Data visualization

Lecture 4

Louis SIRUGUE

CPES 2 - Fall 2022

Last time we saw

Read data

```
starbucks <- read.csv("C:/User/Documents/folder/starbucks.csv", sep = ";", encoding = "UTF-8")
```

***➜ Make sure to use / and not ***

Chaining operations

```
starbucks %>%
  arrange(-Calories) %>%
  select(Beverage_category, Beverage_prep, Calories) %>%
  filter(row_number() <= 3)
```

```
##             Beverage_category Beverage_prep Calories
## 1   Signature Espresso Drinks       2% Milk      510
## 2   Signature Espresso Drinks       Soymilk      460
## 3 Frappuccino® Blended Coffee    Whole Milk      460
```

➜ Make sure to view your data at each step

Last time we saw

Important functions of the dplyr grammar

Function	Meaning
mutate()	Modify or create a variable
select()	Keep a subset of variables
filter()	Keep a subset of observations
arrange()	Sort the data
group_by()	Group the data
summarise()	Summarizes variables into 1 observation per group
bind_rows()	Append data
left/right/inner/full_join()	Merge data
pivot_longer/wider()	Reshape data

Today we learn how to plot data

1. The `ggplot()` function

- 1.1. Basic structure
- 1.2. Axes
- 1.3. Theme
- 1.4. Annotation

2. Adding dimensions

- 2.1. More axes
- 2.2. More facets
- 2.3. More labels

3. Types of geometry

- 3.1. Points and lines
- 3.2. Barplots and histograms
- 3.3. Densities and boxplots

4. How (not) to lie with graphics

- 4.1. Cumulative representations
- 4.2. Axis manipulations
- 4.3. Interpolation

5. Wrap up!

Today we learn how to plot data

1. The `ggplot()` function

- 1.1. Basic structure
- 1.2. Axes
- 1.3. Theme
- 1.4. Annotation

1. The `ggplot()` function

1.1. Basic structure

- **ggplot()** is the function that we're gonna use for all our plots
- It takes the following **core arguments**:
  - **Data**: the values to plot
  - **Mapping** (aes, for aesthetics): the structure of the plot
  - **Geometry**: the type of plot

- **Data and mapping** should be specified within the **parentheses**
- **Geometry** and any **other element** should be added with a **+** sign

```
ggplot(data, aes()) + geometry + anything_else
```

- You can also **apply** the `ggplot()` function to your data with a **pipe**

```
data %>% ggplot(aes()) + geometry
```

1. The `ggplot()` function

1.1. Basic structure: Example

```
test_data <- tibble(V1 = 1:6, V2 = c(64, 60, 16, 8, 16, 32))
```

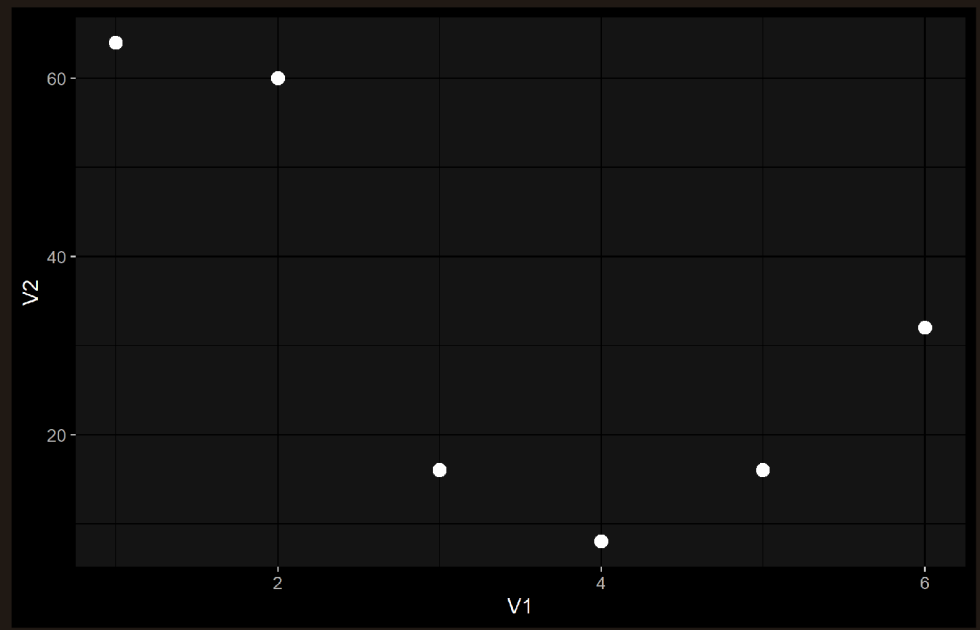
```
ggplot(test_data, aes(x = V1, y = V2)) + geom_point(size = 3)
```

- We first specified our **data**:

V1	1	2	3	4	5	6
V2	64	60	16	8	16	32

- Then assigned in **aes()**
  - V1 to the x-axis
  - V2 to the y-axis

- And chose the **point geometry** with a size of 3



1. The `ggplot()` function

1.2. Axes

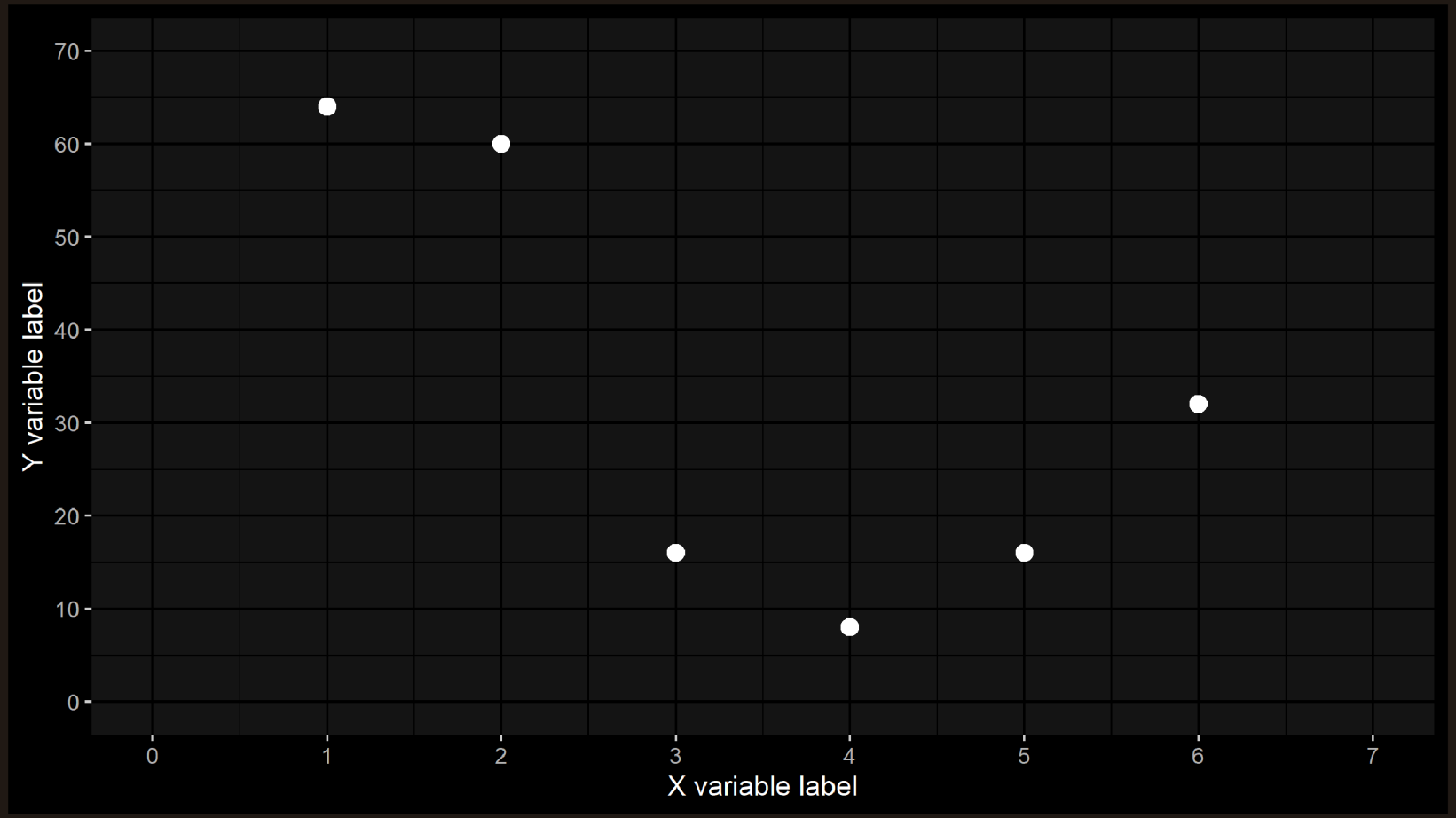
- Axes can be modified using with **scale functions**:
  - **scale_x_continuous()**: when a **continuous** variable is assigned to the **x-axis**
  - **scale_x_discrete()**: when a **discrete** variable is assigned to the **x-axis**
  - **scale_y_continuous()**: when a **continuous** variable is assigned to the **y-axis**
  - **scale_y_discrete()**: when a **discrete** variable is assigned to the **y-axis**

- The following **parameters** can be modified in these scale functions:
  - **name:** The label of the corresponding axis
  - **limits:** Where the axis should start and end
  - **breaks:** Where to put ticks and values on the axis

```
ggplot(test_data, aes(x = V1, y = V2)) + geom_point(size = 3) +
  scale_x_continuous(name = "X variable label", limits = c(0, 7), breaks = 0:7) +
  scale_y_continuous(name = "Y variable label", limits = c(0, 70), breaks = seq(0, 70, 10))
```

1. The `ggplot()` function

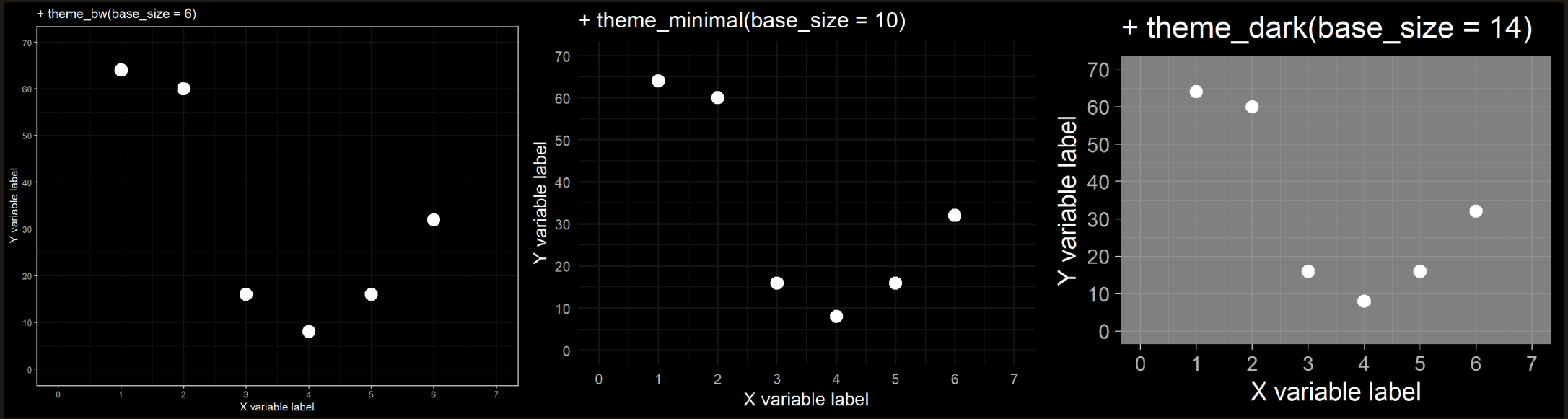
1.2. Axes



1. The `ggplot()` function

1.3. Theme()

- You can use one of the **default R themes** to easily change the layout of your plot
  - ... + theme_**bw**()
  - ... + theme_**minimal**()
  - ... + theme_**dark**()
  - You can also tune the **font size** inside these functions with the **base_size** argument



1. The `ggplot()` function

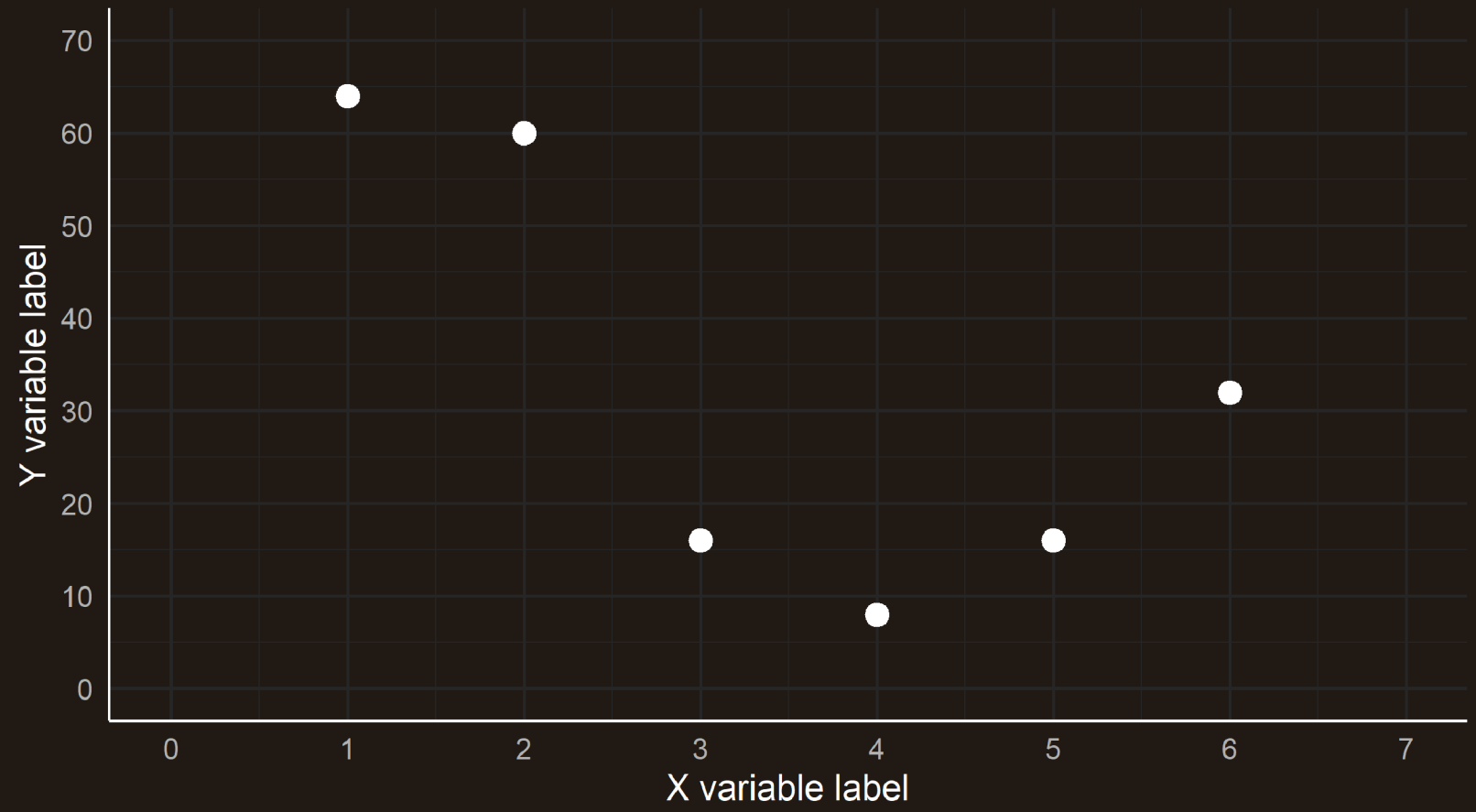
1.3. Theme()

- You can also custom your graph using the **theme()** function
  - It allows to **custom** virtually **anything**
  - Enter ?theme to see the **endless** list of possible **arguments**
  - Obviously we won't go through all of them but here are a few

```
ggplot(test_data, aes(x = V1, y = V2)) +
  geom_point(size = 4) +
  # Axes
  scale_x_continuous(name = "X variable label", limits = c(0, 7), breaks = 0:7) +
  scale_y_continuous(name = "Y variable label", limits = c(0, 70), breaks = seq(0, 70, 10)) +
  # Theme
  theme_minimal(base_size = 14) +
  theme(# Color of the background and of its border
        plot.background = element_rect(fill = "#DFE6EB", colour = "#DFE6EB"),
        # Size of the axis lines
        axis.line = element_line(size = rel(0.8)),
        # Color of the grid lines
        panel.grid = element_line(color = "gray85"))
```

1. The `ggplot()` function

1.3. Theme()

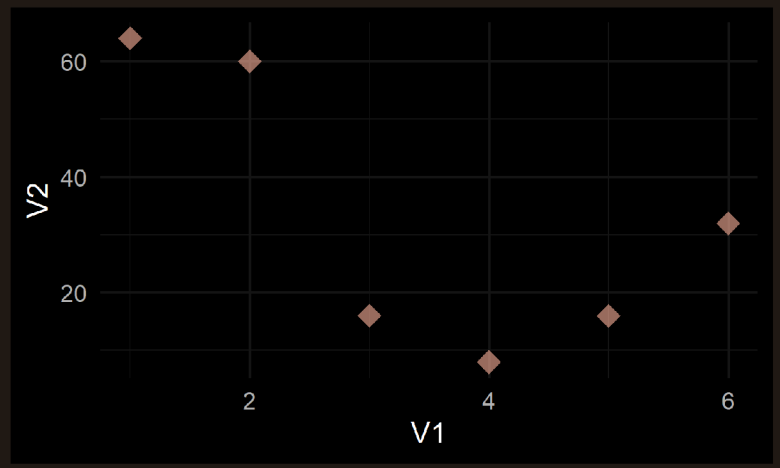


1. The `ggplot()` function

1.3. Theme()

- Geometries can also be modified
  - **alpha:** opacity from 0 to 1
  - **color:** color of the geometry (for geometries that are filled such as bars, it will color the border)
  - **fill:** fill color for geometries such as bars
  - **size:** size of the geometry
  - **shape:** change shape for geometries like points
  - **linetype:** solid, dashed, dotted, etc., for line geometries
  - ...

```
ggplot(test_data, aes(x = V1, y = V2)) +
  geom_point(size = 3,
             color = "#6794A7",
             alpha = .6,
             shape = 18)  +
  theme_minimal(base_size = 14)
```



1. The `ggplot` function

1.4. Annotation

- It is sometimes useful to **annotate** a graph so that certain things become **more salient**
  - **Separate** two groups with a **dashed line**
  - Add a few **words somewhere** for clarity
  - **Circle** a specific group of **data points**
  - Add **labels** to data points

- **Straight lines** can easily be added with their respective geometry

```
+ geom_hline(yintercept = , linetype = )
```

```
+ geom_vline(xintercept = , linetype = )
```

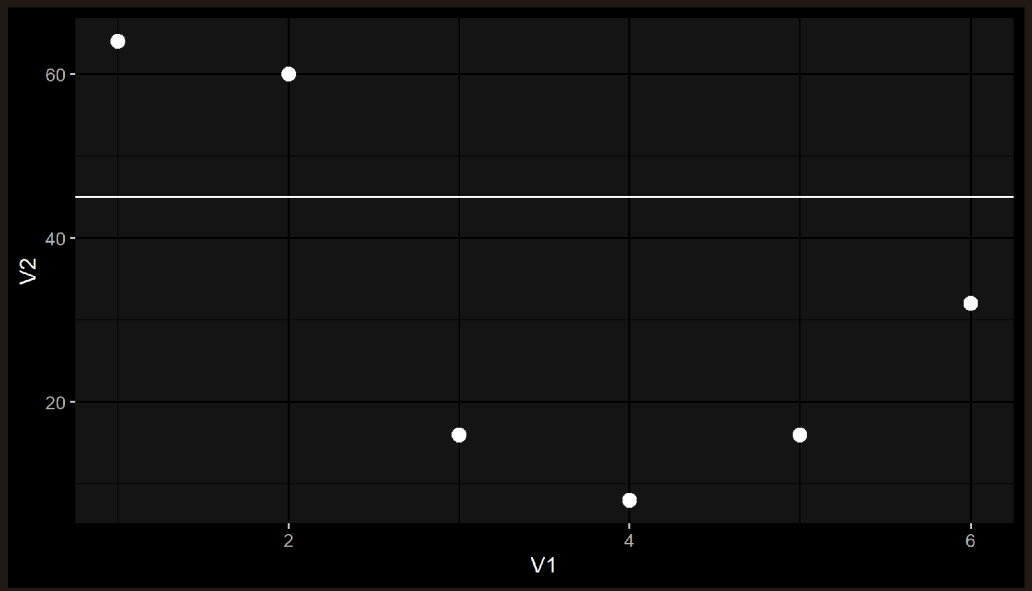
- And **punctual text annotations** can be added with `annotate()`

```
+ annotate("text", x = , y = , label = )
```

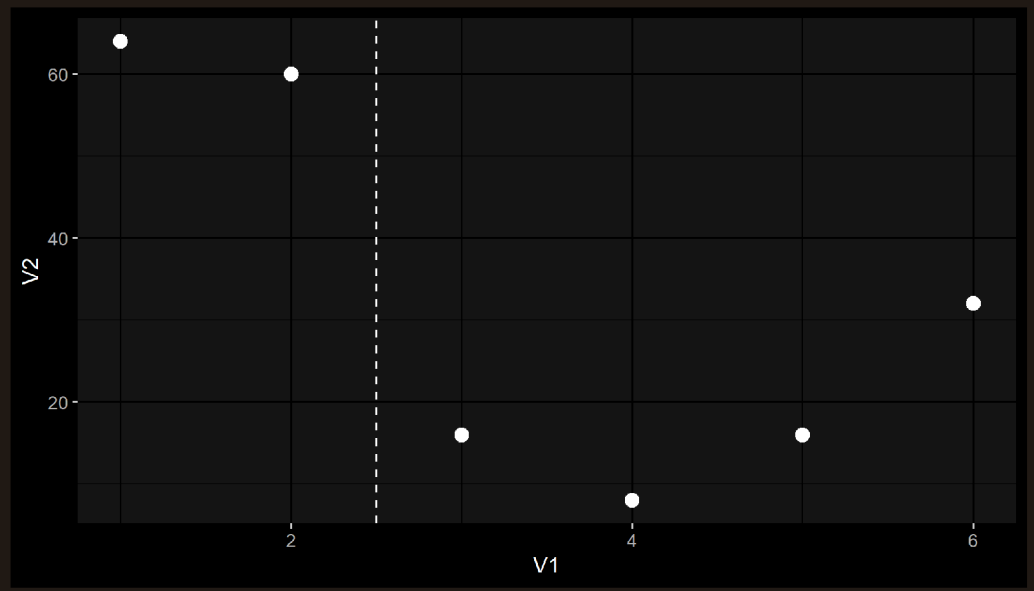
1. The `ggplot` function

1.4. Annotation: Adding lines

```
ggplot(test_data, aes(x = V1, y = V2)) +
  geom_point(size = 3) +
  geom_hline(yintercept = 45)
```



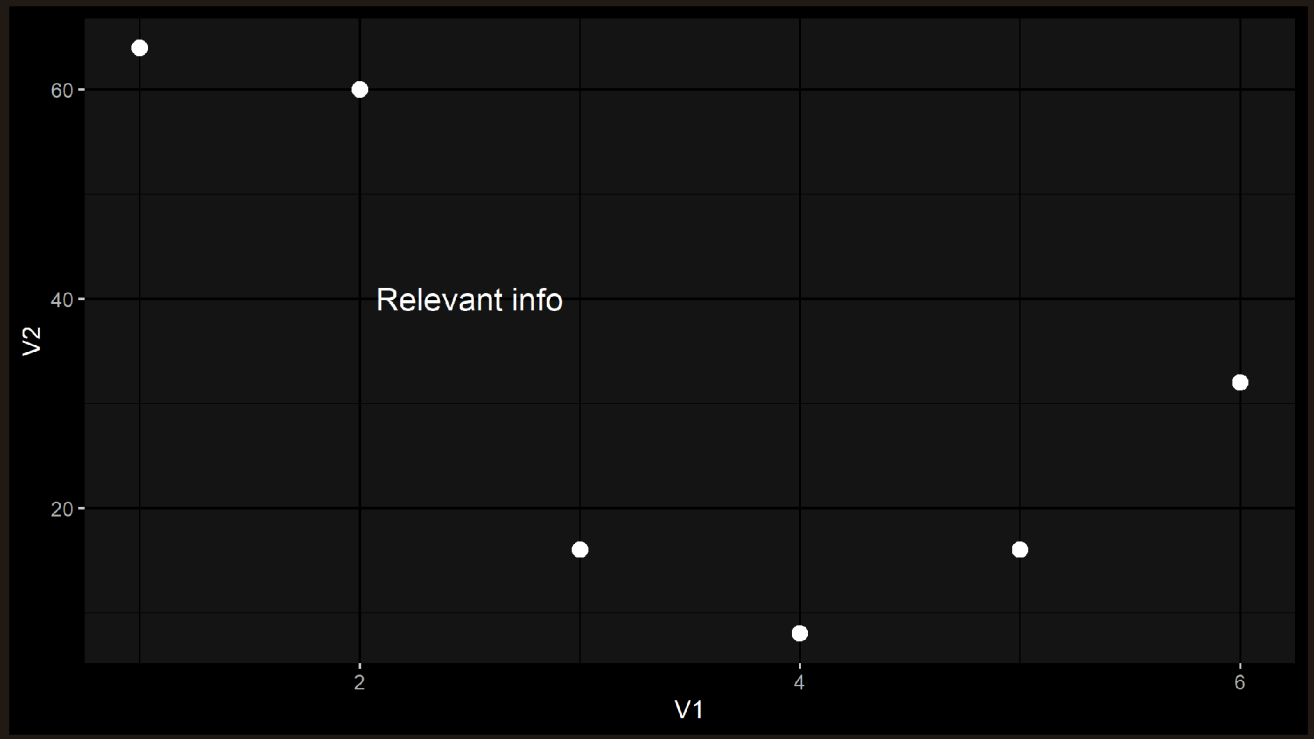

```
ggplot(test_data, aes(x = V1, y = V2)) +
  geom_point(size = 3) +
  geom_vline(xintercept = 2.5,
             linetype = "dashed")
```



1. The `ggplot` function

1.4. Annotation: Adding text

```
ggplot(test_data, aes(x = V1, y = V2)) + geom_point(size = 3) +
  annotate("text", x = 2.5, y = 40, label = "Relevant info", size = 5)
```



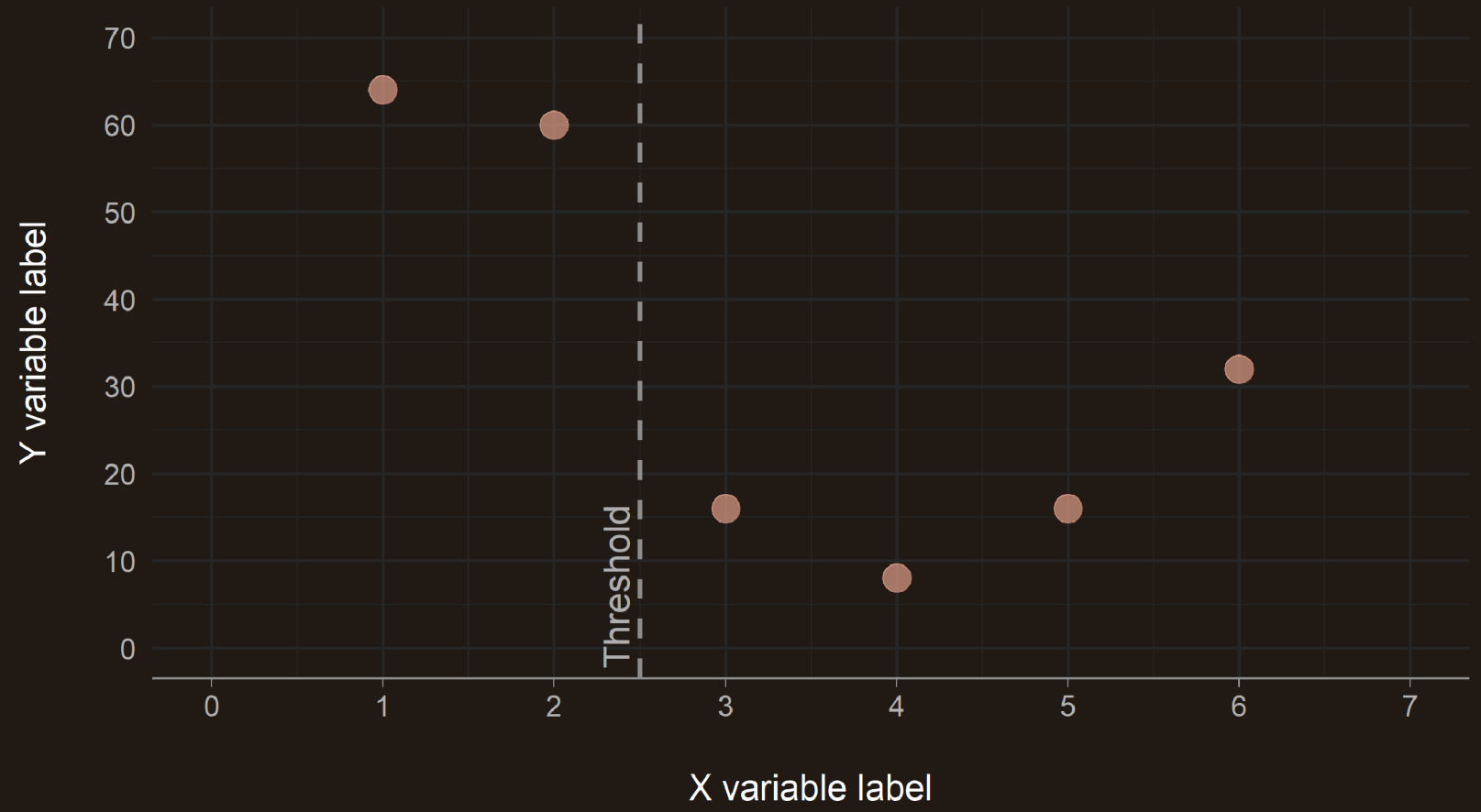
1. The `ggplot` function

Combining everything

```
ggplot(test_data, aes(x = V1, y = V2)) +
  geom_point(size = 5, color = "#014D64", alpha = .6) +
  # Annotations
  geom_vline(xintercept = 2.5, linetype = "dashed", size = 1, color = "#727272") +
  annotate("text", x = 2.36, y = 7, label = "Threshold", angle = 90, size = 5, color = "#727272") +
  # Axes
  scale_x_continuous(name = "X variable label", limits = c(0, 7), breaks = 0:7) +
  scale_y_continuous(name = "Y variable label", limits = c(0, 70), breaks = seq(0, 70, 10)) +
  # Theme
  theme_minimal(base_size = 14) +
        # Custom background
  theme(plot.background = element_rect(fill = "#DFE6EB", colour = "#DFE6EB"),
        panel.grid = element_line(color = "gray85"),
        # Custom axes
        axis.line.x = element_line(size = rel(0.8), color = "#727272"),
        axis.ticks.x = element_line(size = rel(0.4), color = "#727272"),
        axis.line.y = element_blank(), axis.ticks.y = element_blank(),
        axis.title.y = element_text(margin = margin(t = 0, r = 20, b = 0, l = 0)),
        axis.title.x = element_text(margin = margin(t = 20, r = 0, b = 0, l = 0)),
```

1. The `ggplot` function

Combining everything



Overview

1. The `ggplot()` function ✓

- 1.1. Basic structure
- 1.2. Axes
- 1.3. Theme
- 1.4. Annotation

2. Adding dimensions

- 2.1. More axes
- 2.2. More facets
- 2.3. More labels

3. Types of geometry

- 3.1. Points and lines
- 3.2. Barplots and histograms
- 3.3. Densities and boxplots

4. How (not) to lie with graphics

- 4.1. Cumulative representations
- 4.2. Axis manipulations
- 4.3. Interpolation

5. Wrap up!

Overview

1. The `ggplot()` function ✓

- 1.1. Basic structure
- 1.2. Axes
- 1.3. Theme
- 1.4. Annotation

2. Adding dimensions

- 2.1. More axes
- 2.2. More facets
- 2.3. More labels

2. Adding dimensions

2.1. More axes

- We now know how to produce a **scatter plot** involving two variables, x and y
  - But often times it may be useful to add information from a **third variable**

- To illustrate that, let's open an **actual dataset**
  - These data come from the World Inequality database

```
wid <- read.csv("wid.csv")
```

- It contains the following variables
  - **Observation level**: Continent/Country/Year
  - **f_share**: Female labor income share
  - **top1**: Top 1% income share
  - **inc_head**: Per adult national income

```
names(wid)
```

```
## [1] "country"   "continent" "year"      "fshare"    "top1"      "inc_head"
```

2. Adding dimensions

2.1. More axes

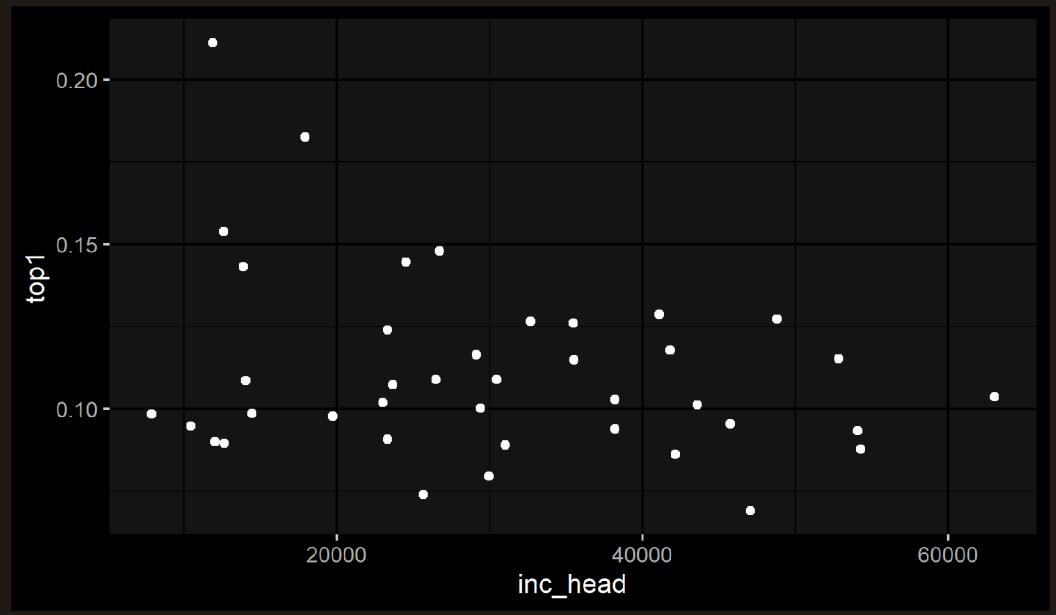
	country	continent	year	fshare	top1	inc_head
1	Algeria	Africa	2010	0.0992	0.1003	12610.6266
2	Algeria	Africa	2011	0.1120	0.0991	12619.9841
3	Algeria	Africa	2012	0.1201	0.0991	12634.0257
4	Algeria	Africa	2013	0.1206	0.0991	12531.9880
5	Algeria	Africa	2014	0.1160	0.0991	12546.4300
6	Algeria	Africa	2015	0.1221	0.0991	12533.3886
7	Algeria	Africa	2016	0.1232	0.0991	12955.5190
8	Algeria	Africa	2017	0.1282	0.0991	12740.5483
9	Algeria	Africa	2018	0.1265	0.0991	12590.2065
10	Algeria	Africa	2019	0.1248	0.0991	12533.5310
11	Angola	Africa	2010	0.2664	0.1744	11180.7619
12	Angola	Africa	2011	0.2707	0.1851	11251.6206
13	Angola	Africa	2012	0.2682	0.1958	11898.0819
14	Angola	Africa	2013	0.2631	0.2065	12194.3769
15	Angola	Africa	2014	0.2614	0.2172	12523.9286
16	Angola	Africa	2015	0.2610	0.2278	12344.2155
17	Angola	Africa	2016	0.2658	0.2385	11565.1393
18	Angola	Africa	2017	0.2677	0.2491	11044.4066

2. Adding dimensions

2.1. More axes

- Let's start with a **simple scatterplot** for European countries in 2019
  - **Female labor income share** on the y axis, and **national income** per adult on the x axis

```
wid %>% filter(year == 2019 & continent == "Europe") %>%
  ggplot(aes(x = inc_head, y = top1)) + geom_point()
```



2. Adding dimensions

2.1. More axes

- In some cases you would **convey information** with other means than a position on axis
  - The **color, size, or shape** of a geometry can be used to represent a **third variable**:

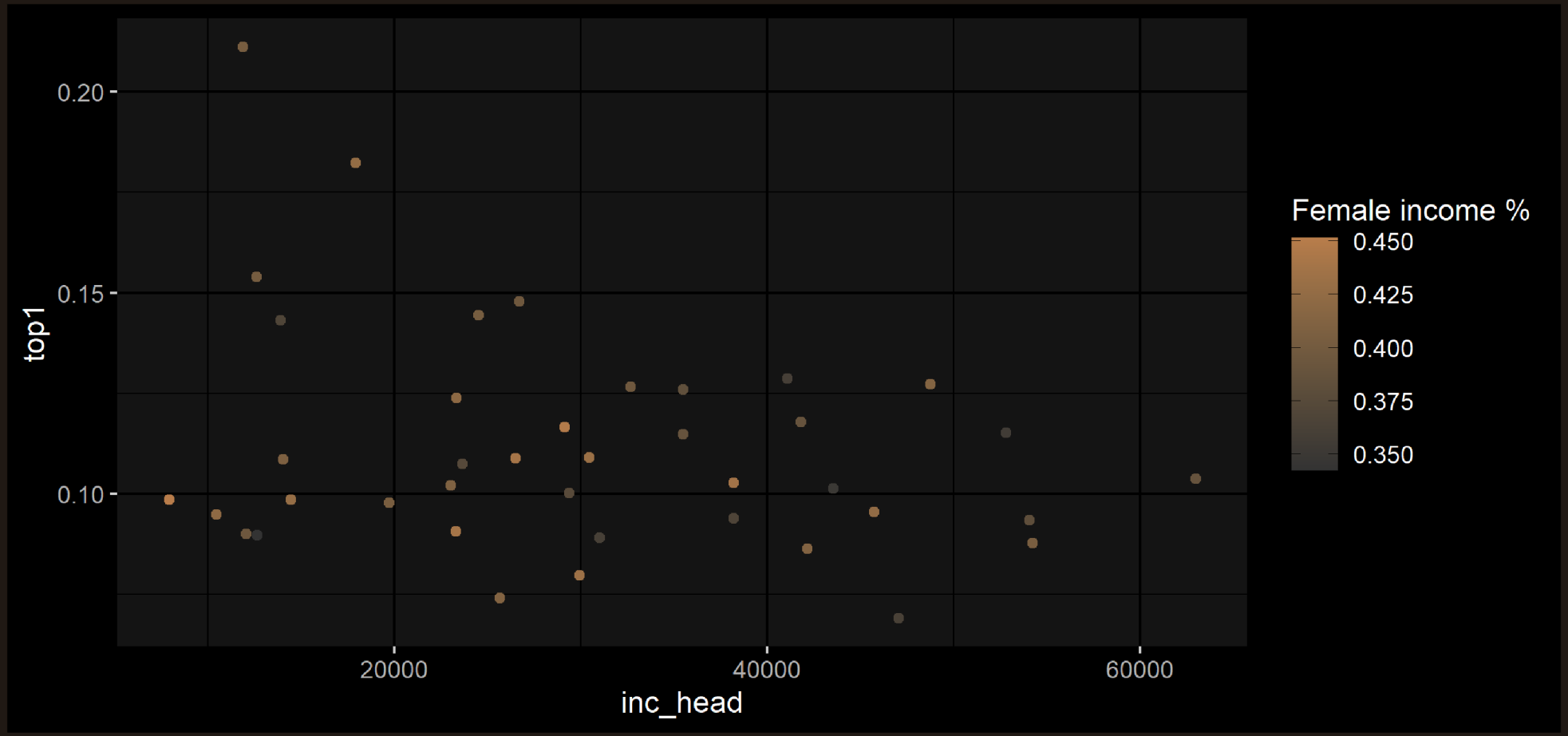
- We can assign a **different color to each point** depending on its female labor income share
  - This allows to see how x and y are related to this variable

- Here, the **color shade** should be viewed as a **dimension** just like x and y
  - fshare should be assigned to the *"color axis"* in **aes()**
  - But because there is no proper *"color axis"*, a **legend** will be generated
  - And just like **scale functions** custom axes of the x and y dimensions...
  - ... scale functions also custom the legend of the color dimension

```
wid %>%
  filter(year == 2019 & continent == "Europe") %>%
  ggplot(aes(x = inc_head, y = top1, color = fshare)) +
  geom_point() +
  scale_color_gradient(name = "Female income %", low = "grey80", high = "steelblue")
```

2. Adding dimensions

2.1. More axes



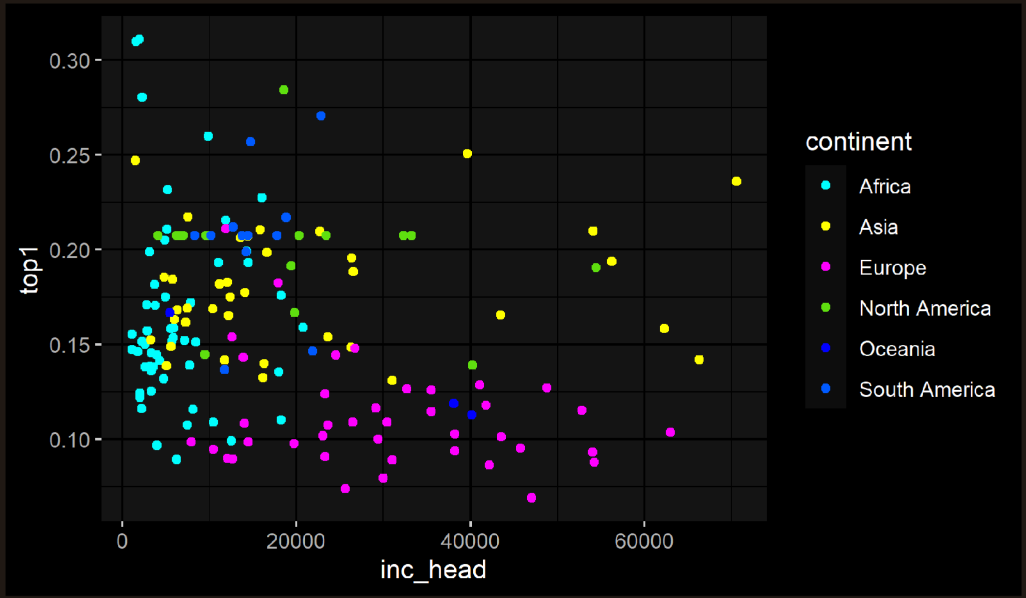
2. Adding dimensions

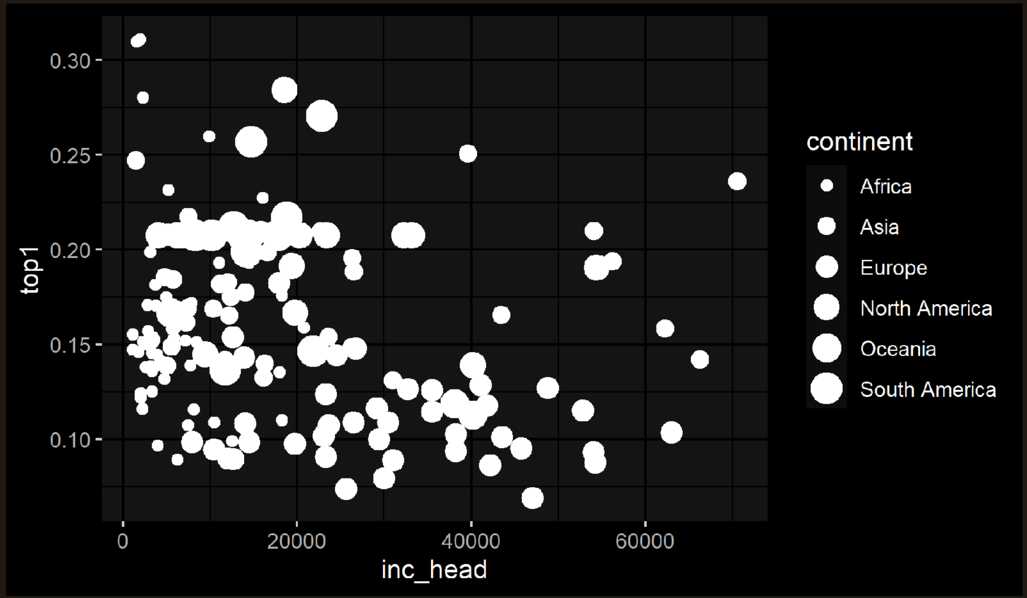
2.1. More axes

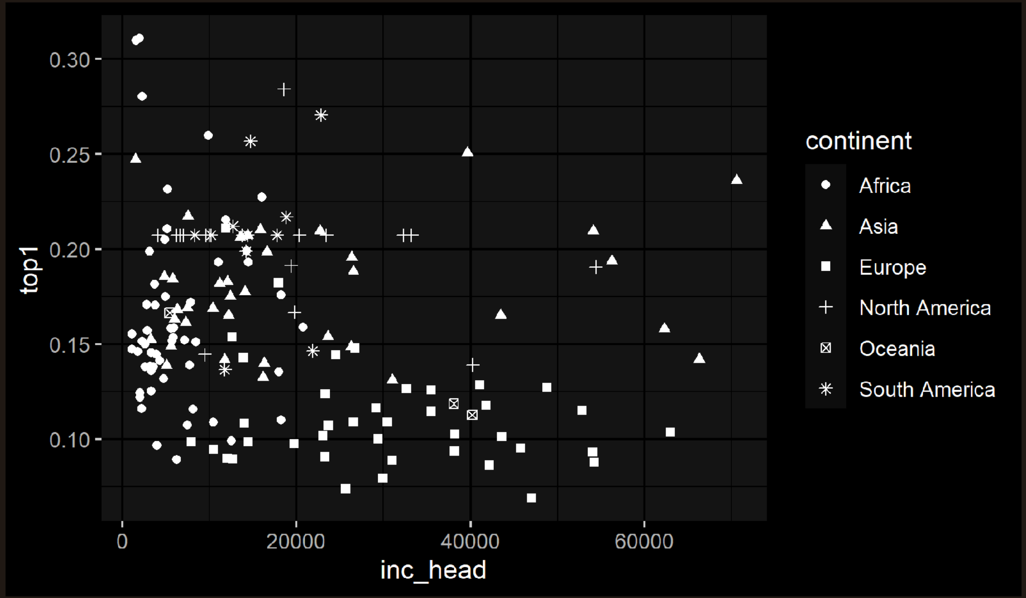
- The color axis can also be used to **distinguish** different **groups**
  - We can indeed assign a **discrete** variable to the **color** axis

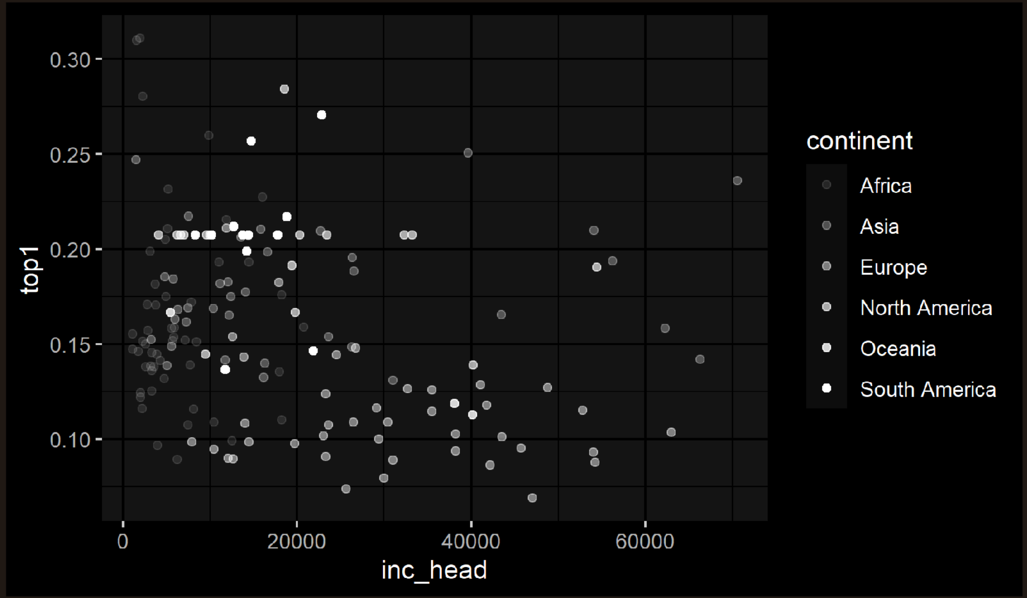
```
wid %>%
  filter(year == 2019) %>%
  ggplot(aes(x = inc_head, y = top1, color = continent)) +
  geom_point() +
  scale_color_manual(values = c("red", "blue", "green", "purple", "yellow", "orange"))
```

- But color is not this only **property** that can be used as a **dimension**, you can use:
  - **fill**
  - **size**
  - **shape**
  - **alpha**
  - **linetype**
  - ...


top1
0.30
0.25
0.20
0.15
0.10
0
20000
40000
60000
inc_head
continent
Africa
Asia
Europe
North America
Oceania
South America


top1
0.30
0.25
0.20
0.15
0.10
0
20000
40000
60000
inc_head
continent
Africa
Asia
Europe
North America
Oceania
South America


top1
0.30
0.25
0.20
0.15
0.10
0
20000
40000
60000
inc_head
continent
Africa
Asia
Europe
North America
Oceania
South America


top1
0.30
0.25
0.20
0.15
0.10
0
20000
40000
60000
inc_head
continent
Africa
Asia
Europe
North America
Oceania
South America

2. Adding dimensions

2.2. More facets

- Another way to **distinguish groups** is to divide the plot into **facets**
  - To do so, indicate your faceting variable into the **facet_wrap()** function

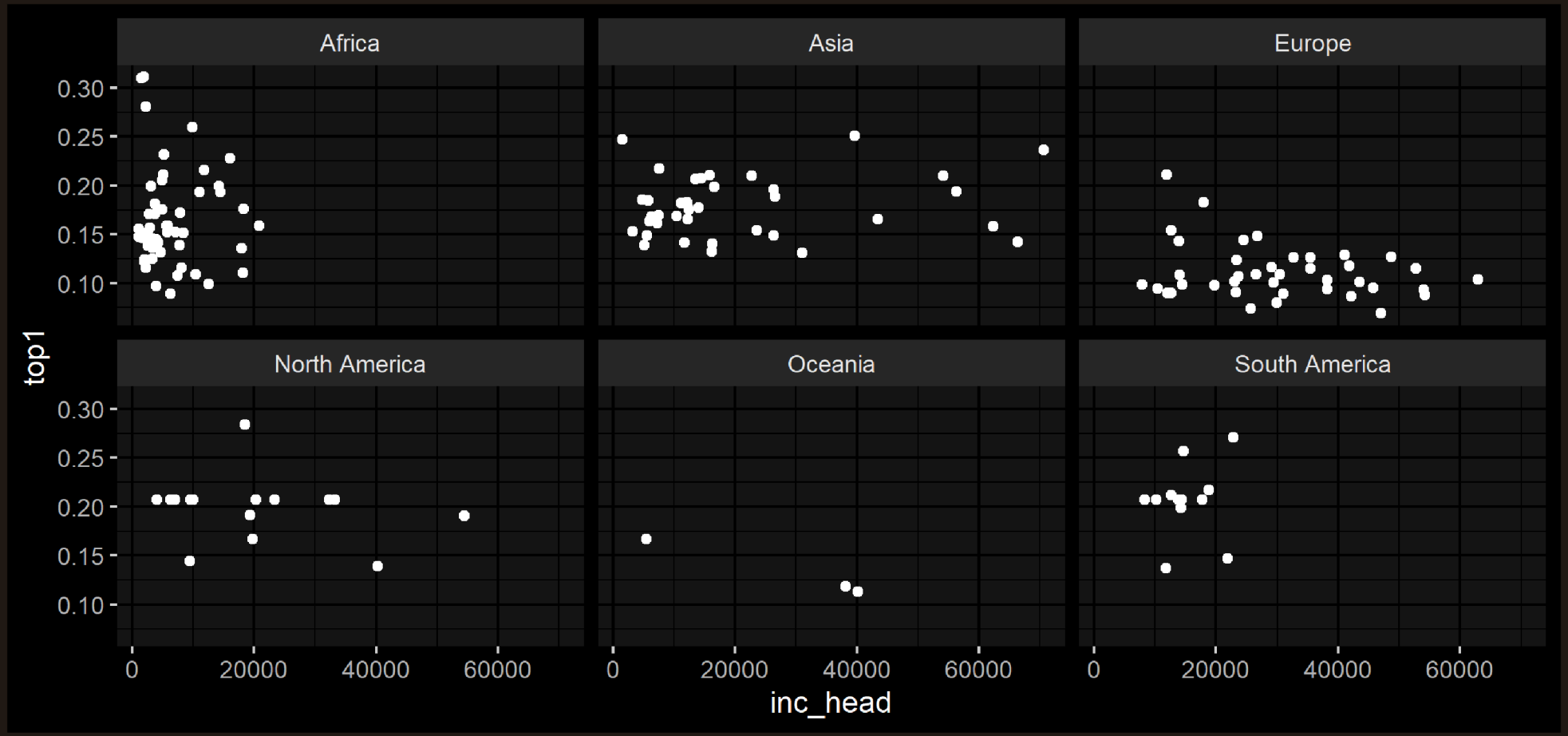
```
wid %>%
  filter(year == 2019) %>%
  ggplot(aes(x = inc_head, y = top1)) +
  geom_point() +
  facet_wrap(~continent, ncol = 3, scales = "fixed")
```

- You should specify the number of columns or rows:
  - **nrow** to indicate the number of rows
  - **ncol** to indicate the number of columns

- As well as which **scale** should be free or fixed
  - The default **"fixed"** will set all facet on the same x and y scales
  - Set scales to **"free_y"** to let the y axis fit the domain of each facet
  - Set scales to **"free_x"** to let the x axis fit the domain of each facet
  - Set scales to **"free"** to let the both axis fit the domain of each facet

2. Adding dimensions

2.2. More facets

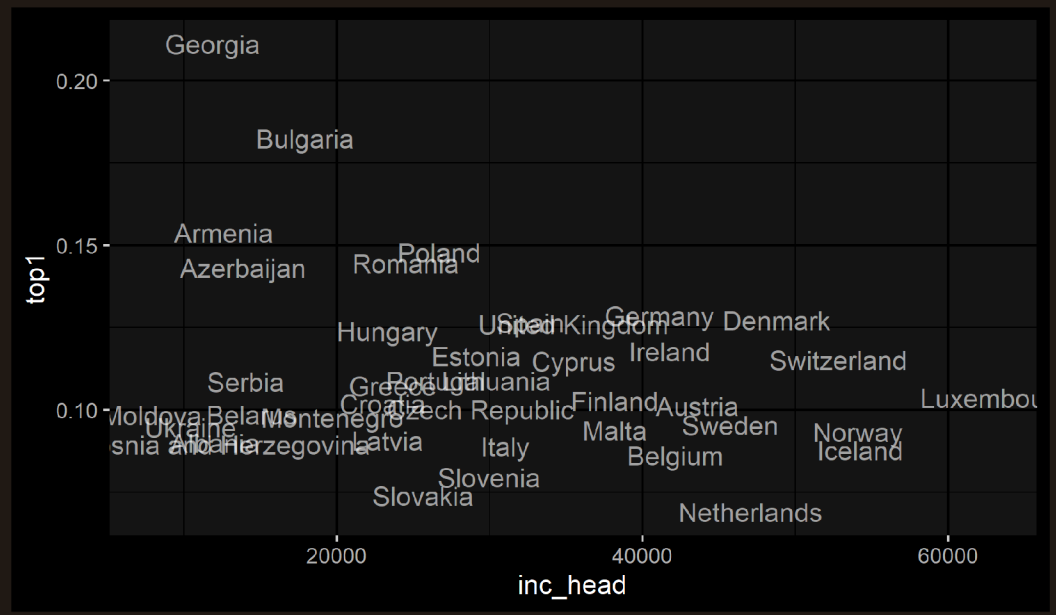


2. Adding dimensions

2.3. More labels

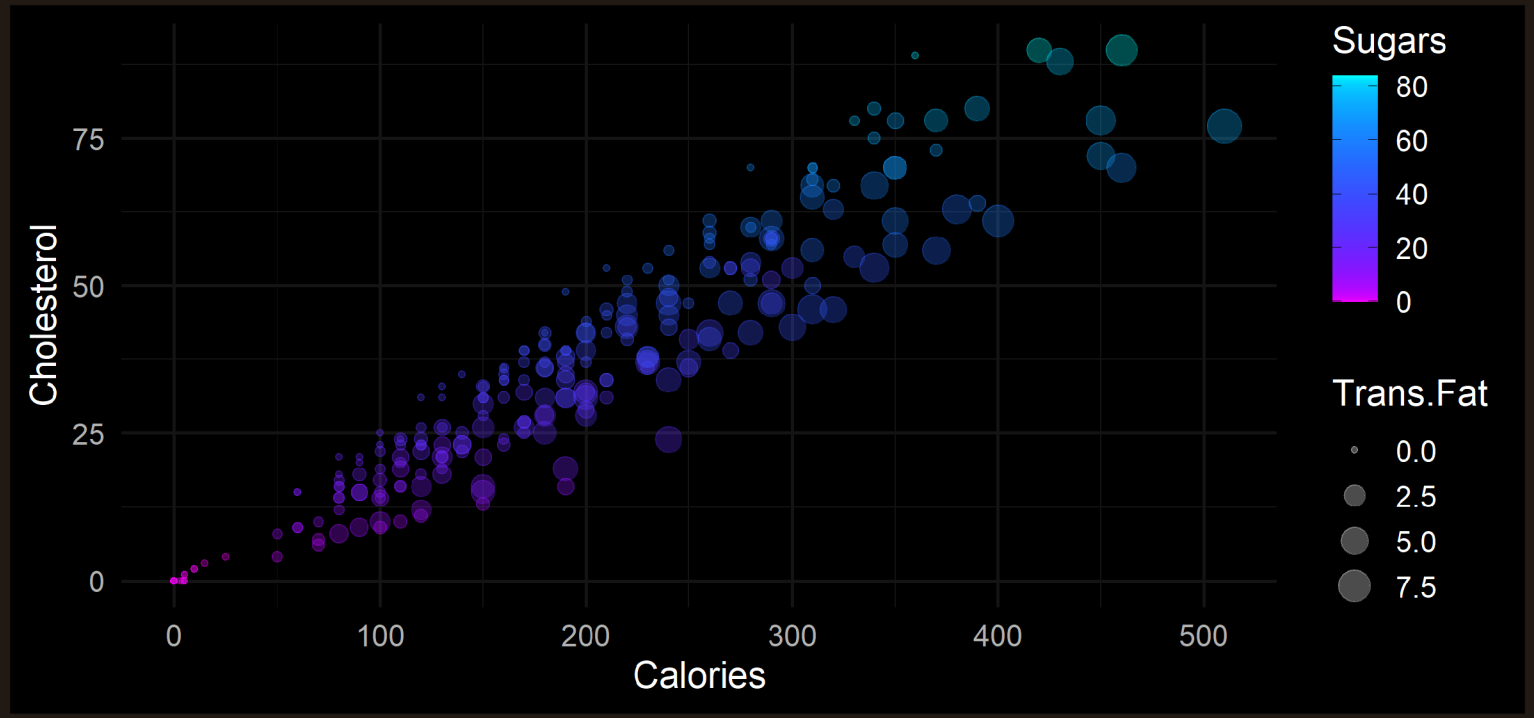
- The last dimension I want to mention is the ***label* axis**
  - When using **geom_text()** instead of geom_point(), it will plot the corresponding **text instead of points**

```
wid %>% filter(year == 2019 & continent == "Europe") %>%
  ggplot(aes(x = inc_head, y = top1, label = country)) + geom_text(alpha = .6)
```



Practice

1) Open the `starbucks.csv` data and reproduce that graph:



You've got 10 minutes!

Solution

1) Open the `starbucks.csv` data and reproduce that graph:

```
starbucks <- read.csv("starbucks.csv", sep = ";", encoding = "UTF-8") # Same as last week
# Use the ggplot function and specify the data
ggplot(starbucks,
       # In aes, assign the Calories variable to the x axis
       aes(x = Calories,
           # Cholesterol to the y axis
           y = Cholesterol,
           # Trans.Fat to the size axis
           size = Trans.Fat,
           # Sugars to the color axis
           color = Sugars)) +
  # Add a point geometry and set opacity to 30%
  geom_point(alpha = .3) +
  # Custom the color gradient from green to red
  scale_color_gradient(low = "green", high = "red") +
  # Apply the minimal default theme with a base font size of 14
  theme_minimal(base_size = 14)
```

Overview

1. The `ggplot()` function ✓

- 1.1. Basic structure
- 1.2. Axes
- 1.3. Theme
- 1.4. Annotation

2. Adding dimensions ✓

- 2.1. More axes
- 2.2. More facets
- 2.3. More labels

3. Types of geometry

- 3.1. Points and lines
- 3.2. Barplots and histograms
- 3.3. Densities and boxplots

4. How (not) to lie with graphics

- 4.1. Cumulative representations
- 4.2. Axis manipulations
- 4.3. Interpolation

5. Wrap up!

Overview

1. The `ggplot()` function ✓

- 1.1. Basic structure
- 1.2. Axes
- 1.3. Theme
- 1.4. Annotation

2. Adding dimensions ✓

- 2.1. More axes
- 2.2. More facets
- 2.3. More labels

3. Types of geometry

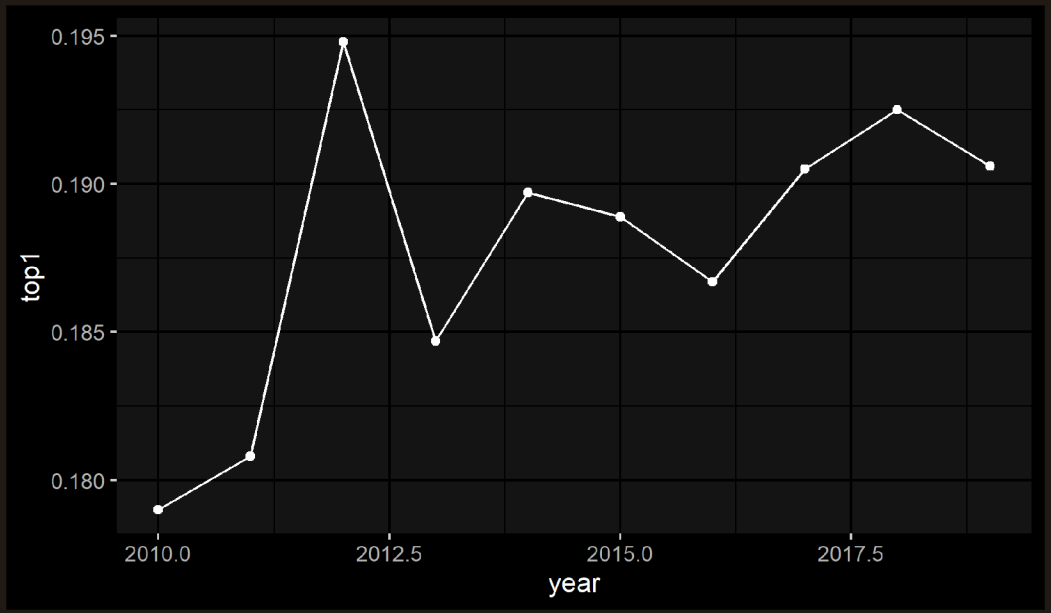
- 3.1. Points and lines
- 3.2. Barplots and histograms
- 3.3. Densities and boxplots

3. Types of geometry

3.1. Points and lines

- So far we only represented scatterplots, but **many other geometries** can be used
  - For instance, **lines** are particularly suited for **evolutions** over time

```
ggplot(wid %>% filter(country == "USA"), aes(x = year, y = top1)) +
  geom_point() + geom_line()
```

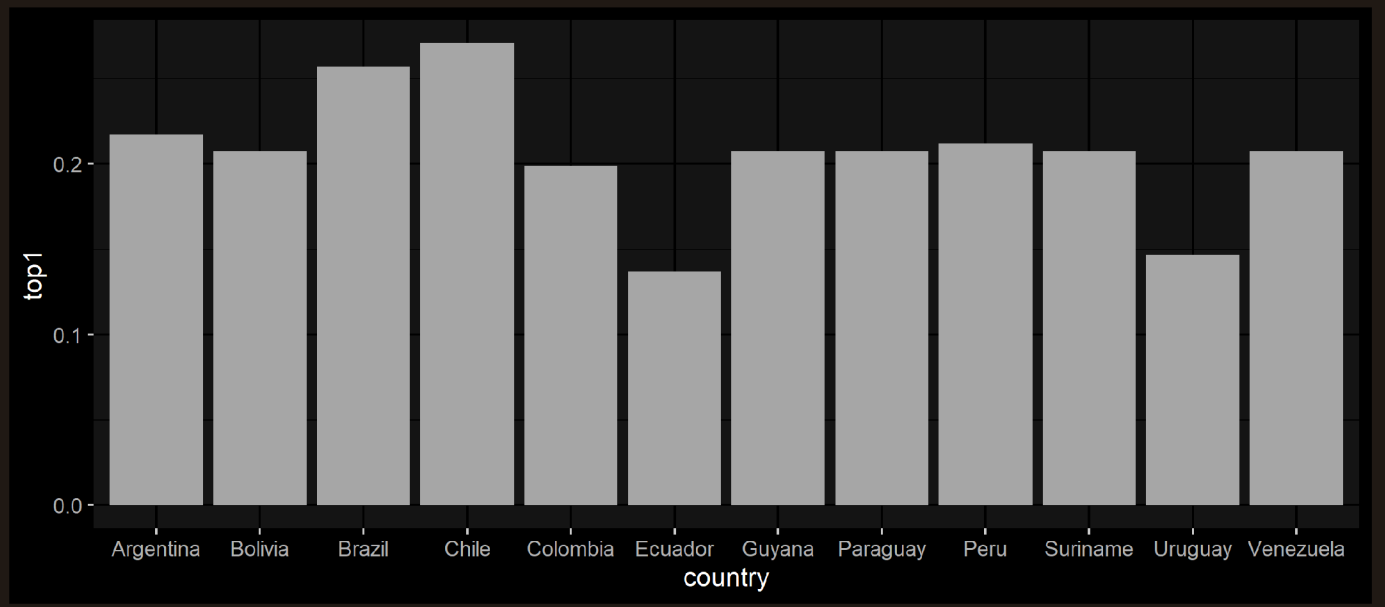


3. Types of geometry

3.2. Barplots and histograms

- **Barplots** however are great for categorical x variables and continuous y variables
  - Setting the **stat** argument to **"identity"** allows to display the corresponding **y value**

```
ggplot(wid %>% filter(continent == "South America" & year == 2019),
       aes(x = country, y = top1)) + geom_bar(stat = "identity")
```

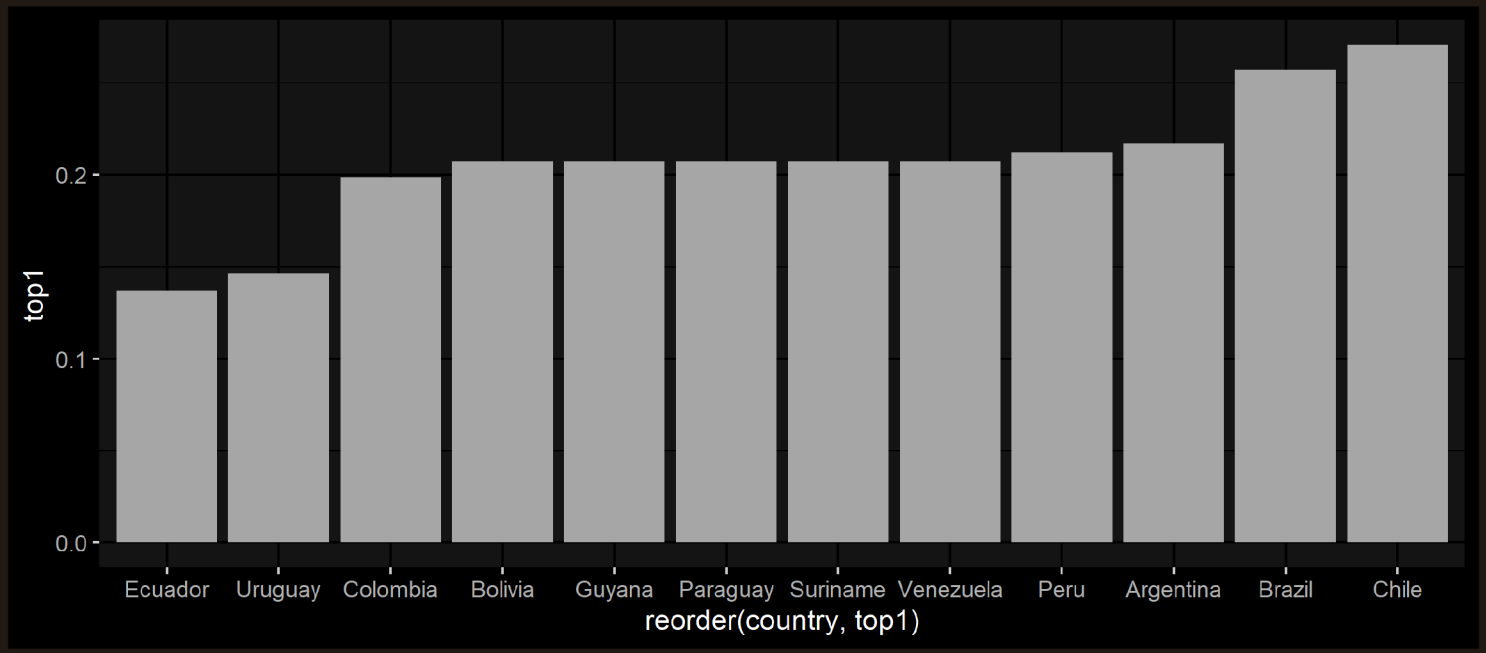


3. Types of geometry

3.2. Barplots and histograms

- Note that you can **reorder the bars** according to their y value using the reorder() function

```
ggplot(wid %>% filter(continent == "South America" & year == 2019),
       aes(x = reorder(country, top1), y = top1)) + geom_bar(stat = "identity")
```

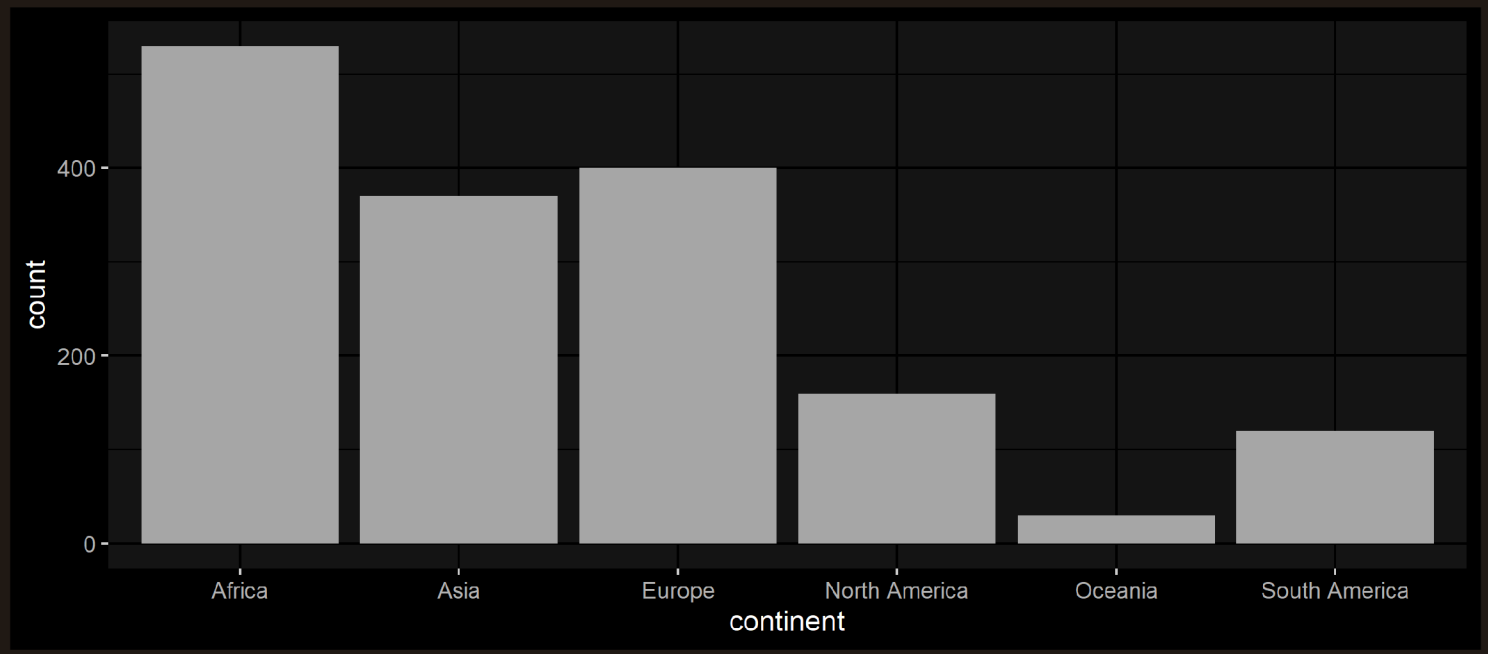


3. Types of geometry

3.2. Barplots and histograms

- You can also set stat to **"count"** to plot the **number of observations** per category
  - In that case, no variable should be assigned to the y axis

```
ggplot(wid, aes(x = continent)) + geom_bar(stat = "count")
```

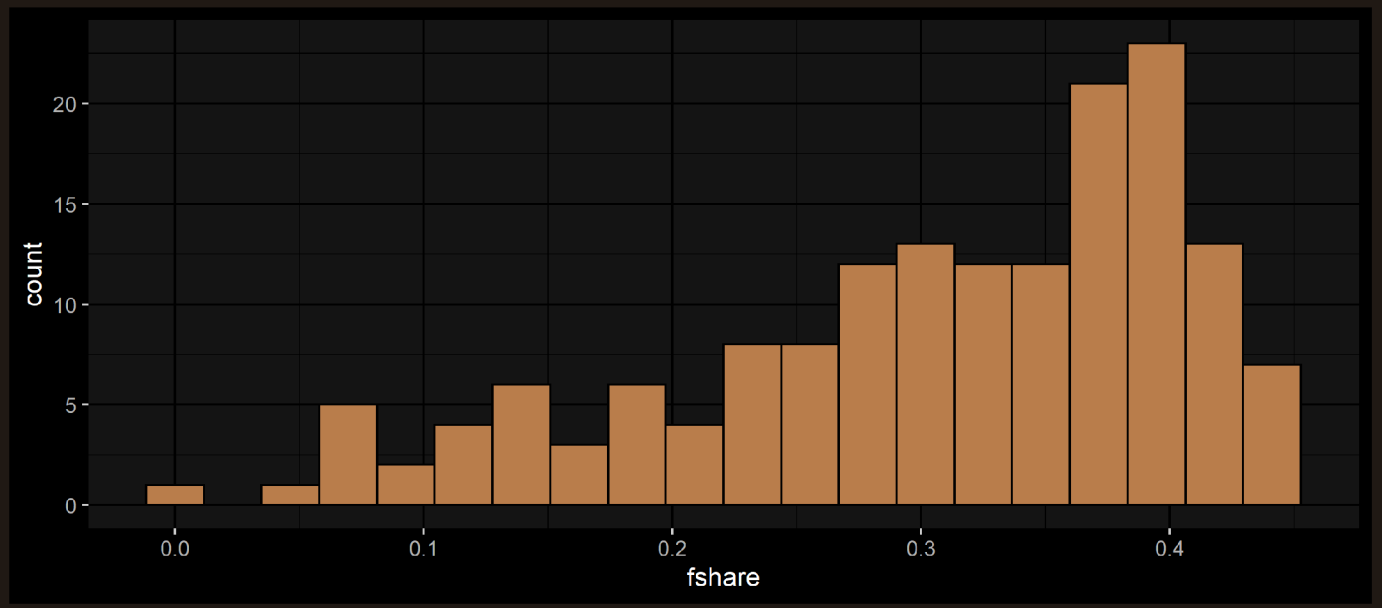


3. Types of geometry

3.2. Barplots and histograms

- Finally, histograms can be used to describe the distribution of a continuous variable
  - You can tune the bin width with **binwidth** or the number of bins with **bins**

```
ggplot(wid %>% filter(year == 2019), aes(x = fshare)) +
  geom_histogram(bins = 20, color = "white", fill = "steelblue")
```

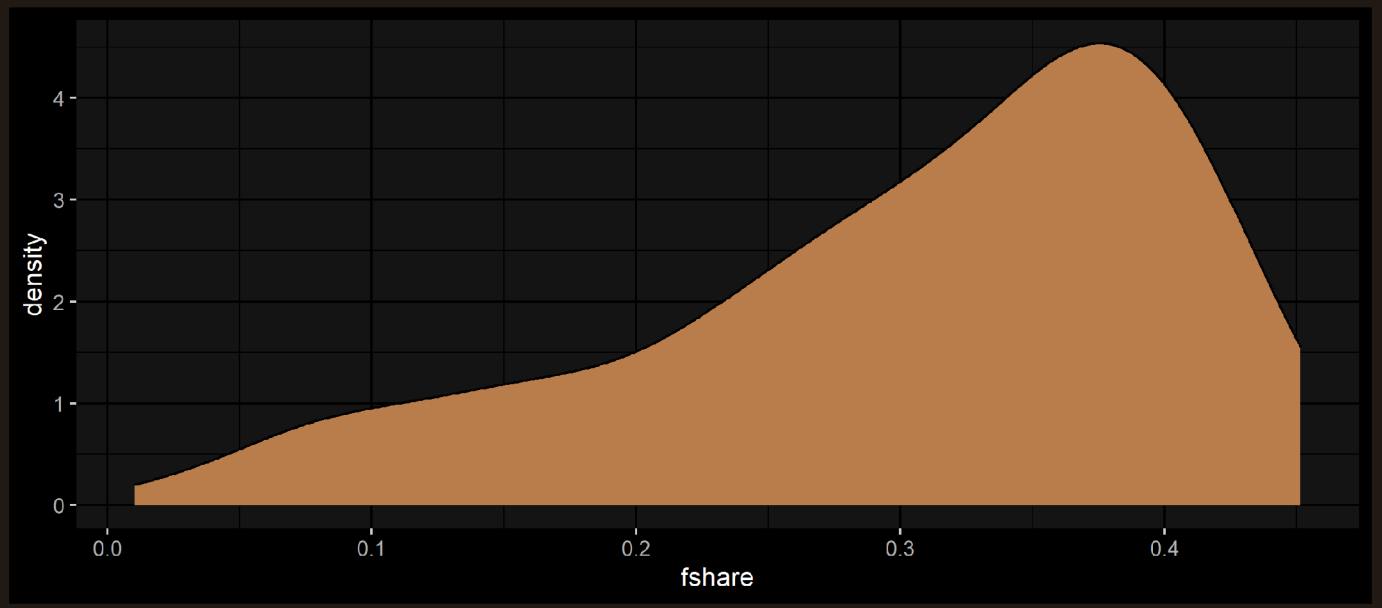


3. Types of geometry

3.3. Densities and boxplots

- As you may remember from last class, the **continuous** equivalent of the histogram is the **density**
  - Similarly you can tune the **bandwidth** with the **bw** argument *(don't do it)*

```
ggplot(wid %>% filter(year == 2019), aes(x = fshare)) +
    geom_density(color = "white", fill = "steelblue")
```

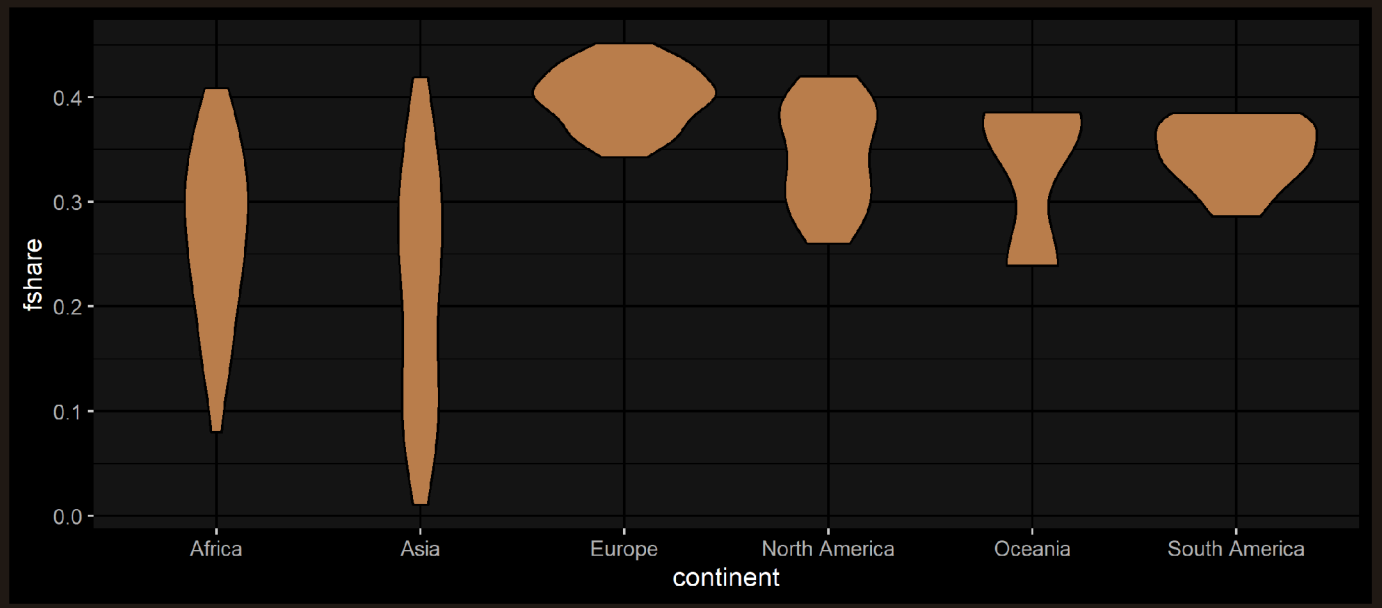


3. Types of geometry

3.3. Densities and boxplots

- A handy geometry to plot **densities** for different **groups** is the **violin**
  - Note that the **grouping variable** should be assigned to the x **axis**

```
ggplot(wid %>% filter(year == 2019), aes(x = continent, y = fshare)) +
    geom_violin(color = "white", fill = "steelblue")
```

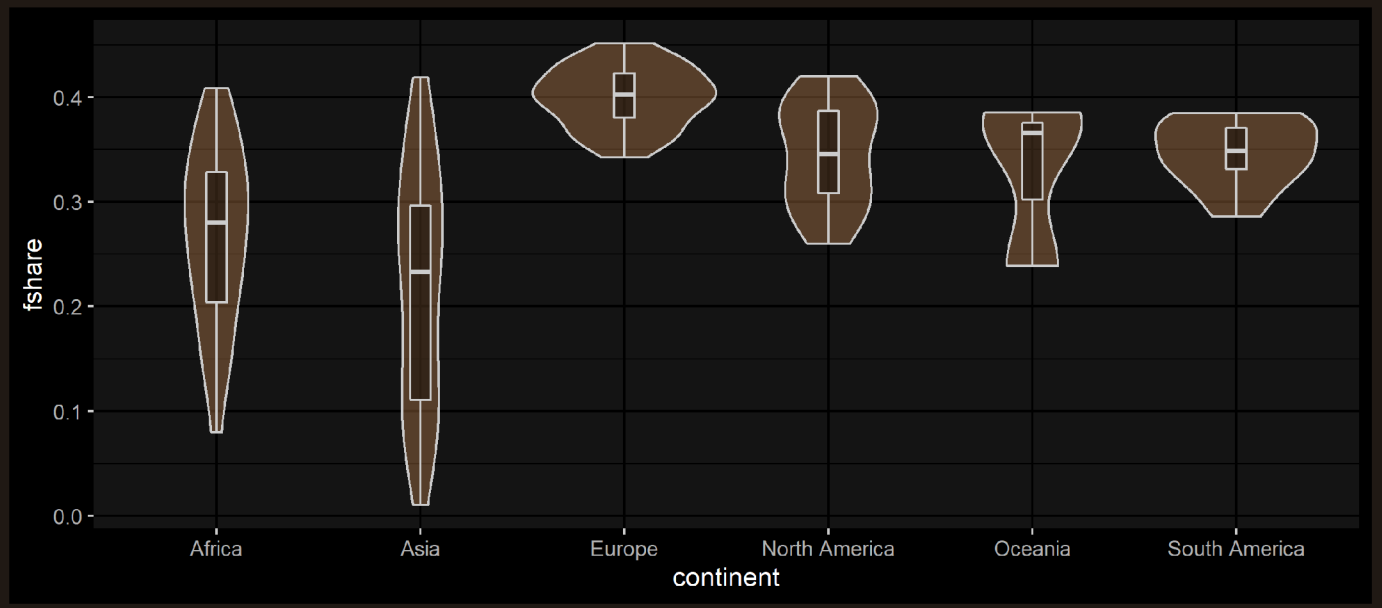


3. Types of geometry

3.3. Densities and boxplots

- **Violins** are particularly interesting when **combined with boxplots**
  - When overlaying these geometries, make sure to tune the **width and opacity** appropriately

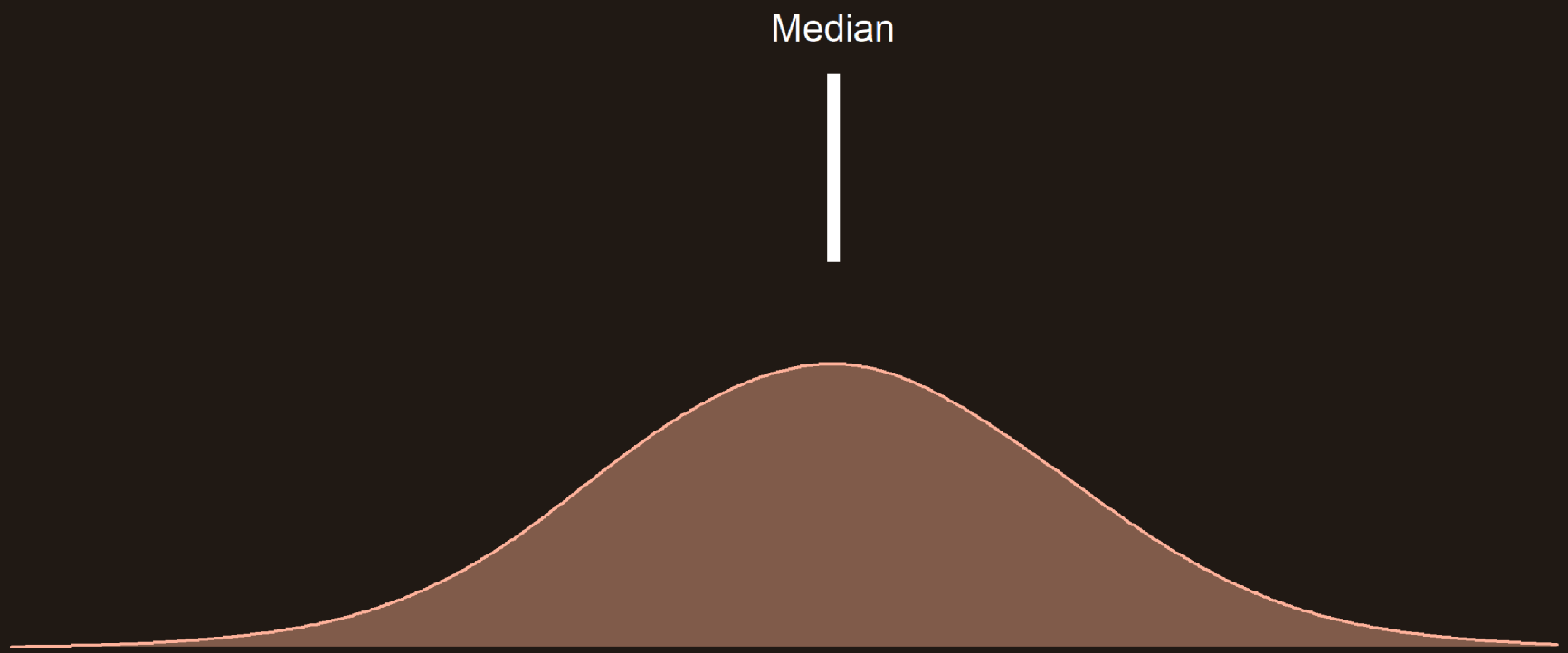
```
ggplot(wid %>% filter(year == 2019), aes(x = continent, y = fshare)) +
    geom_violin(fill = "steelblue", alpha = .4)  + geom_boxplot(width = .1, alpha = .4)
```



3. Types of geometry

3.3. Densities and boxplots

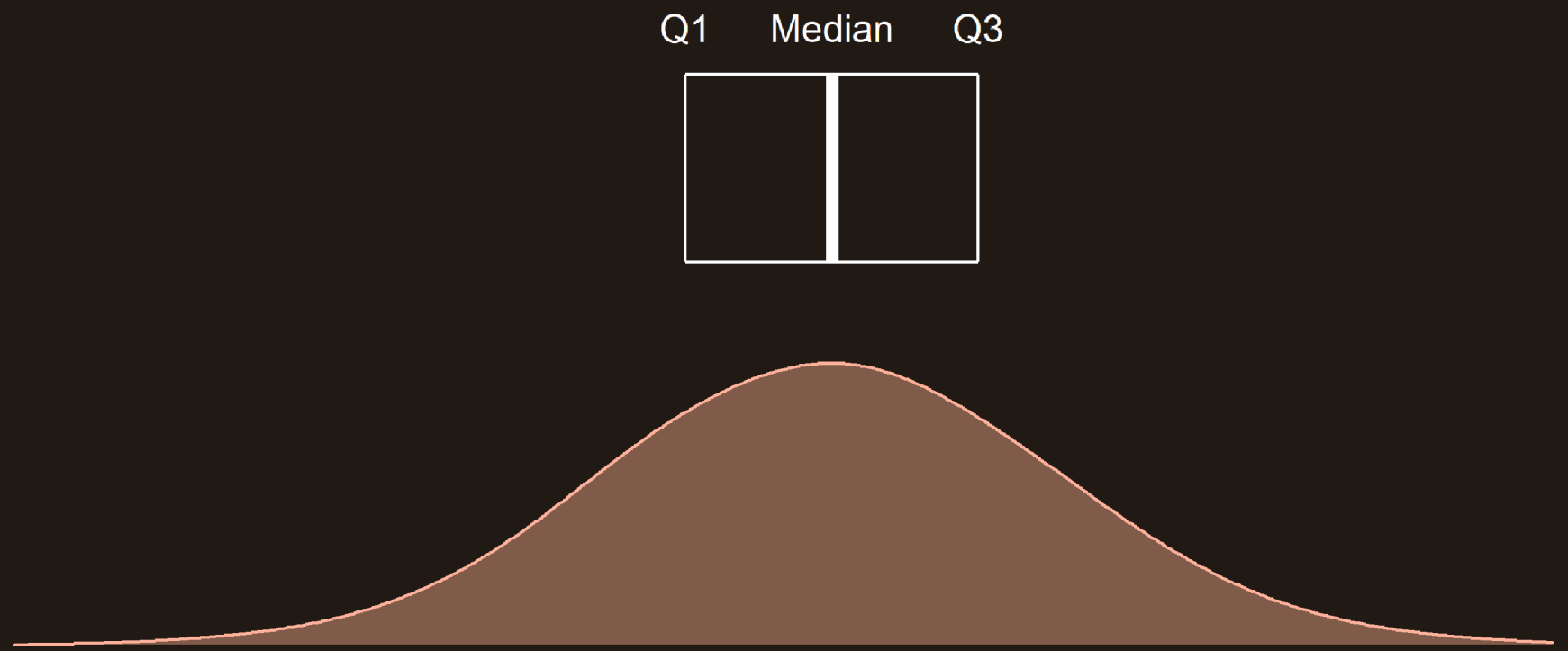
- This is how **boxplots** are constructed:



3. Types of geometry

3.3. Densities and boxplots

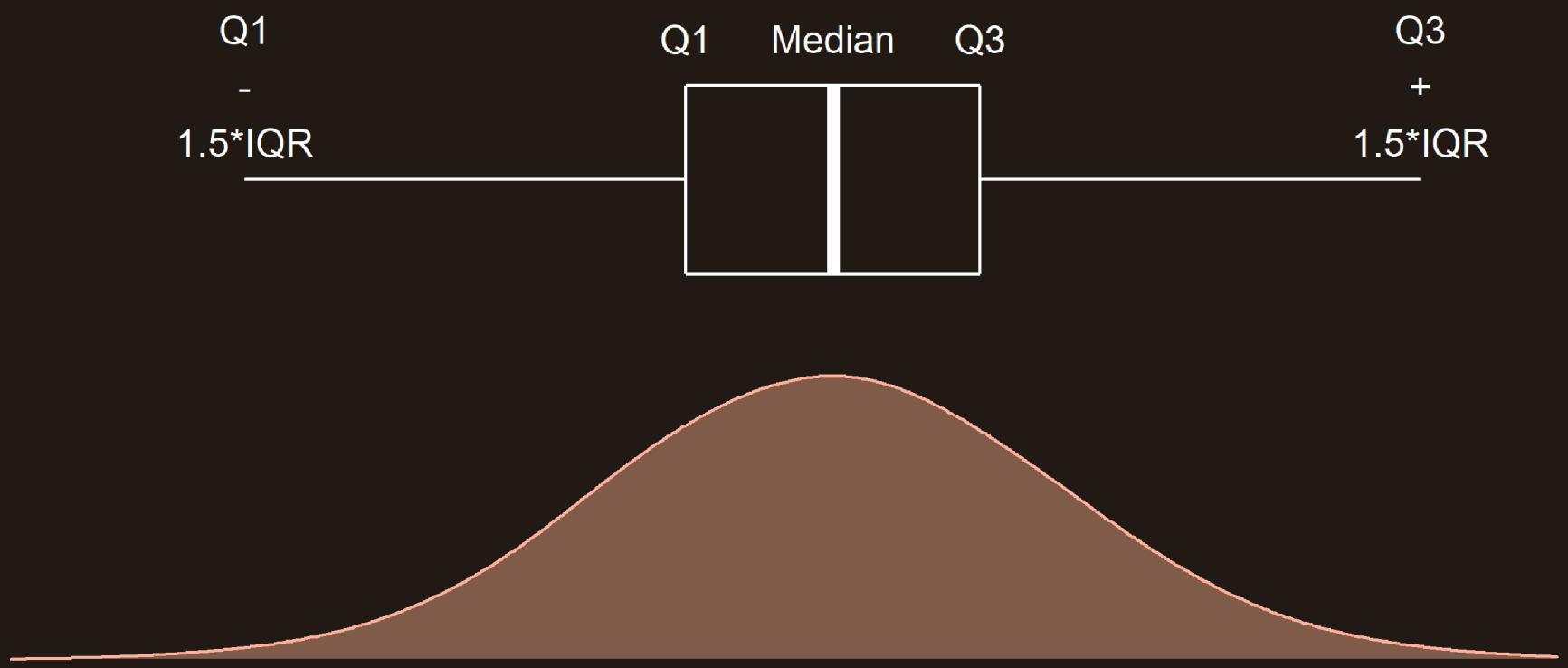
- This is how **boxplots** are constructed:



3. Types of geometry

3.3. Densities and boxplots

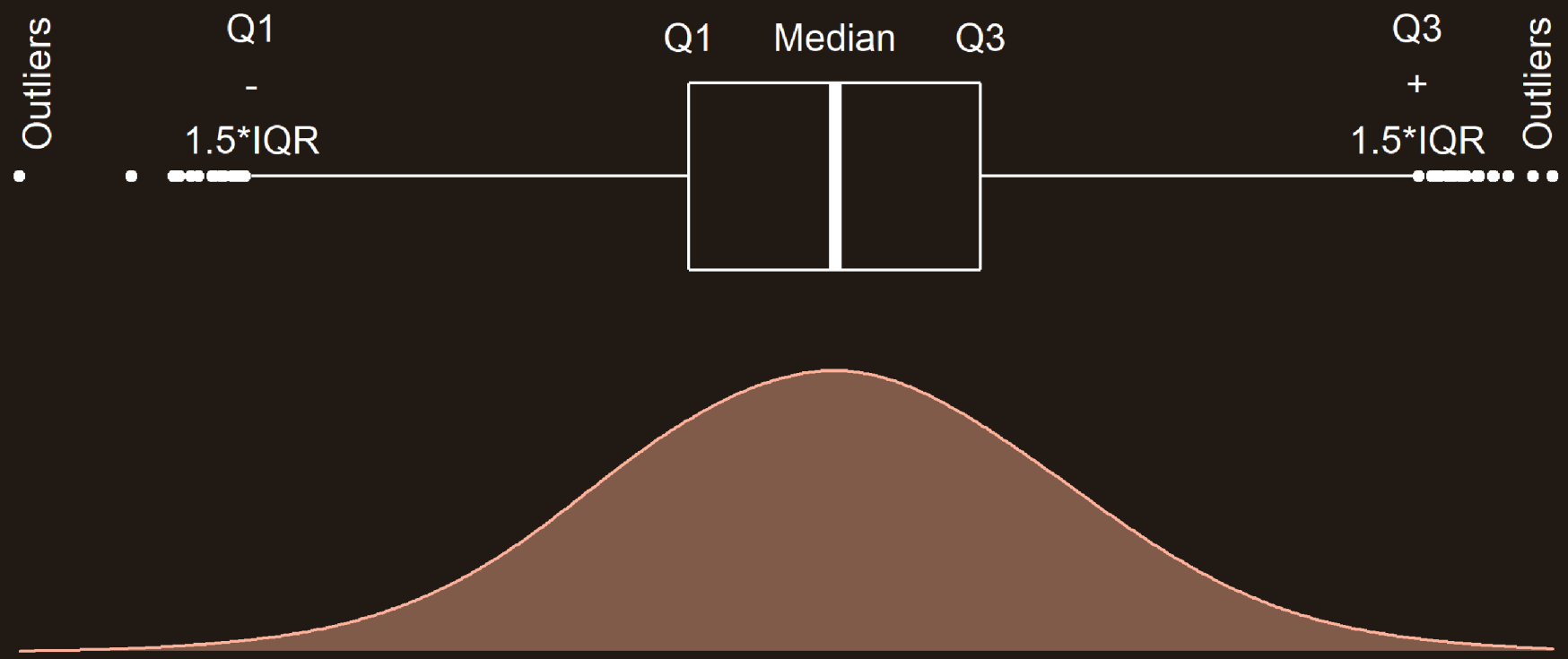
- This is how **boxplots** are constructed:



3. Types of geometry

3.3. Densities and boxplots

- This is how **boxplots** are constructed:



Overview

1. The `ggplot()` function ✓

- 1.1. Basic structure
- 1.2. Axes
- 1.3. Theme
- 1.4. Annotation

2. Adding dimensions ✓

- 2.1. More axes
- 2.2. More facets
- 2.3. More labels

3. Types of geometry ✓

- 3.1. Points and lines
- 3.2. Barplots and histograms
- 3.3. Densities and boxplots

4. How (not) to lie with graphics

- 4.1. Cumulative representations
- 4.2. Axis manipulations
- 4.3. Interpolation

5. Wrap up!

Overview

1. The `ggplot()` function ✓

- 1.1. Basic structure
- 1.2. Axes
- 1.3. Theme
- 1.4. Annotation

2. Adding dimensions ✓

- 2.1. More axes
- 2.2. More facets
- 2.3. More labels

3. Types of geometry ✓

- 3.1. Points and lines
- 3.2. Barplots and histograms
- 3.3. Densities and boxplots

4. How (not) to lie with graphics

- 4.1. Cumulative representations
- 4.2. Axis manipulations
- 4.3. Interpolation

4. How (not) to lie with graphics

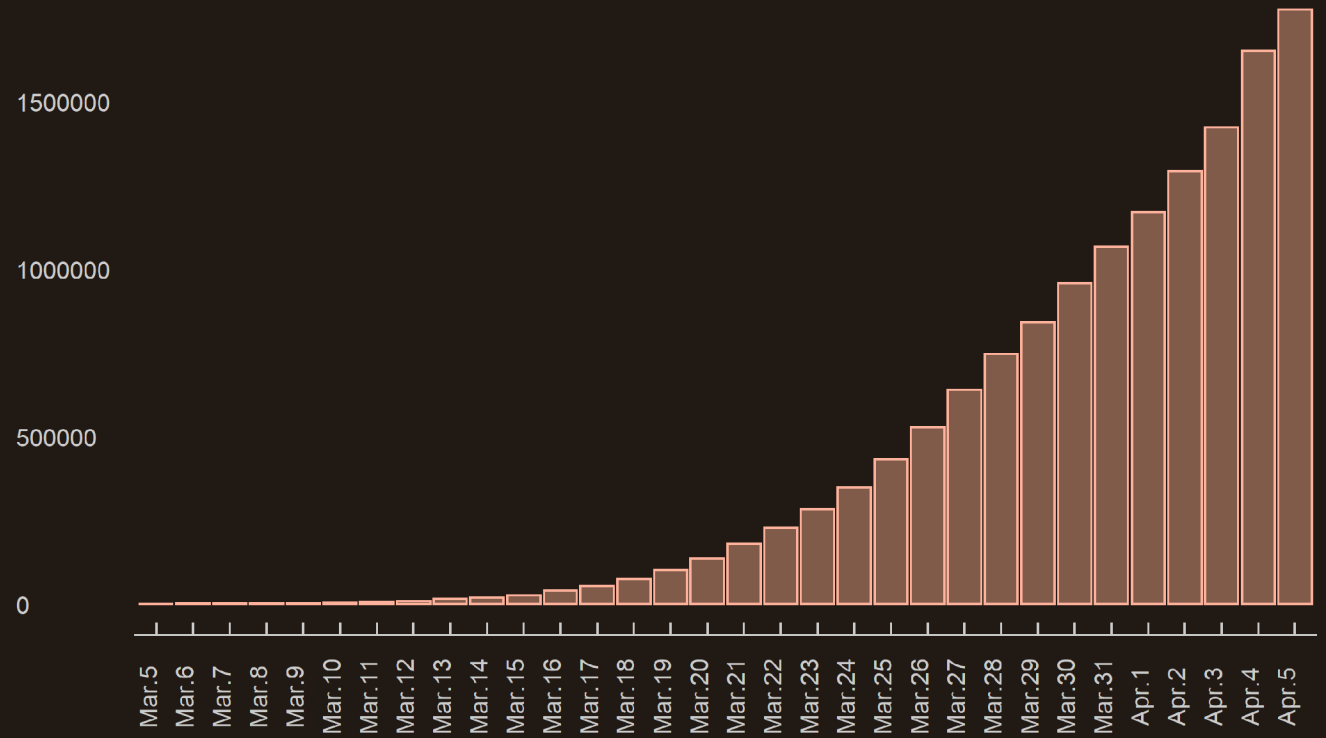
4.1. Cumulative representations

Donald Trump during his daily coronavirus task force briefing on April 6, 2020

The legend indicates:
''>1,790,000 tests completed through April 5''

4. How (not) to lie with graphics

4.1. Cumulative representations



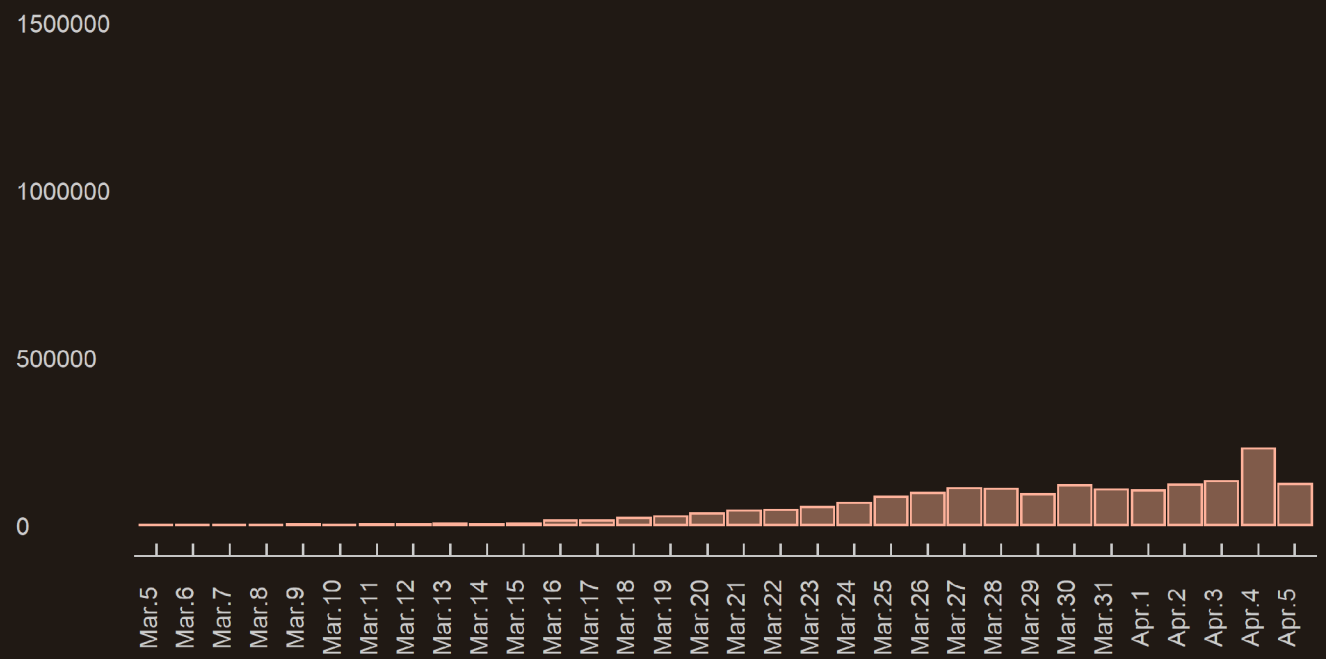
Let's take a closer look

">1,790,000 tests completed through April 5"

Isn't there something tricky here?

4. How (not) to lie with graphics

4.1. Cumulative representations

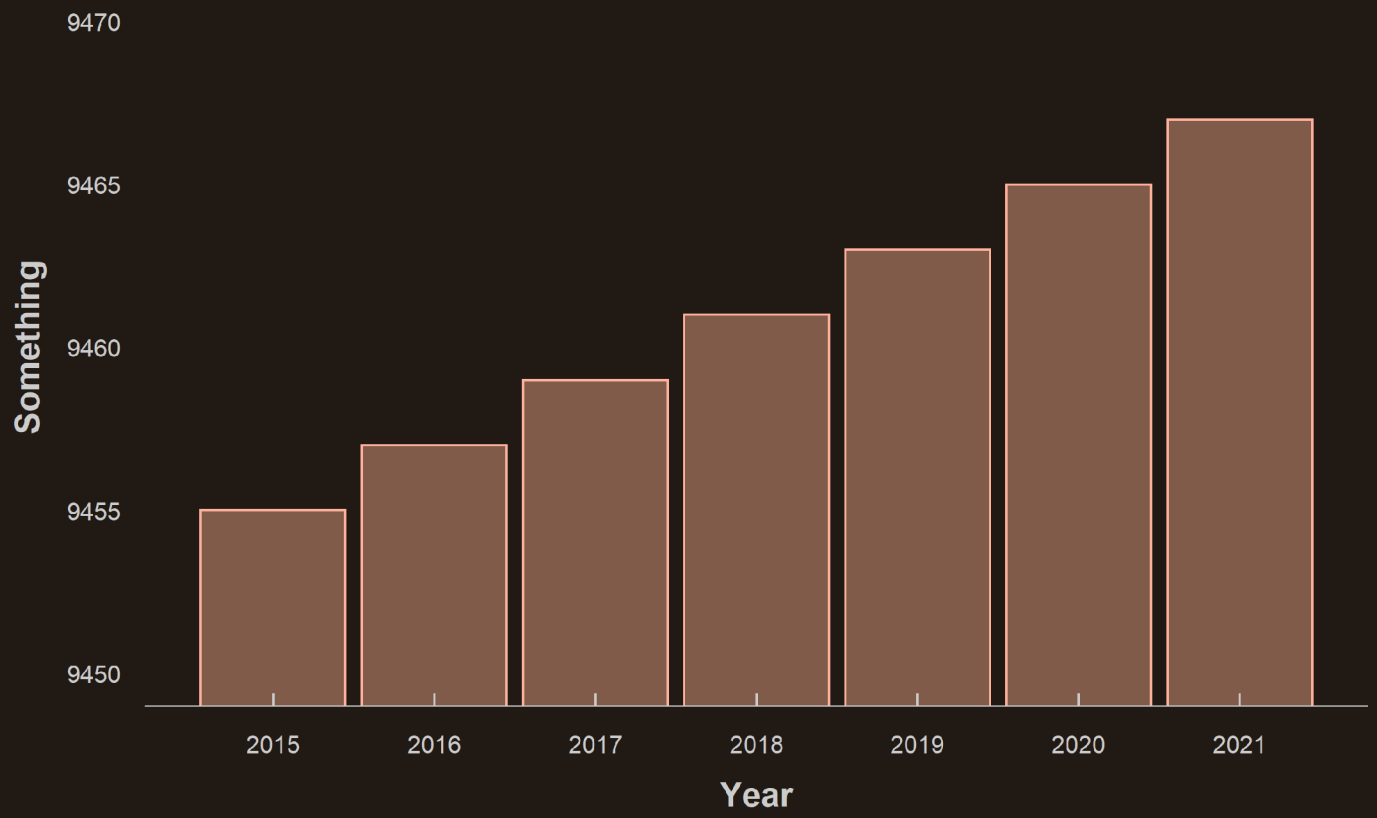


*They plotted the **cumulative** number tests!*

- This make it **looks** like an **exponential** progression
- While the daily number of tests **actually did not increase that exponentially**

4. How (not) to lie with graphics

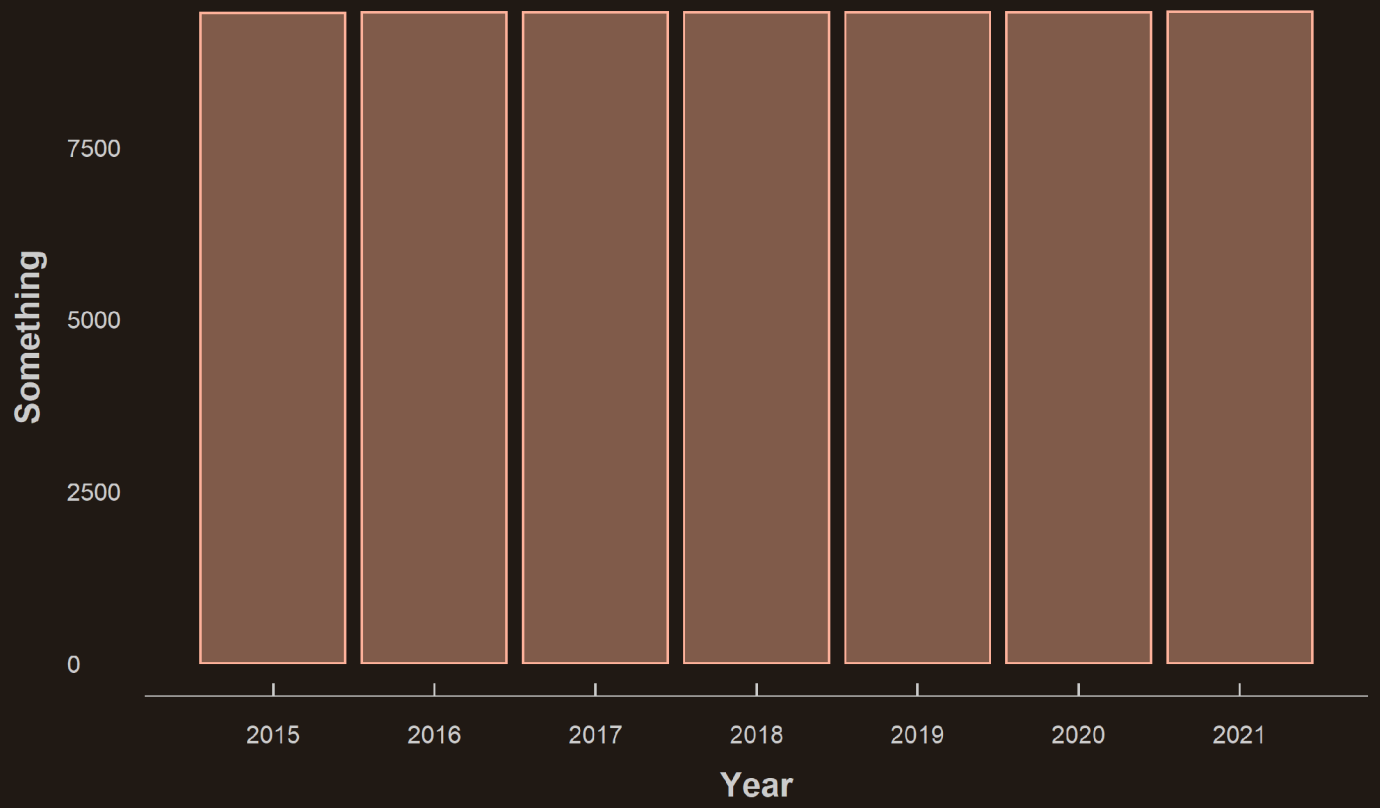
4.2. Axis manipulations



What about that increase?

4. How (not) to lie with graphics

4.2. Axis manipulations

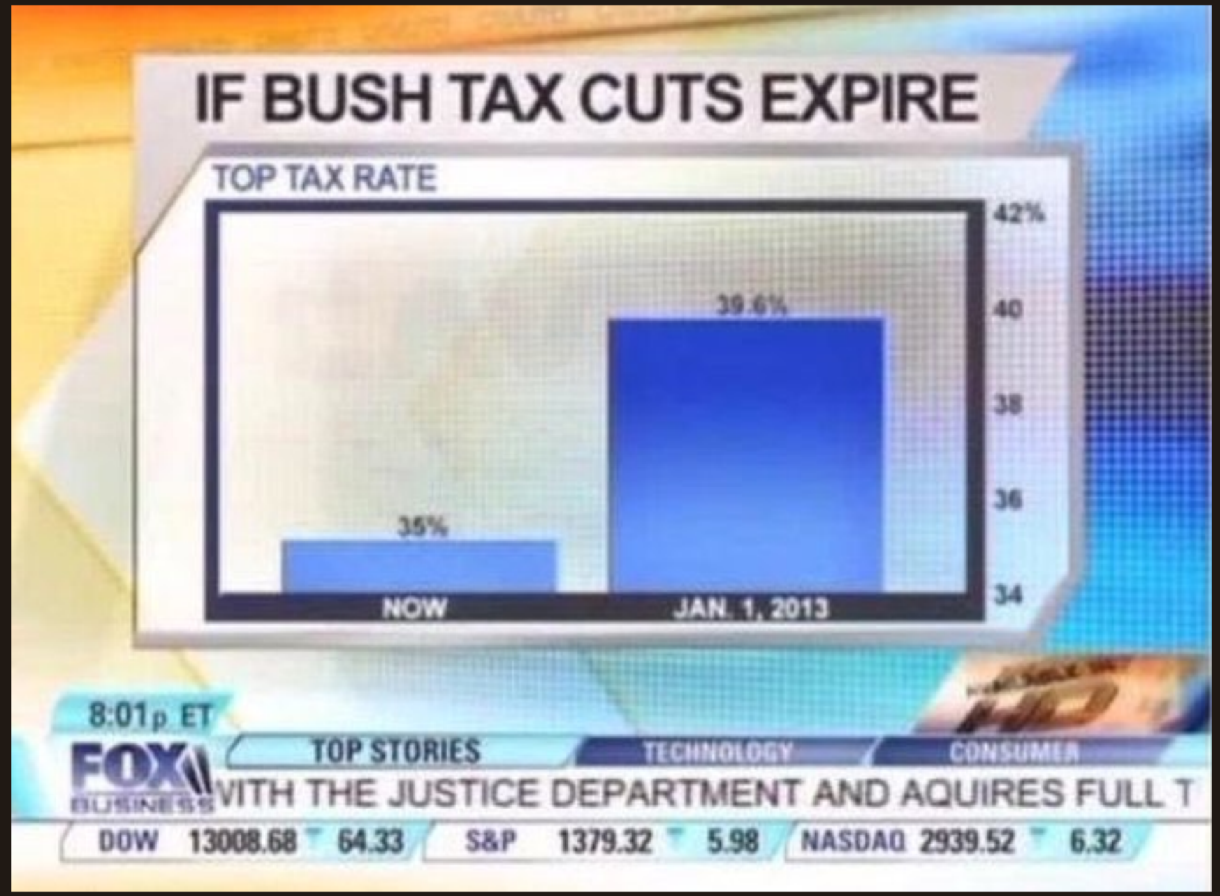


Same data**, but starting **from 0

→ **Zooming** or unzooming on a graph can be very **misleading**

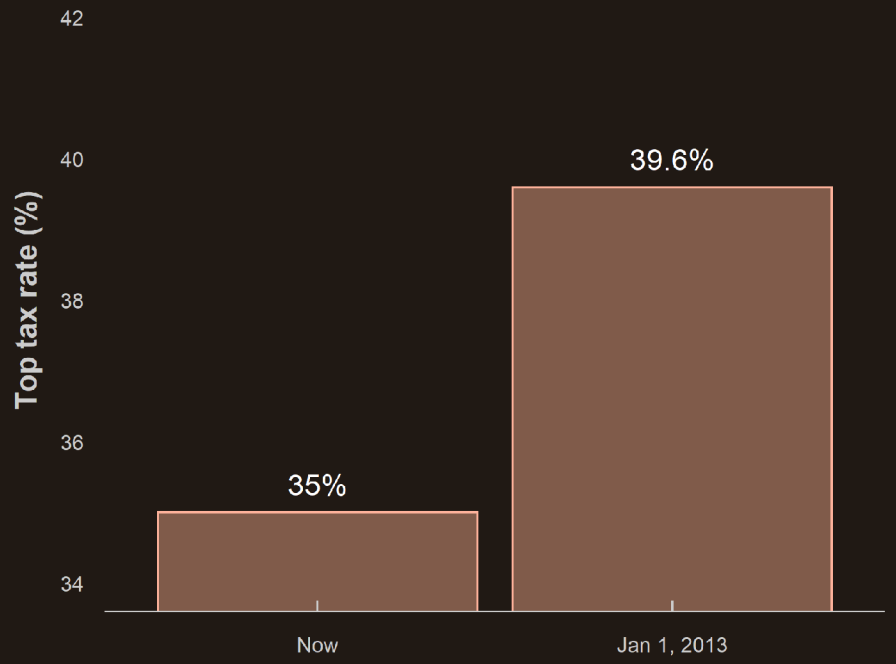
4. How (not) to lie with graphics

4.2. Axis manipulations

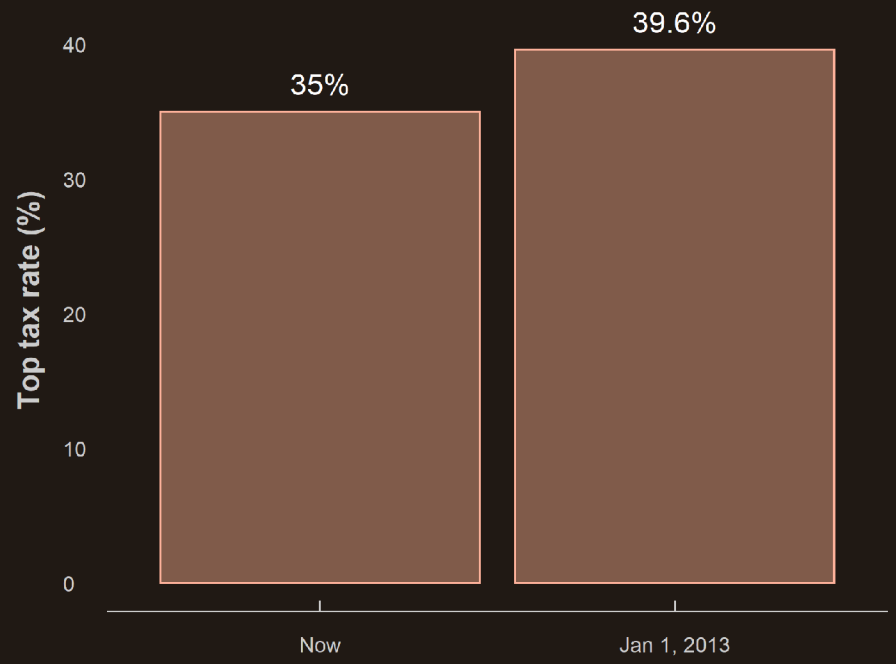


4. How (not) to lie with graphics

4.2. Axis manipulations



Misleading

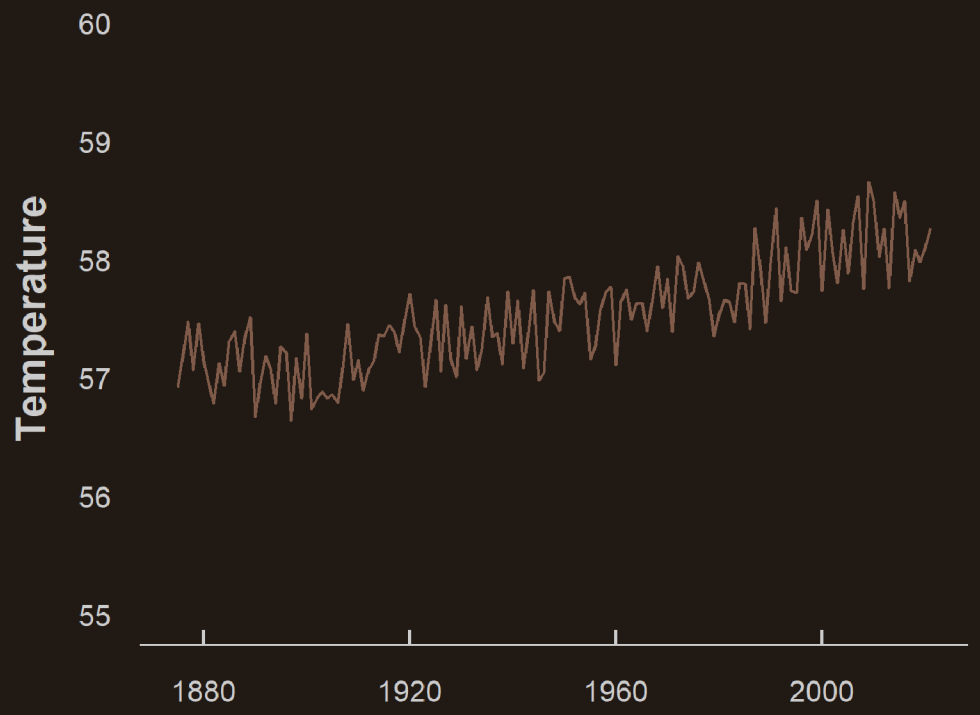


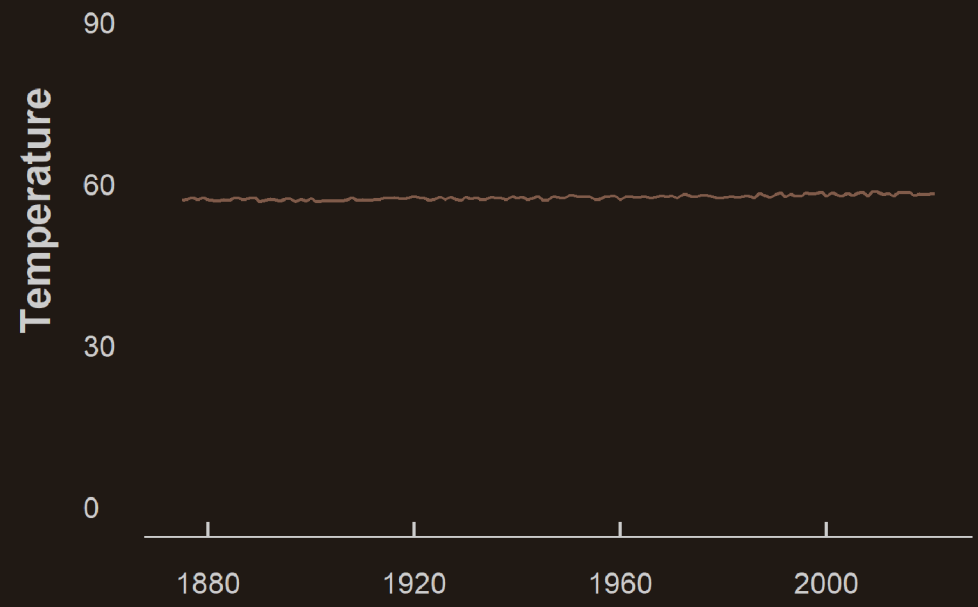
Not misleading

4. How (not) to lie with graphics

4.2. Axis manipulations

- But in this case which is the most adequate representation?

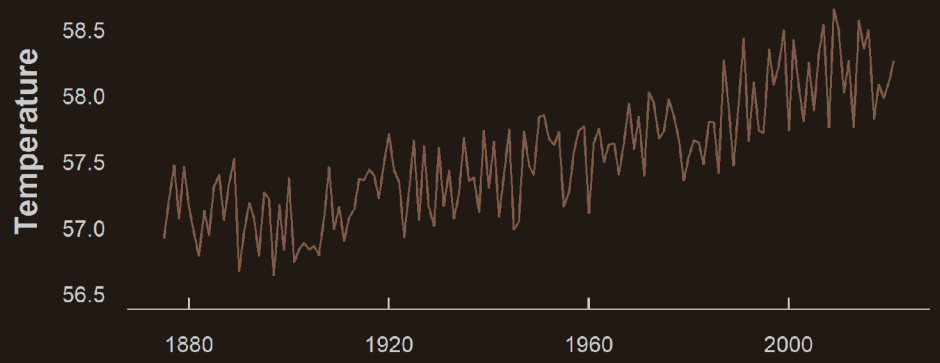


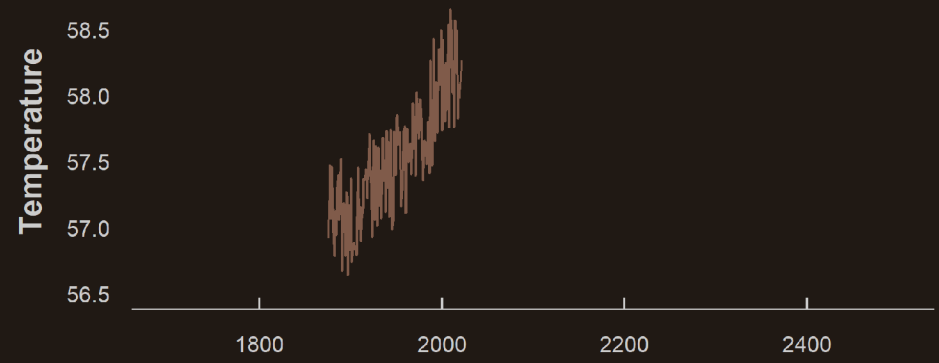


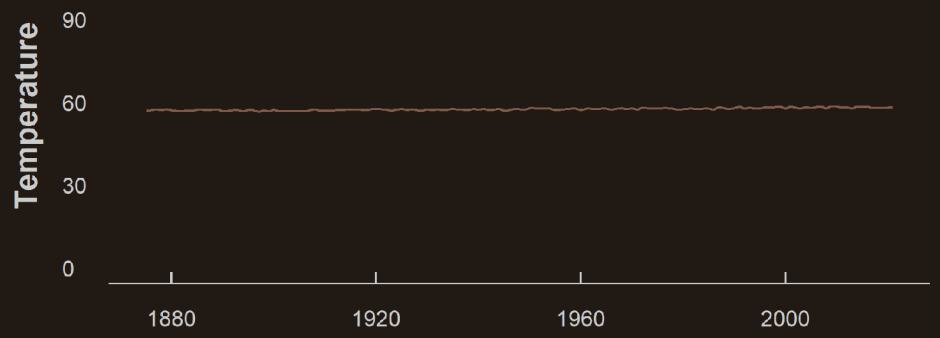
4. How (not) to lie with graphics

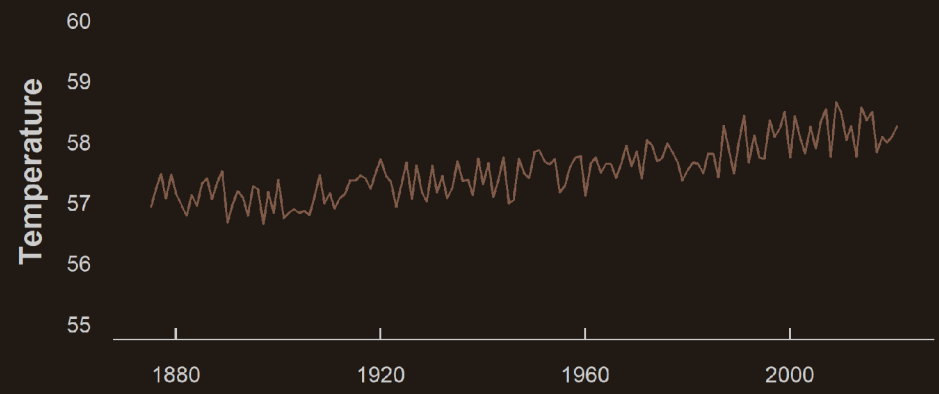
4.2. Axis manipulations

- There is no universal rule, but always **pay attention to axes and scales**



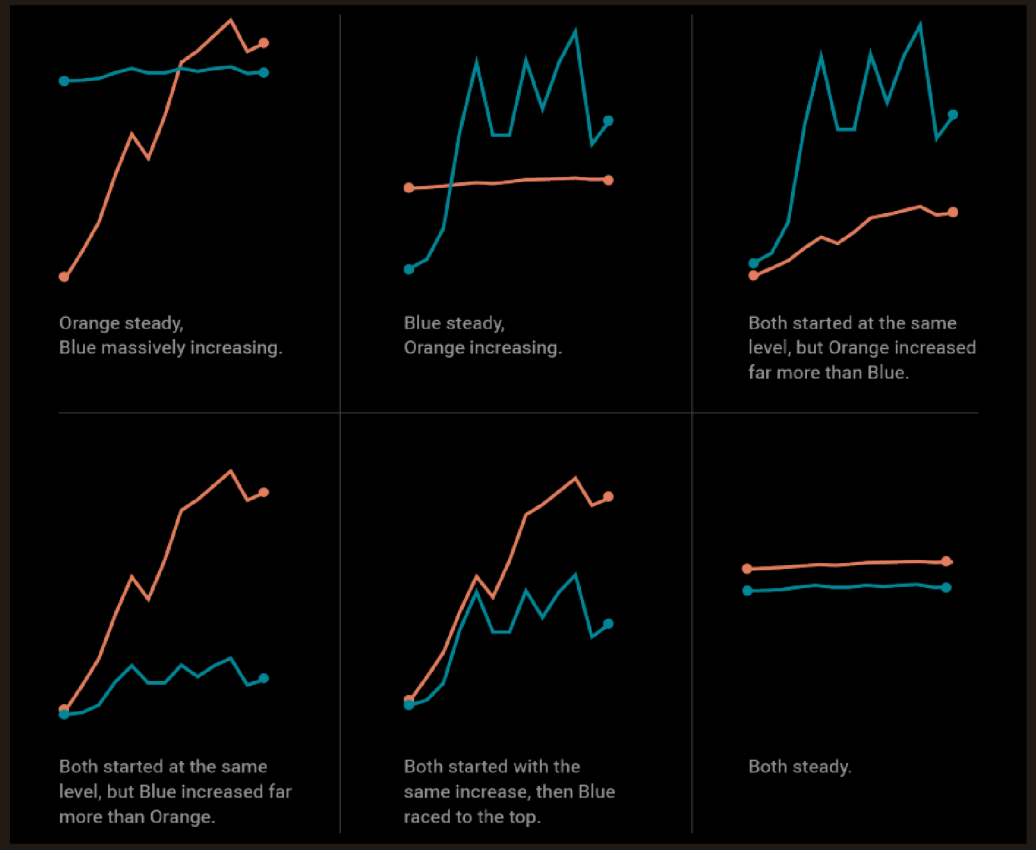






4. How (not) to lie with graphics

4.2. Axis manipulations



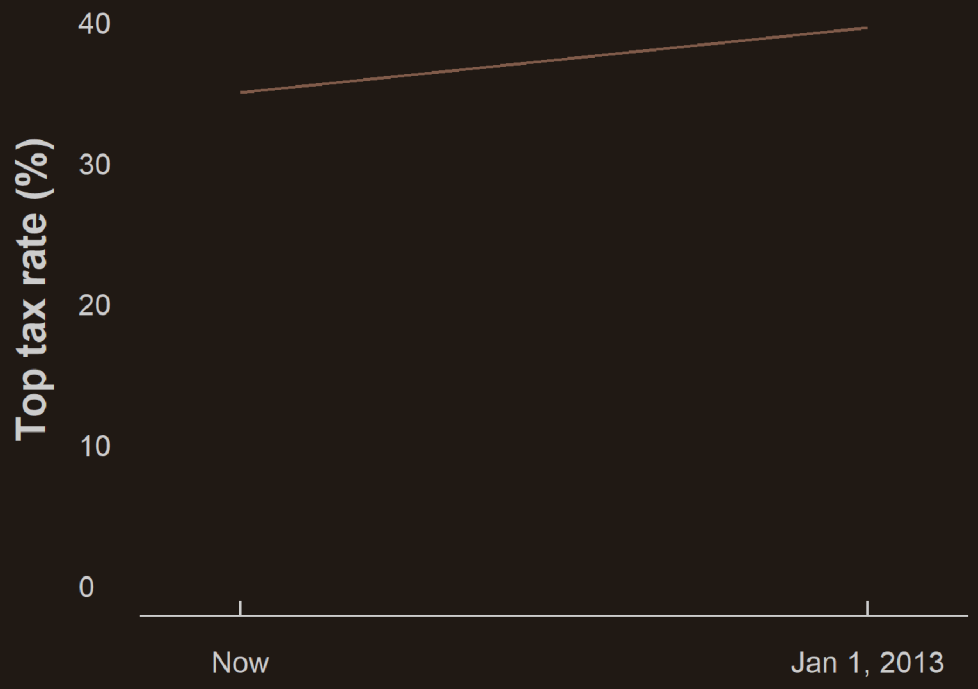
⚠ Be very careful with double axes ⚠

You can make a them tell basically everything

4. How (not) to lie with graphics

4.2. Interpolation

- Here is the **previous graph** on the tax increase using a **line geometry**



This line has **infinitely many** points

But **only two** of them are **correct**

4. How (not) to lie with graphics

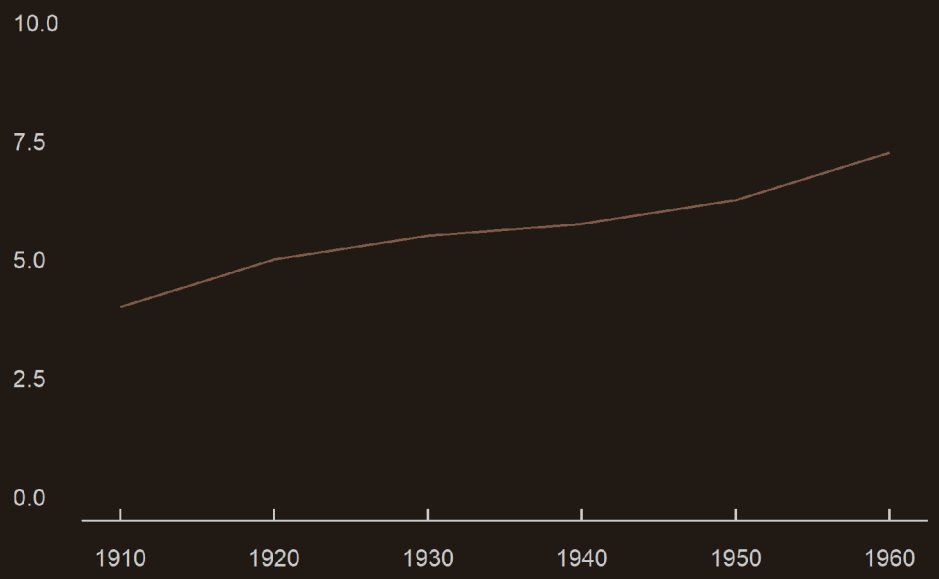
4.2. Interpolation

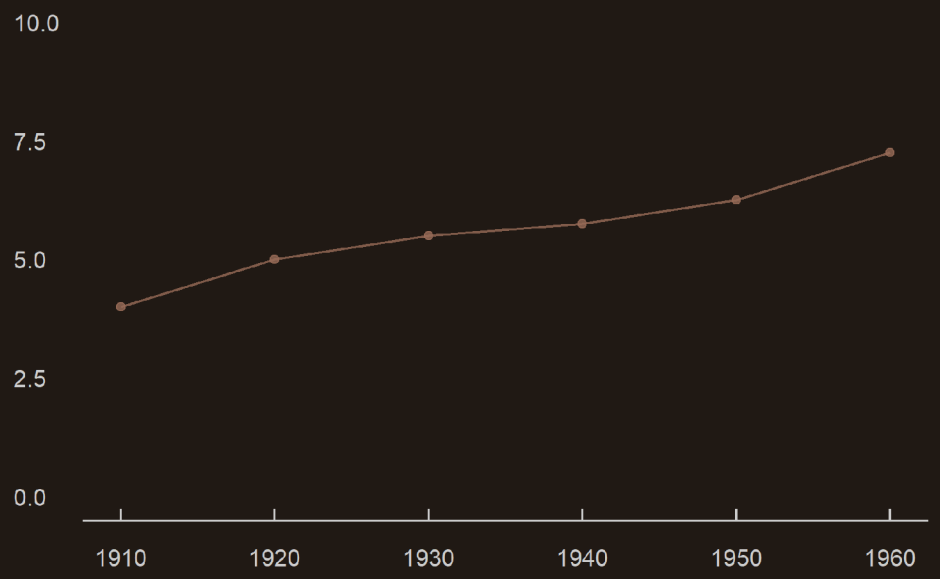
- This figure also has **finitely many** values but feels more natural
  - This is because values are **sufficiently close** to each other to be **considered** as **continuous**

4. How (not) to lie with graphics

4.2. Interpolation

- There is no rule either on when **lines** should be used or not
  - But the **observation level** should be **clear** on the graph





Overview

1. The `ggplot()` function ✓

- 1.1. Basic structure
- 1.2. Axes
- 1.3. Theme
- 1.4. Annotation

2. Adding dimensions ✓

- 2.1. More axes
- 2.2. More facets
- 2.3. More labels

3. Types of geometry ✓

- 3.1. Points and lines
- 3.2. Barplots and histograms
- 3.3. Densities and boxplots

4. How (not) to lie with graphics

- 4.1. Cumulative representations
- 4.2. Axis manipulations
- 4.3. Interpolation

5. Wrap up!

5. Wrap up!

The 3 core components of the ggplot() function

Component	Contribution	Implementation
Data	Underlying values	ggplot(data, | data %>% ggplot(.,
Mapping	Axis assignment	aes(x = V1, y = V2, ...))
Geometry	Type of plot	+ geom_point() + geom_line() + ...

- Any **other element** should be added with a **`+` sign**

```
ggplot(data, aes(x = V1, y = V2)) +
  geom_point() + geom_line() +
  anything_else()
```

5. Wrap up!

Main customization tools

Item to customize	Main functions
Axes	scale_[x/y]_[continuous/discrete]
Baseline theme	theme_[void/minimal/.../dark]()
Annotations	geom_[[h/v]line/text](), annotate()
Theme	theme(axis.[line/ticks].[x/y] = ...,

Main types of geometry

Geometry	Function
Bar plot	geom_bar()
Histogram	geom_histogram()
Area	geom_area()
Line	geom_line()
Density	geom_density()
Boxplot	geom_boxplot()
Violin	geom_violin()
Scatter plot	geom_point()

5. Wrap up!

Main types of aesthetics

Argument	Meaning
alpha	opacity from 0 to 1
color	color of the geometry
fill	fill color of the geometry
size	size of the geometry
shape	shape for geometries like points
linetype	solid, dashed, dotted, etc.

- If specified **in the geometry**
  - It will apply uniformly to every **all the geometry**
- If assigned to a variable **in aes**
  - it will **vary with the variable** according to a scale documented in legend

```
ggplot(data, aes(x = V1, y = V2, size = V3)) +
  geom_point(color = "steelblue", alpha = .6)
```