Data visualization

Lecture 4

Louis SIRUGUE

CPES 2 - Fall 2022

Quick reminder

1. Packages

```
library(dplyr)
```

2. Main dplyr functions

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

Quick reminder

3. Merge data

```
a <- data.frame(x = c(1, 2, 3), y = c("a", "b", "c"))
b <- data.frame(x = c(4, 5, 6), y = c("d", "e", "f"))
c <- data.frame(x = 1:6, z = c("alpha", "bravo", "charlie", "delta", "echo", "foxtrot"))
```

```
a %>% bind_rows(b) %>% left_join(c, by = "x")
```

x	y	z
1	a	alpha
2	b	bravo
3	c	charlie
4	d	delta
5	e	echo
6	f	foxtrot

Quick reminder

4. Reshape data

country	year	share_tertiary	share_gdp
FRA	2015	44.69	3.40
USA	2015	46.52	3.21

```
data %>% pivot_longer(c(share_tertiary, share_gdp), names_to = "Variable", values_to = "Value")
```

country	year	Variable	Value
FRA	2015	share_tertiary	44.69
FRA	2015	share_gdp	3.40
USA	2015	share_tertiary	46.52
USA	2015	share_gdp	3.21

Warm up practice

10:00

1) Import `starbucks.csv` and `View()` the data

2) Inspect the structure of the data using `str()`

3) Use `summarise()` to compute for each beverage category the average number of calories and the number of different declinations (there is 1 row per declination)

4) Create a subset of the data called `maxcal` containing the variables `Beverage_category`, `Beverage_prep`, and `Calories`, for the 10 observations with the highest calorie values

You can use the row_number() function within filter() to use the row numbers as any other variable

You've got 10 minutes!

Solution

1) Import `starbucks.csv` and `View()` the data

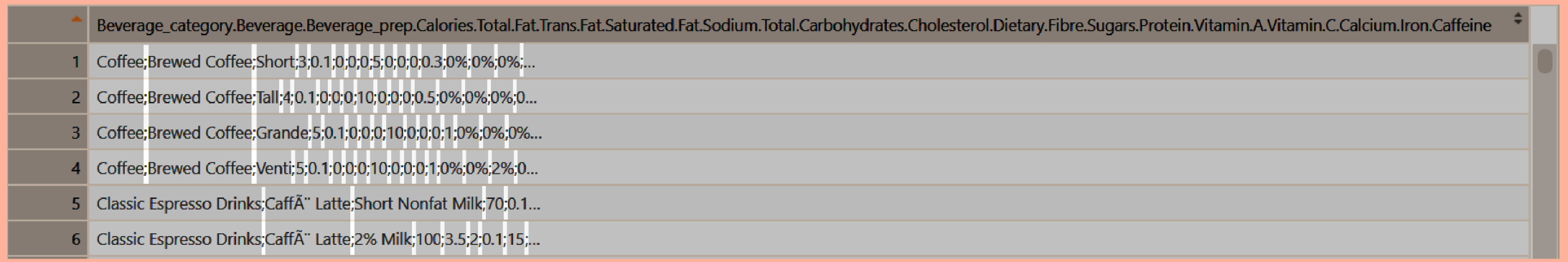
```
starbucks <- read.csv("C:/User/Documents/starbucks.csv")
View(starbucks)
```

	Beverage_category.Beverage.Beverage_prep.Calories.Total.Fat.Trans.Fat.Saturated.Fat.Sodium.Total.Carbohydrates.Cholesterol.Dietary.Fibre.Sugars.Protein.Vitamin.A.Vitamin.C.Calcium.Iron.Caffeine
1	Coffee;Brewed Coffee;Short;3;0.1;0;0;0;5;0;0;0;0.3;0%;0%;0%;...
2	Coffee;Brewed Coffee;Tall;4;0.1;0;0;0;10;0;0;0;0.5;0%;0%;0%;0...
3	Coffee;Brewed Coffee;Grande;5;0.1;0;0;0;10;0;0;0;1;0%;0%;0%...
4	Coffee;Brewed Coffee;Venti;5;0.1;0;0;0;10;0;0;0;1;0%;0%;2%;0...
5	Classic Espresso Drinks;CaffÃ¨ Latte;Short Nonfat Milk;70;0.1...
6	Classic Espresso Drinks;CaffÃ¨ Latte;2% Milk;100;3.5;2;0.1;15;...

Solution

1) Import `starbucks.csv` and `View()` the data

```
starbucks <- read.csv("C:/User/Documents/starbucks.csv")
View(starbucks)
```



	Beverage_category.Beverage.Beverage_prep.Calories.Total.Fat.Trans.Fat.Saturated.Fat.Sodium.Total.Carbohydrates.Cholesterol.Dietary.Fibre.Sugars.Protein.Vitamin.A.Vitamin.C.Calcium.Iron.Caffeine
1	Coffee;Brewed Coffee;Short;3;0.1;0;0;0;5;0;0;0;0.3;0%;0%;0%;...
2	Coffee;Brewed Coffee;Tall;4;0.1;0;0;0;10;0;0;0;0.5;0%;0%;0%;0...
3	Coffee;Brewed Coffee;Grande;5;0.1;0;0;0;10;0;0;0;1;0%;0%;0%...
4	Coffee;Brewed Coffee;Venti;5;0.1;0;0;0;10;0;0;0;1;0%;0%;2%;0...
5	Classic Espresso Drinks;CaffÃ¨ Latte;Short Nonfat Milk;70;0.1...
6	Classic Espresso Drinks;CaffÃ¨ Latte;2% Milk;100;3.5;2;0.1;15;...

- We only have **one variable** in which all values are **separated by semicolons**
  - We need to set the **sep** argument of the function accordingly

Solution

1) Import `starbucks.csv` and `View()` the data

```
starbucks <- read.csv("C:/User/Documents/starbucks.csv")
View(starbucks)
```

	Beverage_category.Beverage.Beverage_prep.Calories.Total.Fat.Trans.Fat.Saturated.Fat.Sodium.Total.Carbohydrates.Cholesterol.Dietary.Fibre.Sugars.Protein.Vitamin.A.Vitamin.C.Calcium.Iron.Caffeine
1	Coffee;Brewed Coffee;Short;3;0.1;0;0;0;5;0;0;0;0.3;0%;0%;0%;...
2	Coffee;Brewed Coffee;Tall;4;0.1;0;0;0;10;0;0;0;0.5;0%;0%;0%;0...
3	Coffee;Brewed Coffee;Grande;5;0.1;0;0;0;10;0;0;0;1;0%;0%;0%...
4	Coffee;Brewed Coffee;Venti;5;0.1;0;0;0;10;0;0;0;1;0%;0%;2%;0...
5	Classic Espresso Drinks;CaffÃ¨ Latte;Short Nonfat Milk;70;0.1...
6	Classic Espresso Drinks;CaffÃ¨ Latte;2% Milk;100;3.5;2;0.1;15;...

- We only have **one variable** in which all values are **separated by semicolons**
  - We need to set the **sep** argument of the function accordingly
  - Like last time, we also need to set the **encoding** argument correctly

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

Solution

2) Inspect the structure of the data using `str()`

```
str(starbucks)
```

```
## 'data.frame':    242 obs. of  18 variables:
##  $ Beverage_category  : chr  "Coffee" "Coffee" "Coffee" "Coffee" ...
##  $ Beverage           : chr  "Brewed Coffee" "Brewed Coffee" "Brewed Coffee" "Brewed Coffee" ...
##  $ Beverage_prep      : chr  "Short" "Tall" "Grande" "Venti" ...
##  $ Calories           : int  3 4 5 5 70 100 70 100 150 110 ...
##  $ Total.Fat          : chr  "0.1" "0.1" "0.1" "0.1" ...
##  $ Trans.Fat          : num  0 0 0 0 0.1 2 0.4 0.2 3 0.5 ...
##  $ Saturated.Fat      : num  0 0 0 0 0 0.1 0 0 0.2 0 ...
##  $ Sodium             : int  0 0 0 0 5 15 0 5 25 0 ...
##  $ Total.Carbohydrates: int  5 10 10 10 75 85 65 120 135 105 ...
##  $ Cholesterol        : int  0 0 0 0 10 10 6 15 15 10 ...
##  $ Dietary.Fibre      : int  0 0 0 0 0 0 1 0 0 1 ...
##  $ Sugars             : int  0 0 0 0 9 9 4 14 14 6 ...
##  $ Protein            : num  0.3 0.5 1 1 6 6 5 10 10 8 ...
##  $ Vitamin.A          : chr  "0%" "0%" "0%" "0%" ...
##  $ Vitamin.C          : chr  "0%" "0%" "0%" "0%" ...
##  . ...                . ...  ...  ...  ...  ...  ...
```

Solution

3) Use `summarise()` to compute for each beverage category the average number of calories and the number of different declinations (there is 1 row per declination)

```
starbucks %>%
  group_by(Beverage_category) %>%
  summarise(Declinations = n(),
            Mean_cal = mean(Calories))
```

```
## # A tibble: 9 x 3
##   Beverage_category                Declinations Mean_cal
##   <chr>                                   <int>    <dbl>
## 1 Classic Espresso Drinks                    58   140.
## 2 Coffee                                      4     4.25
## 3 Frappuccino® Blended Coffee                36   277.
## 4 Frappuccino® Blended Crème                 13   233.
## 5 Frappuccino® Light Blended Coffee          12   162.
## 6 Shaken Iced Beverages                      18   114.
## 7 Signature Espresso Drinks                  40   250
## 8 Smoothies                                   9   282.
## 9 Tazo® Tea Drinks                           52   177.
```

Solution

4) Create a subset of the data called `maxcal` containing the variables `Beverage_category`, `Beverage_prep`, and `Calories`, for the 10 observations with the highest calorie values

```
maxcal <- starbucks %>%
  arrange(-Calories) %>%
  select(Beverage_category, Beverage_prep, Calories) %>%
  filter(row_number() <= 10)

maxcal
```

```
##              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
## 4    Signature Espresso Drinks Venti Nonfat Milk      450
## 5              Tazo® Tea Drinks           2% Milk      450
## 6  Frappuccino® Blended Coffee           Soymilk      430
## 7  Frappuccino® Blended Coffee Venti Nonfat Milk      420
## 8    Signature Espresso Drinks           2% Milk      400
## 9              Tazo® Tea Drinks           Soymilk      390
## 10 Frappuccino® Blended Coffee        Whole Milk      390
```

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

- Let's use **ggplot** on data from the World Inequality database

```
wid <- read.csv("C:/User/Documents/wid.csv")
str(wid)
```

```
## 'data.frame':    1610 obs. of  6 variables:
##  $ country  : chr  "Algeria" "Algeria" "Algeria" "Algeria" ...
##  $ continent: chr  "Africa" "Africa" "Africa" "Africa" ...
##  $ year     : int  2010 2011 2012 2013 2014 2015 2016 2017 2018 2019 ...
##  $ fshare   : num  0.0992 0.112 0.1201 0.1206 0.116 ...
##  $ top1     : num  0.1003 0.0991 0.0991 0.0991 0.0991 ...
##  $ inc_head : num  12611 12620 12634 12532 12546 ...
```

- It contains 1610 observations and 6 variables:
  - **continent/country/year**: Observation level
  - **f_share**: Female labor income share
  - **top1**: Top 1% income share
  - **inc_head**: Per adult national income

1. The `ggplot()` function

1.1. Basic structure

- **ggplot()** from ggplot2 is what 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
```

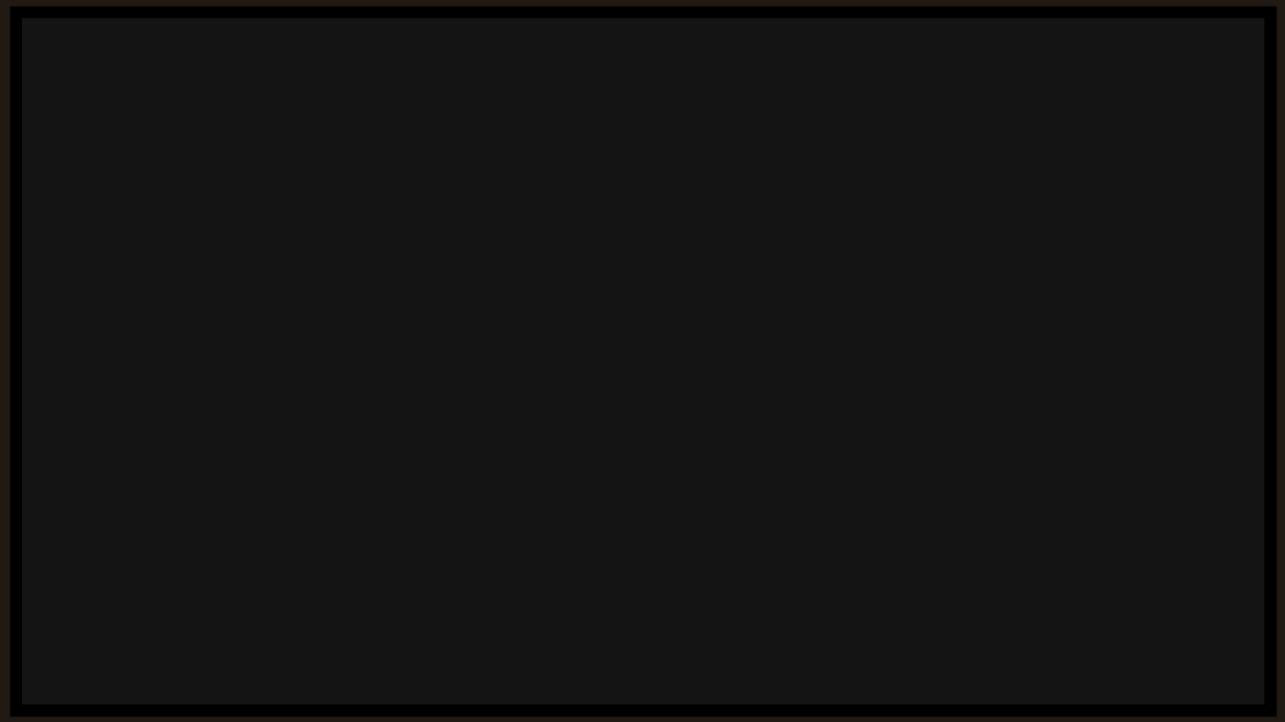
- You can also **apply** the `ggplot()` function to your data with a **pipe**

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

1. The `ggplot()` function

1.1. Basic structure

```
ggplot(wid)                                    # Data
                                               #
```



- We assigned data to ggplot()
  - But our plot is empty

1. The `ggplot()` function

1.1. Basic structure

```
ggplot(wid, aes(x = inc_head, y = top1))          # Data & aesthetics
                                                  #
```

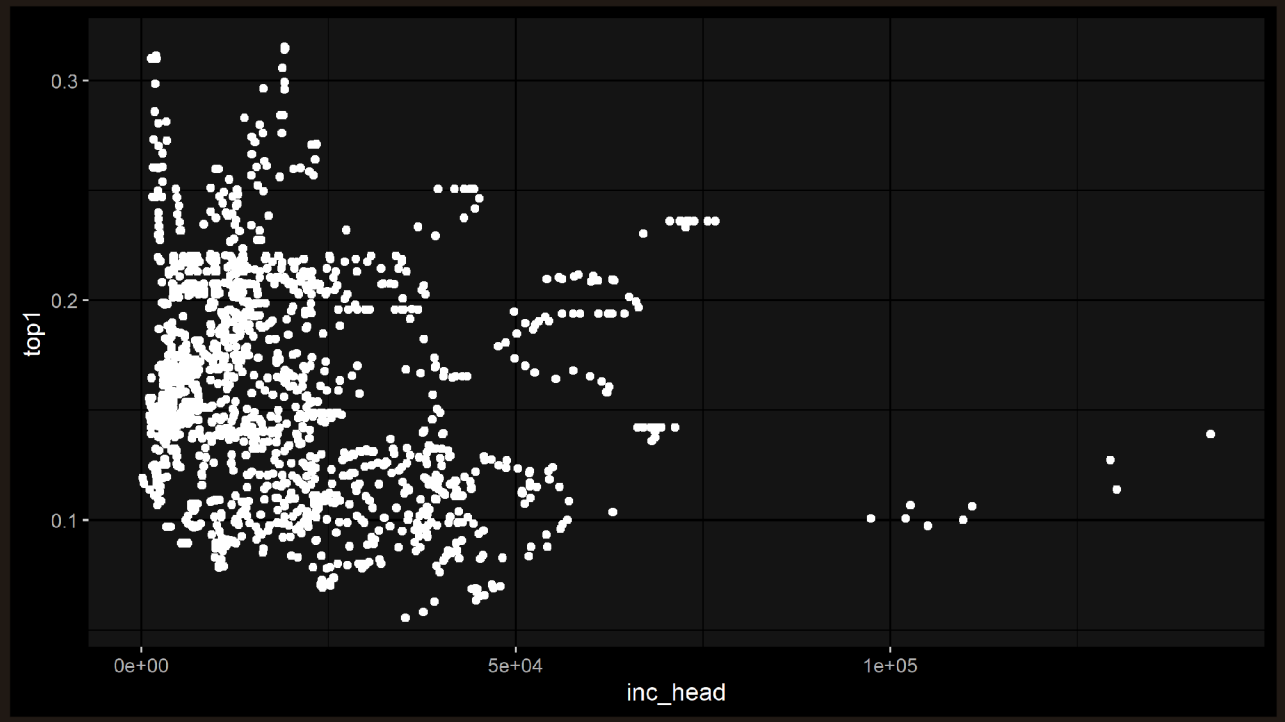
- We assigned data to ggplot()
  - But our plot is empty

- We assigned variables to axes
  - But still nothing

1. The `ggplot()` function

1.1. Basic structure

```
ggplot(wid, aes(x = inc_head, y = top1)) +          # Data & aesthetics
  geom_point()                                      # Geometry
```



- We assigned data to ggplot()
  - But our plot is empty
- We assigned variables to axes
  - But still nothing
- We need a geometry
  - Points for instance

1. The `ggplot()` function

1.1. Basic structure

- You can the save the plot using the **ggsave()** function
  - You just need to specify the **output destination** and it will **save** what is in your **plot panel**

```
ggsave("C:/User/Documents/wid.png")
```

- You can also **modify** the following options, which take the **parameters of your plot** panel if unspecified:
  - **plot:** ggplot object
  - **width:** width of the plot
  - **height:** height of the plot
  - **unit:** unit of the plot size ("in", "cm", "mm", "px")
  - **dpi:** pixel density, default to 300px/in

```
ggsave("wid.png", plot = last_plot(), width = 16, height = 9, unit = "cm", dpi = 900)
```

1. The `ggplot()` function

1.2. Axes

- Axes can be modified with **scale functions**, whose names depend on:
  - The axis to modify
  - The type of variable assigned to the axis

Basic scale functions

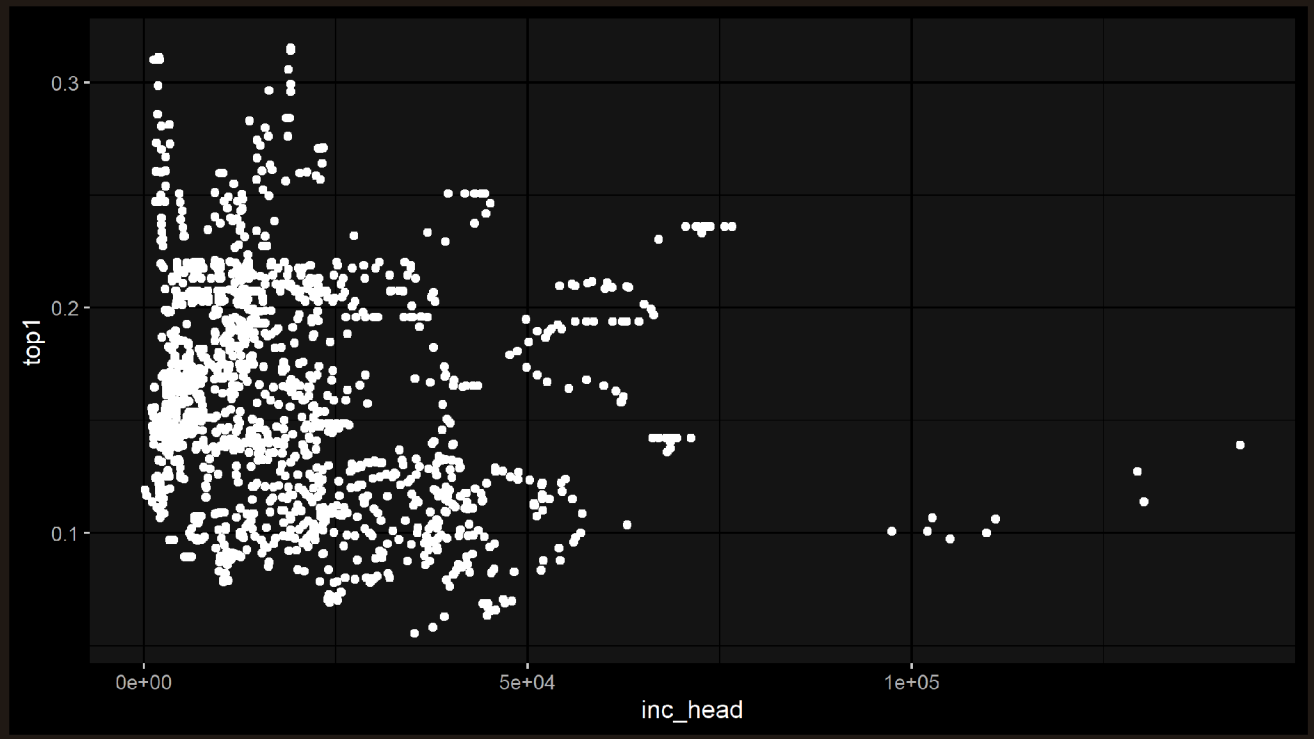
Axis	x-axis	y-axis
Continuous	scale_x_continuous()	scale_y_continuous()
Discrete	scale_x_discrete()	scale_y_discrete()

- 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

1. The `ggplot()` function

1.2. Axes

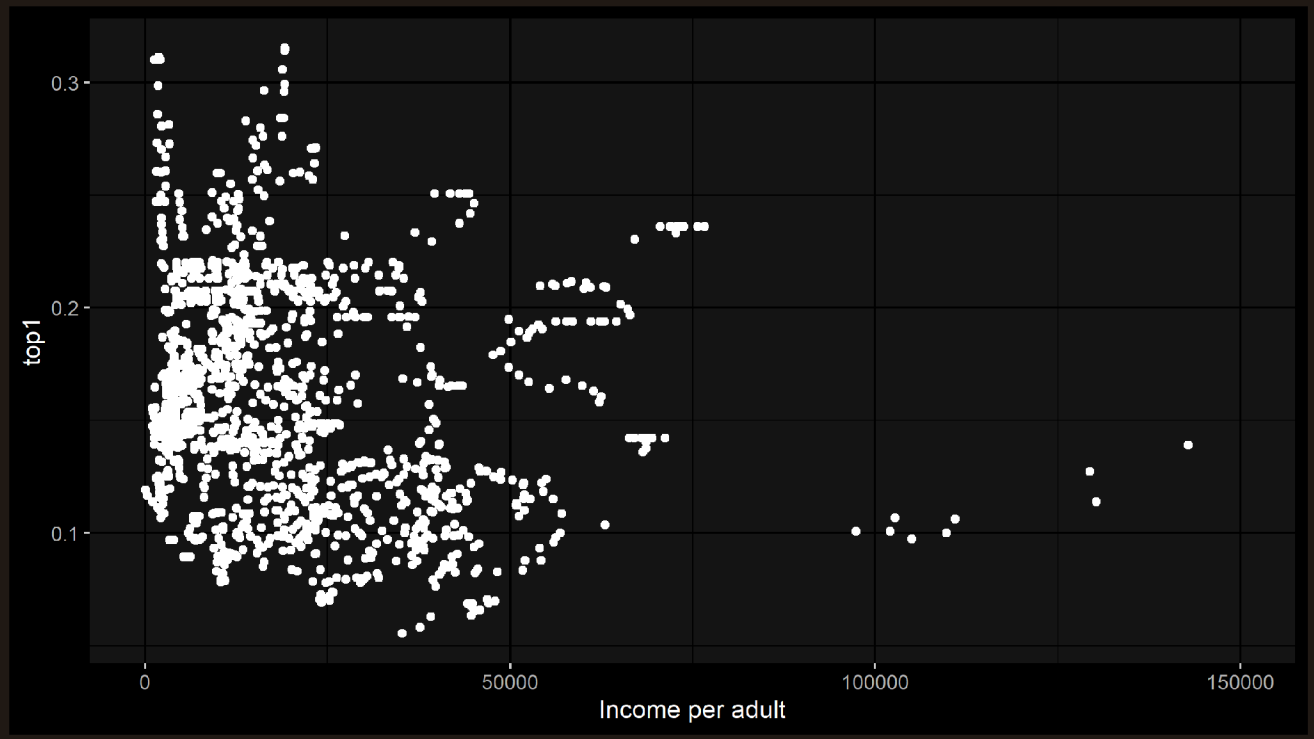
```
ggplot(wid, aes(x = inc_head, y = top1)) + geom_point()                    # Basic structure
                                                                           #
```



1. The `ggplot()` function

1.2. Axes

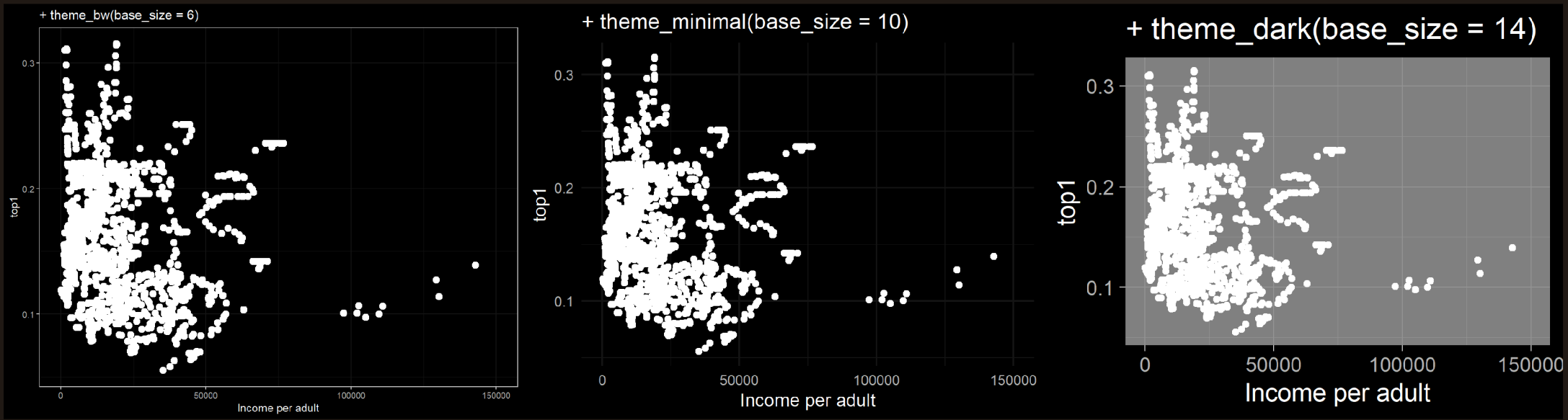
```
ggplot(wid, aes(x = inc_head, y = top1)) + geom_point() +                  # Basic structure
   scale_x_continuous(name = "Income per adult", limits = c(0, 150000))   # Scale function
```



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

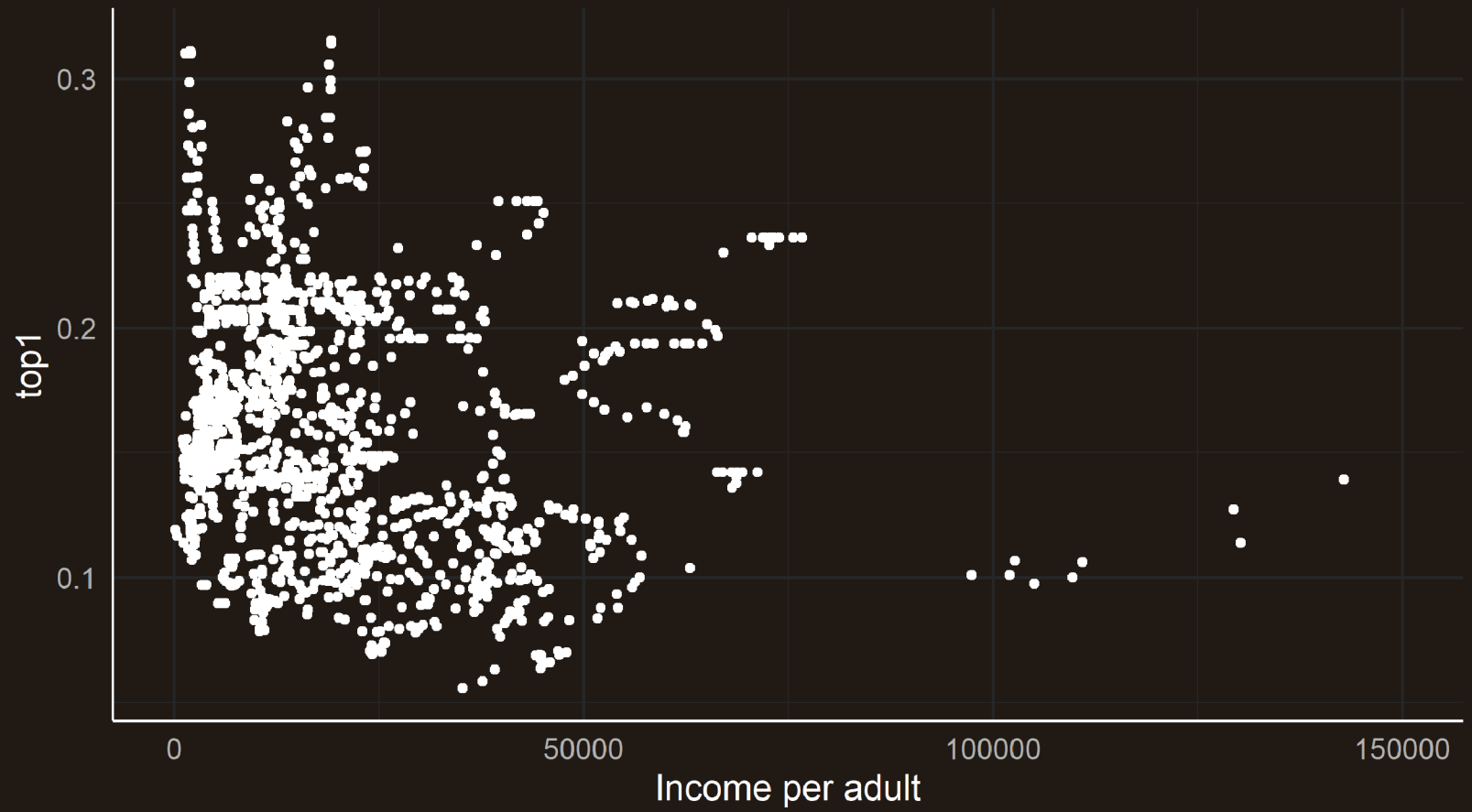
1.3. Theme()

- 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

```
  # Basic structure
ggplot(wid, aes(x = inc_head, y = top1)) + geom_point() +
  # Axis
  scale_x_continuous(name = "Income per adult", limits = c(0, 150000)) +
  # 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()

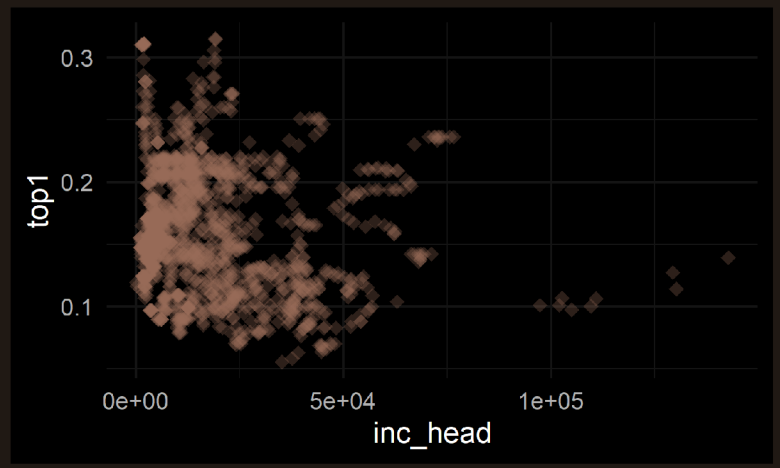


1. The `ggplot()` function

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(wid, aes(x = inc_head, y = top1)) +
  geom_point(size = 3,
             color = "#6794A7",
             alpha = .3,
             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 = )
```

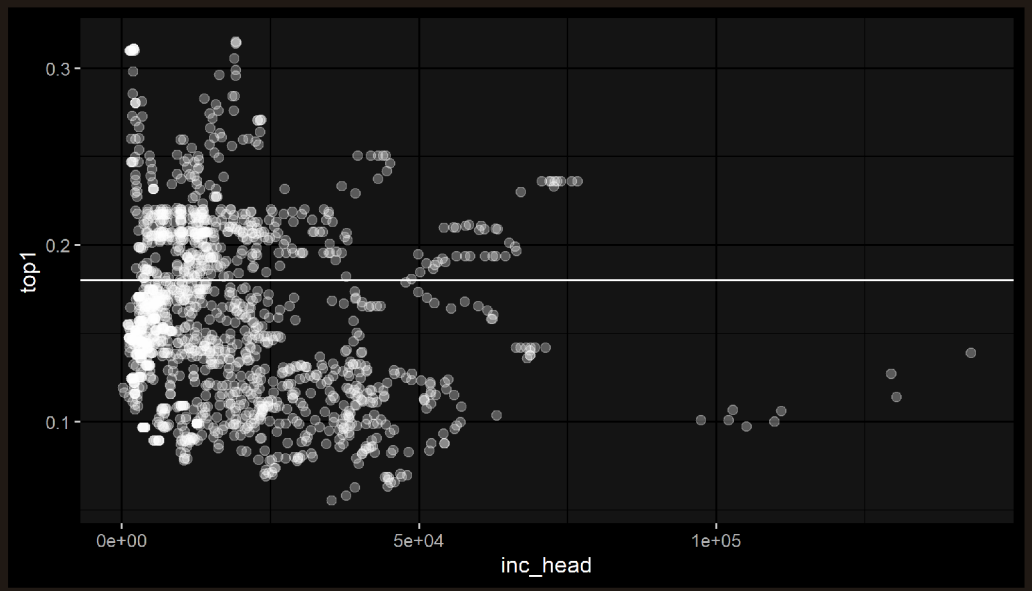
- And **punctual text annotations** can be added with `annotate()`

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

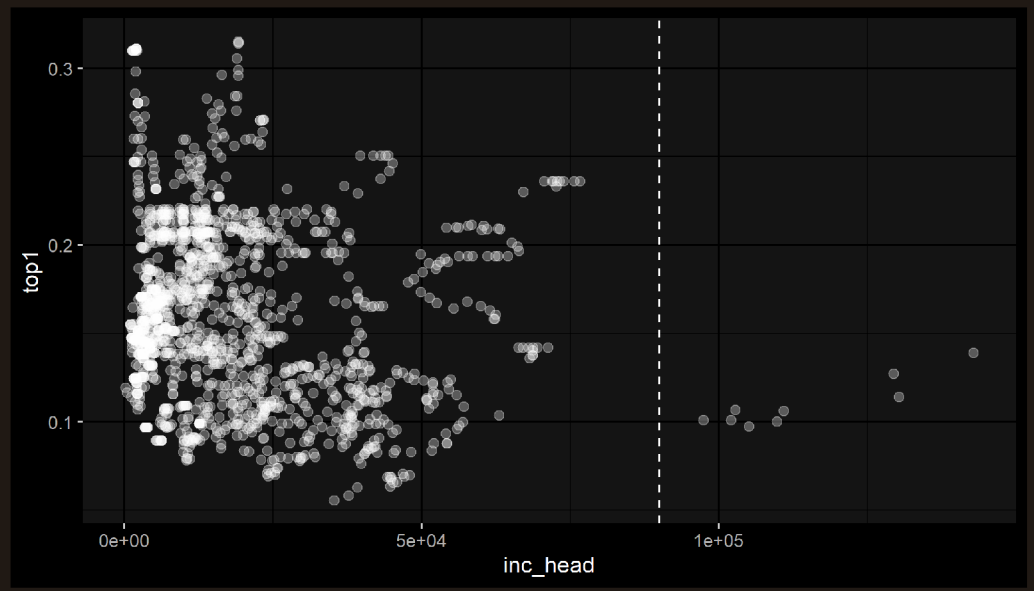
1. The `ggplot()` function

1.4. Annotation: Adding lines

```
ggplot(wid, aes(x = inc_head, y = top1)) +
  geom_point(size = 2, alpha = .3) +
  geom_hline(yintercept = .17)
```



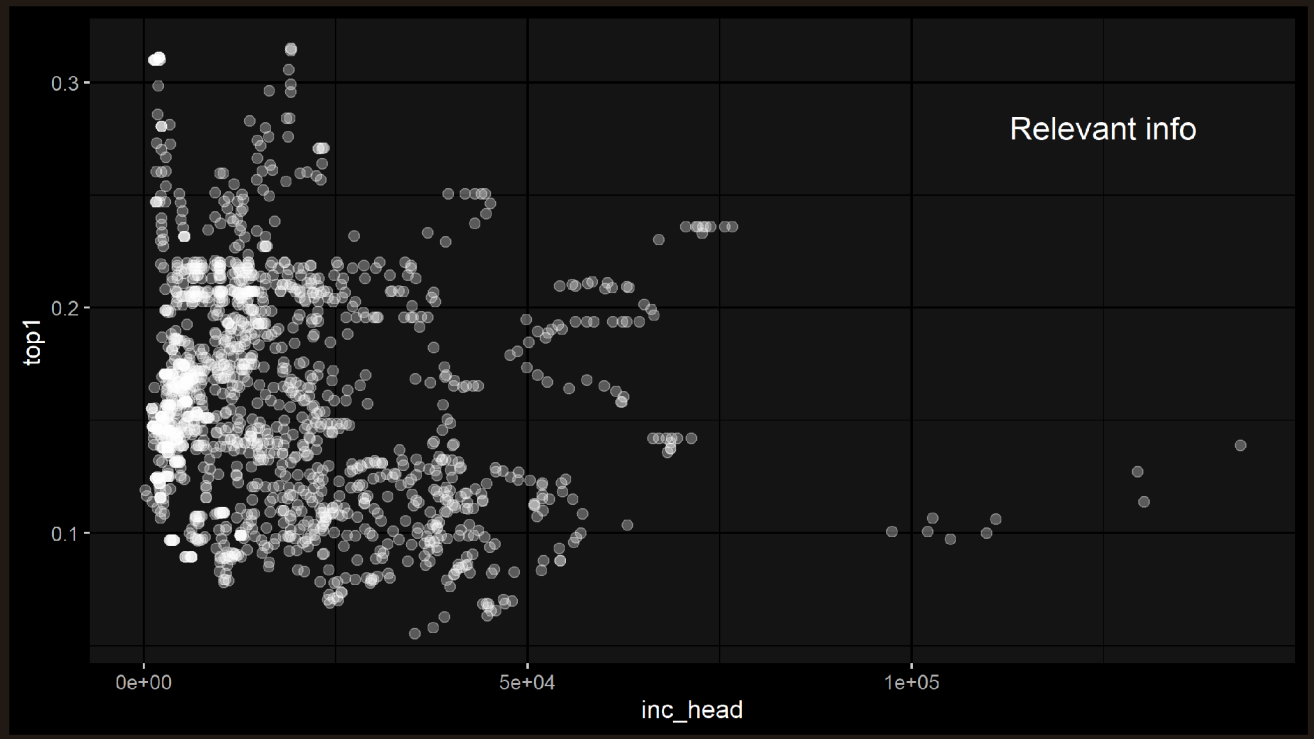

```
ggplot(wid, aes(x = inc_head, y = top1)) +
  geom_point(size = 2, alpha = .3) +
  geom_vline(xintercept = 90000,
             linetype = "dashed")
```



1. The `ggplot()` function

1.4. Annotation: Adding text

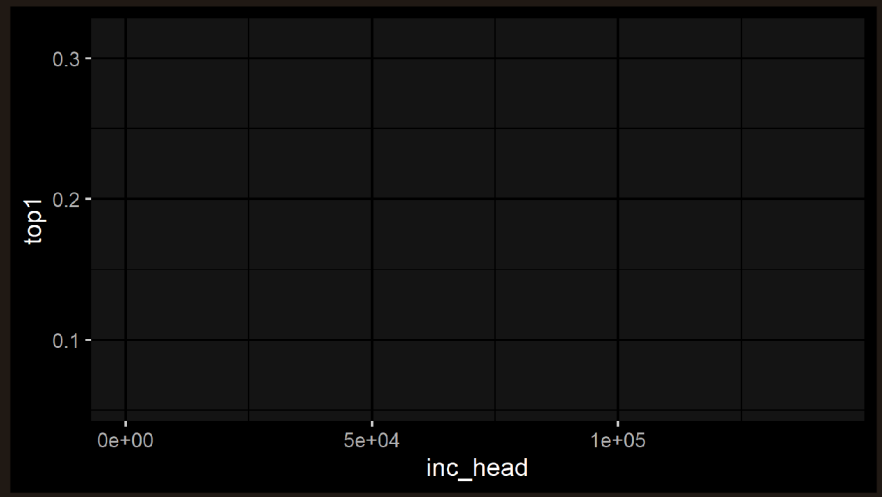
```
ggplot(wid, aes(x = inc_head, y = top1)) + geom_point(size = 2, alpha = .3) +
  annotate("text", x = 125000, y = .28, label = "Relevant info", size = 5)
```



1. The `ggplot()` function

Combining everything

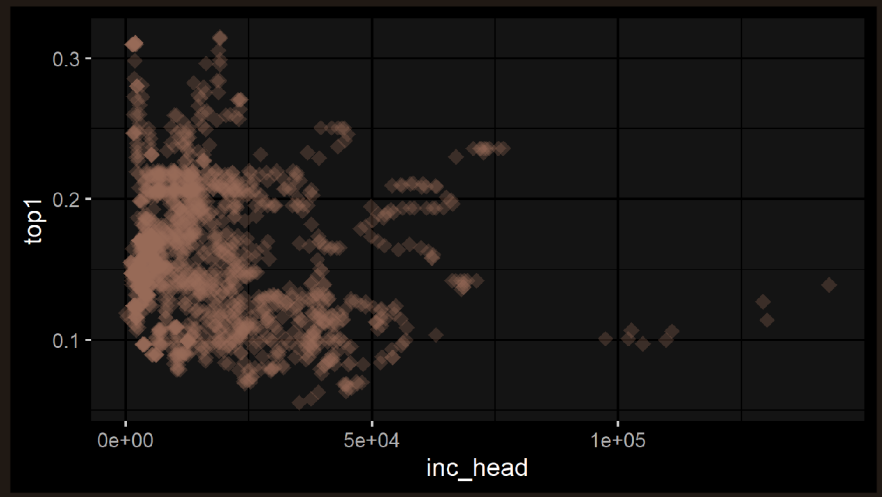
```
ggplot(wid, aes(x = inc_head, y = top1))
                                                                    #
                                                                    #
                                                                    #
                                                                    #
                                                                    #
                                                                    #
                                                                    #
```



1. The `ggplot()` function

Combining everything

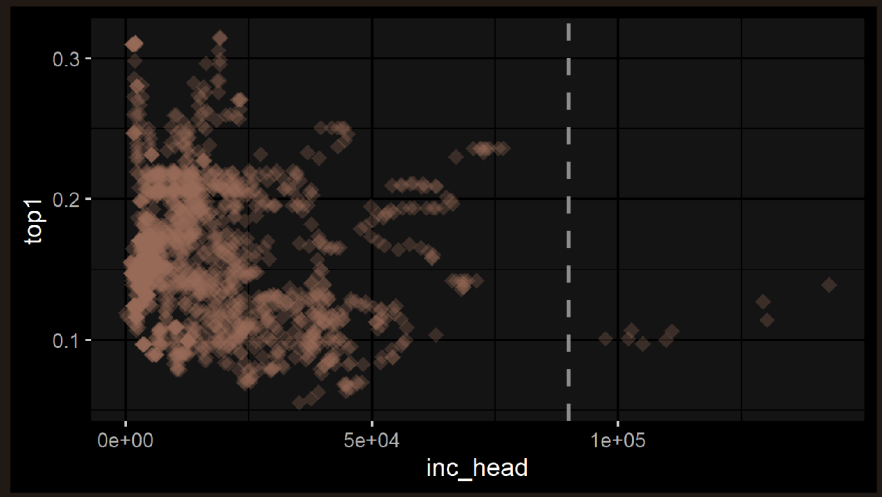
```
ggplot(wid, aes(x = inc_head, y = top1)) +
  geom_point(size = 3,  color = "#6794A7", alpha = .3, shape = 18)
                                                                  #
                                                                  #
                                                                  #
                                                                  #
                                                                  #
                                                                  #
```



1. The `ggplot()` function

Combining everything

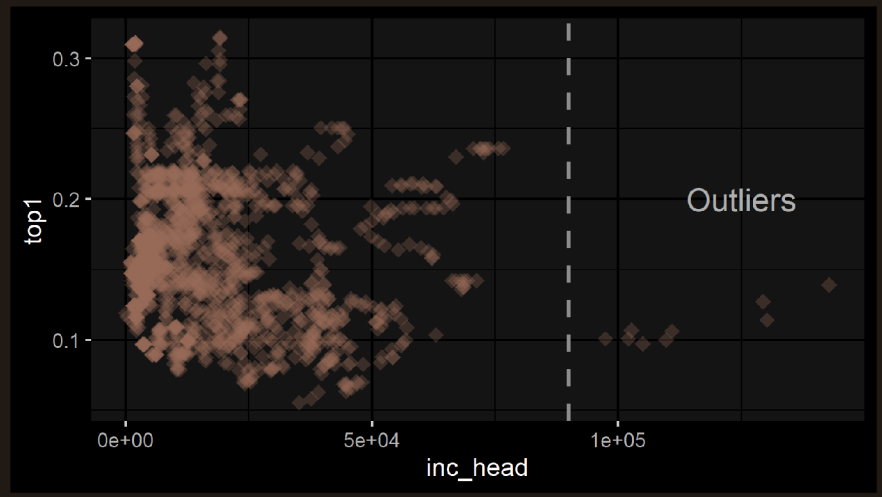
```
ggplot(wid, aes(x = inc_head, y = top1)) +
  geom_point(size = 3,  color = "#6794A7", alpha = .3, shape = 18) +
  geom_vline(xintercept = 90000, linetype = "dashed", size = 1, color = "#727272")
                                                                                #
                                                                                #
                                                                                #
                                                                                #
                                                                                #
```



1. The `ggplot()` function

Combining everything

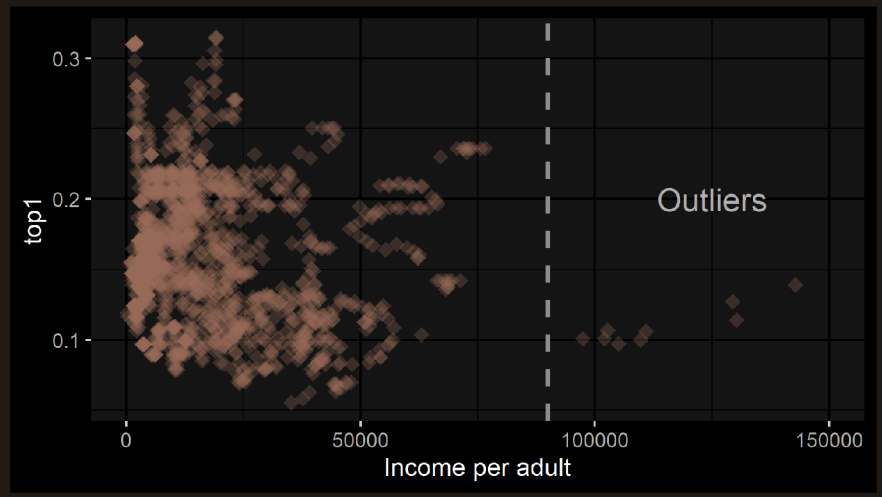
```
ggplot(wid, aes(x = inc_head, y = top1)) +
  geom_point(size = 3,  color = "#6794A7", alpha = .3, shape = 18) +
  geom_vline(xintercept = 90000, linetype = "dashed", size = 1, color = "#727272") +
  annotate("text", x = 125000, y = .2, label = "Outliers", size = 5, color = "#505050")
                                                                                    #
                                                                                    #
                                                                                    #
                                                                                    #
```



1. The `ggplot()` function

Combining everything

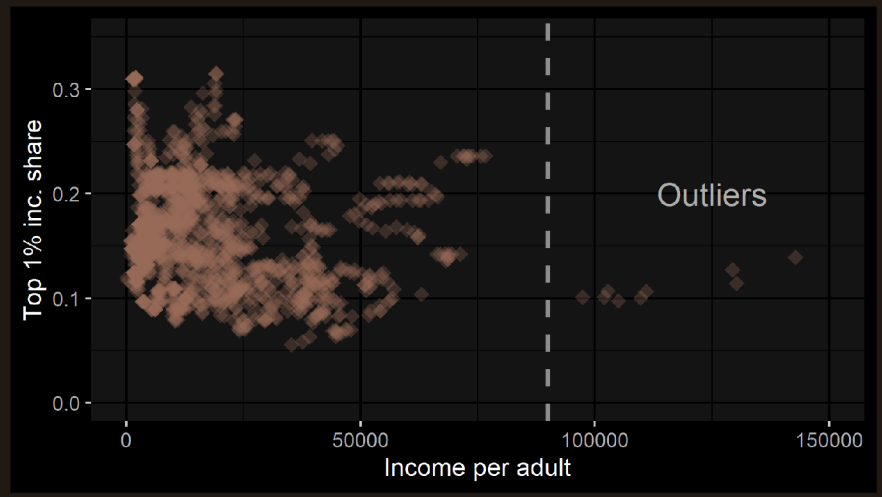
```
ggplot(wid, aes(x = inc_head, y = top1)) +
  geom_point(size = 3,  color = "#6794A7", alpha = .3, shape = 18) +
  geom_vline(xintercept = 90000, linetype = "dashed", size = 1, color = "#727272") +
  annotate("text", x = 125000, y = .2, label = "Outliers", size = 5, color = "#505050") +
  scale_x_continuous(name = "Income per adult", limits = c(0, 150000))
                                                                                    #
                                                                                    #
                                                                                    #
```



1. The `ggplot()` function

Combining everything

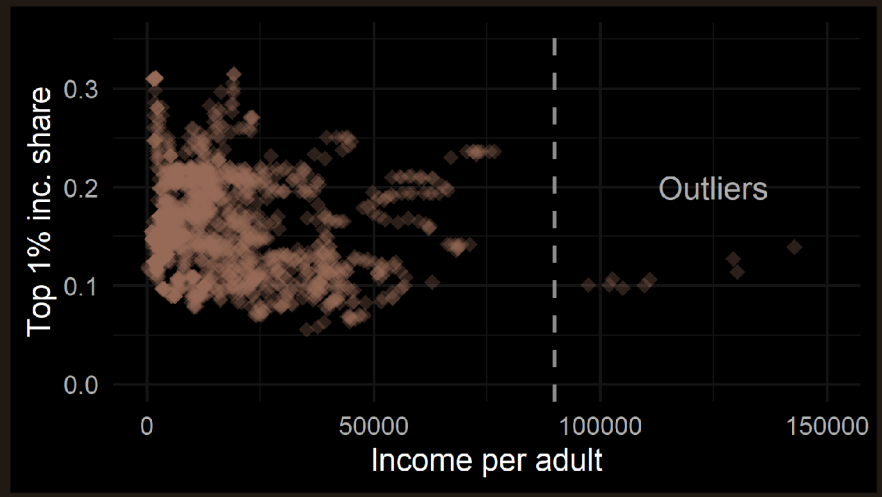
```
ggplot(wid, aes(x = inc_head, y = top1)) +
  geom_point(size = 3,  color = "#6794A7", alpha = .3, shape = 18) +
  geom_vline(xintercept = 90000, linetype = "dashed", size = 1, color = "#727272") +
  annotate("text", x = 125000, y = .2, label = "Outliers", size = 5, color = "#505050") +
  scale_x_continuous(name = "Income per adult", limits = c(0, 150000)) +
  scale_y_continuous(name = "Top 1% inc. share", limits = c(0, .35))
                                                                                    #
                                                                                    #
```



1. The `ggplot()` function

Combining everything

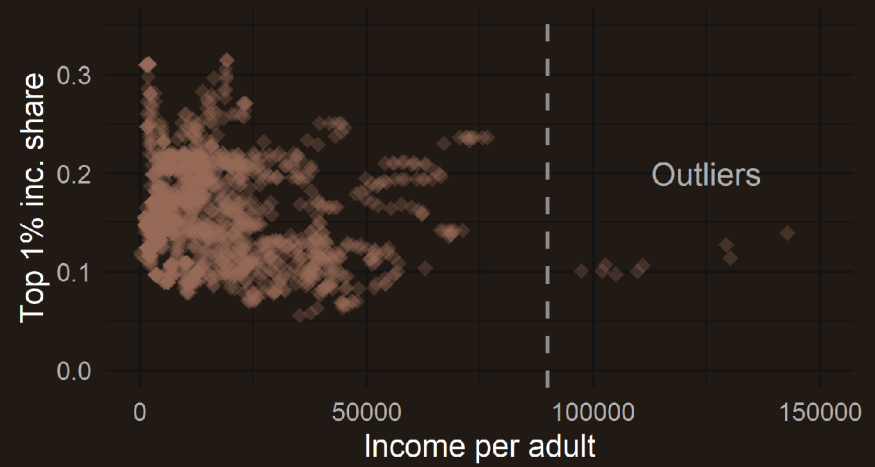
```
ggplot(wid, aes(x = inc_head, y = top1)) +
  geom_point(size = 3,  color = "#6794A7", alpha = .3, shape = 18) +
  geom_vline(xintercept = 90000, linetype = "dashed", size = 1, color = "#727272") +
  annotate("text", x = 125000, y = .2, label = "Outliers", size = 5, color = "#505050") +
  scale_x_continuous(name = "Income per adult", limits = c(0, 150000)) +
  scale_y_continuous(name = "Top 1% inc. share", limits = c(0, .35)) +
  theme_minimal(base_size = 14)
                                                                                    #
```



1. The `ggplot()` function

Combining everything

```
ggplot(wid, aes(x = inc_head, y = top1)) +
  geom_point(size = 3,  color = "#6794A7", alpha = .3, shape = 18) +
  geom_vline(xintercept = 90000, linetype = "dashed", size = 1, color = "#727272") +
  annotate("text", x = 125000, y = .2, label = "Outliers", size = 5, color = "#505050") +
  scale_x_continuous(name = "Income per adult", limits = c(0, 150000)) +
  scale_y_continuous(name = "Top 1% inc. share", limits = c(0, .35)) +
  theme_minimal(base_size = 14) +
  theme(plot.background = element_rect(fill = "#DFE6EB", colour = "#DFE6EB"))
```



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

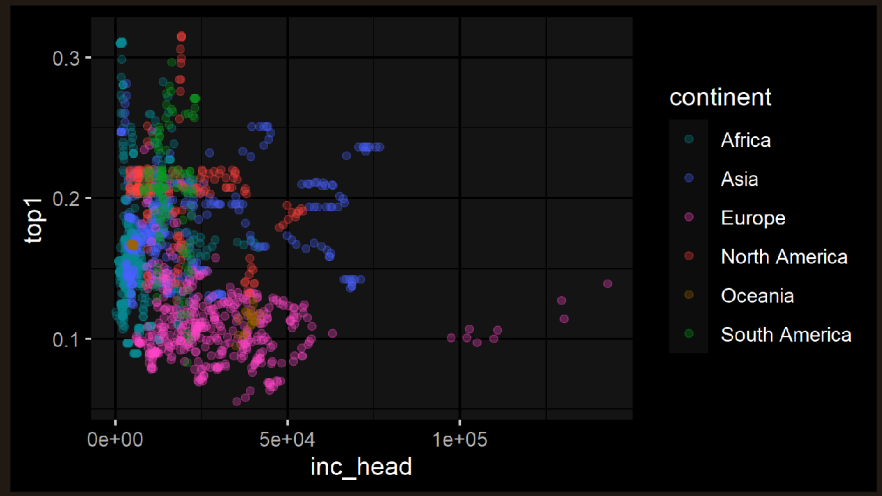
2. Adding dimensions
- 2.1. More axes
- 2.2. More facets
- 2.3. More labels

2. Adding dimensions

2.1. More axes

- In some cases you may want to **convey information** using other means than position on an axis
  - The **color, size, or shape** of a geometry can be used to represent a **third variable**
- We can assign **different colors to different points** depending on the associated continent
  - Continent should be assigned to the *"color axis"* in **aes()**

```
ggplot(wid, aes(x = inc_head, y = top1, color = continent)) + geom_point(alpha = .3)
```

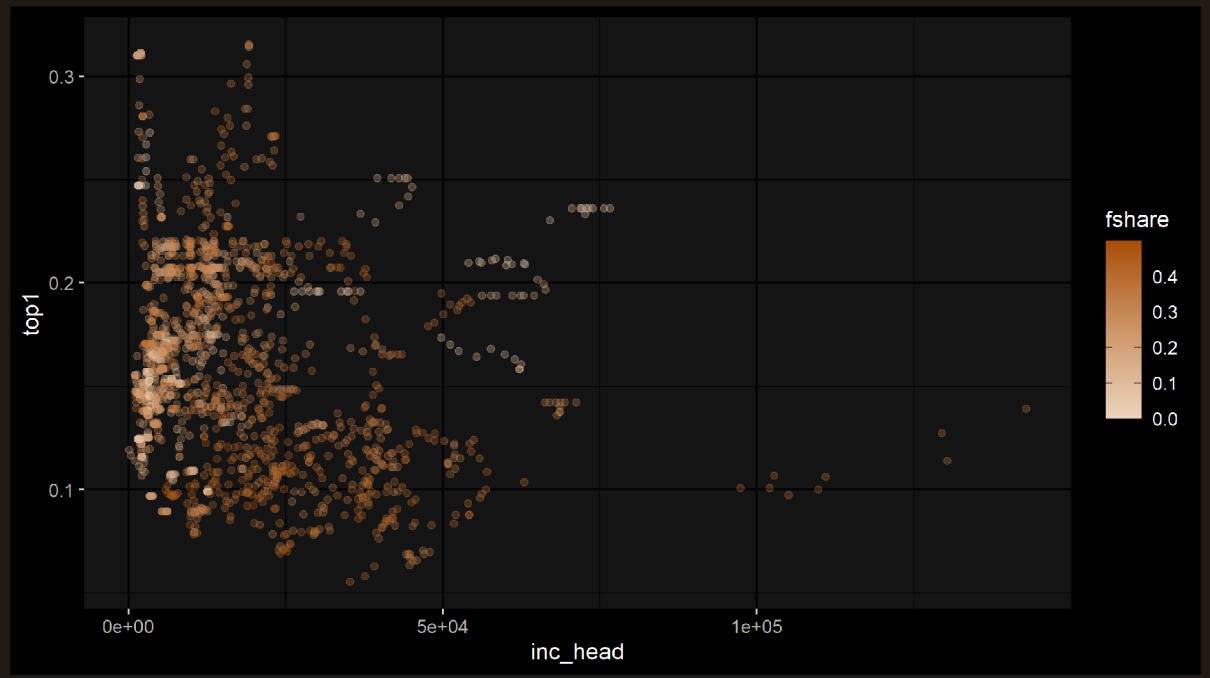


2. Adding dimensions

2.1. More axes

- If the variable assigned to the color axis is continuous, a color gradient will be used

```
ggplot(wid, aes(x = inc_head, y = top1, color = fshare)) + geom_point(alpha = .3)
```



2. Adding dimensions

2.1. More axes

- Because there is no proper *"color axis"*, a **legend** is generated
  - It can be seen as a *"color"* axis, just like the x- and y-axis
  - And should then be modified with a ***scale* function**

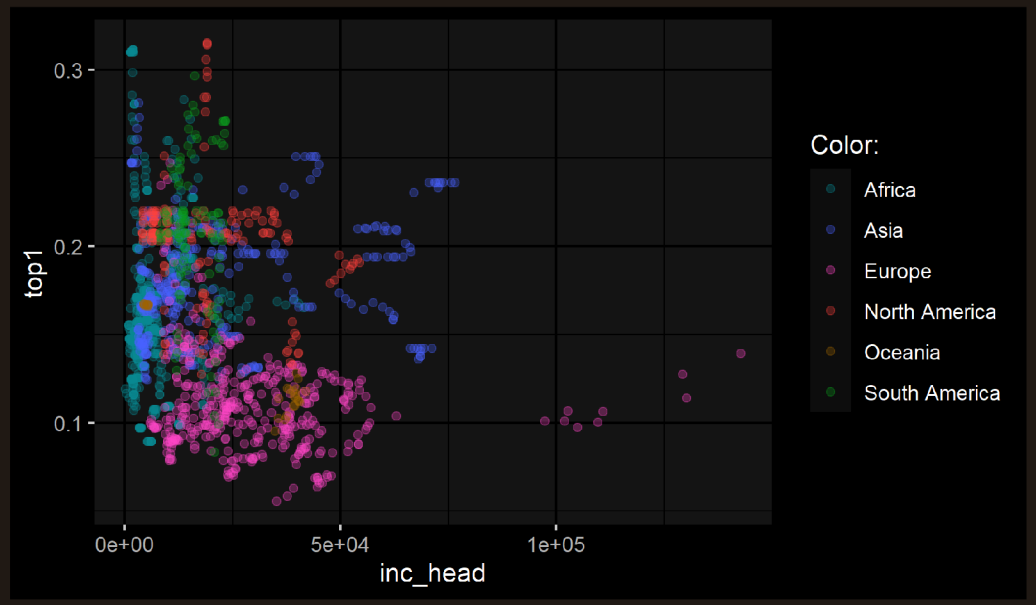
Discrete color variable

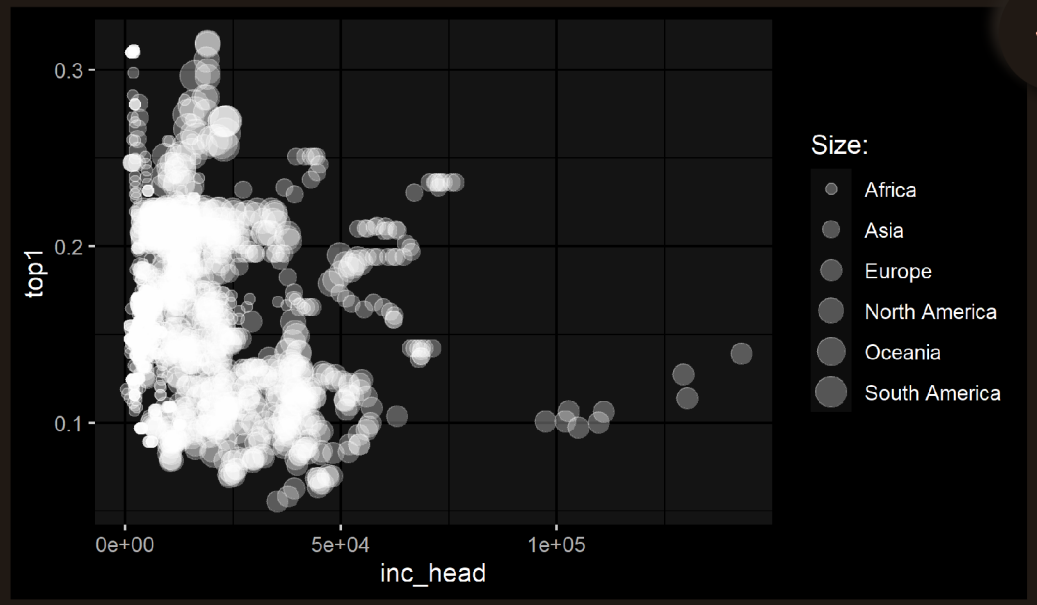
```
plot + scale_color_manual(
  name = "Title", values = c("red", "blue")
)
```

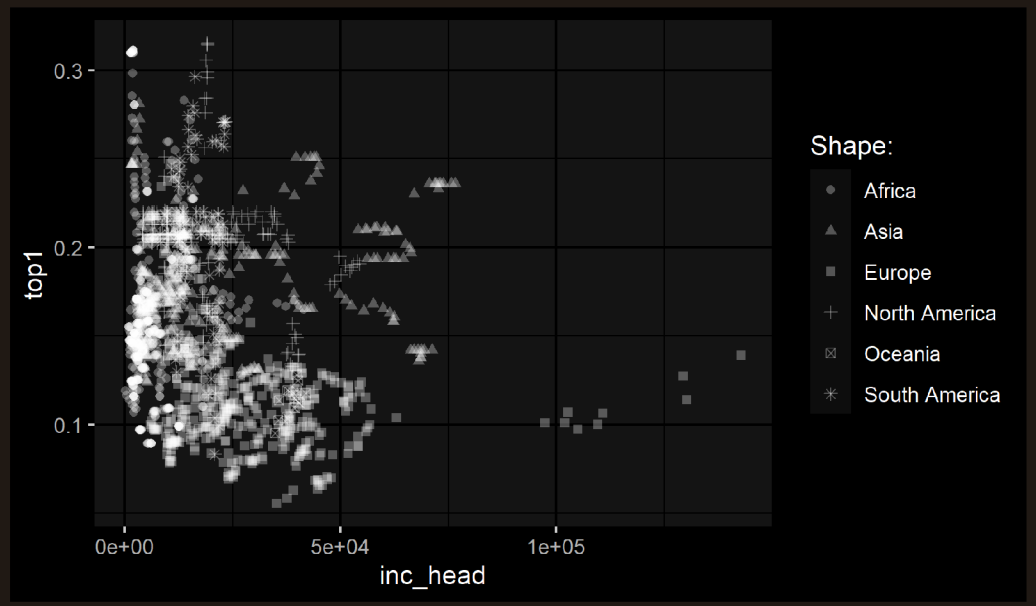
Continuous color variable

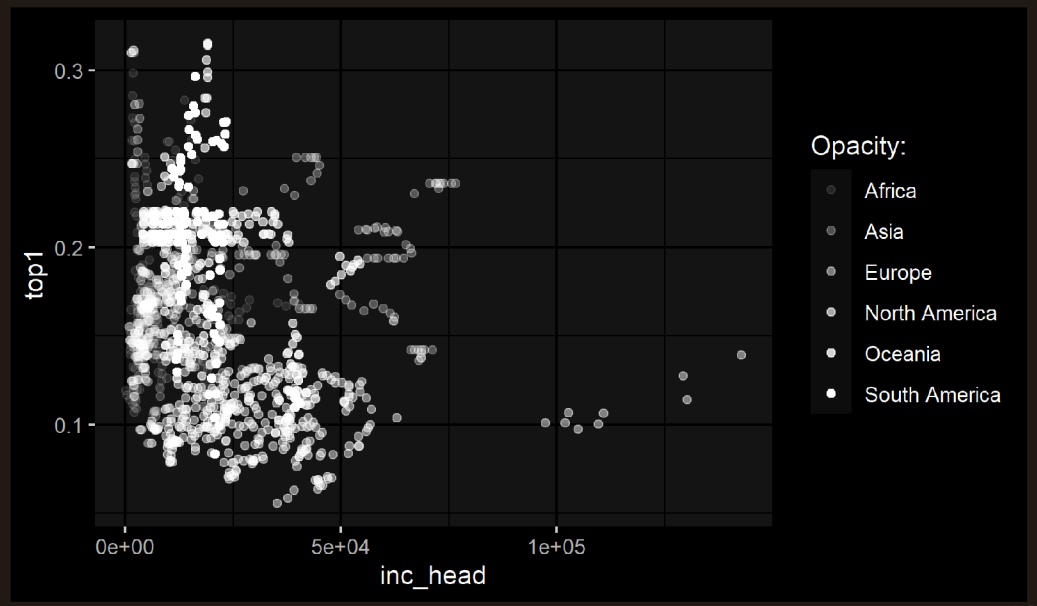
```
plot + scale_color_gradient(
  name = "Title", low = "red", high = "blue"
)
```

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


0.3
0.2
0.1
top1
0e+00
5e+04
1e+05
inc_head
Color:
Africa
Asia
Europe
North America
Oceania
South America


0.3
0.2
0.1
top1
0e+00
5e+04
1e+05
inc_head
Size:
Africa
Asia
Europe
North America
Oceania
South America


0.3
0.2
0.1
top1
0e+00
5e+04
1e+05
inc_head
Shape:
Africa
Asia
Europe
North America
Oceania
South America


0.3
0.2
0.1
top1
0e+00
5e+04
1e+05
inc_head
Opacity:
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
- In `facet_wrap()`, the faceting variable must be preceded by a tilde as the first argument:

```
ggplot(wid, aes(x = inc_head, y = top1)) + geom_point() +
  facet_wrap(~continent)
```

- You can then choose the facet arrangement:
  - **nrow** to indicate the number of rows
  - **ncol** to indicate the number of columns
- As well as which **scale** should be:
  - **free:** adjusted separately to each facet
  - **fixed:** common to all facets

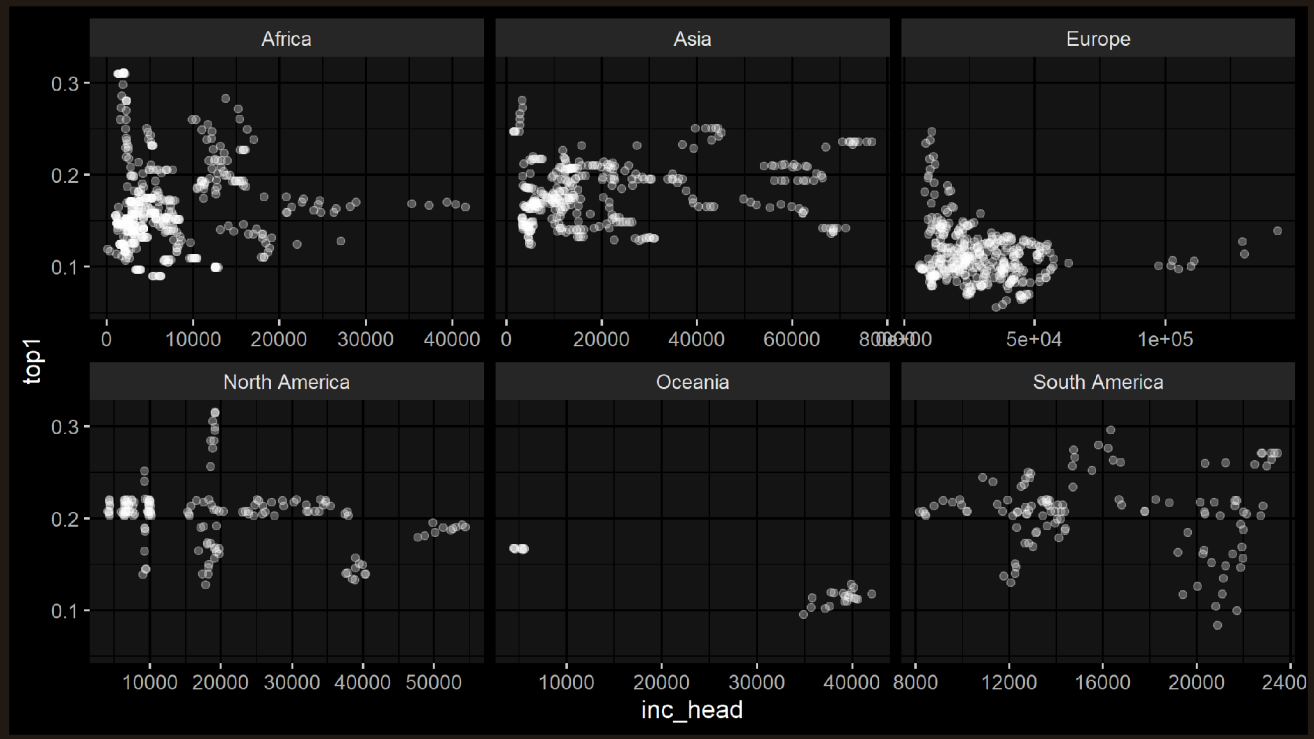
scales argument in facet_wrap()

	x fixed	**x free**
y fixed	scales = "fixed"	scales = "free_x"
y free	scales = "free_y"	scales = "free"

2. Adding dimensions

2.2. More facets

```
ggplot(wid, aes(x = inc_head, y = top1)) + geom_point(alpha = .3) +
  facet_wrap(~continent, ncol = 3, scales = "free_x")
```

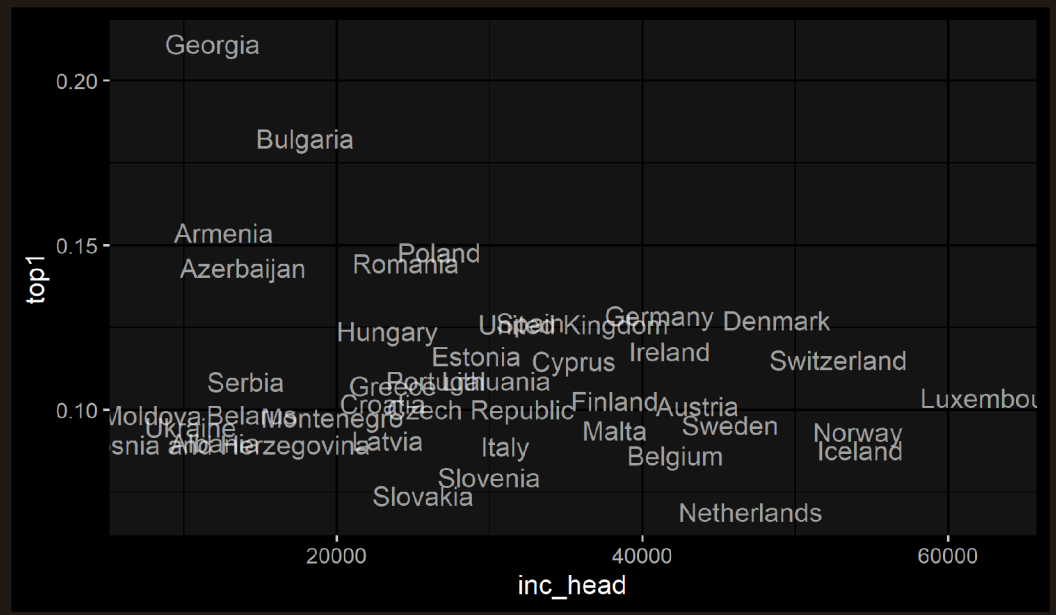


2. Adding dimensions

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**

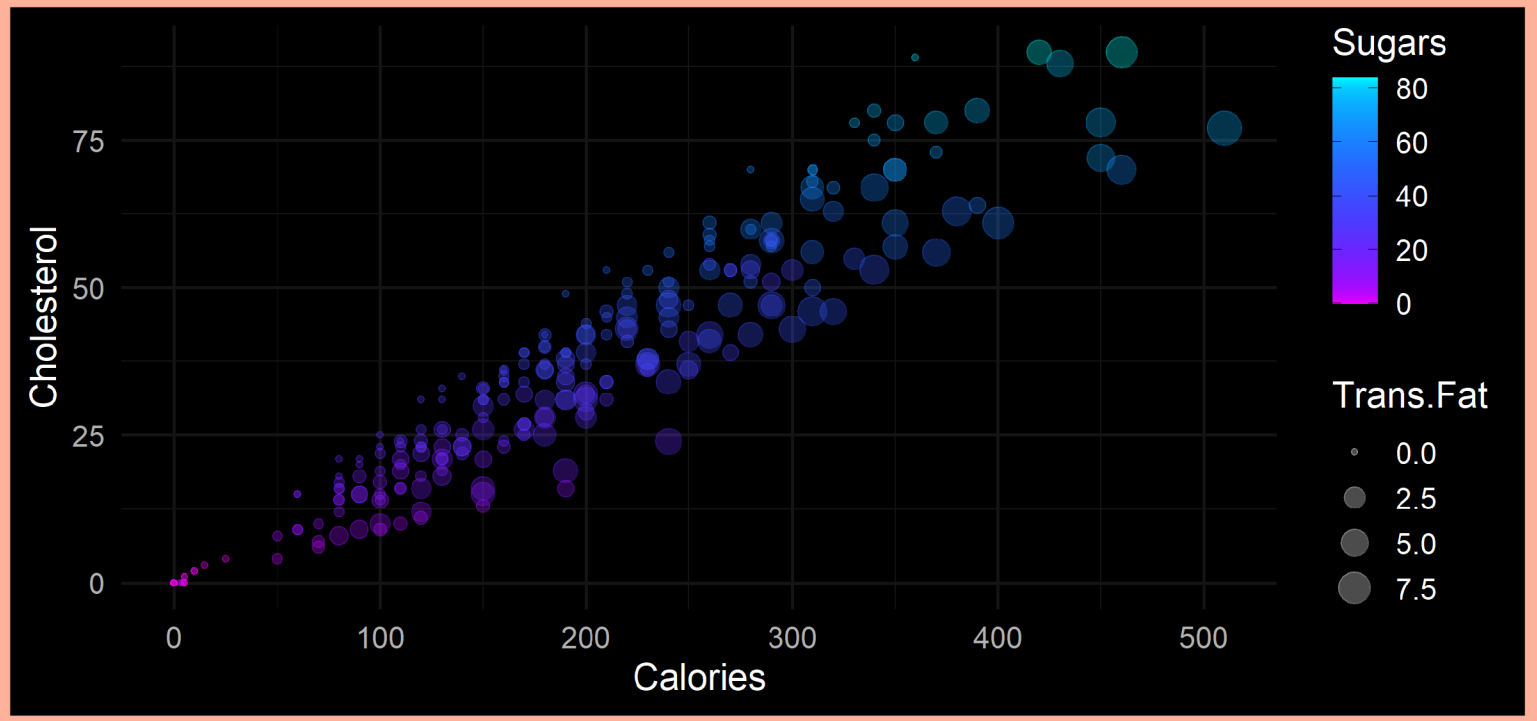
```
ggplot(wid %>% filter(year == 2019 & continent == "Europe"), # subset so that we can see something
  aes(x = inc_head, y = top1, label = country)) + geom_text(alpha = .6)
```



Practice

10:00

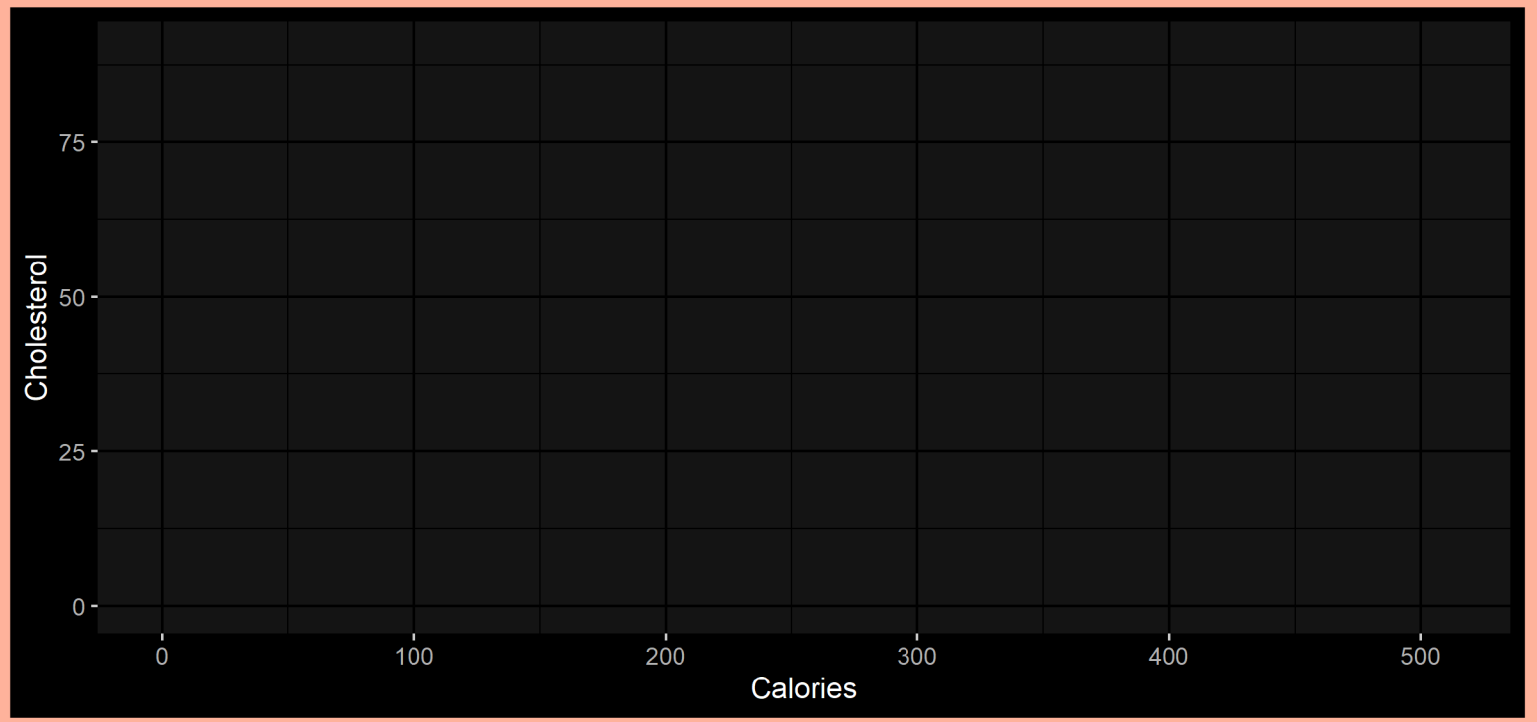
1) Reproduce this graph with the `starbucks` dataset



You've got 10 minutes!

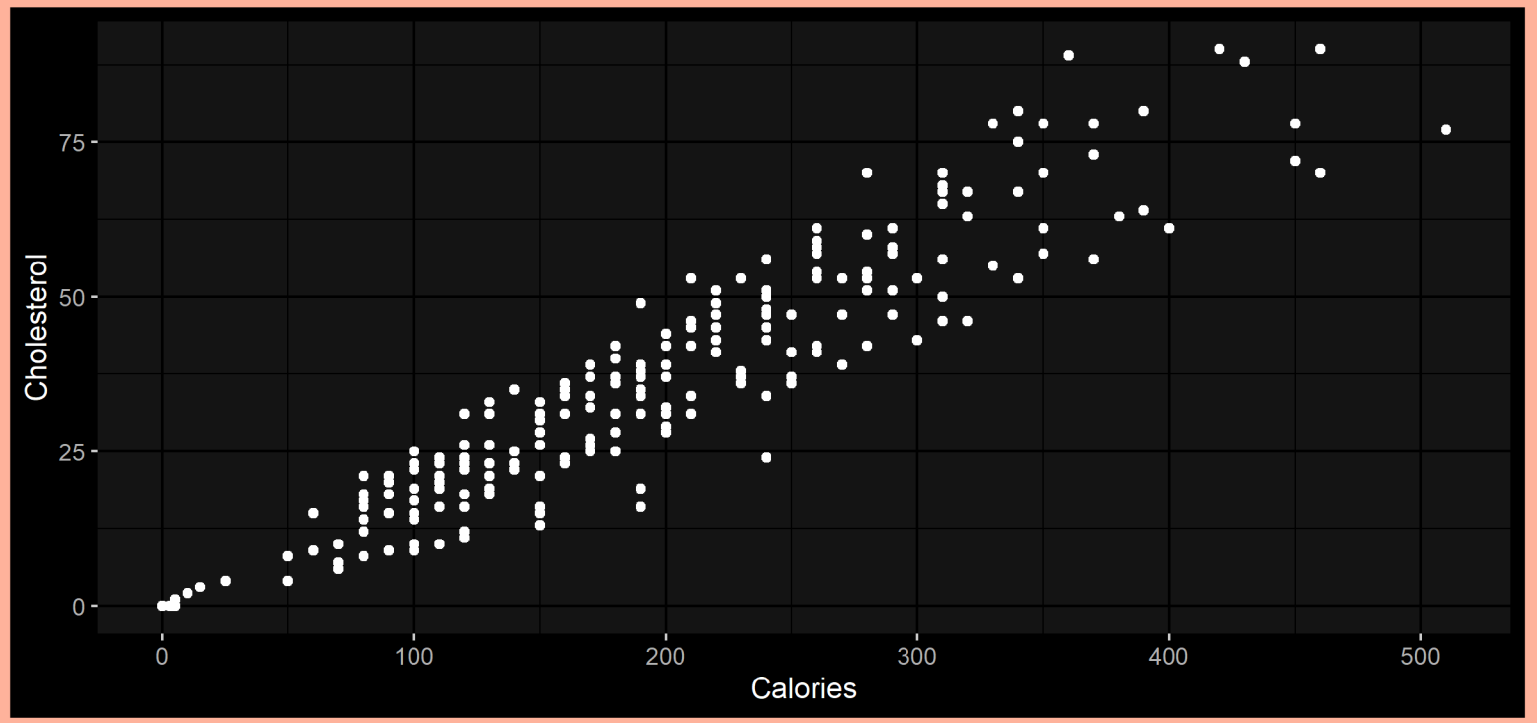
Solution

```
ggplot(starbucks,
       aes(x = Calories, y = Cholesterol))
                                                                    #
                                                                    #
                                                                    #
```



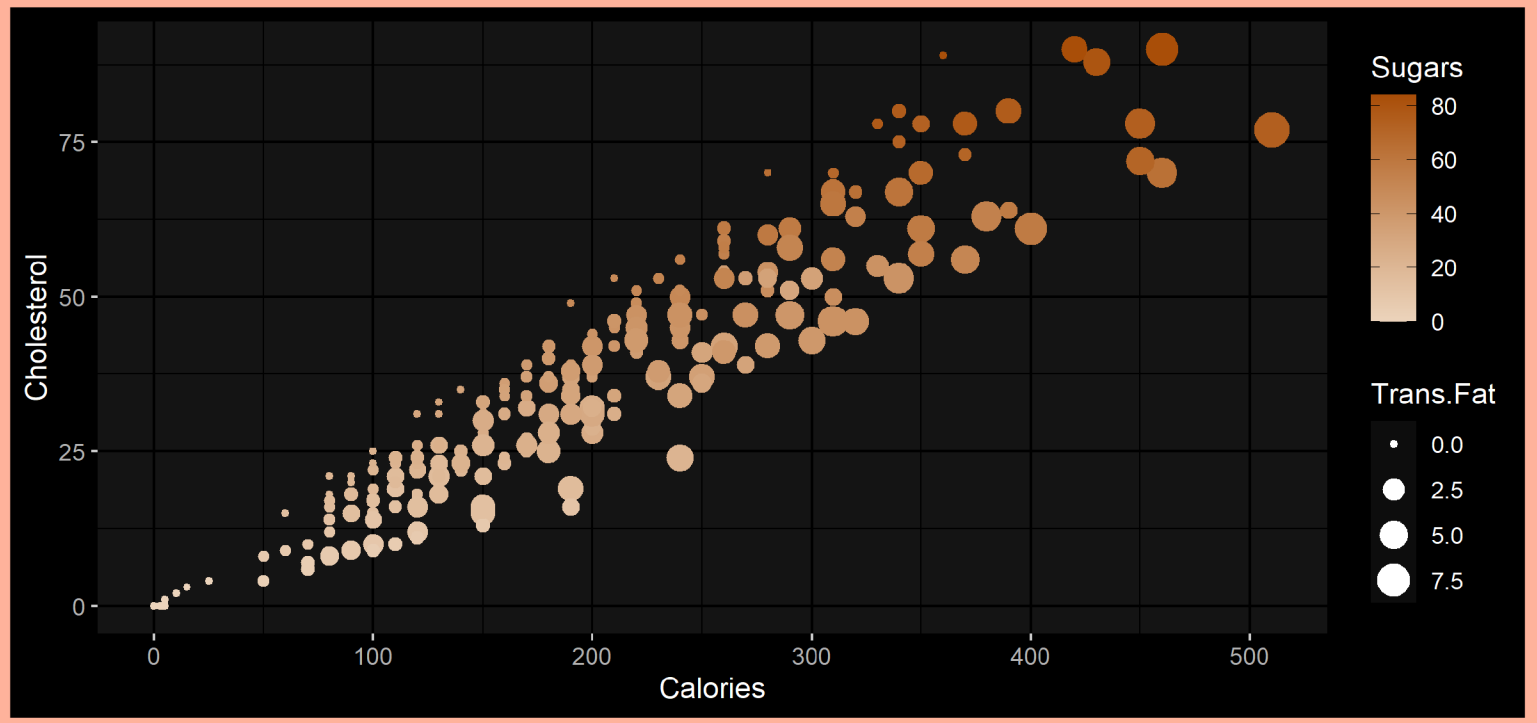
Solution

```
ggplot(starbucks,
       aes(x = Calories, y = Cholesterol)) +
  geom_point()
                                                                    #
                                                                    #
```



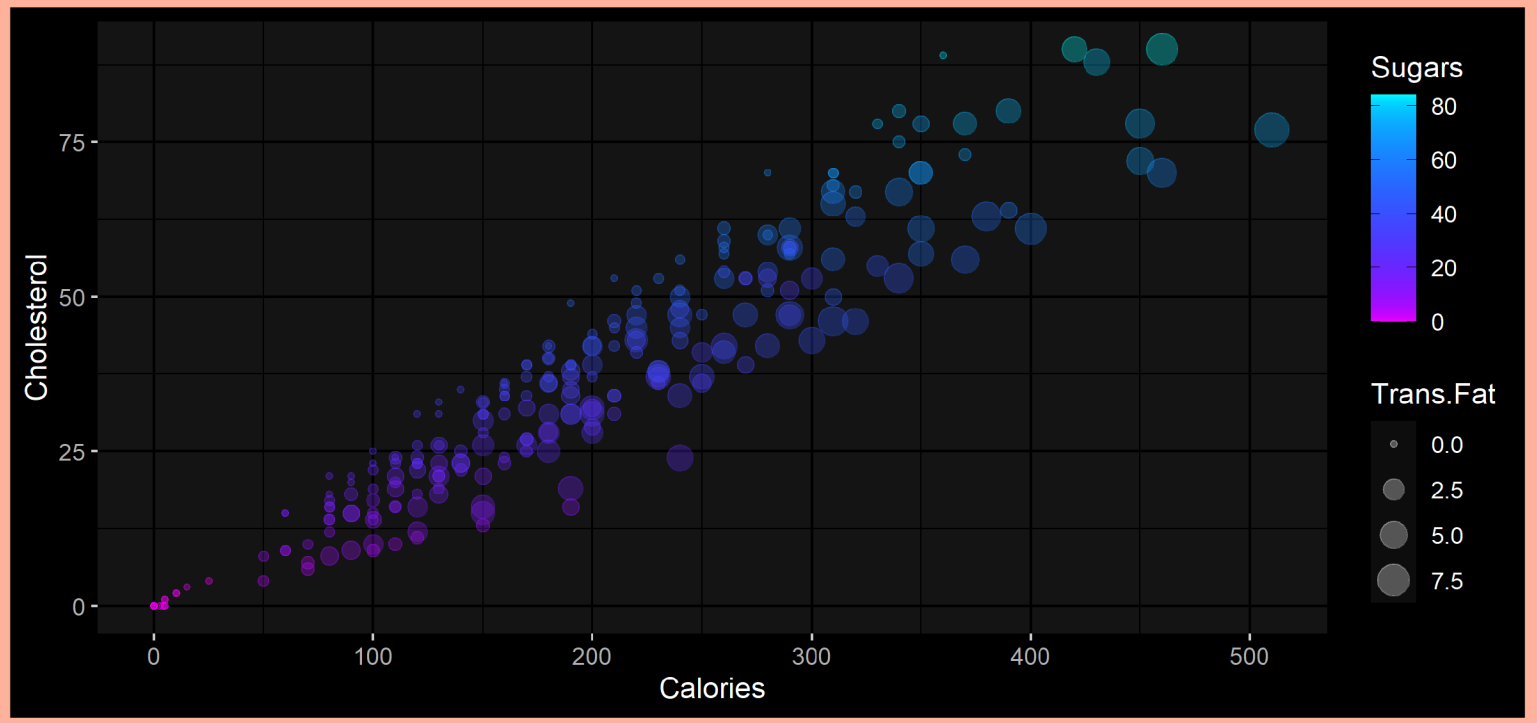
Solution

```
ggplot(starbucks,
       aes(x = Calories, y = Cholesterol, size = Trans.Fat, color = Sugars)) +
  geom_point()
                                                                              #
                                                                              #
```



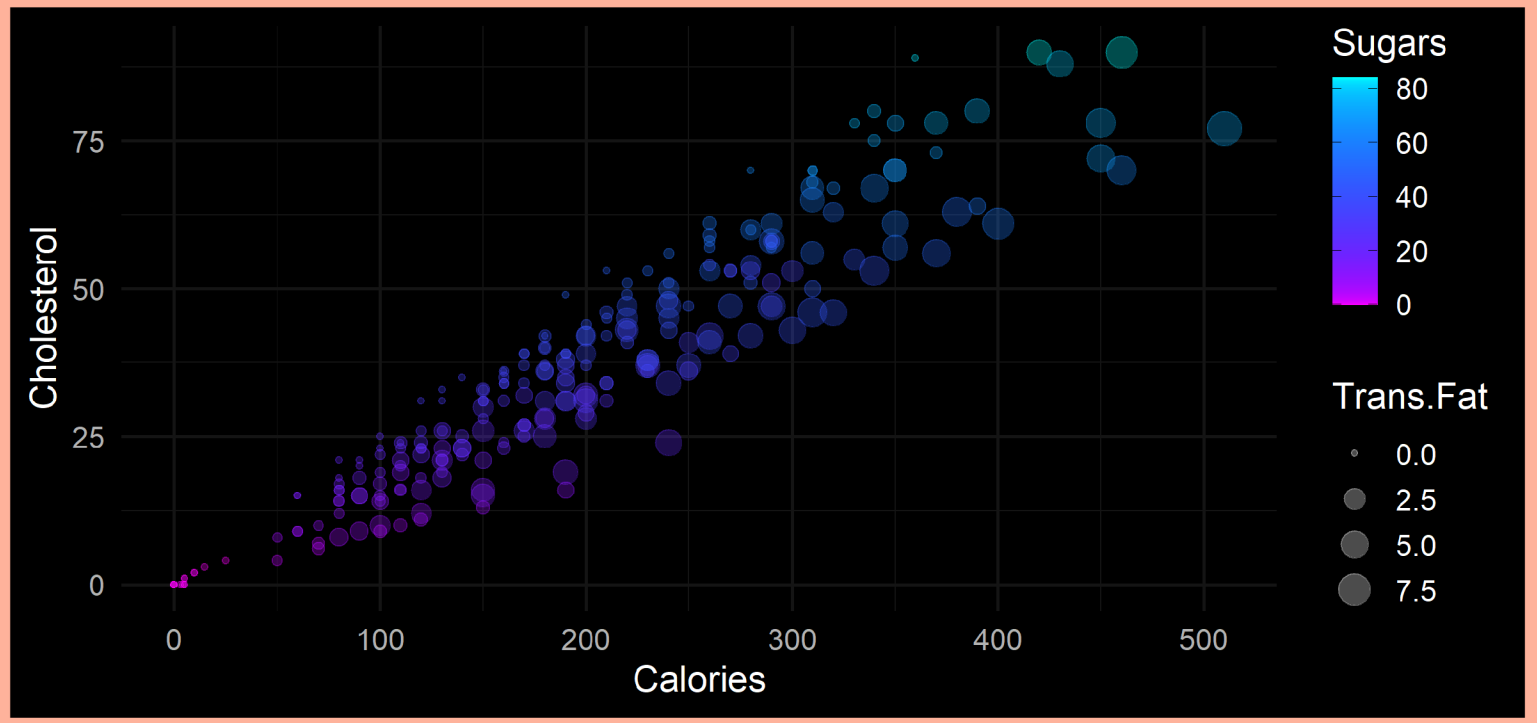
Solution

```
ggplot(starbucks,
       aes(x = Calories, y = Cholesterol, size = Trans.Fat, color = Sugars)) +
  geom_point(alpha = .3) +
  scale_color_gradient(low = "green", high = "red")
                                                                              #
```



Solution

```
ggplot(starbucks,
       aes(x = Calories, y = Cholesterol, size = Trans.Fat, color = Sugars)) +
  geom_point(alpha = .3) +
  scale_color_gradient(low = "green", high = "red") +
  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

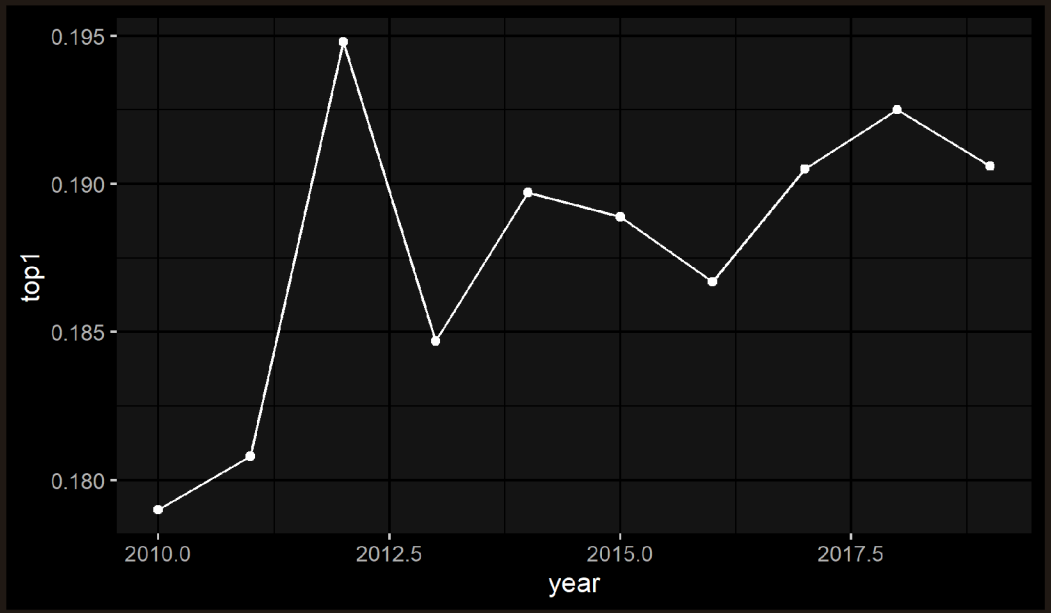
3. Types of geometry
- 3.1. Points and lines
- 3.2. Barplots and histograms
- 3.3. Densities and boxplots

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()
```

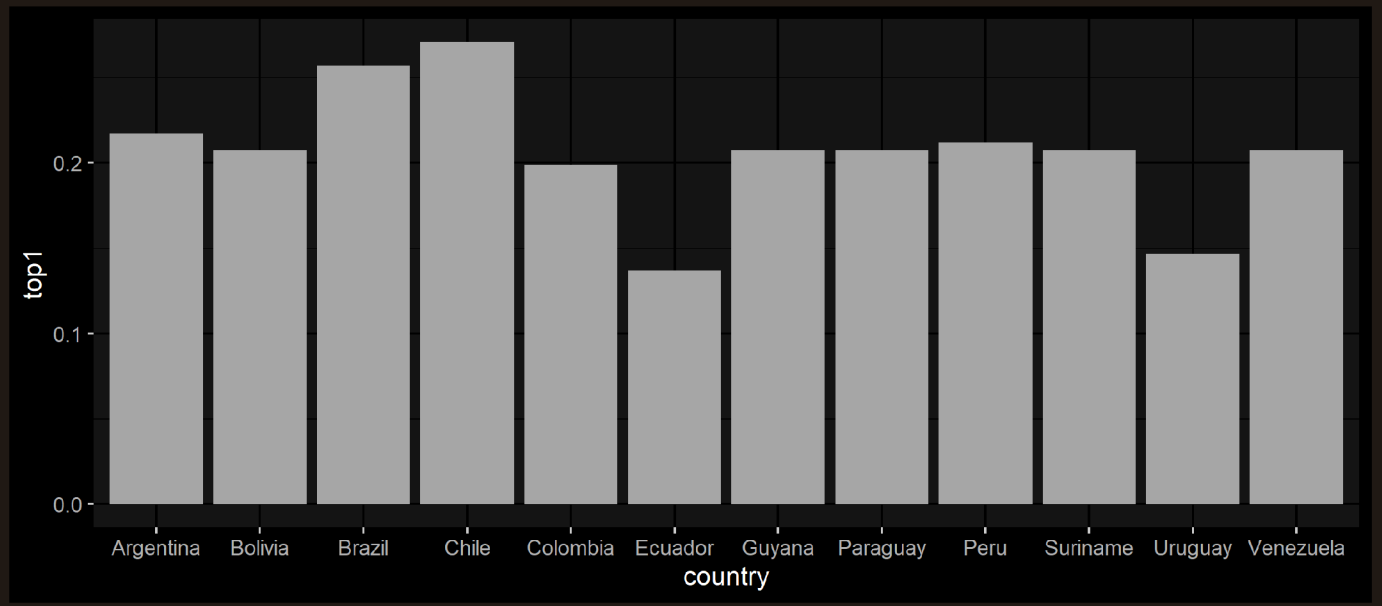


3. Types of geometry

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")
```

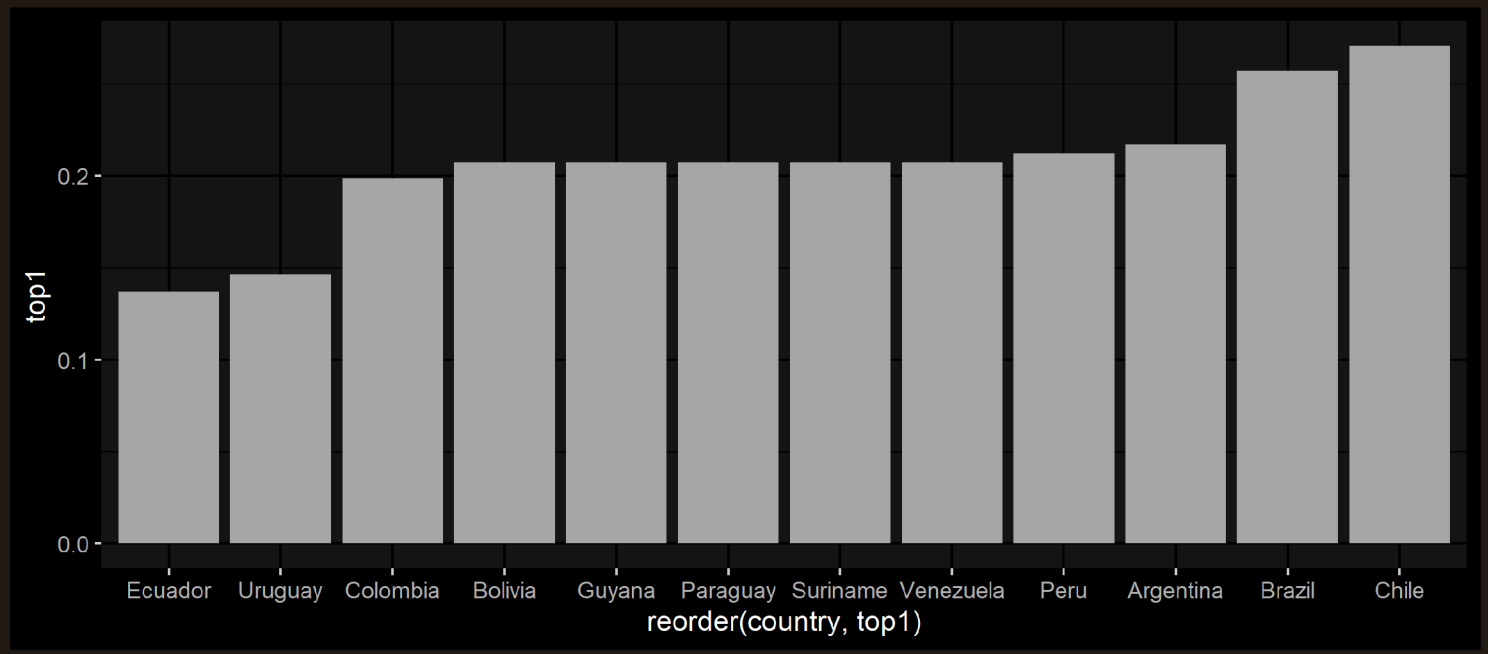


3. Types of geometry

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")
```

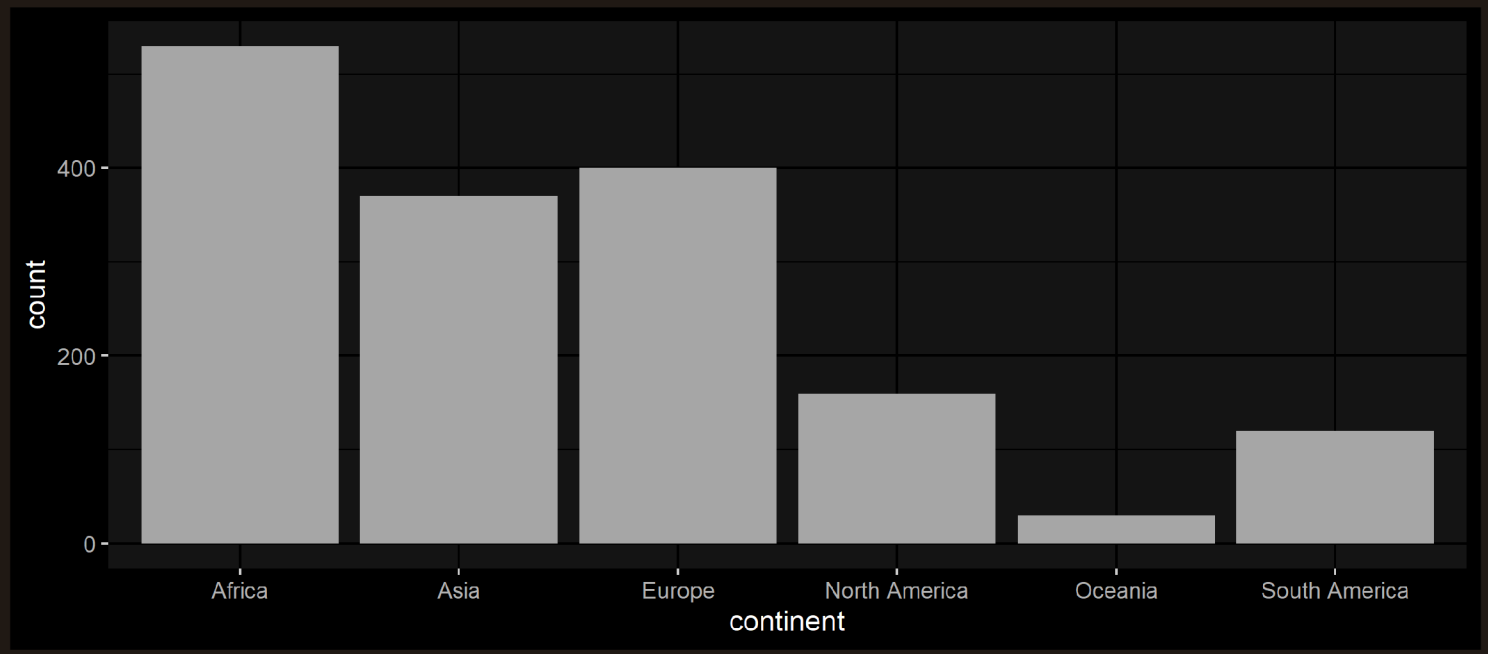


3. Types of geometry

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")
```

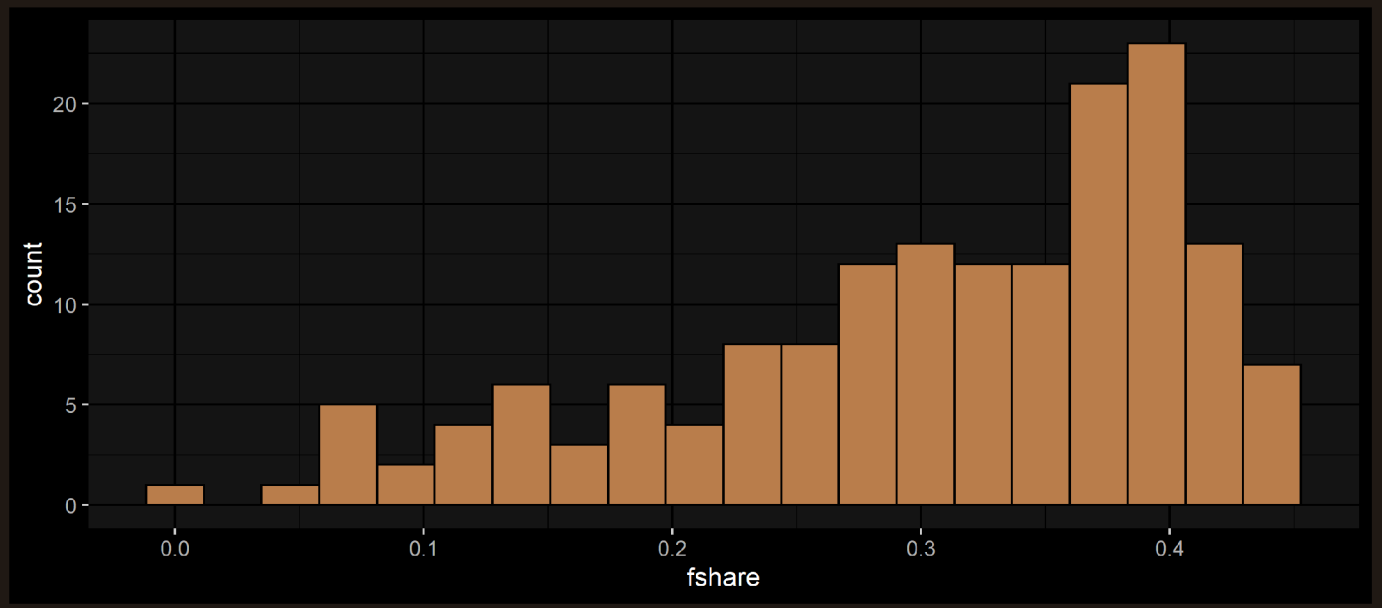


3. Types of geometry

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")
```

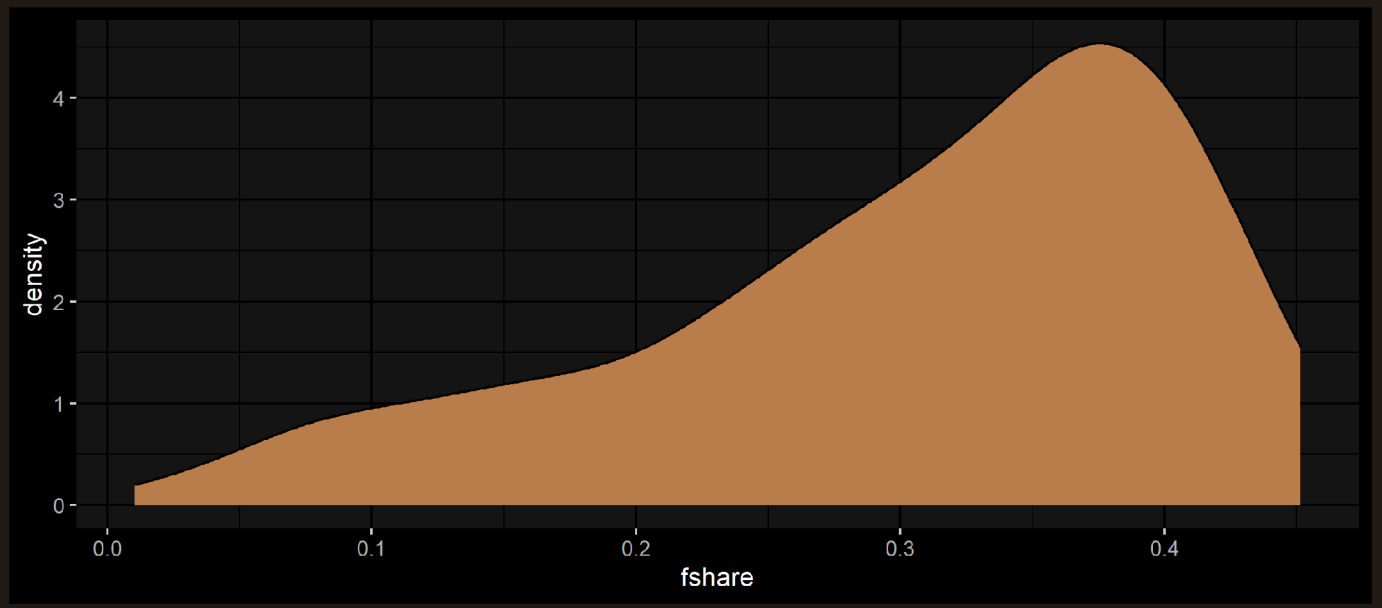


3. Types of geometry

3.3. Densities and boxplots

- 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")
```

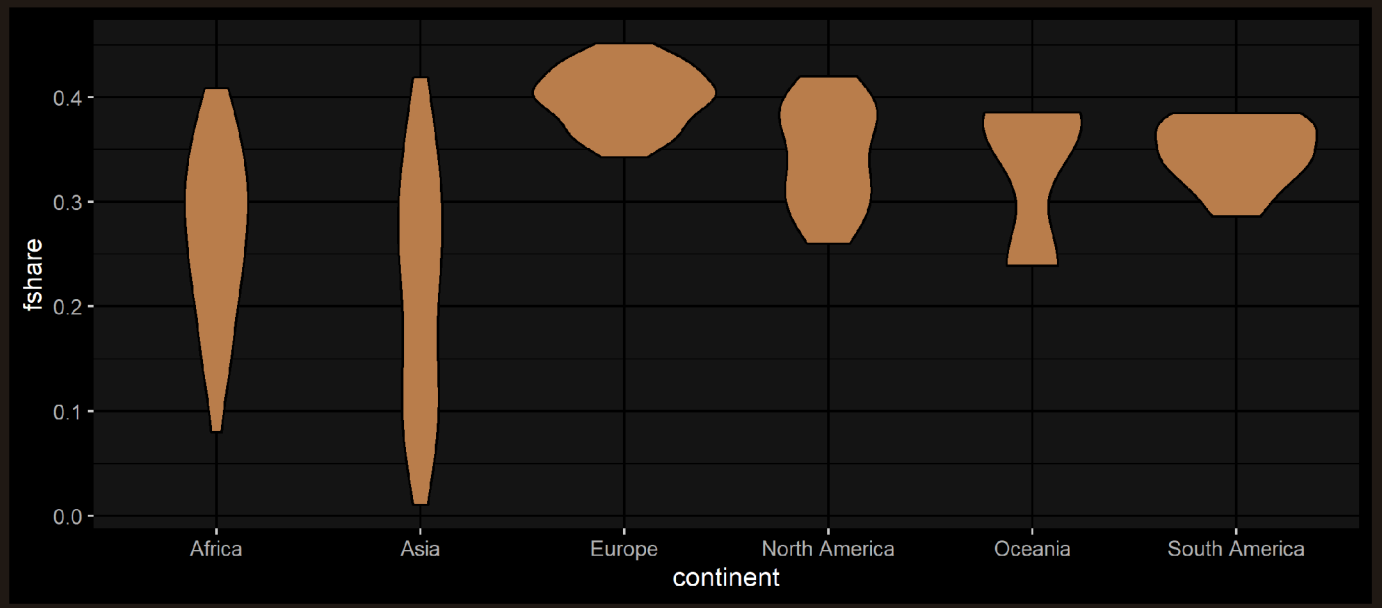


3. Types of geometry

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")
```

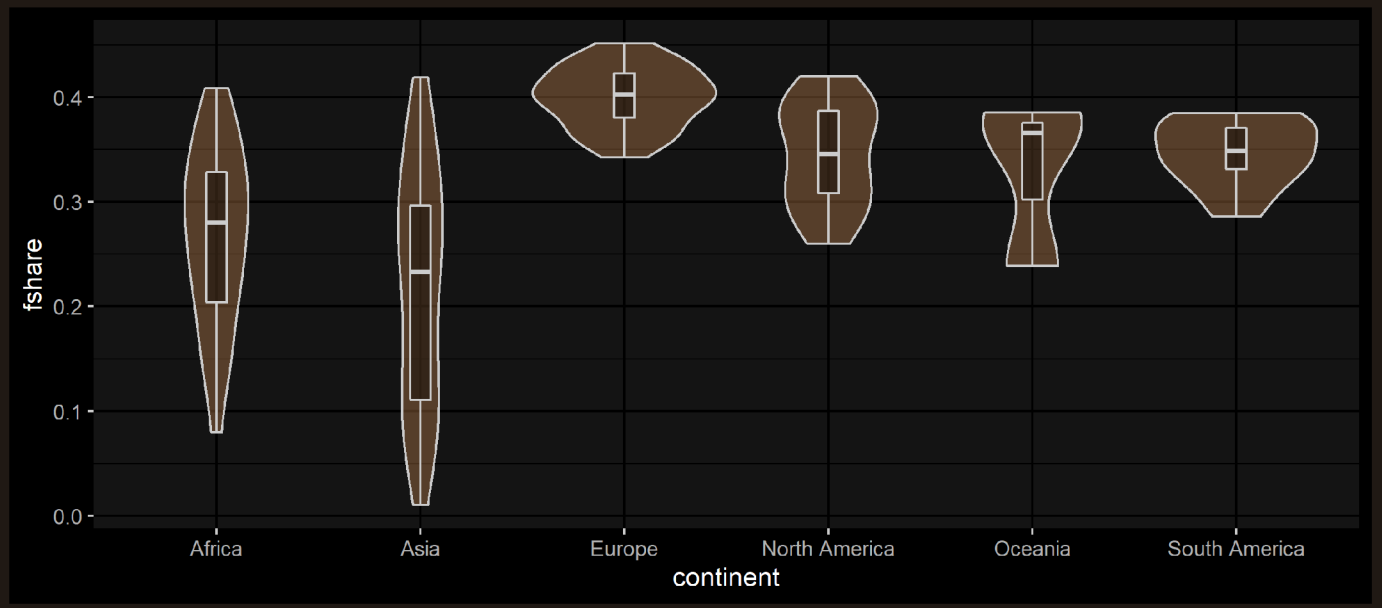


3. Types of geometry

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

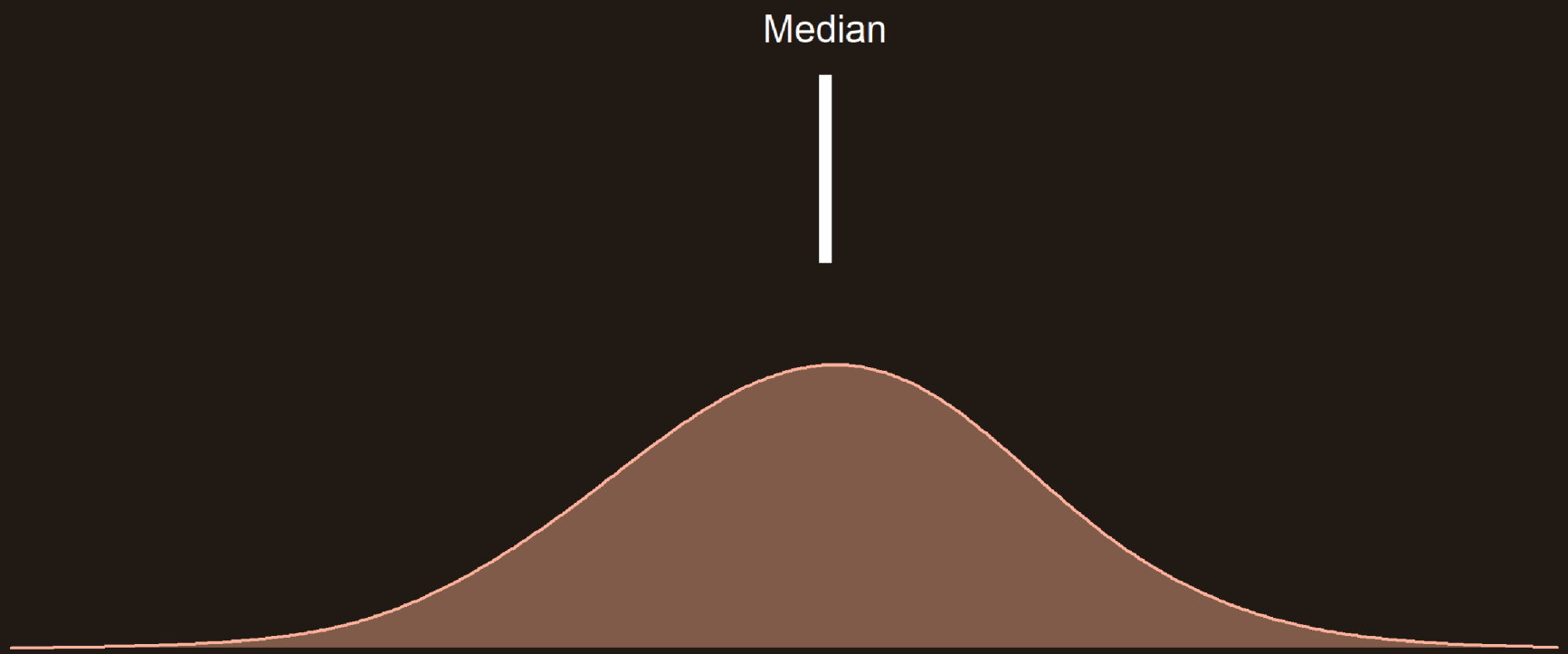
```
ggplot(wid %>% filter(year == 2019), aes(x = continent, y = fshare)) +
    geom_violin(fill = "steelblue", alpha = .4) + geom_boxplot(width = .1, alpha = .4)
```



3. Types of geometry

3.3. Densities and boxplots

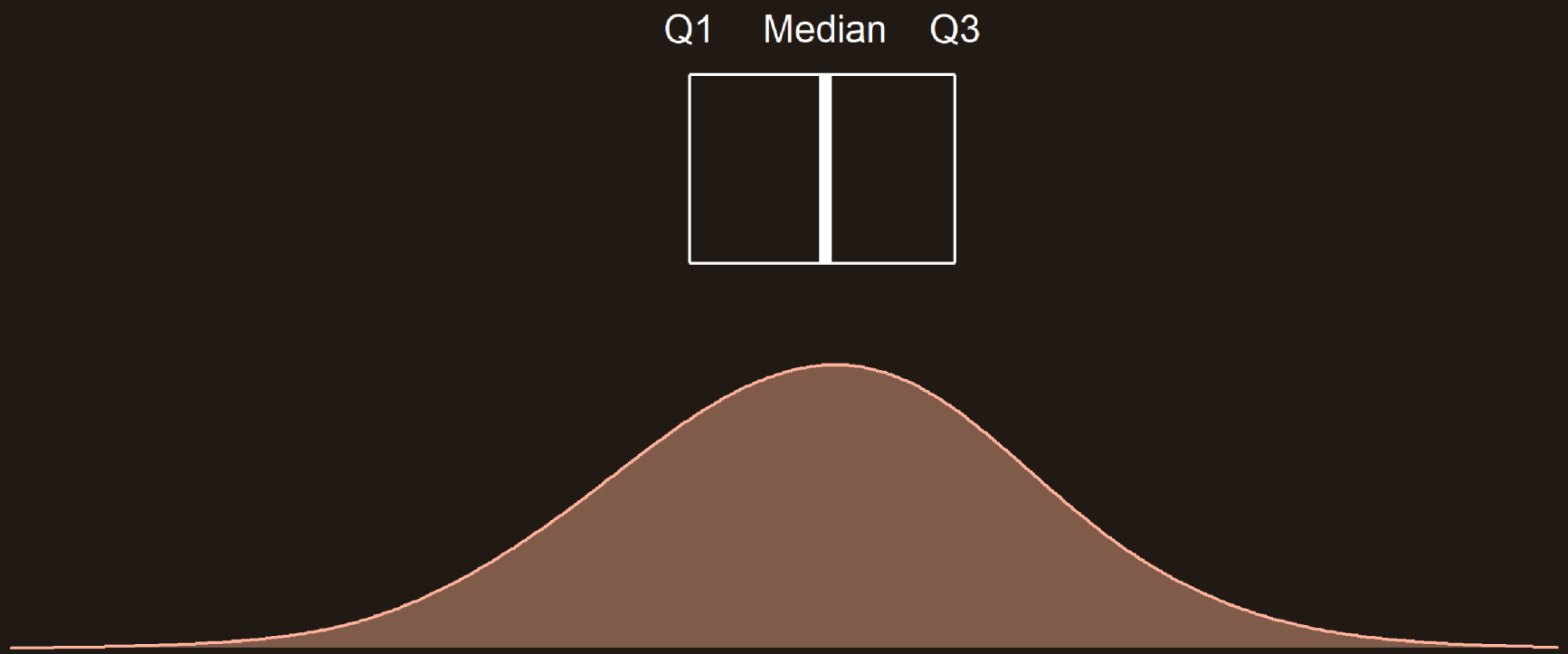
- This is how **boxplots** are constructed:



3. Types of geometry

3.3. Densities and boxplots

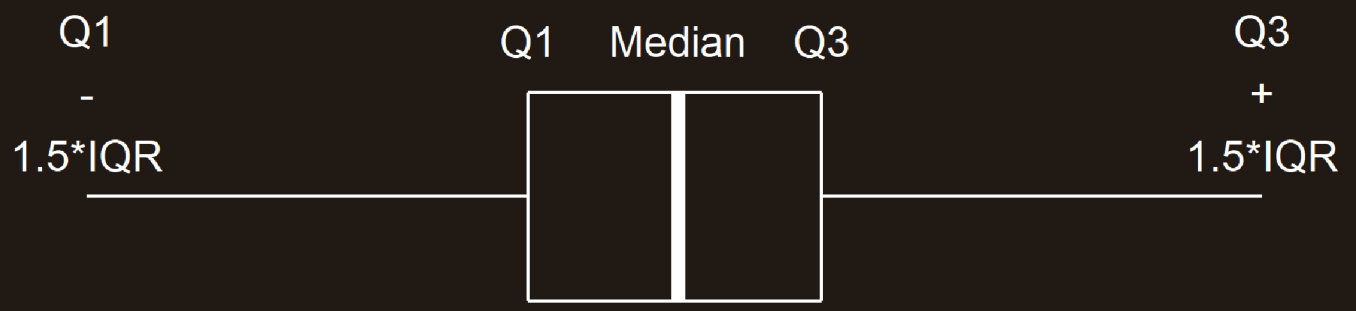
- This is how **boxplots** are constructed:



3. Types of geometry

3.3. Densities and boxplots

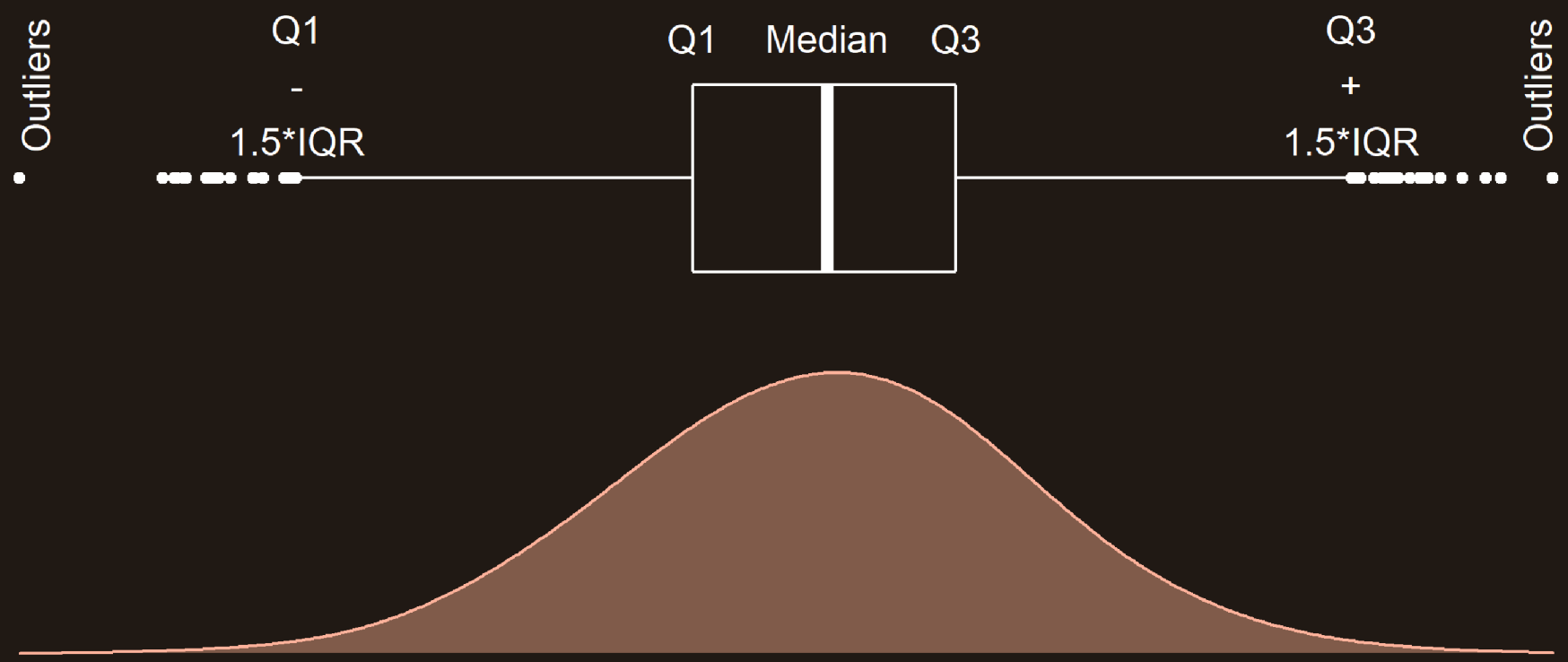
- This is how **boxplots** are constructed:



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

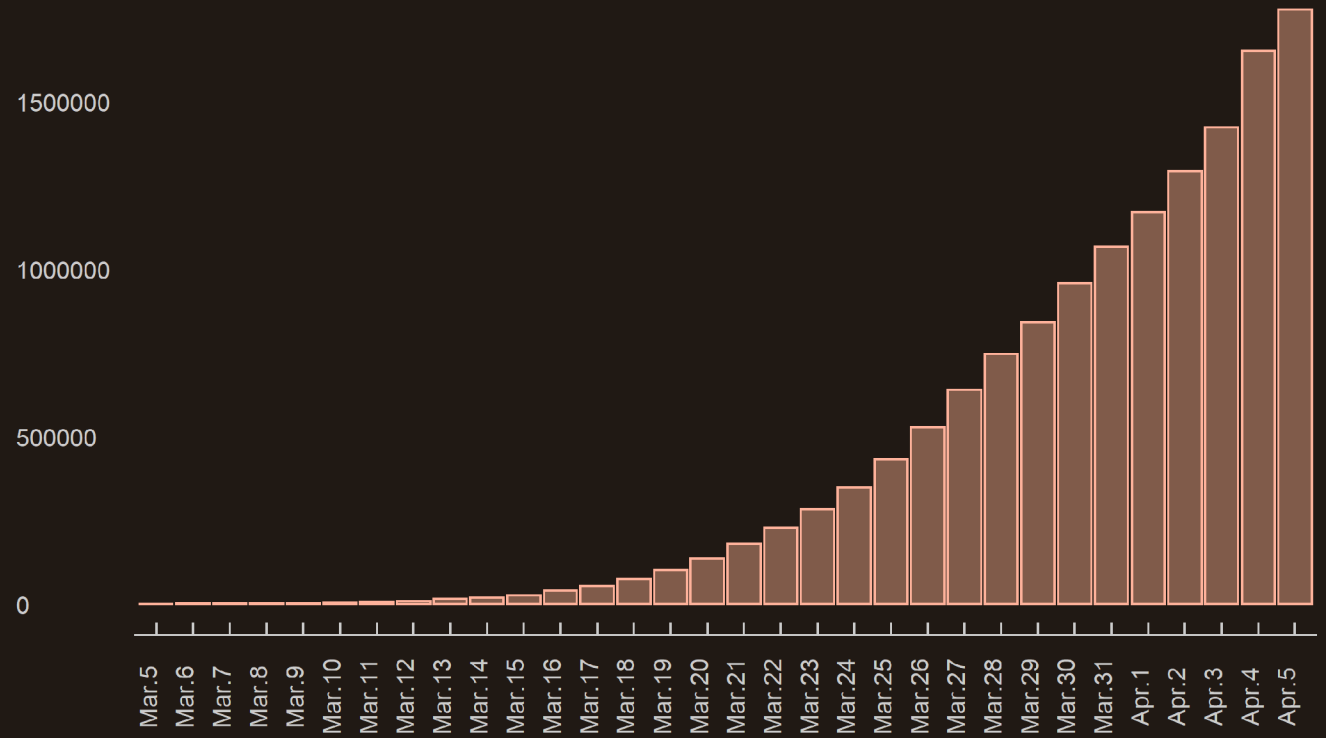
4.1. Cumulative representations

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



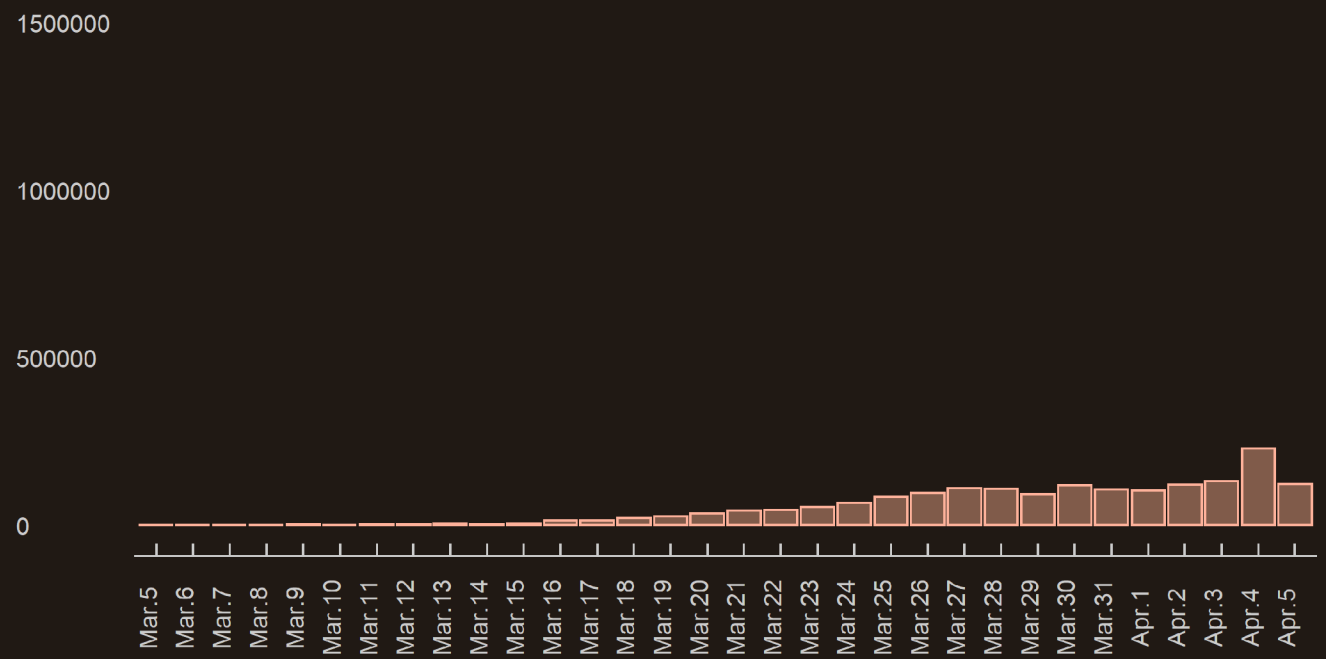
Let's take a closer look

">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

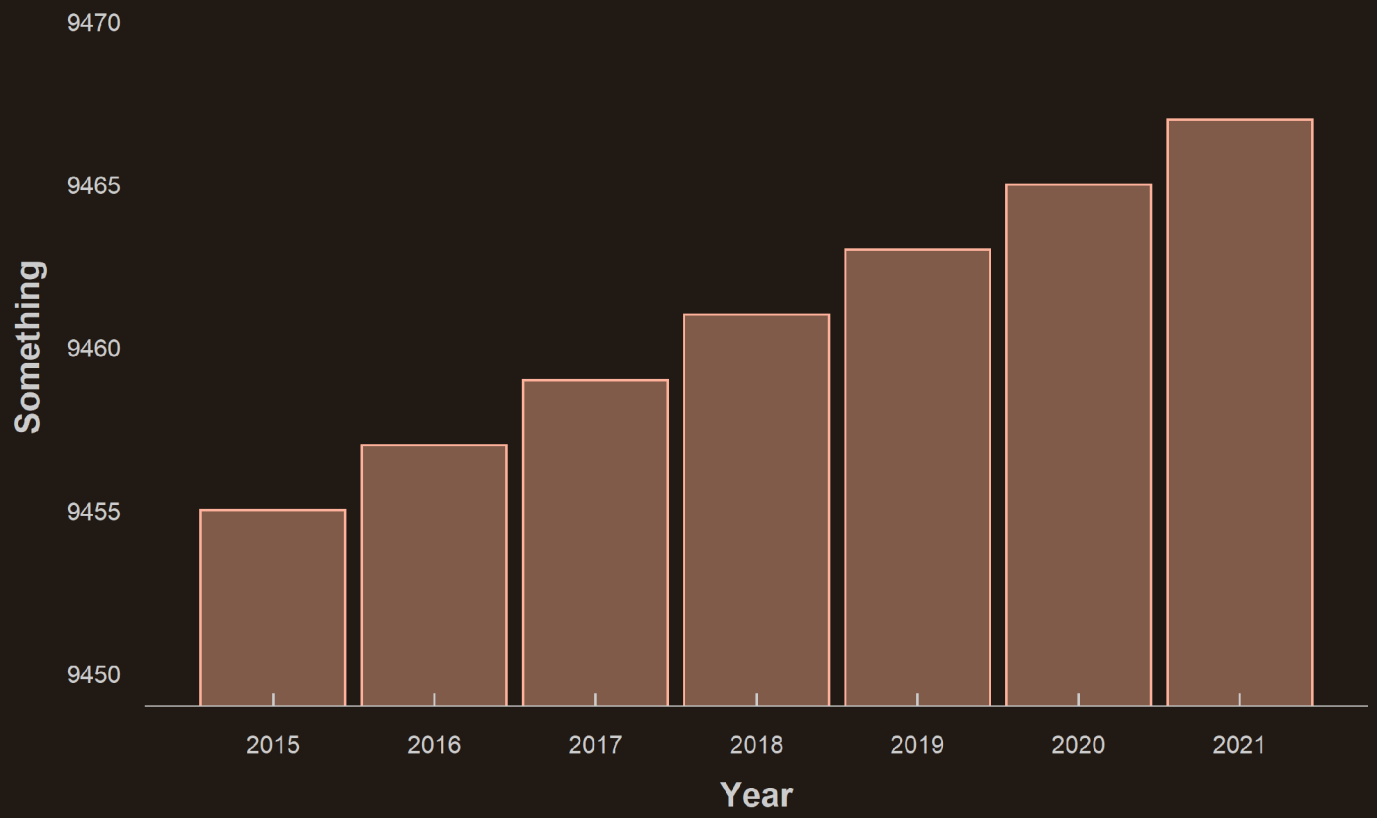


*They plotted the **cumulative** number tests!*

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

4. How (not) to lie with graphics

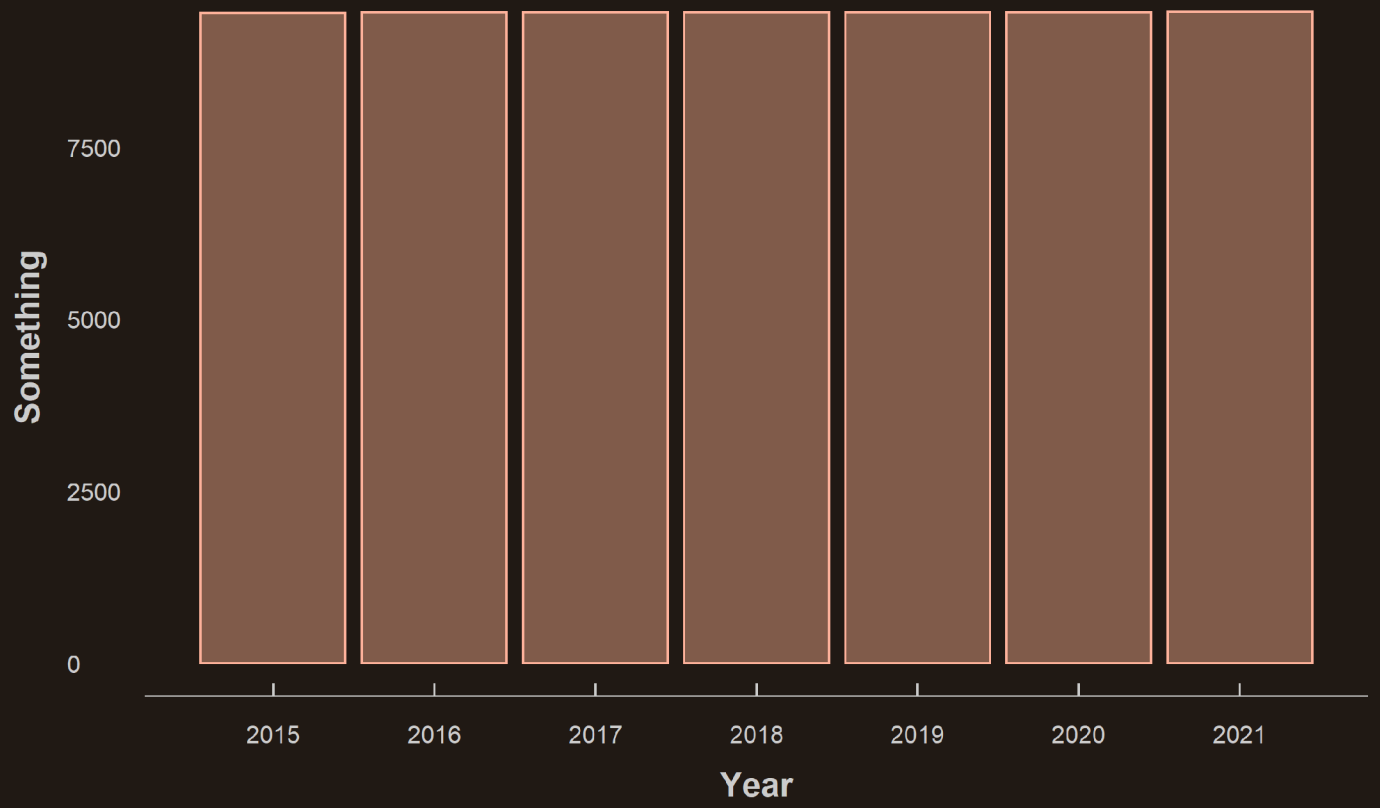
4.2. Axis manipulations



What about this increase?

4. How (not) to lie with graphics

4.2. Axis manipulations

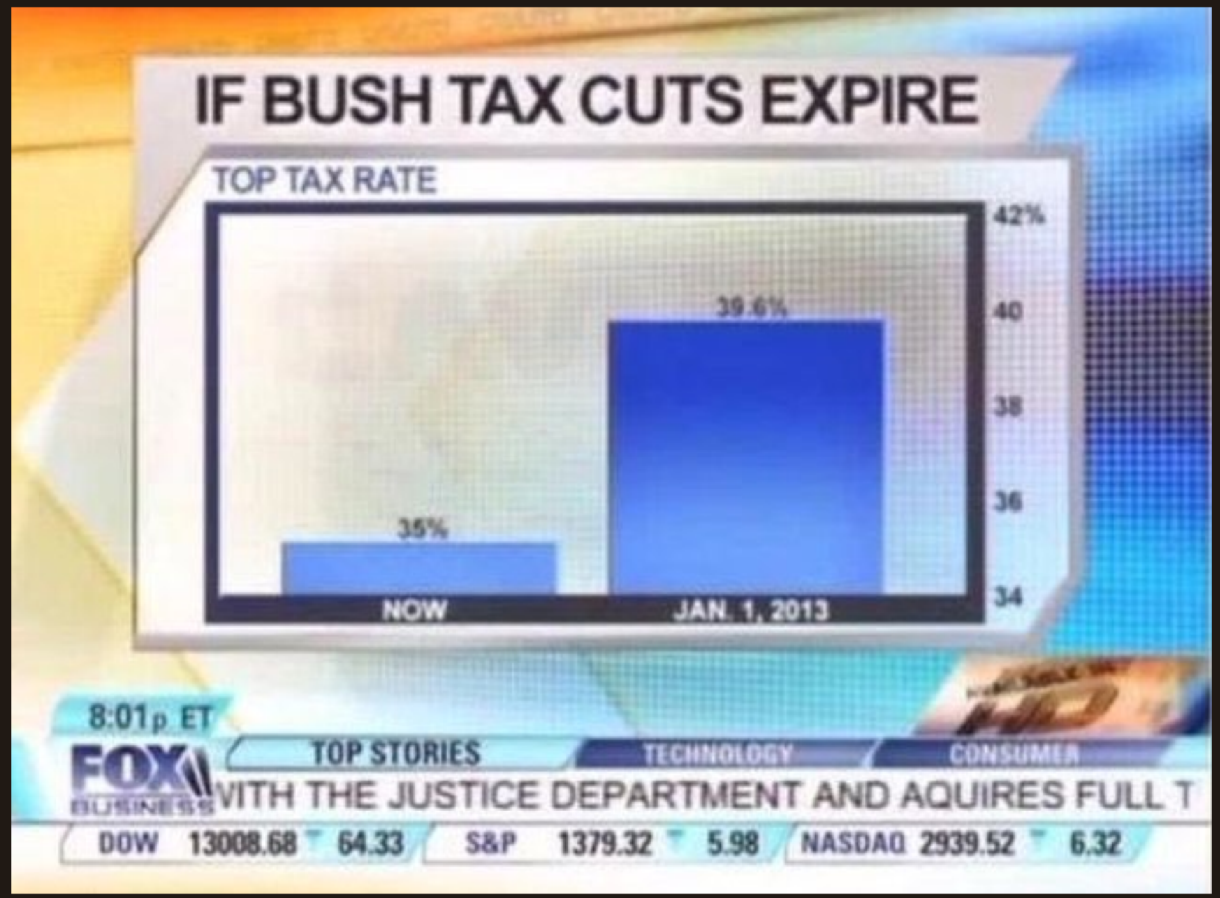


Same data**, but starting **from 0

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

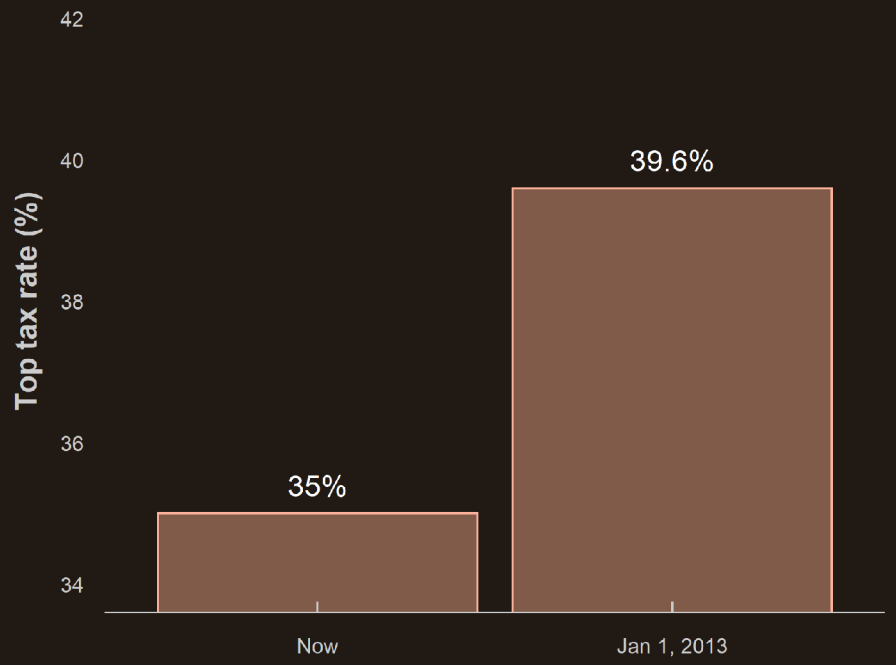
4. How (not) to lie with graphics

4.2. Axis manipulations

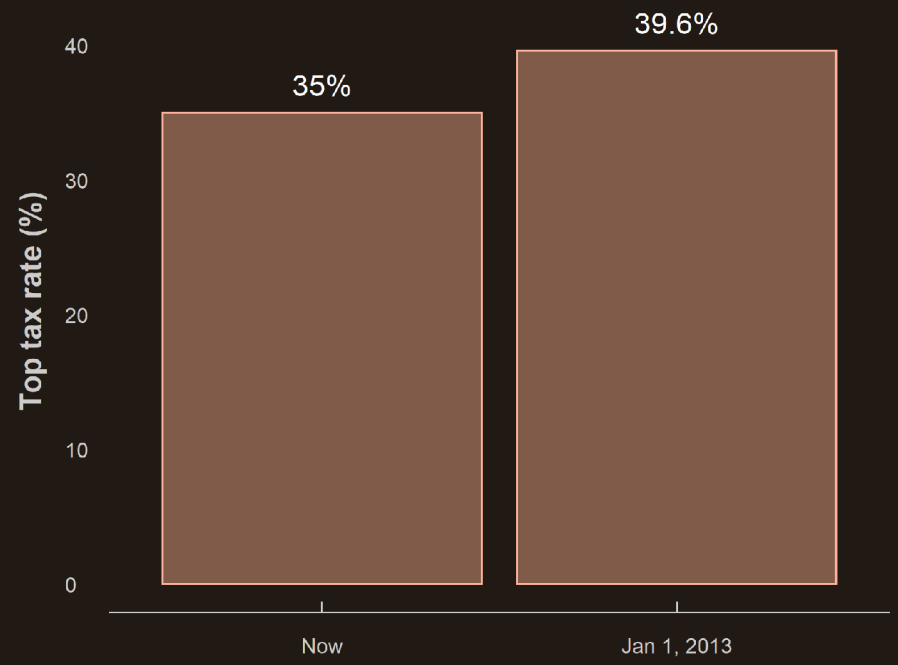


4. How (not) to lie with graphics

4.2. Axis manipulations



Misleading

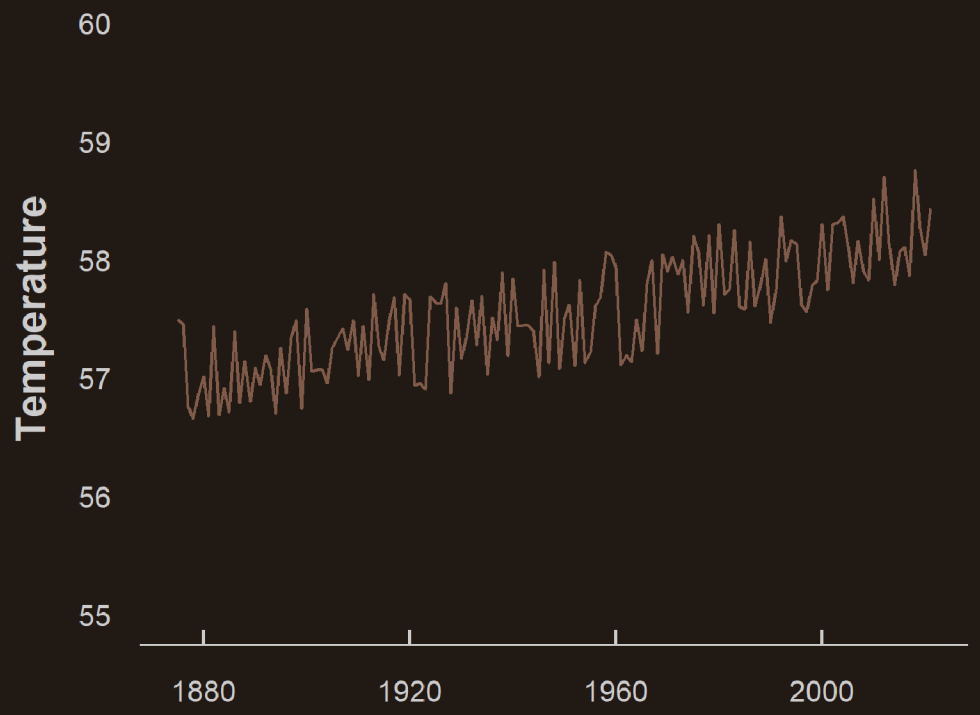


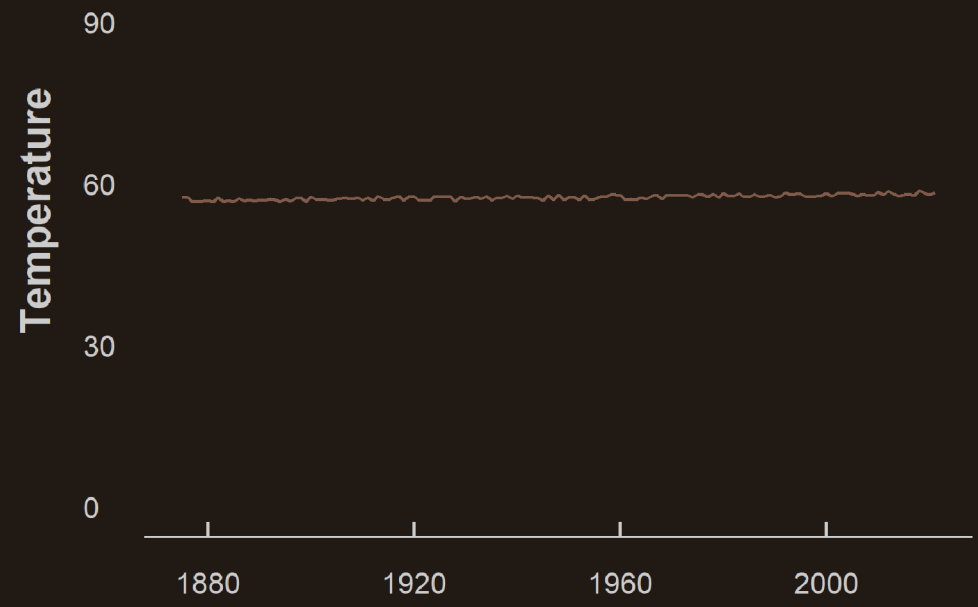
Not misleading

4. How (not) to lie with graphics

4.2. Axis manipulations

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

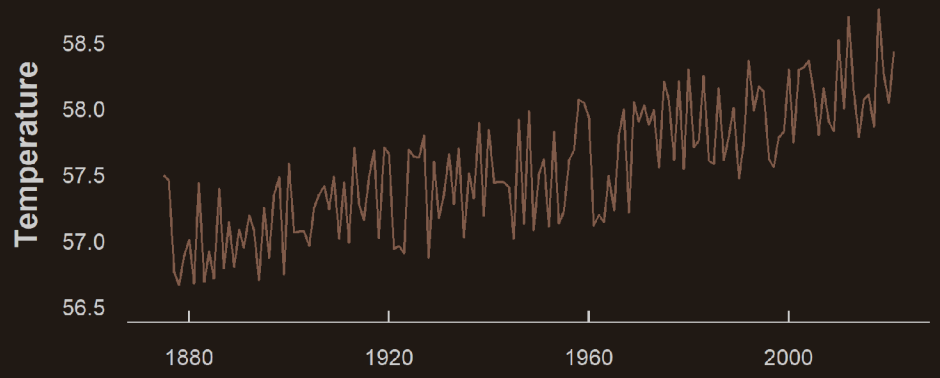


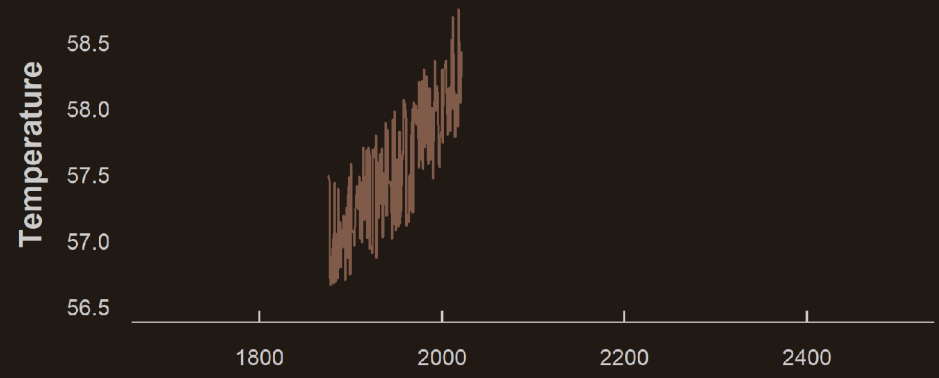


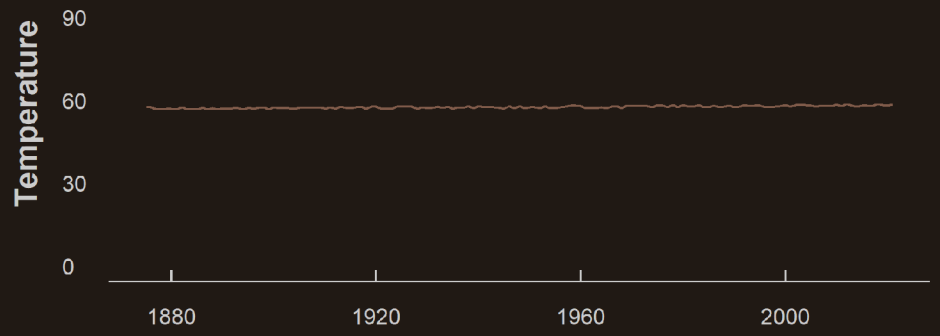
4. How (not) to lie with graphics

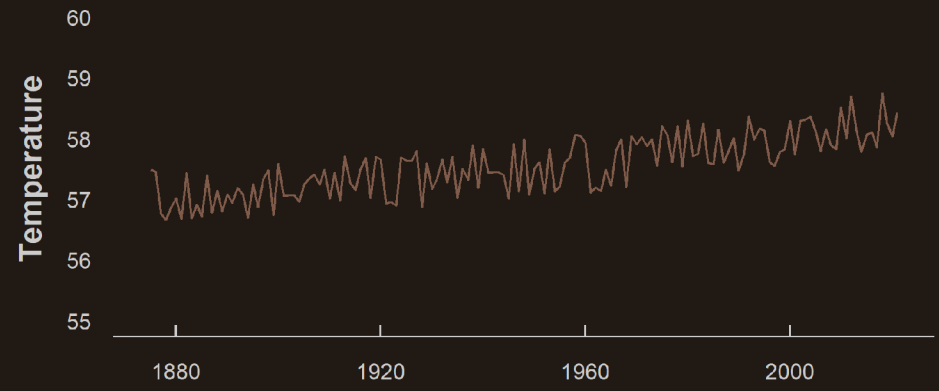
4.2. Axis manipulations

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





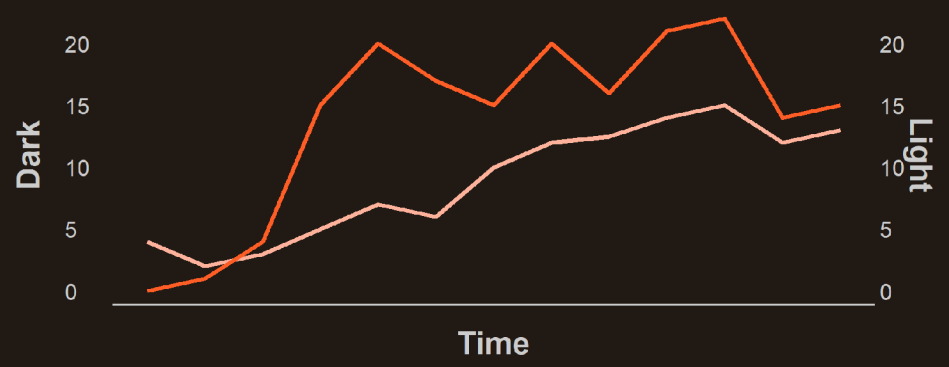


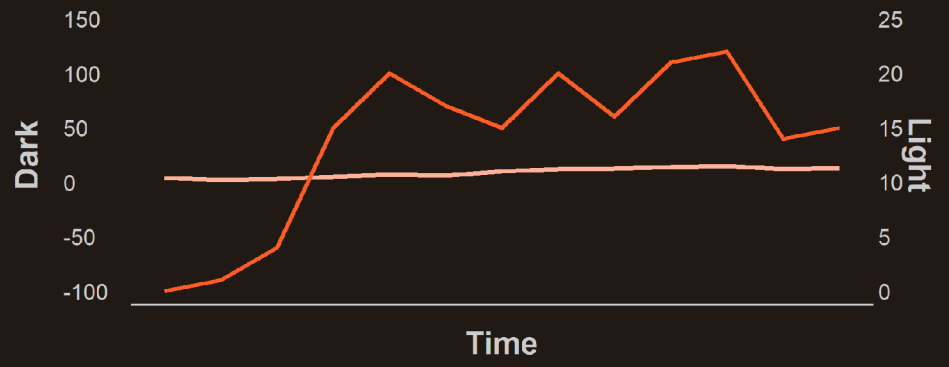


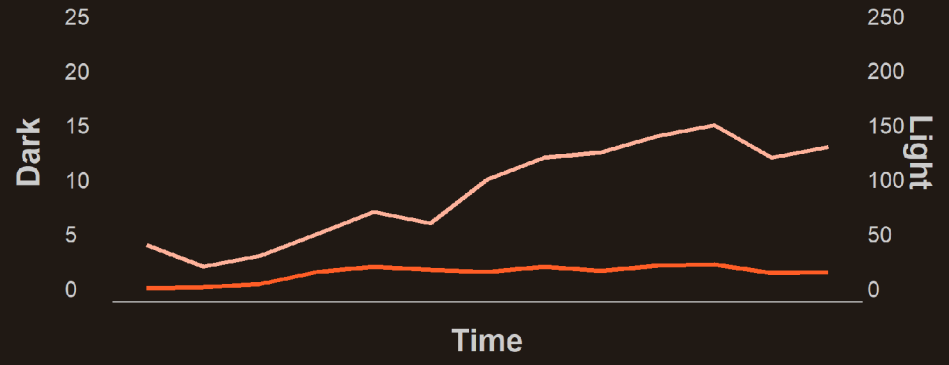
4. How (not) to lie with graphics

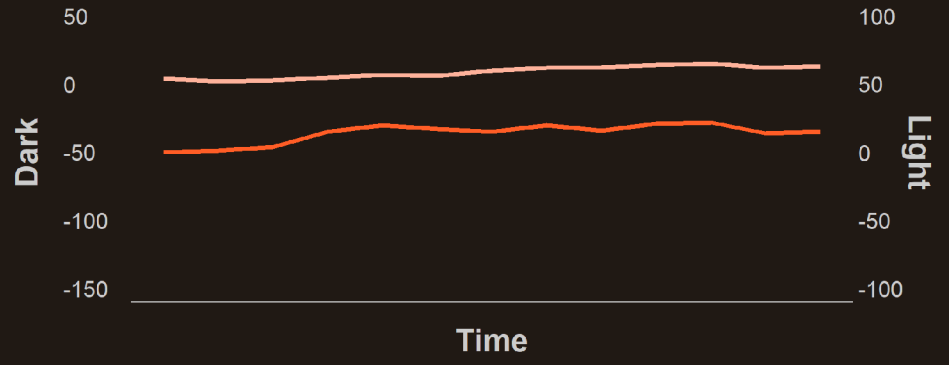
4.2. Axis manipulations

- *⚠ Be very careful with double axes ⚠*
  - You can make them tell basically everything





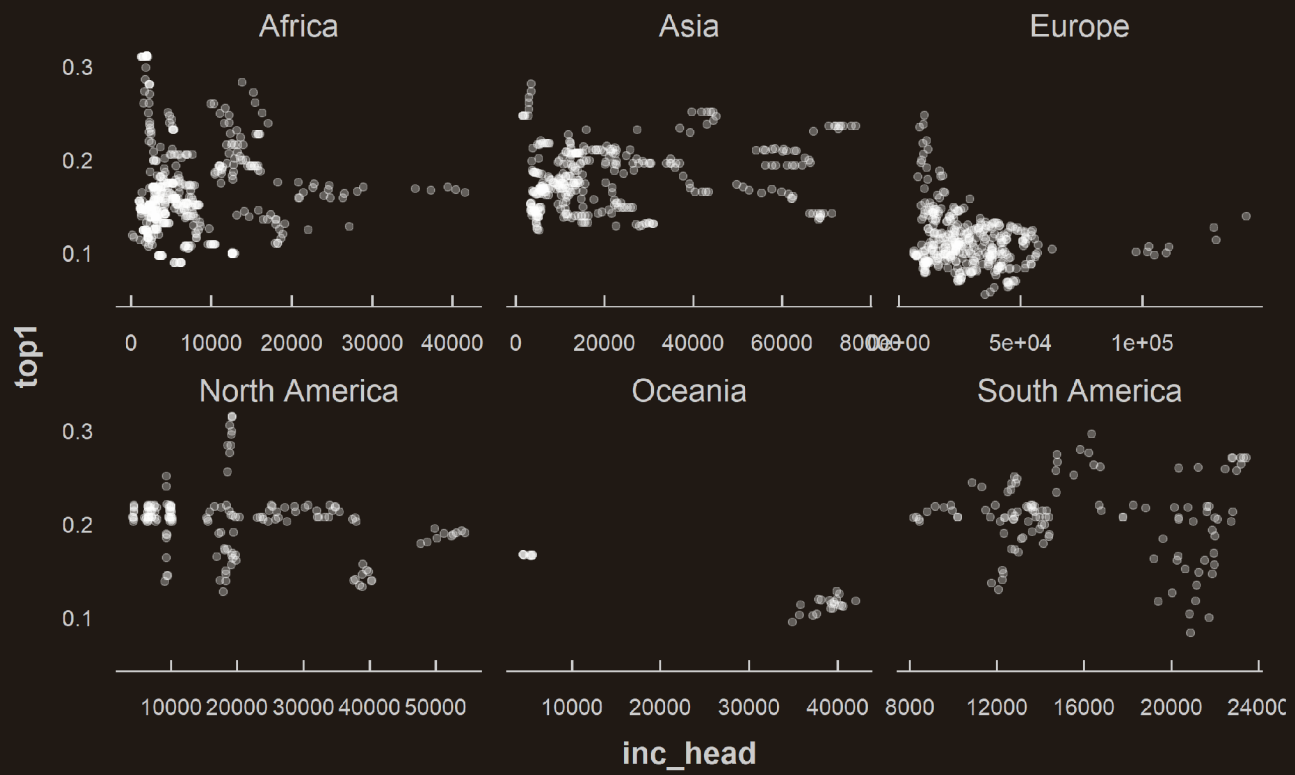




4. How (not) to lie with graphics

4.2. Axis manipulations

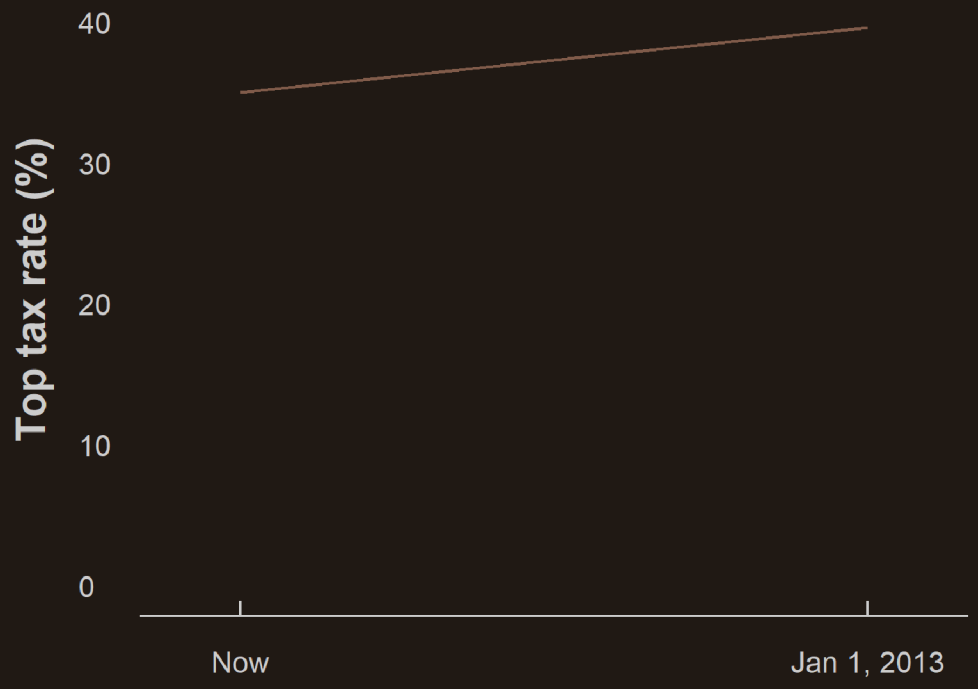
- Be careful with **free scales** in **facet_wrap()** as well
  - It can make things **look more homogeneous** than they actually are



4. How (not) to lie with graphics

4.3. Interpolation

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



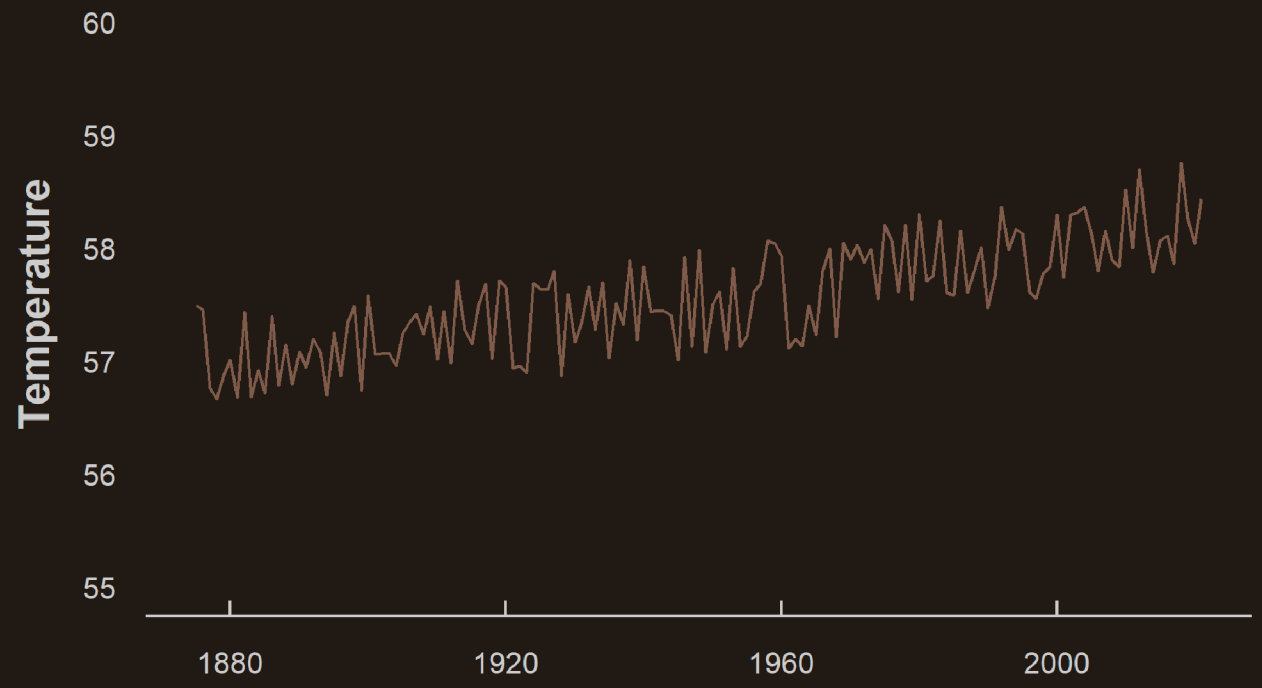
This line has **infinitely many** points

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

4. How (not) to lie with graphics

4.3. Interpolation

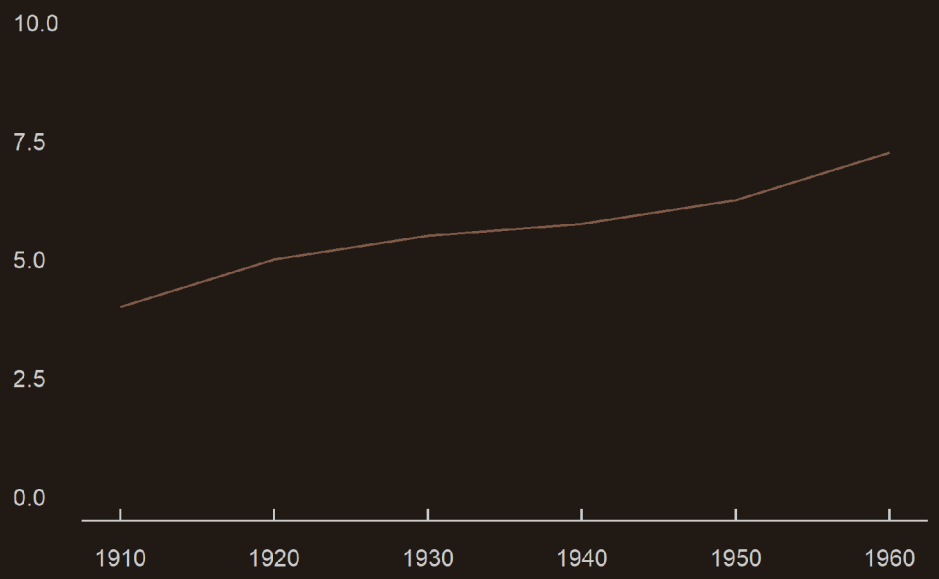
- This figure also has **finitely many actual data points** but feels more natural
  - This is because values are **sufficiently close** to each other to be **considered** as **continuous**

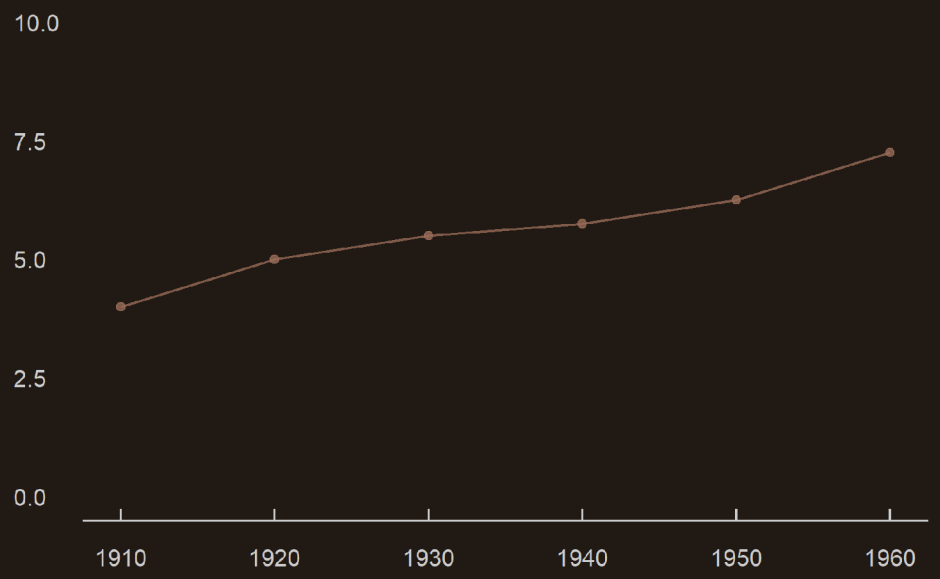


4. How (not) to lie with graphics

4.3. 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 **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)
```