INSTITUTE
OF LAW,
POLITICS AND
DEVELOPMENT

Sant'Anna
School of Advanced Studies – Pisa

Intro to R

Visualization and Programming in R

Prof. Kevin Koehler
kevin.koehler@santannapisa.it

Schedule

We are missing two sessions. Do you prefer adding them to the back or do you want to schedule extra classes in the next weeks?

Today's class

1. Data classes and structures

Sant'Anna
School of Advanced Studies – Pisa

Today's class

1. Data classes and structures
2. The grammar of graphics in `ggplot2`

Today's class

1. Data classes and structures
2. The grammar of graphics in `ggplot2`
3. Writing user-defined functions and automating repetitive tasks with `for` loops

Today's class

1. Data classes and structures
2. The grammar of graphics in `ggplot2`
3. Writing user-defined functions and automating repetitive tasks with `for` loops
4. Exercises

Why visualization?

Data visualization

1. Visualization allows you to better understand data and to efficiently communicate results

Data visualization

1. Visualization allows you to better understand data and to efficiently communicate results
2. Many important journals in Political Science require plots instead of tables for publication

Data visualization

1. Visualization allows you to better understand data and to efficiently communicate results
2. Many important journals in Political Science require plots instead of tables for publication
3. Visual representations are easier to read than full tables

The grammar of graphics

ggplot2

The `ggplot2` package is part of the `tidyverse`. It provides a specific system of constructing plots.

First, some theory

Data classes and structures

There are three (four?) data classes in R

1. Numeric
2. Character
3. Factor
4. (Logical)

You can convert objects between classes with `as.numeric()`, `as.character()`, and `as.factor()`.

Why `factors` are weird

`factors` have **levels** and **labels**. Internally, R represents factors as ordered sequences. This can lead to odd behavior such as:

```
factor <- as.factor(c(1,2,4,2,5,99))
factor
```

```
[1] 1  2  4  2  5  99
Levels: 1 2 4 5 99
```

```
as.numeric(factor)
```

```
[1] 1 2 3 2 4 5
```

What R does here is to order the elements of the factor (in this case by size since the elements are numerical–if we had entered characters they would be ordered alphabetically). It then returns the position in this order as the numerical value.

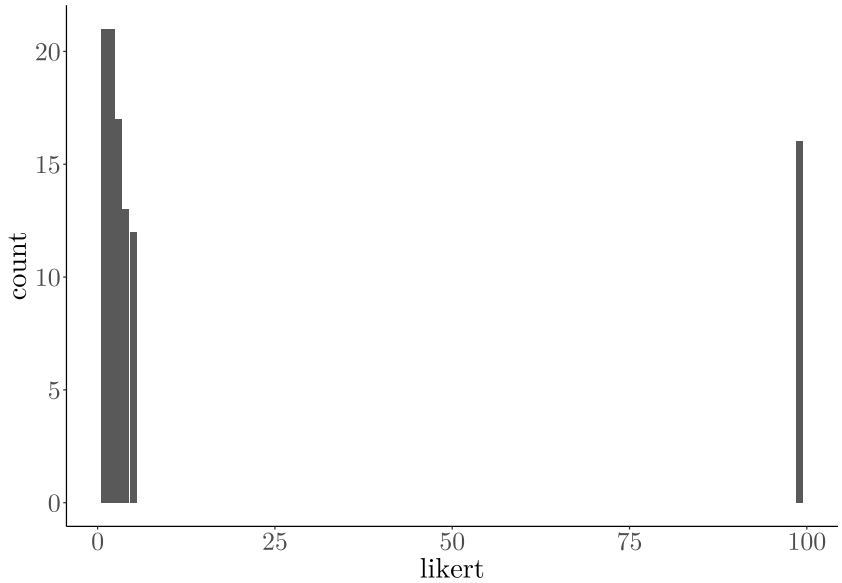
Why `factors` are still usefull

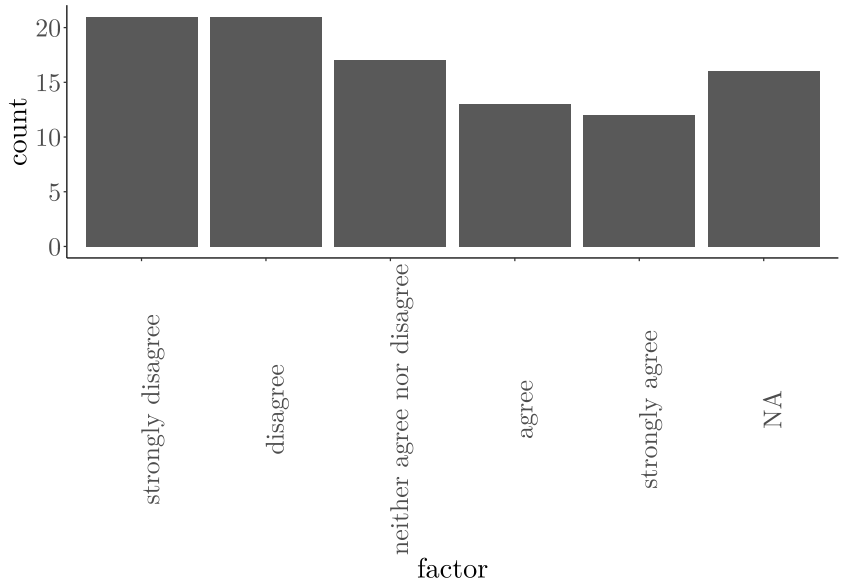
Despite this odd behavior, factors can be useful for plotting.

Say, we have a Likert scale response such as in the Tunisia survey. Plotting this variable as a vector allows us to control the labels:

```
likert <- sample(c(1:5, 99), size = 100, replace = TRUE)
factor <- factor(likert,
                 levels=c(1,2,3,4,5,99),
                 labels=c("strongly disagree",
                          "disagree",
                          "neither agree nor disagree",
                          "agree",
                          "strongly agree",
                          "NA"))
example <- data.frame(likert=likert,
                      factor=factor)
```

Plot both `likert` and `factor` to see the difference





This behavior is similar to what you might know from Stata (where you can assign value labels)

Data structures

1. **Vectors** (numerical or character, only one class): `id <- c(1,2,3,4,5)`, `gender <- c("m","f","m","f","m")`

Data structures

1. **Vectors** (numerical or character, only one class): `id <- c(1,2,3,4,5)`, `gender <- c("m","f","m","f","m")`
2. **Matrices** (two dimensional, only one class): `matrix1 <- matrix(c(1,2,3,4,5,23,34,19,46,38), ncol = 2)`

Data structures

1. **Vectors** (numerical or character, only one class): `id <- c(1,2,3,4,5)`, `gender <- c("m","f","m","f","m")`
2. **Matrices** (two dimensional, only one class): `matrix1 <- matrix(c(1,2,3,4,5,23,34,19,46,38), ncol = 2)`
3. **Lists** (collection of elements of different classes): `list1 <- list(id, gender, matrix1)`

Data structures

1. **Vectors** (numerical or character, only one class): `id <- c(1,2,3,4,5)`, `gender <- c("m","f","m","f","m")`
2. **Matrices** (two dimensional, only one class): `matrix1 <- matrix(c(1,2,3,4,5,23,34,19,46,38), ncol = 2)`
3. **Lists** (collection of elements of different classes): `list1 <- list(id, gender, matrix1)`
4. **Data frames** (two dimensional structure, different classes): `df1 <- data.frame(id = c(1,2,3,4,5), gender = c("m","f","m","f","m"))`

Data structures

R saves the output of many more complex functions as lists. The following code creates random `x` and `y` variables and regresses y on x:

```
y <- runif(100, min = 0, max = 1)
x <- runif(100, min = 10, max = 20)
model1 <- lm(y ~ x)
names(model1)
```

```
 [1] "coefficients"  "residuals"     "effects"       "rank"
 [5] "fitted.values" "assign"        "qr"            "df.residua
 [9] "xlevels"       "call"          "terms"         "model"
```

Data structures

```
summary(model1)
```

```
Call:
lm(formula = y ~ x)

Residuals:
     Min       1Q   Median       3Q      Max
-0.58155 -0.19851  0.06131  0.23177  0.47878

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  0.738717   0.150656   4.903 3.74e-06 ***
x           -0.013389   0.009752  -1.373    0.173
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.2809 on 98 degrees of freedom
Multiple R-squared:  0.01887,   Adjusted R-squared:  0.008857
F-statistic: 1
```

Visualization with `ggplot`

Sant'Anna
School of Advanced Studies – Pisa

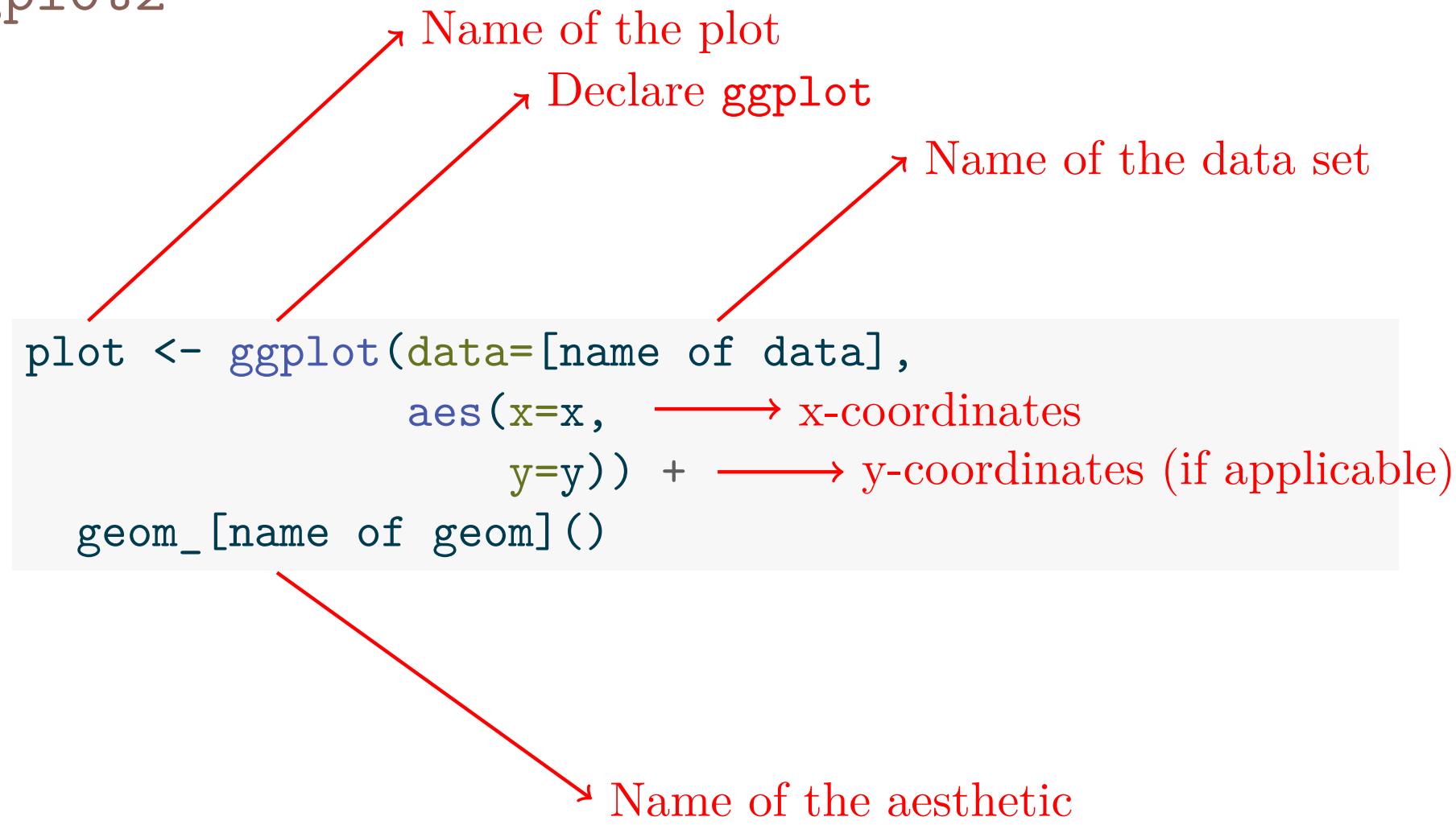
ggplot2

In `ggplot2`, plots consist of three basic components:

1. data
2. aesthetics
3. coordinate system

You can combine as many different aesthetic layers as you like.

ggplot2


Name of the plot
Declare ggplot
Name of the data set
plot <- ggplot(data=[name of data],
aes(x=x,
x-coordinates
y=y)) +
y-coordinates (if applicable)
geom_[name of geom]()
Name of the aesthetic

ggplot2

Data visualization with ggplot2 : : CHEATSHEET

Basics

ggplot2 is based on the **grammar of graphics**, the idea that you can build every graph from the same components: a **data** set, a **coordinate system**, and **geoms**—visual marks that represent data points.

To display values, map variables in the data to visual properties of the geom (**aesthetics**) like **size**, **color**, and **x** and **y** locations.

Complete the template below to build a graph.

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

required

Not required, sensible defaults supplied

ggplot(data = mpg, **aes(**x = cty, y = hwy**))** Begins a plot that you finish by adding layers to. Add one geom function per layer.

last_plot() Returns the last plot.

ggsave("plot.png", width = 5, height = 5**)** Saves last plot as 5' x 5' file named "plot.png" in working directory. Matches file type to file extension.

Aes

Common aesthetic values.

color and **fill** - string ("red", "#RRGGBB")

linetype - integer or string (0 = "blank", 1 = "solid", 2 = "dashed", 3 = "dotted", 4 = "dotdash", 5 = "longdash", 6 = "twodash")

size - integer (in mm for size of points and text)

linewidth - integer (in mm for widths of lines)

shape - integer/shape name or a single character ("a")

posit

Geoms

Use a geom function to represent data points, use the geom's aesthetic properties to represent variables. Each function returns a layer.

GRAPHICAL PRIMITIVES

```
a <- ggplot(economics, aes(date, unemploy))
b <- ggplot(seals, aes(x = long, y = lat))
```

a + geom_blank() and **a + expand_limits()** Ensure limits include values across all plots.

b + geom_curve(aes(yend = lat + 1, xend = long + 1), curvature = 1**)** - x, xend, y, yend, alpha, angle, color, curvature, linetype, size

a + geom_path(lineend = "butt", linejoin = "round", linemitre = 1**)** x, y, alpha, color, group, linetype, size

a + geom_polygon(aes(alpha = 50)**)** - x, y, alpha, color, fill, group, subgroup, linetype, size

b + geom_rect(aes(xmin = long, ymin = lat, xmax = long + 1, ymax = lat + 1)**)** - xmax, xmin, ymax, ymin, alpha, color, fill, linetype, size

a + geom_ribbon(aes(ymin = unemploy - 900, ymax = unemploy + 900)**)** - x, ymax, ymin, alpha, color, fill, group, linetype, size

LINE SEGMENTS

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

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

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

ONE VARIABLE continuous

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

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

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

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

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

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

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

discrete

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

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

TWO VARIABLES both continuous

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

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

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

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

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

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

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

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

THREE VARIABLES

```
seals$z <- with(seals, sqrt(delta_long^2 + delta_lat^2)); l <- ggplot(seals, aes(long, lat))
```

l + geom_contour(aes(z = z)**)** x, y, z, alpha, color, group, linetype, size, weight

l + geom_contour_filled(aes(fill = z)**)** x, y, alpha, color, fill, group, linetype, size, subgroup

continuous bivariate distribution

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

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

h + geom_density_2d() x, y, alpha, color, group, linetype, size

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

continuous function

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

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

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

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

visualizing error

```
df <- data.frame(grp = c("A", "B"), fit = 4:5, se = 1:2)
j <- ggplot(df, aes(grp, fit, ymin = fit - se, ymax = fit + se))
```

j + geom_crossbar(fatten = 2**)** - x, y, ymax, ymin, alpha, color, fill, group, linetype, size

j + geom_errorbar() - x, ymax, ymin, alpha, color, group, linetype, size, width Also **geom_errorbarh()**.

j + geom_linerange() x, ymin, ymax, alpha, color, group, linetype, size

j + geom_pointrange() - x, y, ymin, ymax, alpha, color, fill, group, linetype, shape, size

maps

Draw the appropriate geometric object depending on the simple features present in the data. aes() arguments: map_id, alpha, color, fill, linetype, linewidth.

```
nc <- sf::st_read(system.file("shape/nc.shp", package = "sf"))
ggplot(nc) +
  geom_sf(aes(fill = AREA))
```


Sant'Anna
School of Advanced Studies – Pisa

Plotting a single variable

Plotting a single variable

- We frequently need to plot single variables to visually inspect their distribution

Plotting a single variable

- We frequently need to plot single variables to visually inspect their distribution
- There are various options for doing this:

Plotting a single variable

- We frequently need to plot single variables to visually inspect their distribution
- There are various options for doing this:
  - Histograms (for interval-scaled variables)

Plotting a single variable

- We frequently need to plot single variables to visually inspect their distribution
- There are various options for doing this:
  - Histograms (for interval-scaled variables)
  - Density plots (for interval-scaled variables)

Plotting a single variable

- We frequently need to plot single variables to visually inspect their distribution
- There are various options for doing this:
  - Histograms (for interval-scaled variables)
  - Density plots (for interval-scaled variables)
  - Box plots (for interval-scaled variables)

Plotting a single variable

- We frequently need to plot single variables to visually inspect their distribution
- There are various options for doing this:
  - Histograms (for interval-scaled variables)
  - Density plots (for interval-scaled variables)
  - Box plots (for interval-scaled variables)
  - Bar charts (for categorical variables)

Let's plot the distribution of respondents' age in the Tunisia survey

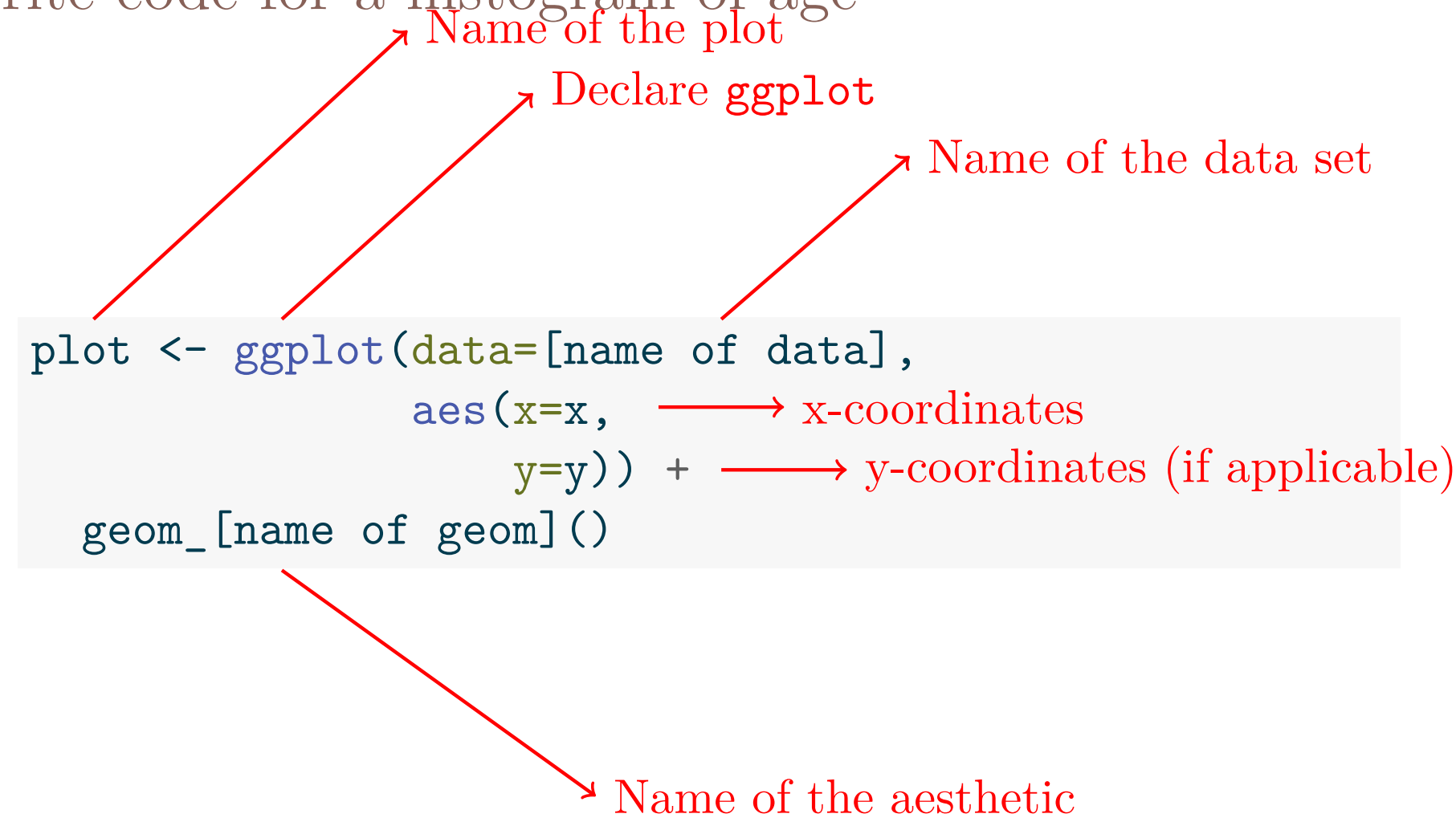
Histograms

- A **histogram** is a type of chart representing the distribution of a single numerical variable.
- In histograms, the x-axis is divided into **bins** which represent ranges of the variable.
- The height of the bars represent **how many observations** fall within a bin.

In `ggplot2`, histogram layers can be produced with `geom_histogram()`.

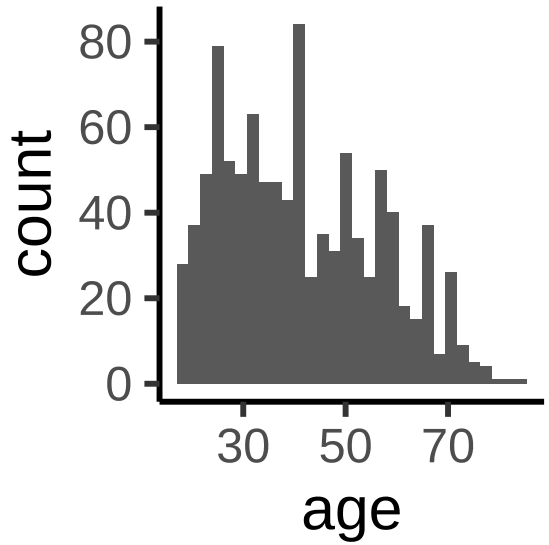
Write code for a histogram of age.

Write code for a histogram of age



Histograms

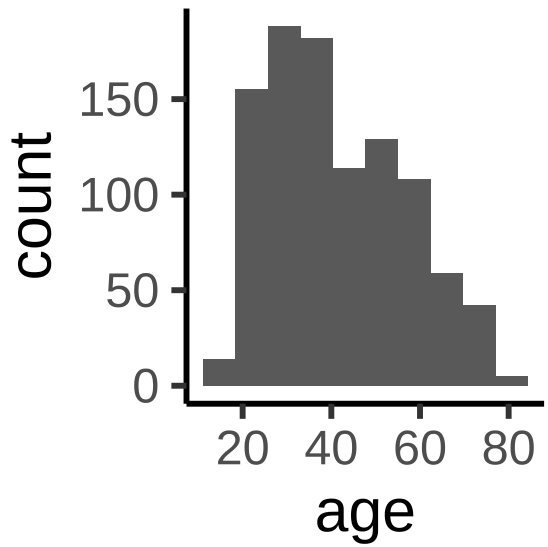
```
plot <- ggplot(data=tun22,
                aes(x=age)) +
  geom_histogram() +
  theme_classic()
plot
```



Histograms

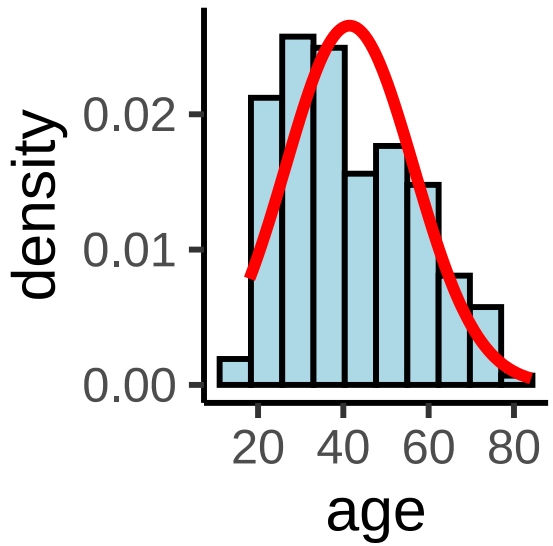
You can set the number of bins with the `bin` option (the default is 30):

```
plot1 <- ggplot(data=tun22,
                aes(x=age)) +
  geom_histogram(bins = 10) +
  theme_classic()
plot1
```



Histograms

We frequently want to check whether a variable aligns with the normal distribution. We can graphically check this by overlaying a normal distribution with the same mean and standard deviation.



Histograms

Here is the code:

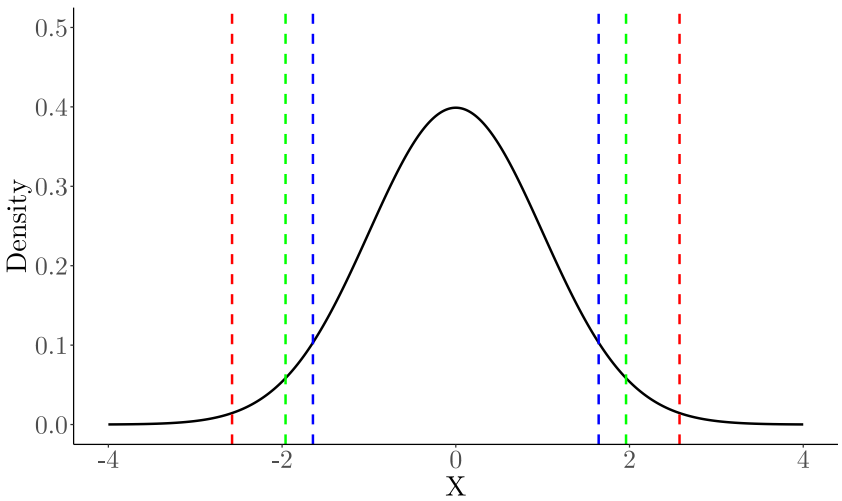
We now plot **density**, not counts

```
plot <- ggplot(data=tun22,
                aes(x=age)) +
  geom_histogram(aes(y=..density..),
                 bins = 10,
                 fill="lightblue",
                 color="black") +
  stat_function(fun = dnorm,
                args = list(mean = mean(tun22$age, na.rm=T),
                            sd = sd(tun22$age, na.rm=T)),
                color = "red", size = 1) +
  theme_classic()
plot
```

Quick detour: What is density?

- **Density** refers to the **probability density function (PDF)**.
- It shows how likely a variable is to take a specific value.
- The total area under the density curve (or histogram) equals **1**.

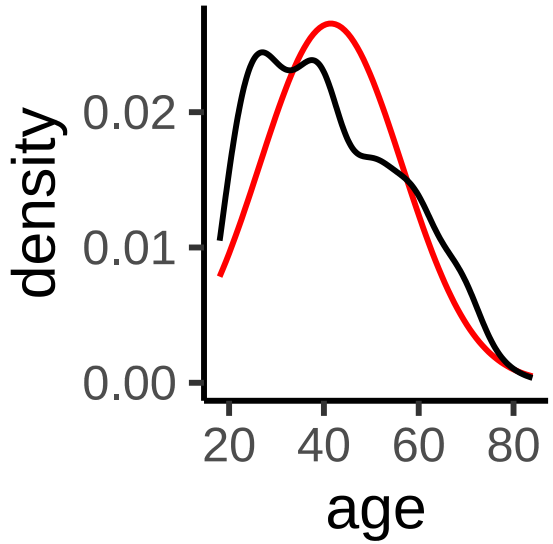
The normal distribution



- 90% of observations fall into the area between the blue lines
- 95% of observations fall into the area between the green lines
- 99% of observations fall into the area between the red lines

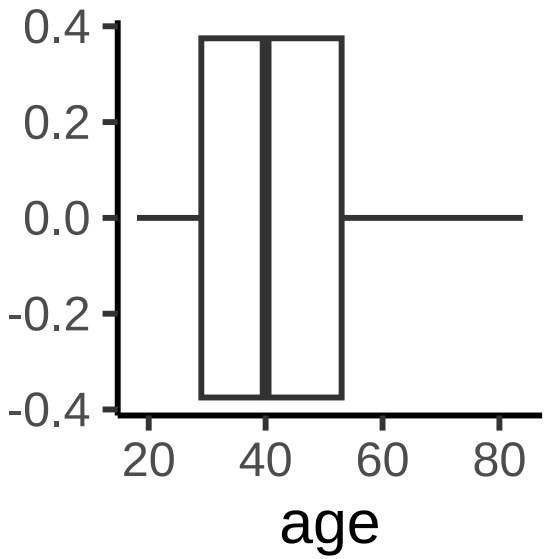
Density plots

```
plot2 <- ggplot(data=tun22,
                aes(x=age)) +
   stat_function(fun = dnorm,
                 args = list(mean = mean(tun22$age, na.rm=T),
                             sd = sd(tun22$age, na.rm=T)),
                 color = "red") +
  geom_density() +
  theme_classic()
plot2
```

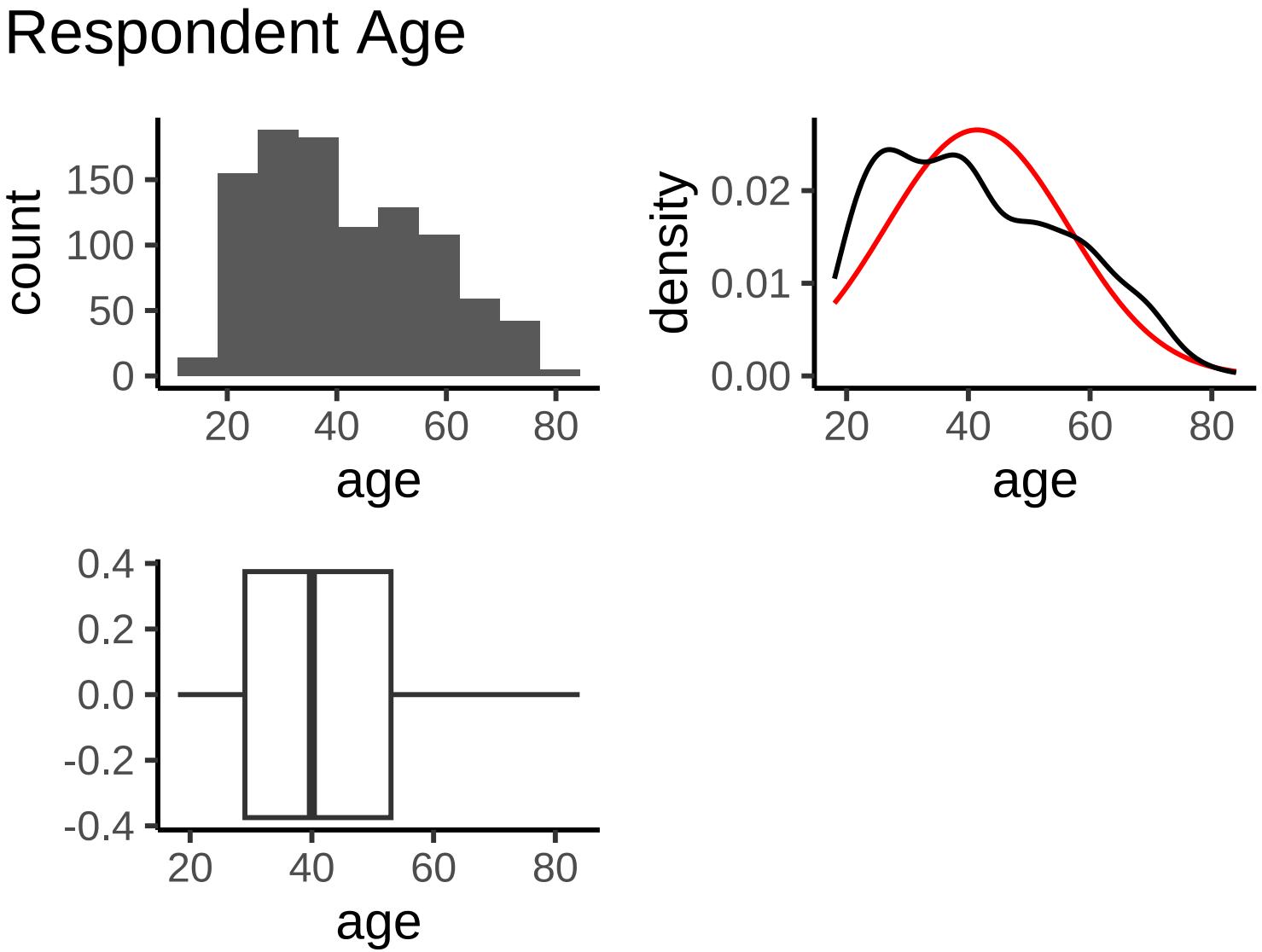


Boxplots

```
plot3 <- ggplot(data=tun22,
                aes(x=age)) +
  geom_boxplot() +
  theme_classic()
plot3
```



Summary


Respondent Age
count
150
100
50
0
20
40
60
80
age
density
0.02
0.01
0.00
20
40
60
80
age
0.4
0.2
0.0
-0.2
-0.4
20
40
60
80
age

ggplot2

We can combine `ggplot` with `tidyverse` data manipulation:

```
tun22 %>%
  group_by(region) %>%
  summarize(n = n()) %>%
  ggplot() +
  geom_bar(
    aes(x = region, y = n),
    stat = "identity"
  ) +
  labs(x="",
       y="") +
  theme_classic() +
  theme(axis.text.x = element_text(angle = 90))
```

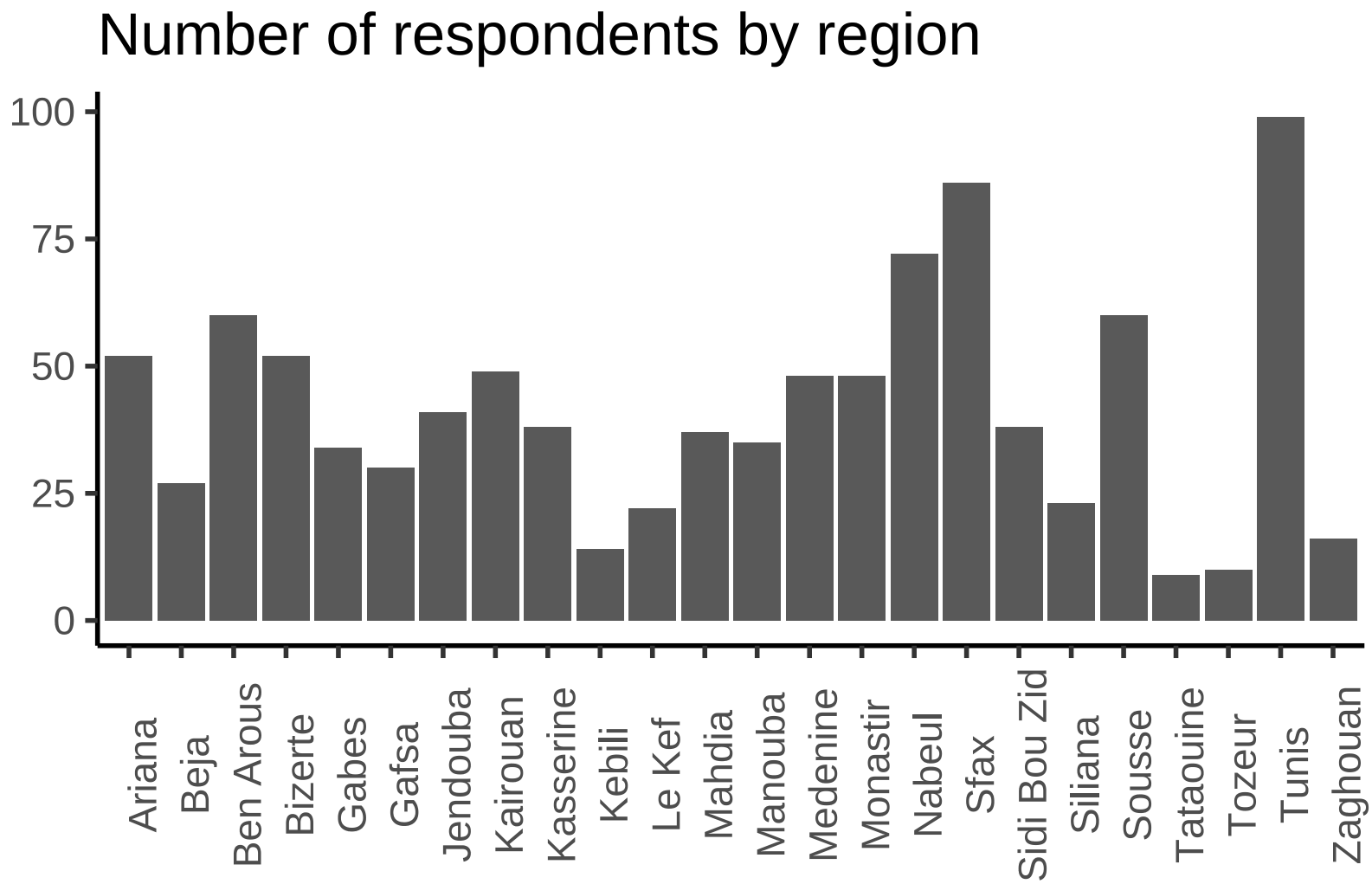
ggplot2

We can combine `ggplot` with `tidyverse` data manipulation:

```
tun22 %>%
  group_by(region) %>%
  summarize(n = n()) %>%
  ggplot() +
  geom_bar(
    aes(x = region, y = n),
    stat = "identity"
  ) +
  labs(x="",
       y="",
       title="Number of respondents by region") +
  theme_classic() +
  theme(axis.text.x = element_text(angle = 90))
```

What will this graph look like?

Bar charts

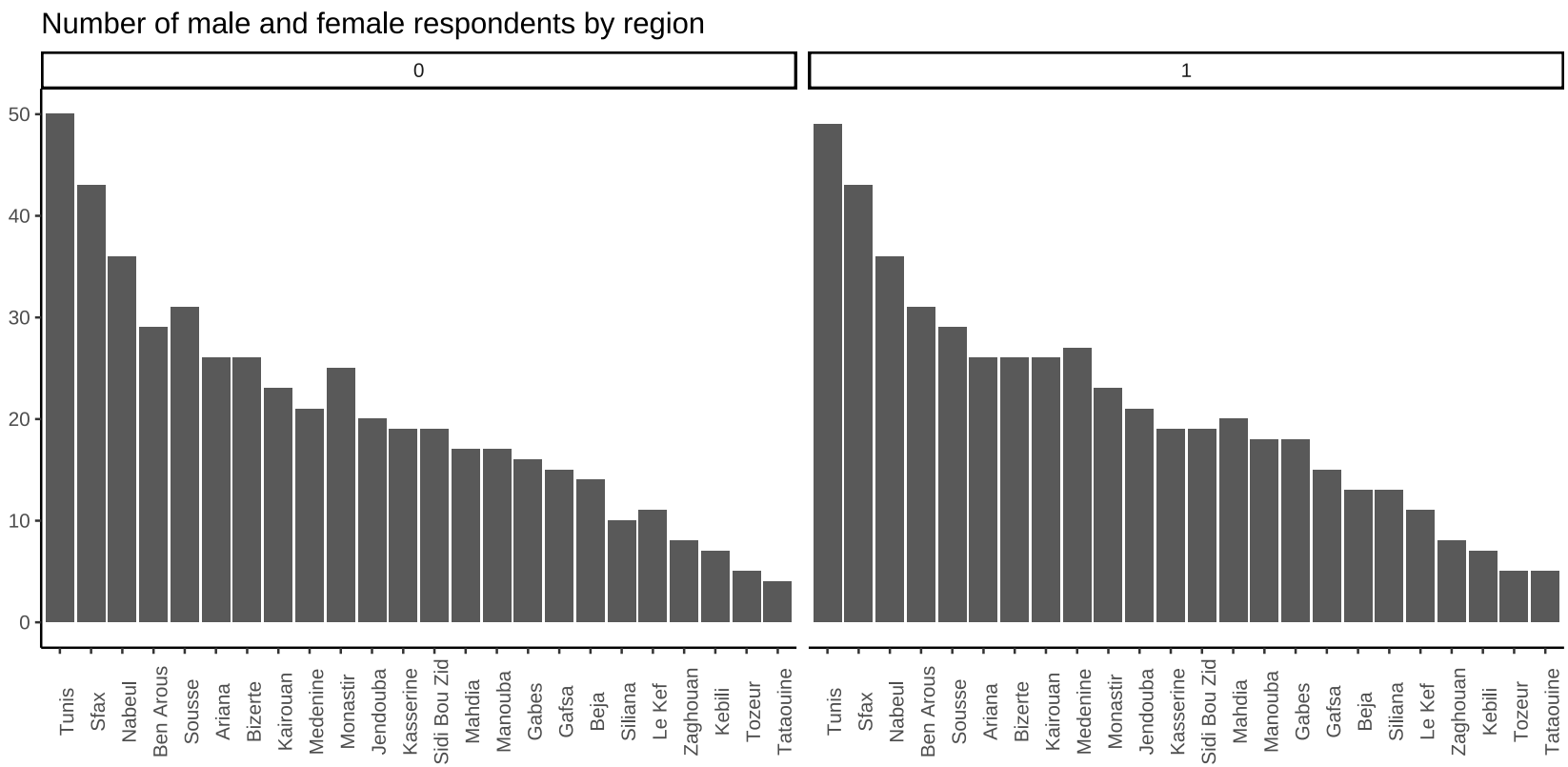

Number of respondents by region
100
75
50
25
0
Ariana
Beja
Ben Arous
Bizerte
Gabes
Gafsa
Jendouba
Kairouan
Kasserine
Kebili
Le Kef
Mahdia
Manouba
Medenine
Monastir
Nabeul
Sfax
Sidi Bou Zid
Siliana
Sousse
Tataouine
Tozeur
Tunis
Zaghouan

ggplot2

We can also produce different plots by group using the `facet()` layer

```
tun22 %>%
  group_by(region,female) %>%
  summarize(n = n()) %>%
  ggplot() +
  geom_bar(
    aes(x = reorder(region,-n), y = n), stat = "identity") +
  labs(x="",
       y="",
       title="Number of male and female respondents by region") +
  theme_classic() +
  theme(axis.text.x = element_text(angle = 90)) +
  facet_wrap(~female)
```

ggplot2


Number of male and female respondents by region
0
1
50
40
30
20
10
0
Tunis
Sfax
Nabeul
Ben Arous
Sousse
Ariana
Bizerte
Kairouan
Medenine
Monastir
Jendouba
Kasserine
Sidi Bou Zid
Mahdia
Manouba
Gabes
Gafsa
Beja
Siliana
Le Kef
Zaghouan
Kebili
Tozeur
Tataouine

Plotting two variables

Scatterplots

- **Scatterplots** are plots displaying the relationship between two numerical variables.
- The variables are plotted on the x and y-axis, each dot represents one observation (or case).

V-Dem data

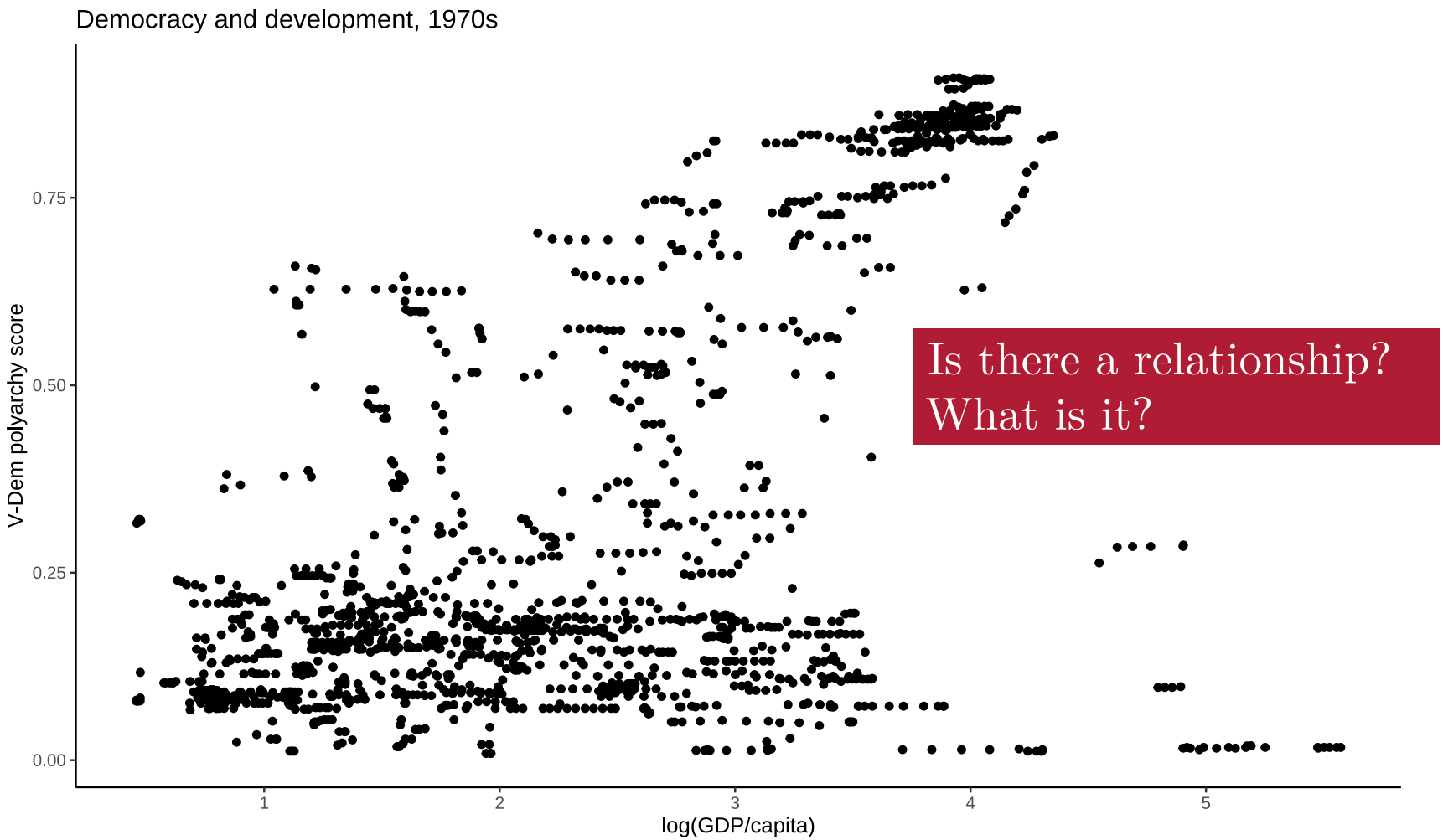
Download the V-Dem data:

```
# First, you need to have the devtools package installed
install.packages("devtools")
# now, install the vdemdata package directly from GitHub
devtools::install_github("vdeminstitute/vdemdata")
# Now load the package
library(vdemdata)

data <- vdem

data <- data %>%
  dplyr::select(v2x_polyarchy,
                e_gdppc,year,
                country_name,
                e_regionpol)
```

Let's create a scatterplot showing the relationship between log GDP/capita and the polyarchy score for the 1970s

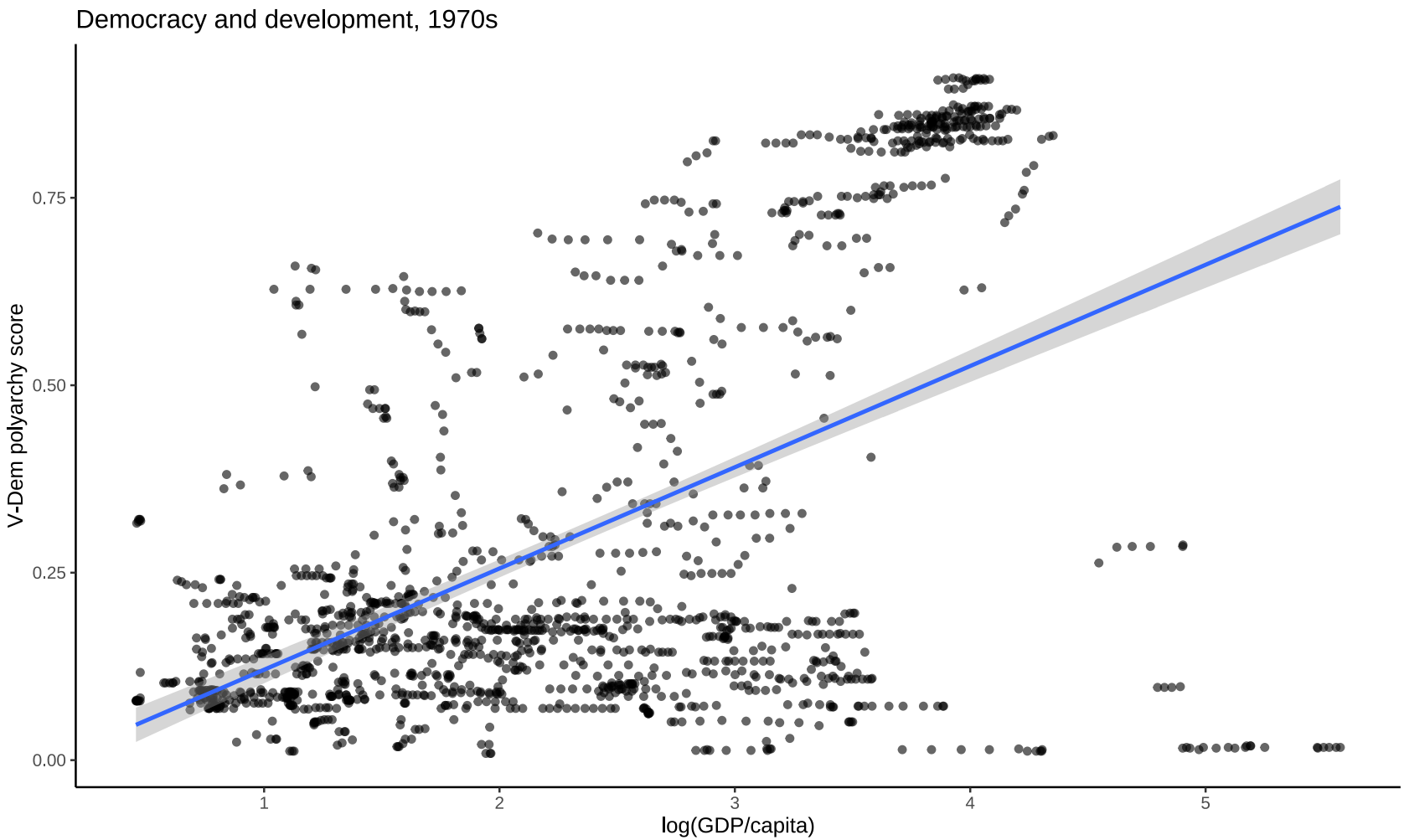
Scatterplots

Scatterplots



Is there a relationship?
What is it?

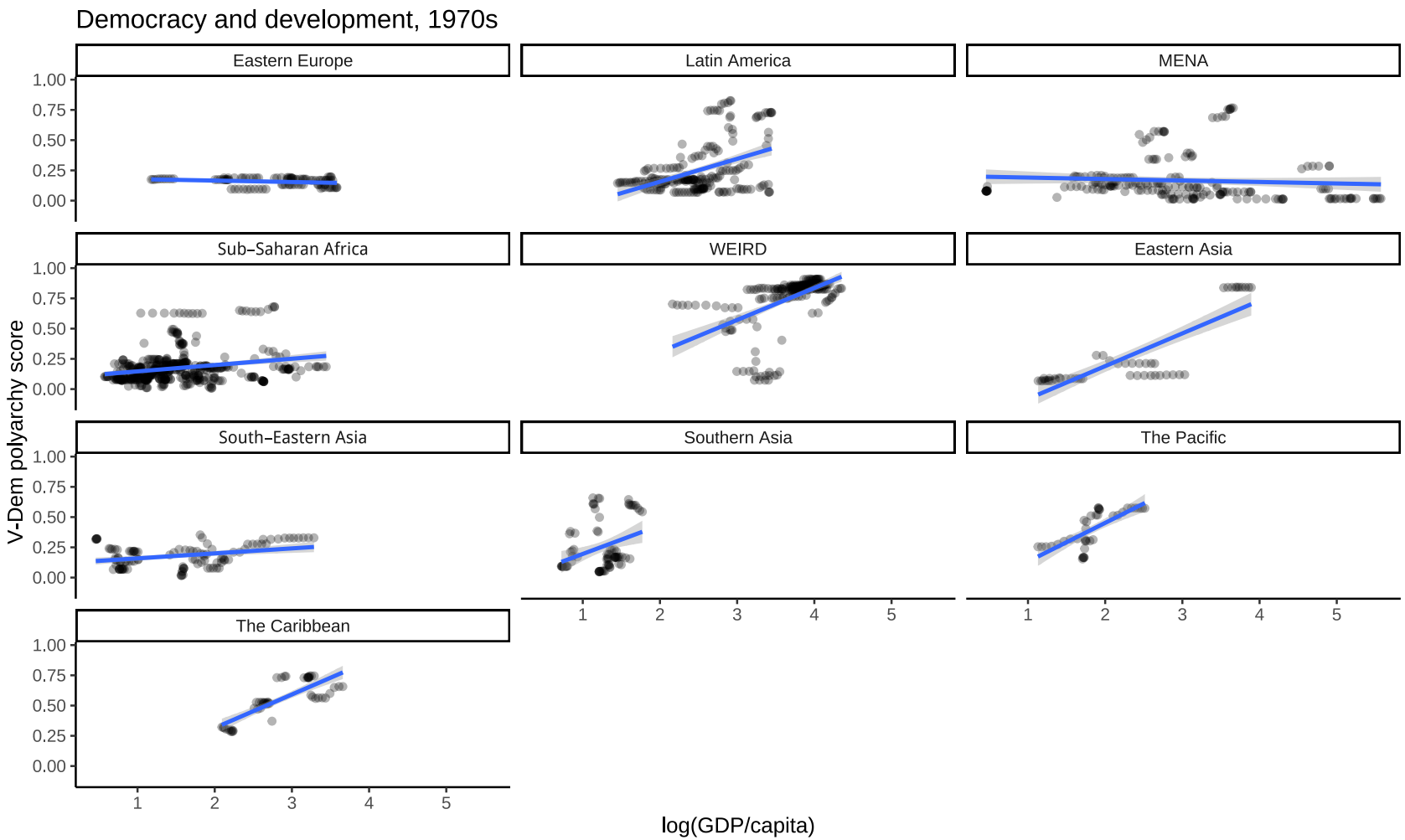
Scatterplots



Scatterplots

```
plot1 <- data %>%
  filter(year>1969 & year<1980) %>%
  rename(democracy=v2x_polyarchy) %>%
  mutate(gdp=log(e_gdppc)) %>%
  ggplot(aes(x=gdp,
             y=democracy)) +
  geom_point(alpha=0.6) +
  geom_smooth(method = "lm") +
  theme_classic() +
  labs(x="log(GDP/capita)",
       y="V-Dem polyarchy score",
       title = "Democracy and development, 1970s")
```

By the way


Democracy and development, 1970s
Eastern Europe
Latin America
MENA
Sub–Saharan Africa
WEIRD
Eastern Asia
South–Eastern Asia
Southern Asia
The Pacific
The Caribbean
V-Dem polyarchy score
1.00
0.75
0.50
0.25
0.00
1
2
3
4
5
log(GDP/capita)

Combining graphs with the patchwork package

```
plot1 <- data %>%
  filter(year>1969 & year<1980) %>%
  rename(democracy=v2x_polyarchy) %>%
  mutate(gdp=log(e_gdppc)) %>%
  ggplot() +
  geom_point(aes(
    x=gdp,
    y=democracy
  ), alpha=0.6, size=1) +
  geom_smooth(aes(
    x=gdp,
    y=democracy),
    method = "lm") +
  theme_classic() +
  labs(x="log(GDP/capita)",
       y="V-Dem polyarchy score",
       title = "All countries")
```

Plot 1

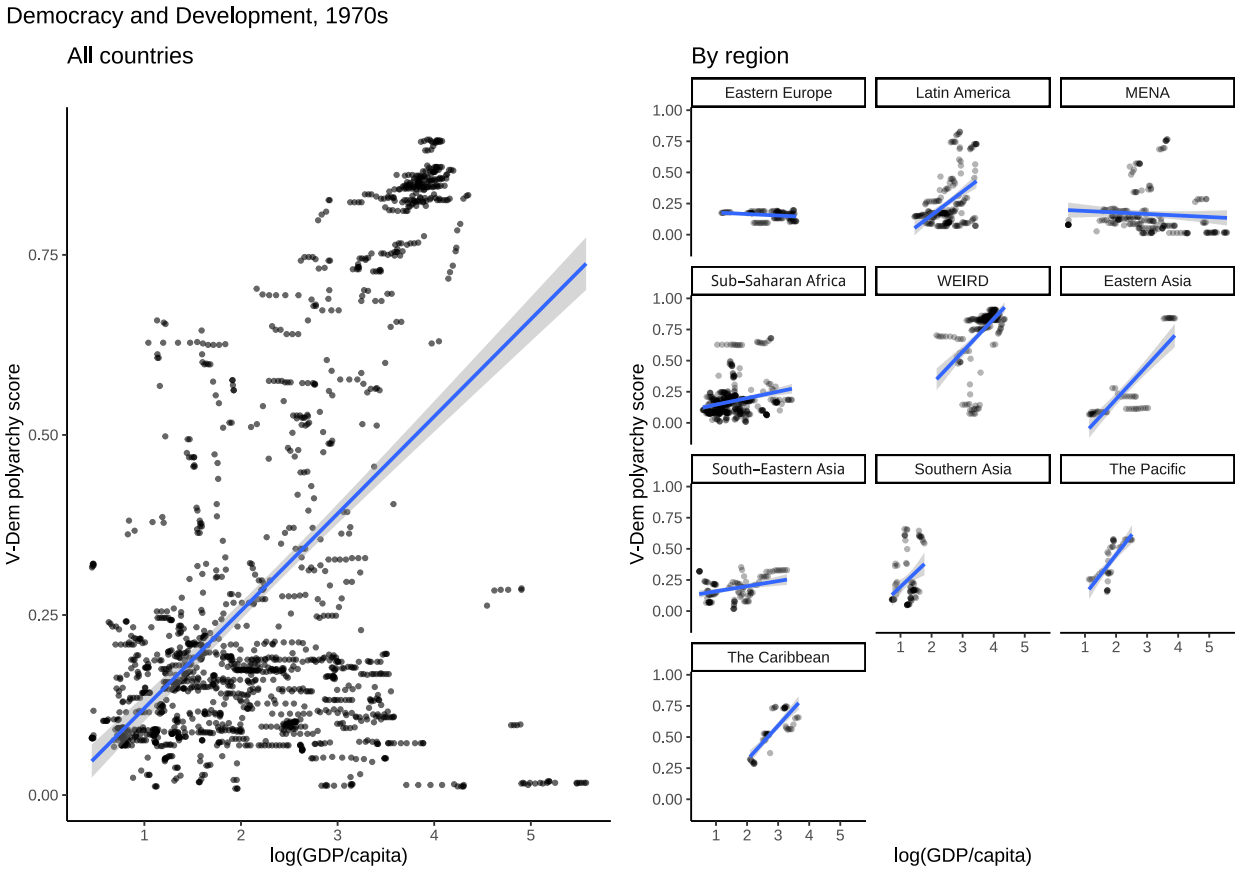
```
library(patchwork)
all_plots <- plot1 + plot2
all_plots + plot_annotation(
  title = "Democracy and Development,
  1970s"
)
```

Combine

```
plot2 <- data %>%
  mutate(region = factor(e_regionpol,
                         levels = 1:10,
                         labels = c("Eastern Europe",
                                    "Latin America",
                                    "MENA",
                                    "Sub-Saharan Africa",
                                    "WEIRD",
                                    "Eastern Asia",
                                    "South-Eastern Asia",
                                    "Southern Asia",
                                    "The Pacific",
                                    "The Caribbean"))) %>%
  filter(year > 1969 & year < 1980) %>%
  rename(democracy = v2x_polyarchy) %>%
  mutate(gdp = log(e_gdppc)) %>%
  ggplot(aes(x = gdp, y = democracy)) +
  geom_point(alpha=0.3, size=1, show.legend = F) +
  geom_smooth(method = "lm", show.legend = FALSE) +
  theme_classic() +
  labs(
    x = "log(GDP/capita)",
    y = "V-Dem polyarchy score",
    title = "By region"
  ) +
  facet_wrap(~ region, ncol = 3)
```

Plot 2

Combining graphs with the `patchwork` package



Basic programming

Sant'Anna
School of Advanced Studies – Pisa

Operators

1. **Relational** operators
   1. `<` less than
   2. `>` greater than
   3. `<=` less than or equal to
   4. `>=` greater than or equal to
   5. `==` equal (identical) to
   6. `!=` not equal to
2. **Logical** operators
   1. `!` not ($\neg$ in propositional logic)
   2. `&` and ($\wedge$ in propositional logic)
   3. `|` or ($\vee$ in propositional logic)

Operators return an object of the class "logical" which can have the values `TRUE` or `FALSE`.

Operators

What will this evaluate to and why?

- `1 < 2 & "left"=="right"` — ?
- `1 < 2 | "left"=="right"` — ?
- `1 < 2 | !"left"=="right"` — ?
- `!(1 < 2 | "left"=="right")` — ?

Operators

What will this evaluate to and why?

- `1 < 2 & "left"=="right"` — `FALSE`
- `1 < 2 | "left"=="right"` — `TRUE`
- `1 < 2 | !"left"=="right"` — `TRUE`
- `!(1 < 2 | "left"=="right")` — `FALSE`

Conditional statements

All programming languages have a way of making conditional statements and R is no exception:

```
if ([test]) {
  [Action if test is passed]
} else {
  [Action if test is not passed]
}
```

Try writing code which returns “positive” if a number is positive and “negative or zero” otherwise

Conditional statements

```
x <- 0
if (x>0) {
  print("positive")
} else {
  print("negative or zero")
}
```

```
[1] "negative or zero"
```

Conditional statements

You can nest if/else statements:

```
x <- 0
if (x>0) {
  print("positive")
} else if (x==0) {
  print("zero")
} else {
  print("negative")
}
```

```
[1] "zero"
```

The `for` loop

`for` loops are useful if we need to repeat the same operation many times on different objects.

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

```
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
[1] 6
[1] 7
[1] 8
[1] 9
[1] 10
```

```
words <- c("this","is",
           "a","for",
           "loop")
for (w in words) {
  print(w)
}
```

```
[1] "this"
[1] "is"
[1] "a"
[1] "for"
[1] "loop"
```

Task

Let's write a function to loop through all columns of a dataset and plot the variable if it is a Likert-scaled question.