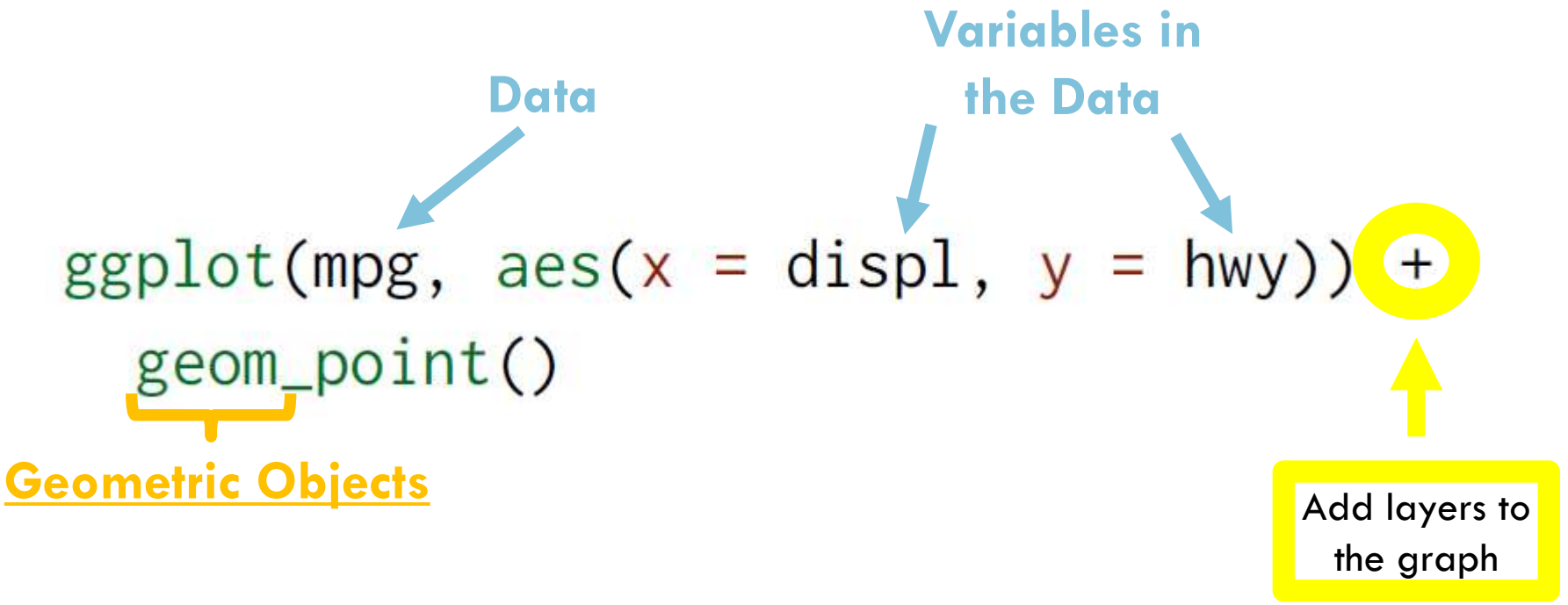
2) LAYERS

```
library(ggplot2)
mpg
```



2) LAYERS

```
library(ggplot2)
mpg
```

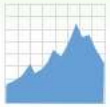


2) LAYERS: BASIC GEOMETRIES

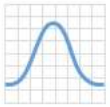
```
library(ggplot2)
mpg
```

ONE VARIABLE continuous

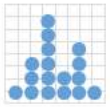
c <- ggplot(mpg, aes(hwy)); c2 <- ggplot(mpg)



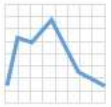
c + geom_area(stat = "bin"**)**
x, y, alpha, color, fill, linetype, size



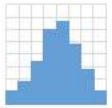
c + geom_density(kernel = "gaussian"**)**
x, y, alpha, color, fill, group, linetype, size, weight



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



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

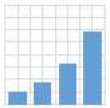


c + geom_histogram(binwidth = 5**)**
x, y, alpha, color, fill, linetype, size, weight

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

discrete

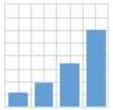
d <- ggplot(mpg, aes(fl))



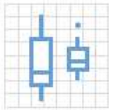
d + geom_bar()
x, alpha, color, fill, linetype, size, weight

one discrete, one continuous

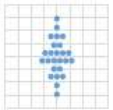
f <- ggplot(mpg, aes(class, hwy))



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

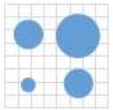


f + geom_dotplot(binaxis = "y", stackdir = "center"**)**
x, y, alpha, color, fill, group

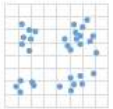
f + geom_violin(scale = "area"**)**
x, y, alpha, color, fill, group, linetype, size, weight

both discrete

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



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



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

Source: https://github.com/rstudio/cheatsheets/blob/main/data-visualization.pdf

EXERCISE 3.1

- From slide 20 choose one graph from the following groups and save them as:

A=*From the "one continues variable"*

B=*From the "one discrete one continuous variables"*

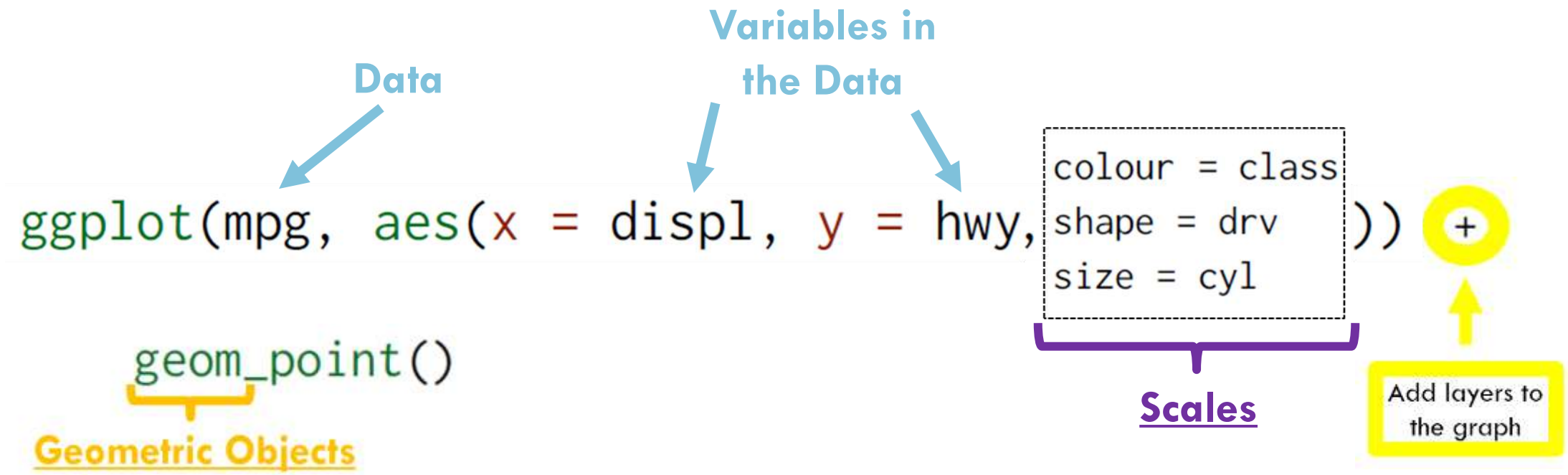
2) LAYERS: A NON-INTUITIVE CASE

What do the following functions do?

```
ggplot(mpg, aes(displ, hwy)) + geom_point(aes(colour = "blue"))
```

```
ggplot(mpg,aes(displ,hwy))+geom_point(aes(colour="blue"))
ggplot(mpg,aes(displ,hwy))+geom_point(aes(colour=class))
ggplot(mpg,aes(displ,hwy))+geom_point(colour="blue")
```

3) SCALES: COLOR, SIZE, SHAPE, AND OTHER AESTHETIC ATTRIBUTES



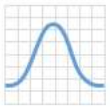
3) SCALES: COLOR, SIZE, SHAPE, ETC.

```
library(ggplot2)
mpg
```

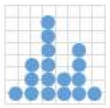
ONE VARIABLE continuous

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

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

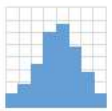


c + geom_density(kernel = "gaussian"**)**
x, y, alpha, color, fill, group, linetype, size, weight

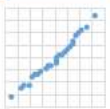


c + geom_dotplot()
x, y, alpha, color, fill

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



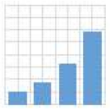
c + geom_histogram(binwidth = 5**)**
x, y, alpha, color, fill, linetype, size, weight



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

discrete

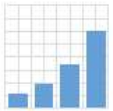
d <- ggplot(mpg, aes(fl))



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

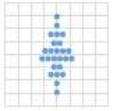
one discrete, one continuous

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

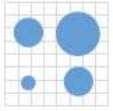


f + geom_dotplot(binaxis = "y", stackdir = "center"**)**
x, y, alpha, color, fill, group

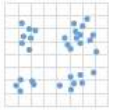
f + geom_violin(scale = "area"**)**
x, y, alpha, color, fill, group, linetype, size, weight

both discrete

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



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



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

Source: https://github.com/rstudio/cheatsheets/blob/main/data-visualization.pdf

EXERCISE 3.2

If possible, add some of the following scales to the graphs you chose in exercise 3.1. How does the graph change when…?

- You add “color”
- You add “shape”
- You add “size”

What happens when you choose more than one scale?

If possible, choose two of the previous scales for each graph created in exercise 3.1 and update the variables A and B.

3) SCALES: SUBGROUPS POSITIONS

- `position_stack()`: stack overlapping bars (or areas) on top of each other.
- `position_fill()`: stack overlapping bars, scaling so the top is always at 1.
- `position_dodge()`: place overlapping bars (or boxplots) side-by-side.

```
dplot <- ggplot(diamonds, aes(color, fill = cut)) +
  xlab(NULL) + ylab(NULL) + theme(legend.position = "none")

dplot + geom_bar()
dplot + geom_bar(position = "fill")
dplot + geom_bar(position = "dodge")
```

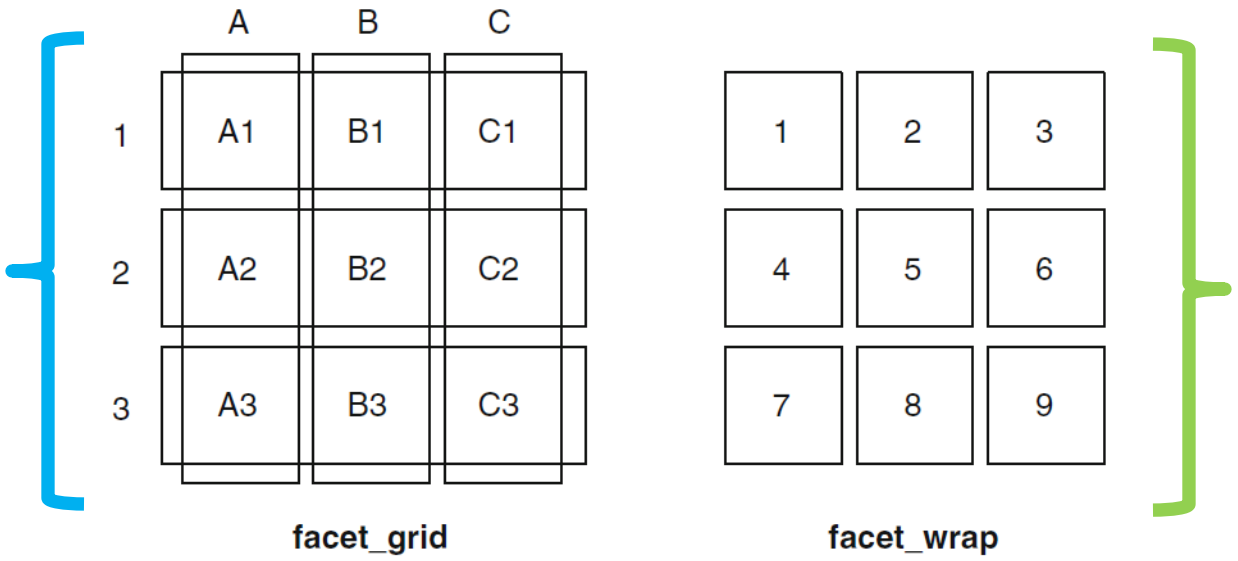
EXERCISE 3.3

- For the "*one discrete*" option (geom_bar) of slide 20, use the different discrete positions. Remember to add the "fill" scale. What do the Y do to the graph?

4) FACETS

- `facet_null()`: a single plot, the default.
- `facet_wrap()`: "wraps" a 1d ribbon of panels into 2d.
- `facet_grid()`: produces a 2d grid of panels defined by variables which form the rows and columns.

They share axes



Each chart has its axes

EXERCISE 3.4

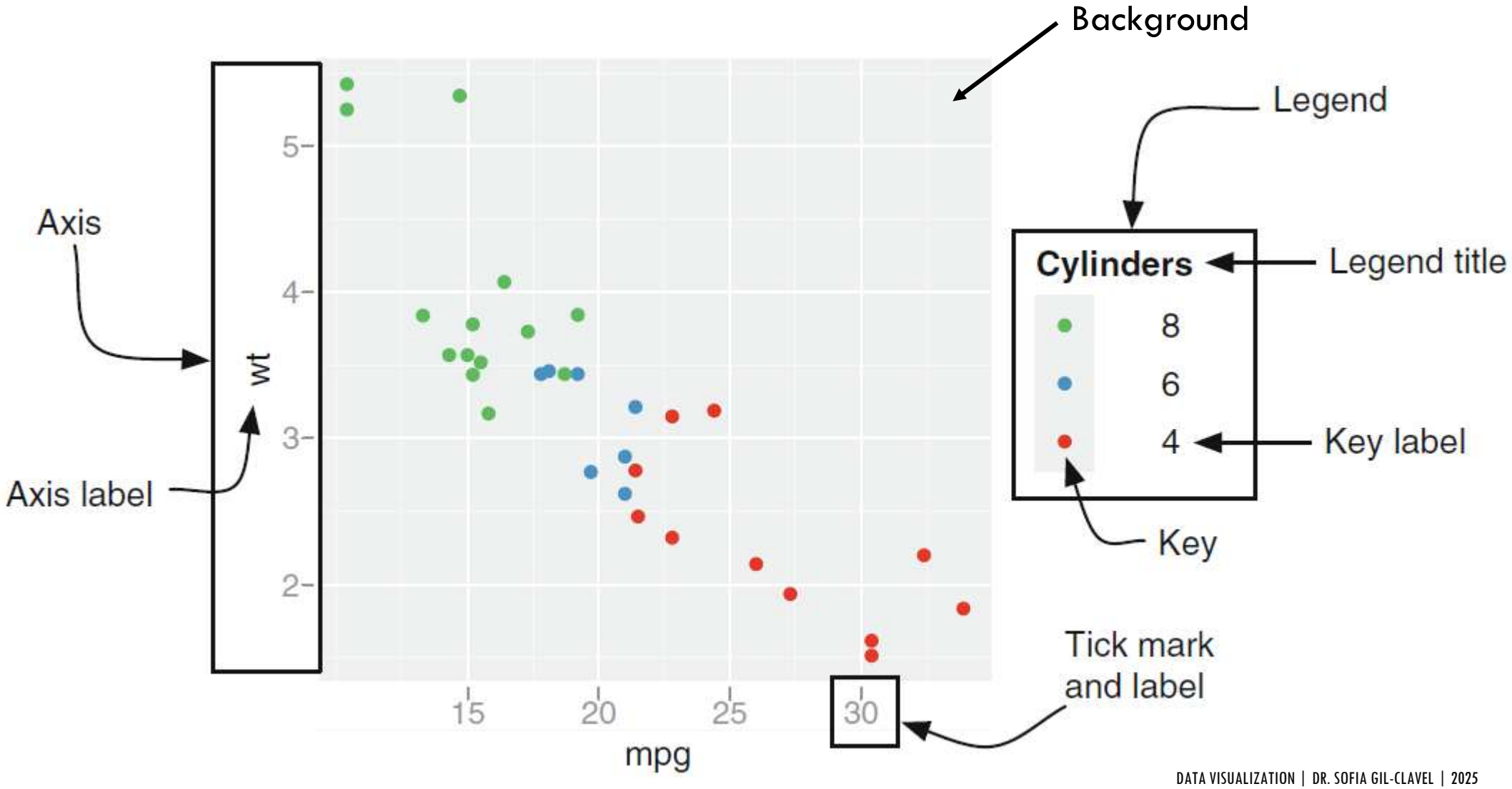
From the **mpg** database, what variables would you choose to group the graph by facet?

Apply the grouping to each of A and B graphs from exercise 3.2.

5) THEMES

Themes allows you to exercise fine **control** over **the non-data elements of your plot.** The theme system does not affect how the data is rendered by geoms, or how it is transformed by scales. Themes don't change the perceptual properties of the plot, but **they do help you make the plot aesthetically pleasing** or match an existing style guide. **Themes give you control over** things like **fonts, ticks, panel strips,** and **backgrounds.**

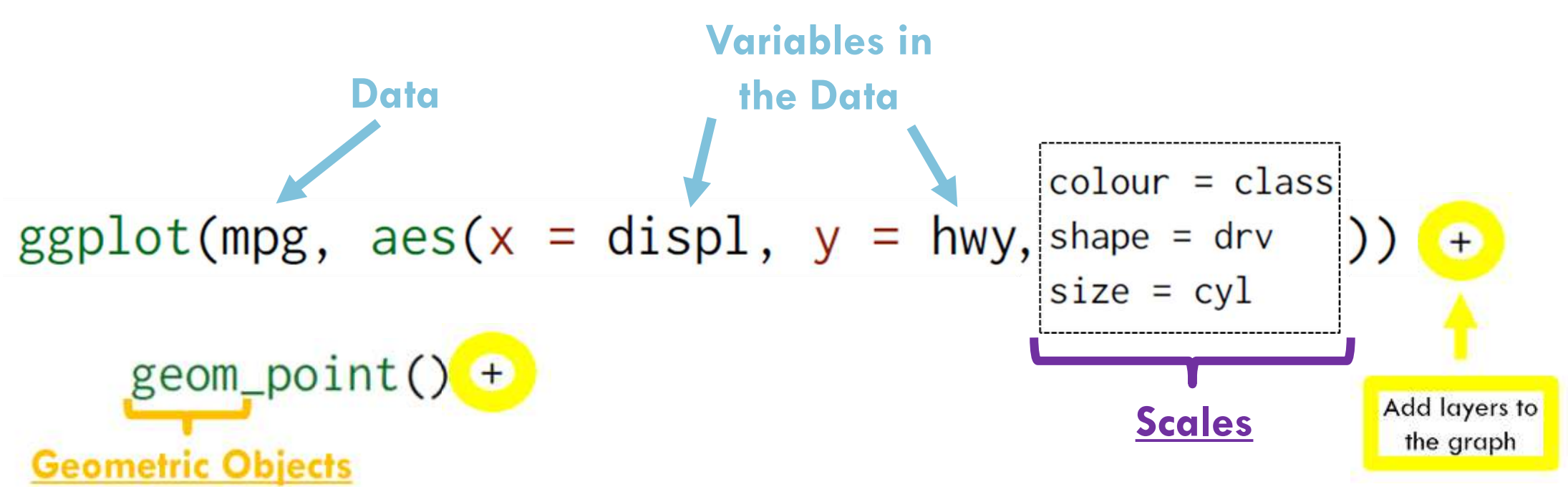
5) THEMES: MODIFYING THE APPEARANCE



5) THEMES: PREDEFINED

- `theme_bw()`: a variation on `theme_grey()` that uses a white background and thin grey grid lines.
- `theme_linedraw()`: A theme with only black lines of various widths on white backgrounds, reminiscent of a line drawing.
- `theme_light()`: similar to `theme_linedraw()` but with light grey lines and axes, to direct more attention towards the data.
- `theme_dark()`: the dark cousin of `theme_light()`, with similar line sizes but a dark background. Useful to make thin coloured lines pop out.
- `theme_minimal()`: A minimalistic theme with no background annotations.
- `theme_classic()`: A classic-looking theme, with x and y axis lines and no gridlines.
- `theme_void()`: A completely empty theme.

5) THEMES FROM SCRATCH




```
theme(
  plot.title = element_text(face = "bold", size = 12),
  legend.background = element_rect(fill = "white", size = 4, colour = "white"),
  legend.justification = c(0, 1),
  legend.position = c(0, 1),
  axis.ticks = element_line(colour = "grey70", size = 0.2),
  panel.grid.major = element_line(colour = "grey70", size = 0.2),
  panel.grid.minor = element_blank()
)
```

EXERCISE 3.5

Using the different **themes'** elements on your A and B graphs, create the most horrible graph anyone has ever seen and update the variables A and B.

SAVING THE GRAPHS

ggplot2 provides a convenient shorthand to save graphs created with ggplot: **ggsave()**

```
ggplot(mpg, aes(displ, cty)) + geom_point()
ggsave("output.pdf")
```

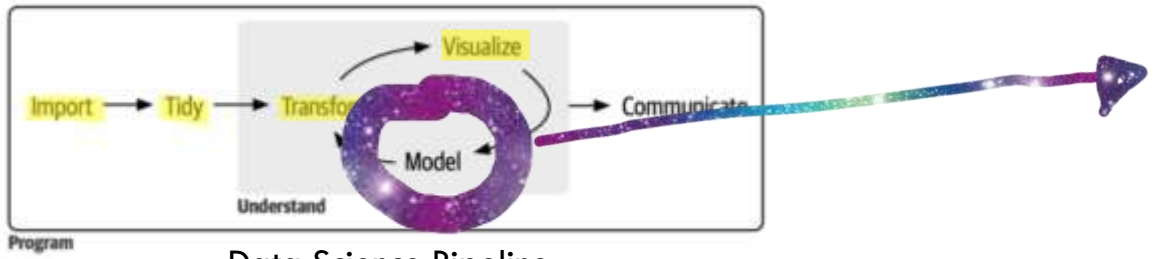
EXERCISE 3.6

- In the “help” section, check the parameters of **ggsave.**
- Use **ggsave** to save your graphs A and B. Play around with the parameters until you manage to save your graph in a good quality format.

REFERENCES

- Albert, Jim, and Maria Rizzo. *R by Example: Concepts to Code*. Use R! New York, NY: Springer New York, 2012. https://doi.org/10.1007/978-1-4614-1365-3.
- Davies, Tilman M. *The Book of R: A First Course in Programming and Statistics*. San Francisco: No Starch Press, 2016. https://web.itu.edu.tr/~tokerem/The_Book_of_R.pdf
- Wickham, Hadley, Mine Çetinkaya-Rundel, and Garrett Grolemund. *R for data science*. " O'Reilly Media, Inc.", 2023. Accessed May 7, 2024. https://r4ds.hadley.nz/.
- Wickham, Hadley. *Ggplot2*. Use R! Cham: Springer International Publishing, 2016. https://doi.org/10.1007/978-3-319-24277-4.
- Wilke, C. *Fundamentals of Data Visualization: A Primer on Making Informative and Compelling Figures*. First edition. Sebastopol, CA: O'Reilly Media, 2019. https://clauswilke.com/dataviz/

SESSION 3: STATISTICS



Data Science Pipeline

stats-package {stats} R Documentation

The R Stats Package

Description

R statistical functions

Details

This package contains functions for statistical calculations and random number generation.

For a complete list of functions, use `library(help = "stats")`.

Author(s)

R Core Team and contributors worldwide

Maintainer: R Core Team R-core@r-project.org

THANKS!

Dr. Sofia Gil-Clavel

https://sofiag1l.github.io/