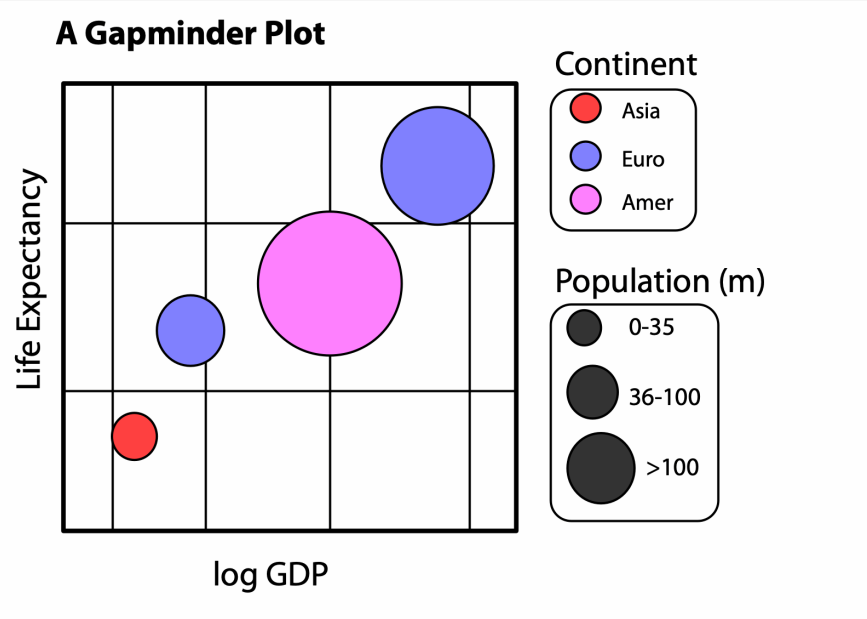
ggplot's flow of action

Data Visualization: Session 2

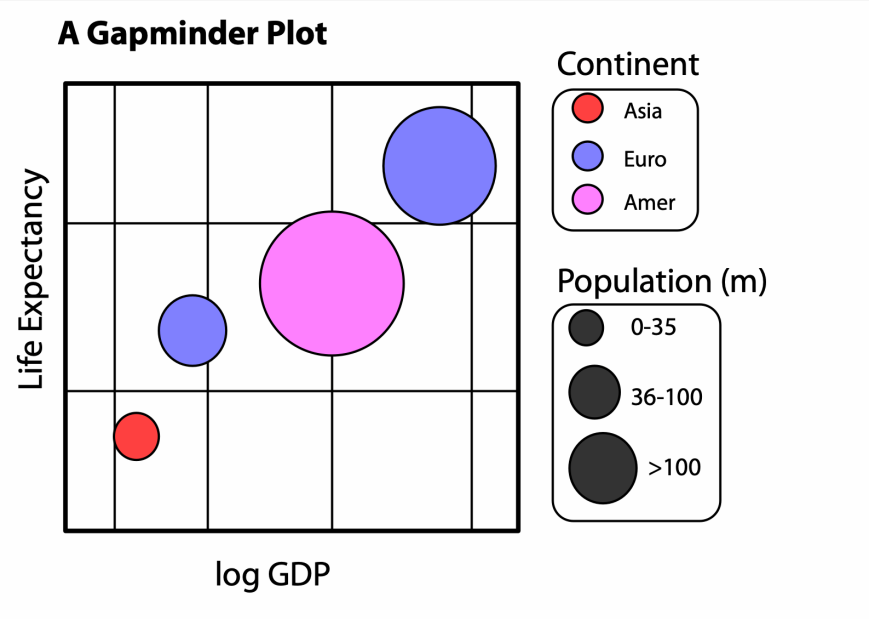
Kieran Healy
Duke University, May 2023

A Plot's Components

What we need our code to make

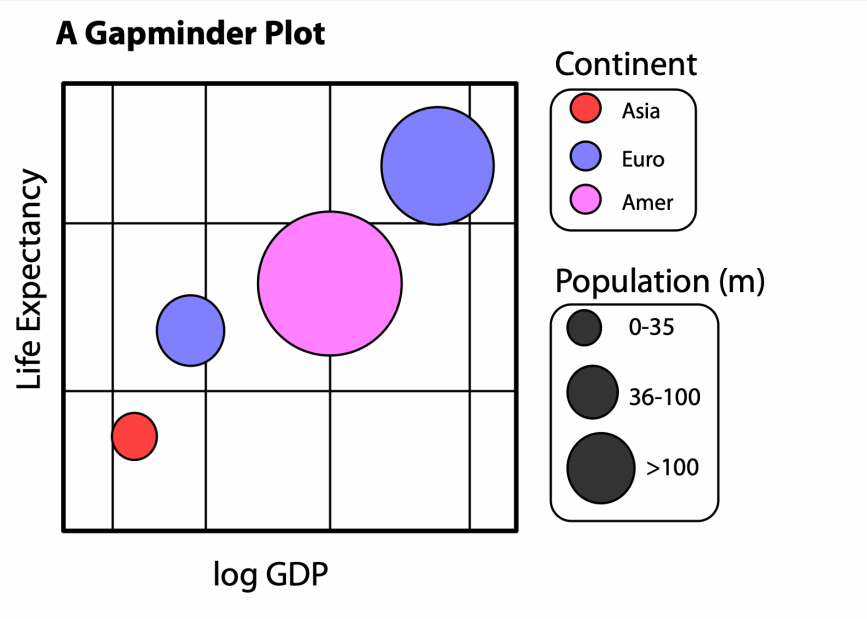


What we need our code to make



Data **represented** by visual elements;

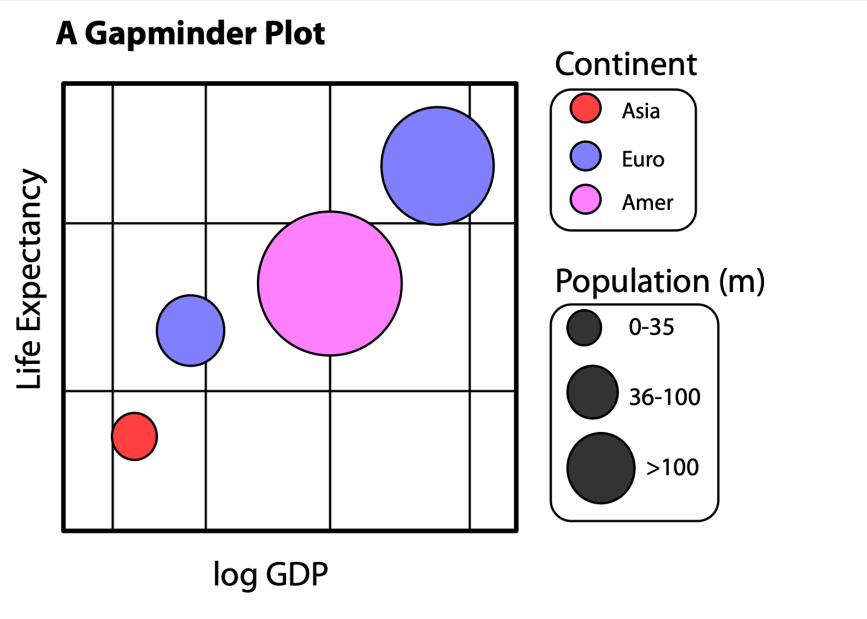
What we need our code to make



Data **represented** by visual elements;

like *position*, *length*, *color*, and *size*;

What we need our code to make

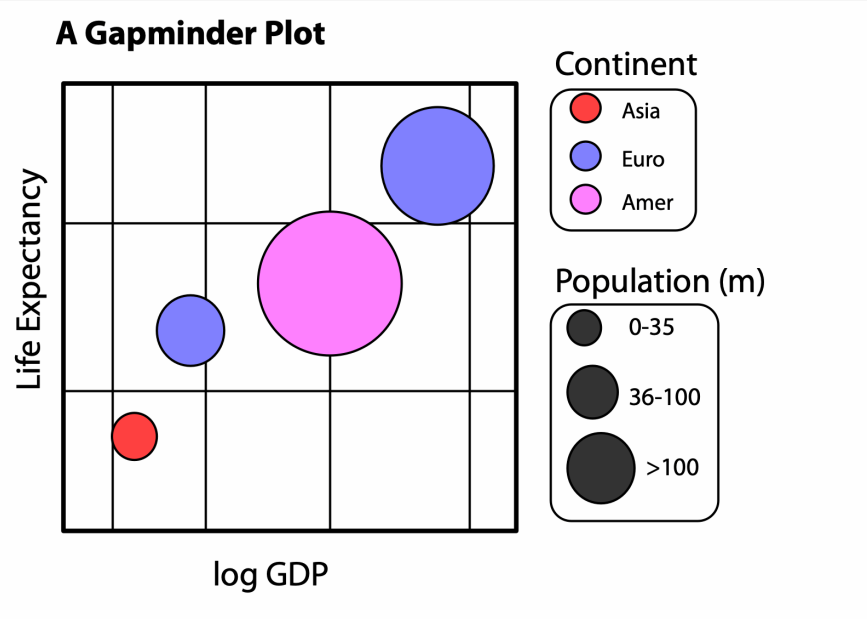


Data **represented** by visual elements;

like *position*, *length*, *color*, and *size*;

Each measured on some **scale**;

What we need our code to make



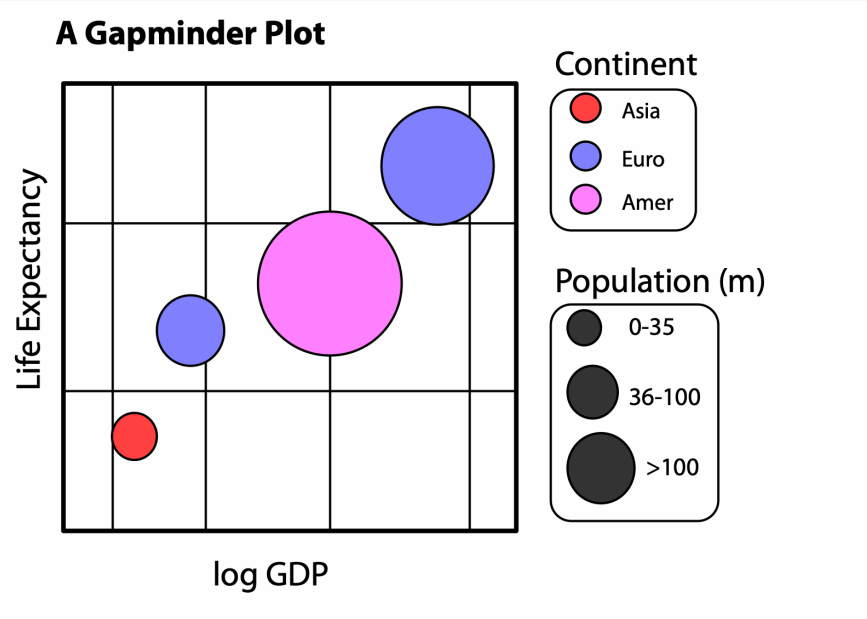
Data **represented** by visual elements;

like *position*, *length*, *color*, and *size*;

Each measured on some **scale**;

Each scale with a labeled **guide**;

What we need our code to make



Data **represented** by visual elements;

like *position*, *length*, *color*, and *size*;

Each measured on some **scale**;

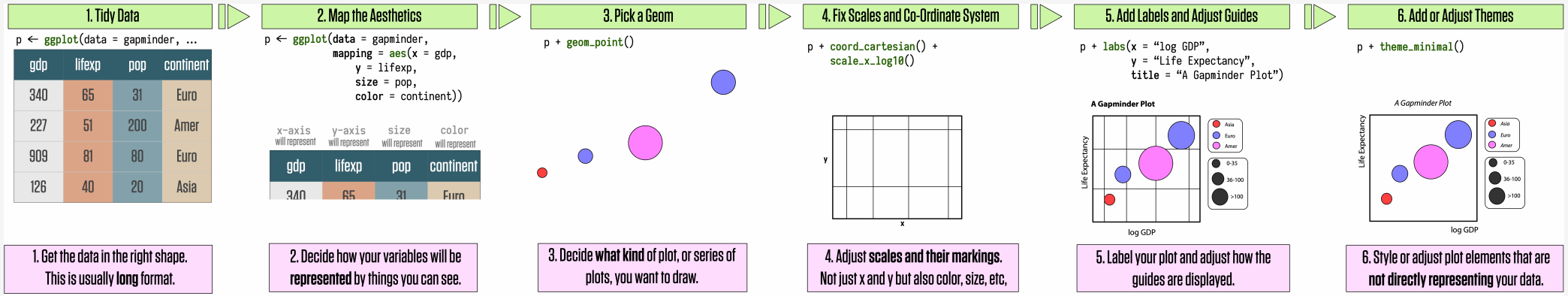
Each scale with a labeled **guide**;

With the plot itself also **titled** and labeled.

How ggplot does this

ggplot's flow of action

Here's the whole thing from start to finish

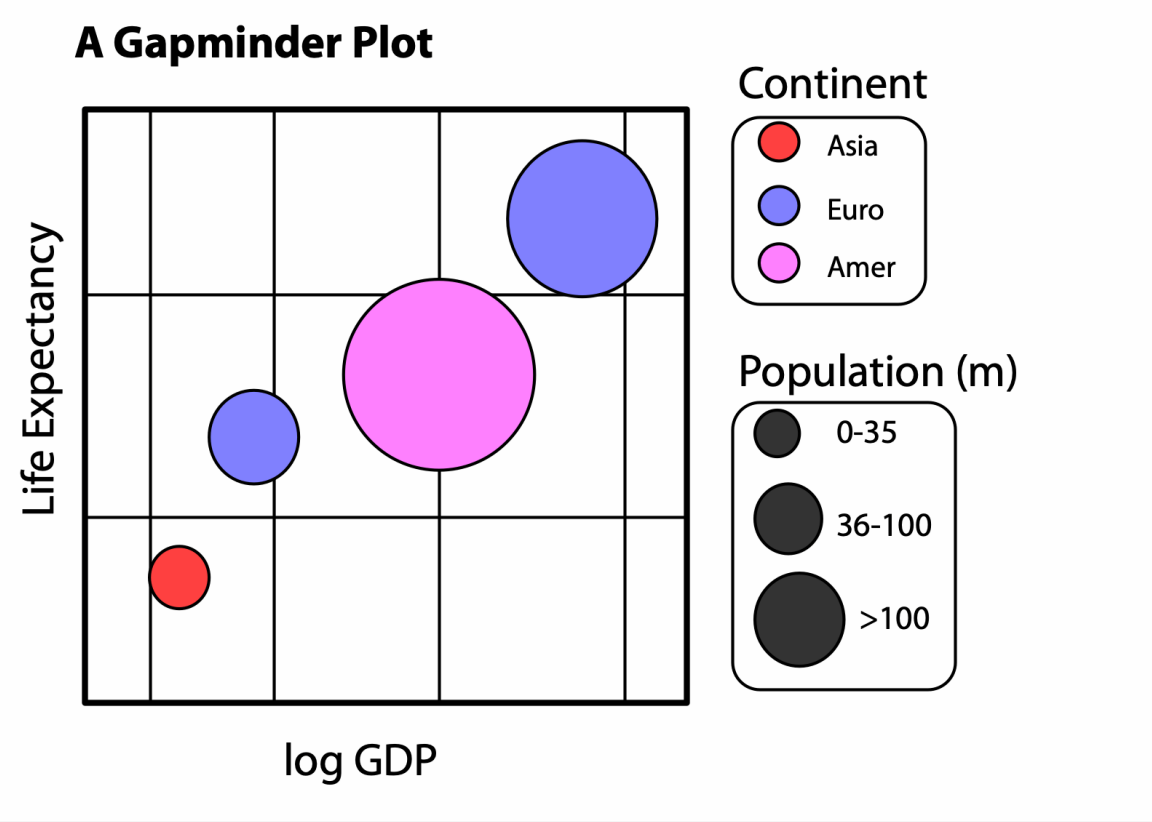


We'll go through it step by step

ggplot's flow of action

gdp	lifexp	pop	continent
340	65	31	Euro
227	51	200	Amer
909	81	80	Euro
126	40	20	Asia

ggplot's flow of action



ggplot's flow of action

1. Tidy Data

```
p ← ggplot(data = gapminder, ...
```

gdp	lifexp	pop	continent
340	65	31	Euro
227	51	200	Amer
909	81	80	Euro
126	40	20	Asia

1. Get the data in the right shape.
This is usually **long** format.

2. Map the Aesthetics

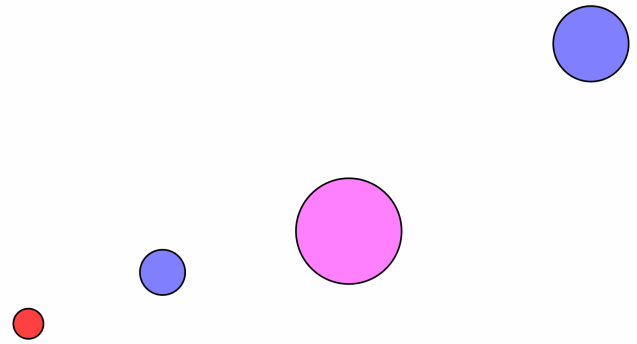
```
p ← ggplot(data = gapminder,
           mapping = aes(x = gdp,
                         y = lifexp,
                         size = pop,
                         color = continent))
```

x-axis will represent	y-axis will represent	size will represent	color will represent
gdp	lifexp	pop	continent
340	65	31	Euro

2. Decide how your variables will be **represented** by things you can see.

3. Pick a Geom

```
p + geom_point()
```

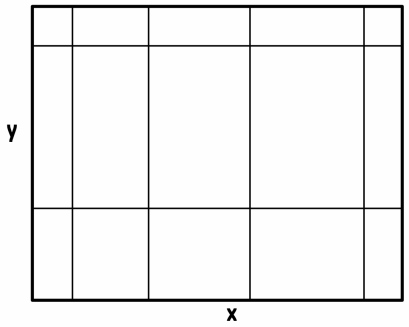


3. Decide **what kind** of plot, or series of plots, you want to draw.

ggplot's flow of action

4. Fix Scales and Co-Ordinate System

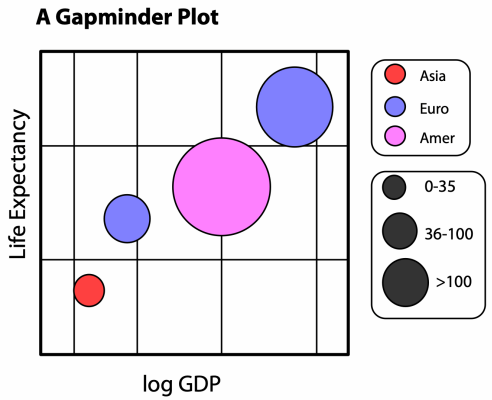
```
p + coord_cartesian() +
    scale_x_log10()
```



4. Adjust **scales and their markings**. Not just x and y but also color, size, etc,

5. Add Labels and Adjust Guides

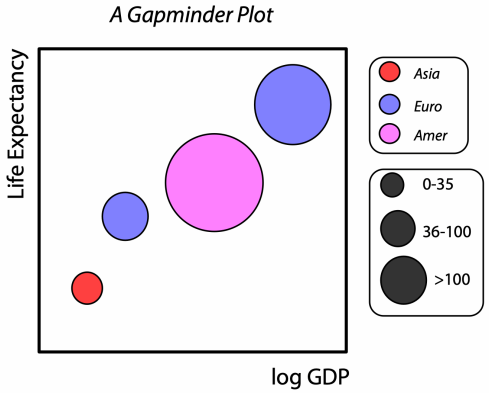
```
p + labs(x = "log GDP",
         y = "Life Expectancy",
         title = "A Gapminder Plot")
```



5. Label your plot and adjust how the guides are displayed.

6. Add or Adjust Themes

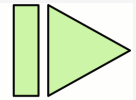
```
p + theme_minimal()
```



6. Style or adjust plot elements that are **not directly representing** your data.

ggplot's flow of action: required

1. Tidy Data



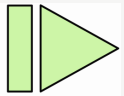
```
p ← ggplot(data = gapminder, ...
```

gdp	lifexp	pop	continent
340	65	31	Euro
227	51	200	Amer
909	81	80	Euro
126	40	20	Asia

1. Get the data in the right shape.
This is usually **long** format.

ggplot's flow of action: required

2. Map the Aesthetics



```
p ← ggplot(data = gapminder,
           mapping = aes(x = gdp,
                         y = lifexp,
                         size = pop,
                         color = continent))
```

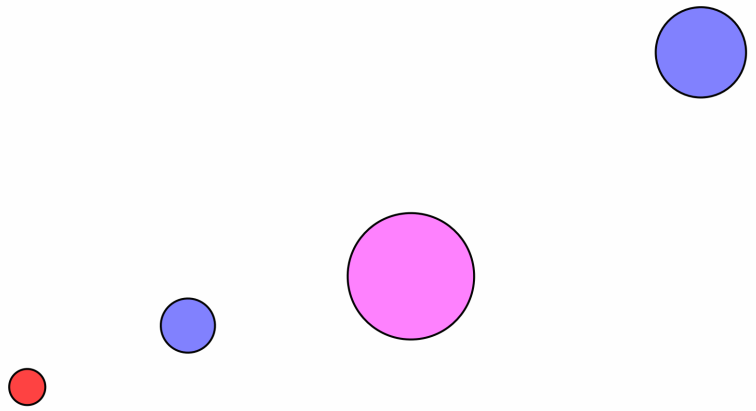
x-axis will represent	y-axis will represent	size will represent	color will represent
gdp	lifexp	pop	continent
340	65	31	Euro

2. Decide how your variables will be represented by things you can see.

ggplot's flow of action: required

3. Pick a Geom

```
p + geom_point()
```



3. Decide **what kind** of plot, or series of plots, you want to draw.

Let's go piece by piece

Start with the data

```
gapminder
```

```
## # A tibble: 1,704 × 6
##    country     continent  year lifeExp      pop gdpPercap
##    <fct>       <fct>     <int>   <dbl>    <int>     <dbl>
##  1 Afghanistan Asia       1952    28.8  8425333      779.
##  2 Afghanistan Asia       1957    30.3  9240934      821.
##  3 Afghanistan Asia       1962    32.0 10267083      853.
##  4 Afghanistan Asia       1967    34.0 11537966      836.
##  5 Afghanistan Asia       1972    36.1 13079460      740.
##  6 Afghanistan Asia       1977    38.4 14880372      786.
##  7 Afghanistan Asia       1982    39.9 12881816      978.
##  8 Afghanistan Asia       1987    40.8 13867957      852.
##  9 Afghanistan Asia       1992    41.7 16317921      649.
## 10 Afghanistan Asia       1997    41.8 22227415      635.
## # ℹ 1,694 more rows
```

```
dim(gapminder)
```

```
## [1] 1704    6
```

Create a plot object

Data is the `gapminder` tibble.

```
p ← ggplot(data = gapminder)
```

Map variables to aesthetics

Tell `ggplot` the variables you want represented by visual elements on the plot

```
p ← ggplot(data = gapminder,
           mapping = aes(x = gdpPercap,
                         y = lifeExp))
```

Map variables to aesthetics

The `mapping` = `aes( ... )` call links variables to things you will see on the plot.

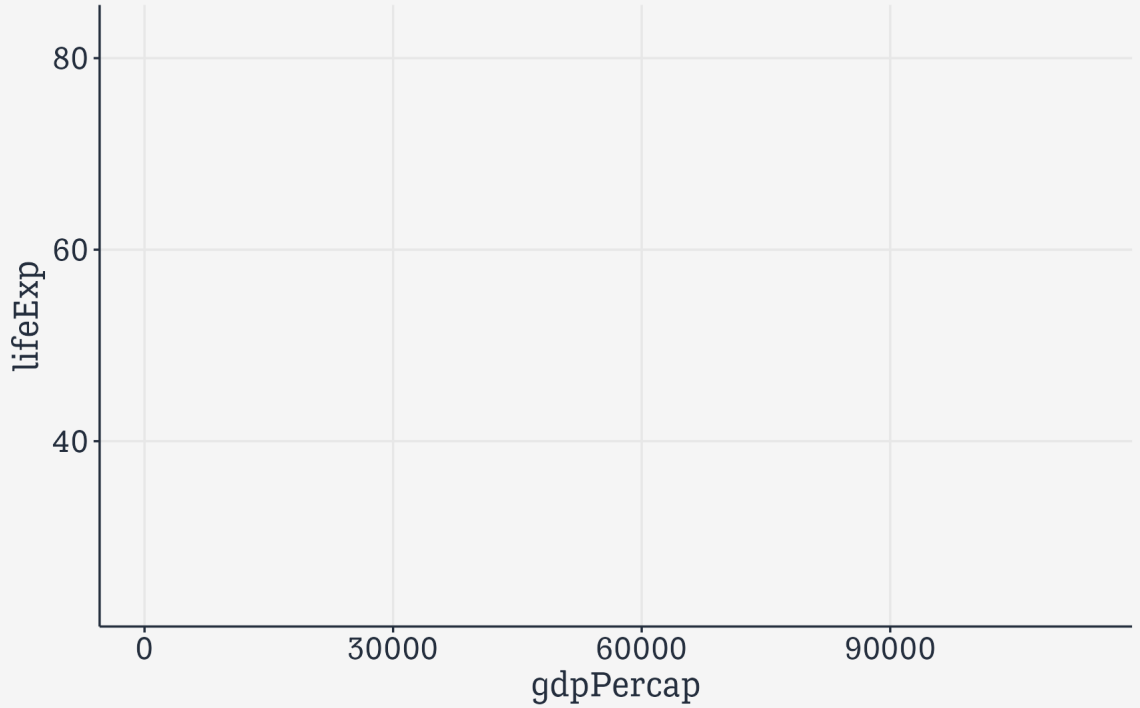
`x` and `y` represent the quantities determining position on the `x` and `y` axes.

Other aesthetic mappings can include, e.g., `color`, `shape`, `size`, and `fill`.

Mappings do not *directly* specify the particular, e.g., colors, shapes, or line styles that will appear on the plot. Rather, they establish *which variables* in the data will be represented by *which visible elements* on the plot.

p has data and mappings but no geom

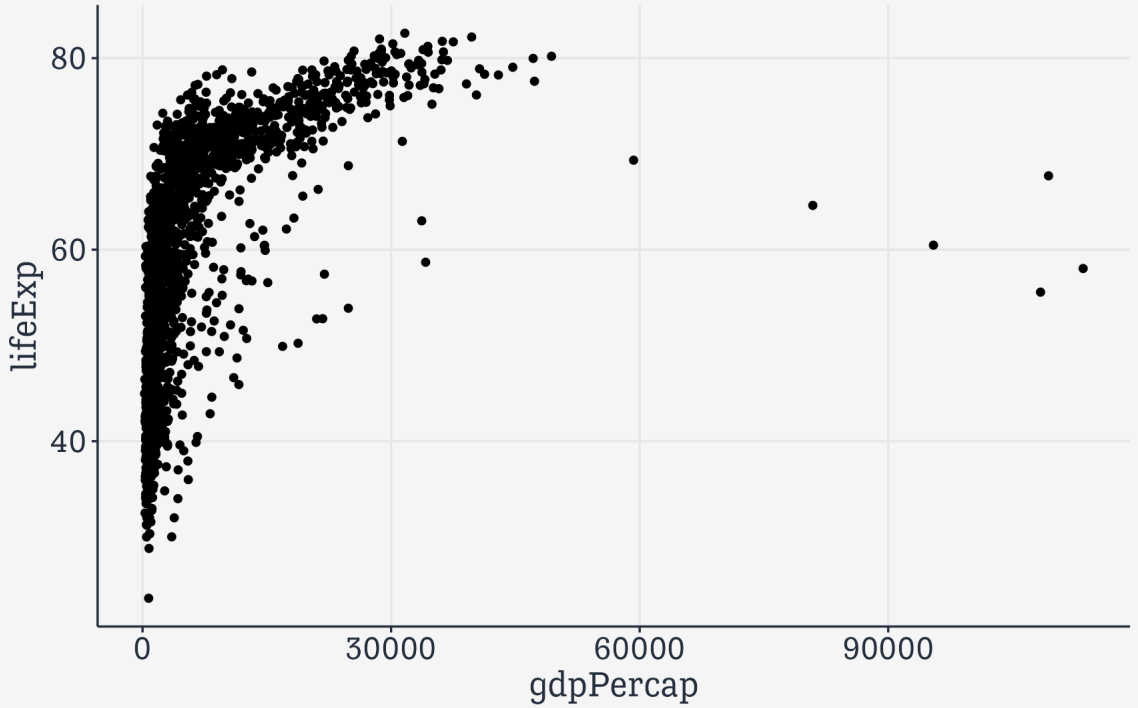
```
p
```



This empty plot has no geoms.

Add a geom

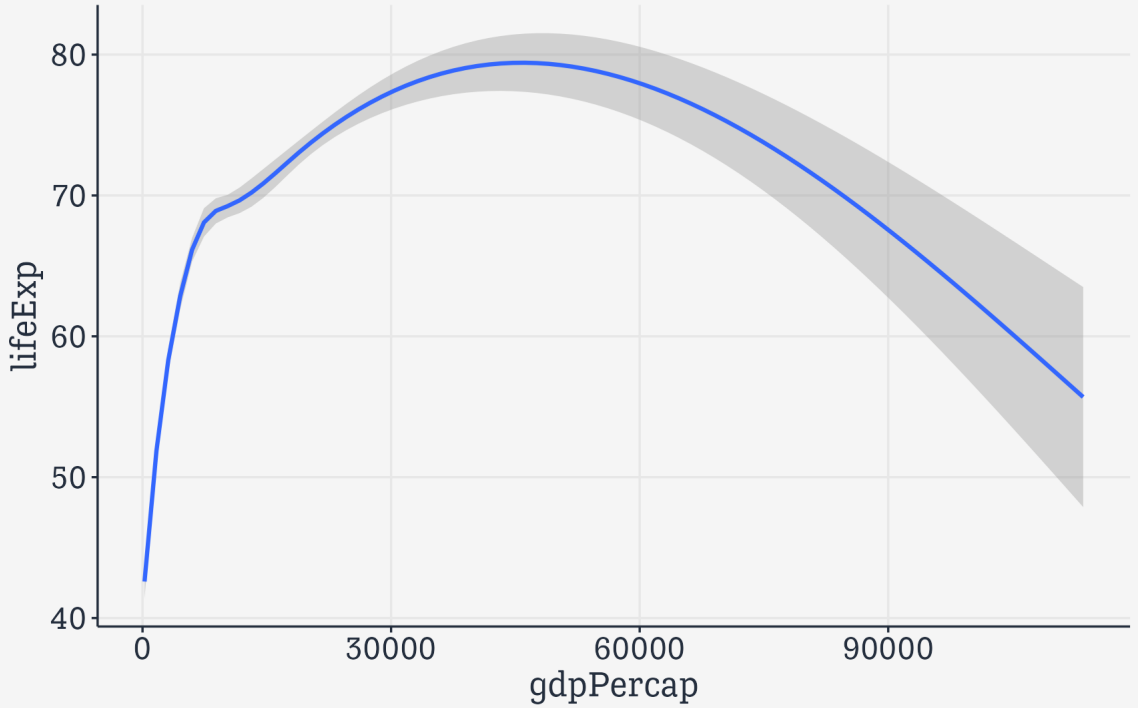
```
p + geom_point()
```



A scatterplot of Life Expectancy vs GDP

Try a different geom

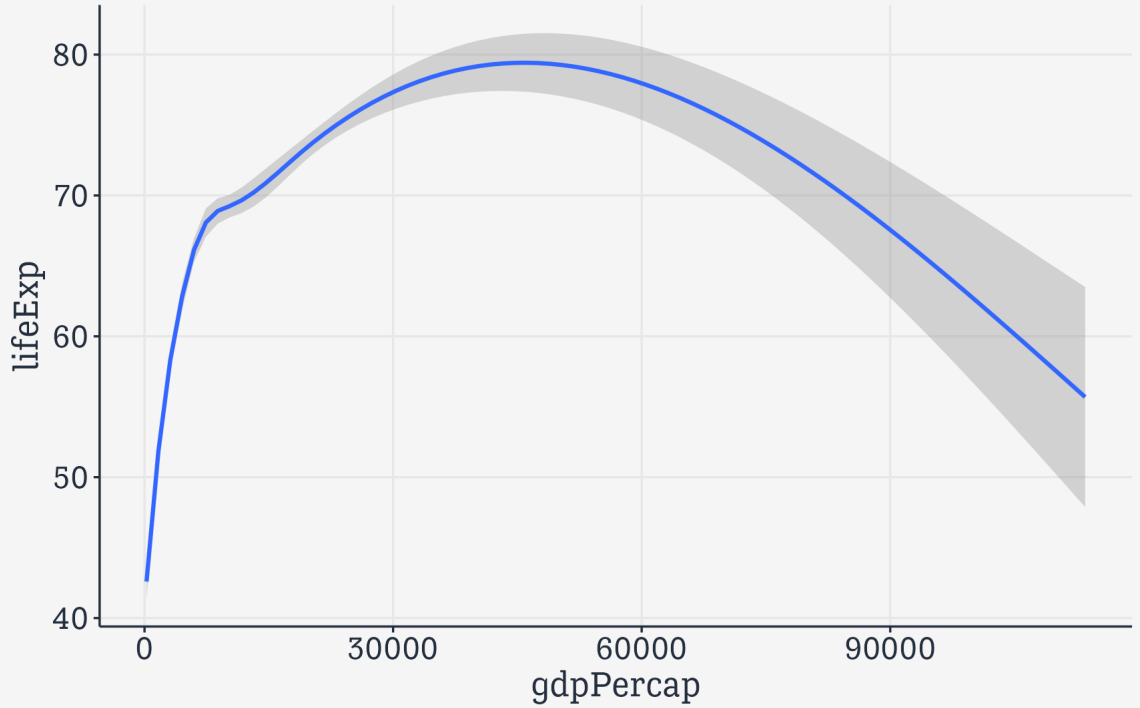
```
p + geom_smooth()
```



A scatterplot of Life Expectancy vs GDP

Build your plots layer by layer

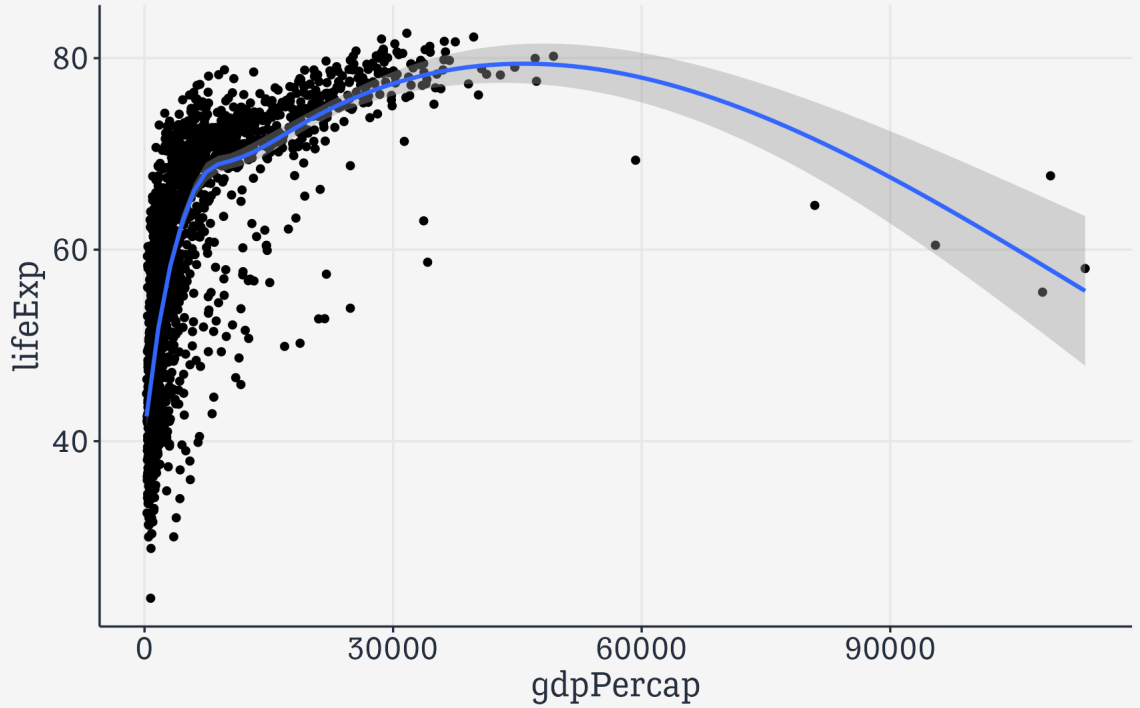
```
p ← ggplot(data = gapminder,
           mapping = aes(x = gdpPercap,
                         y=lifeExp))
p + geom_smooth()
```



Life Expectancy vs GDP, using a smoother.

This process is additive

```
p ← ggplot(data = gapminder,
           mapping = aes(x = gdpPercap,
                         y=lifeExp))
p + geom_point() + geom_smooth()
```



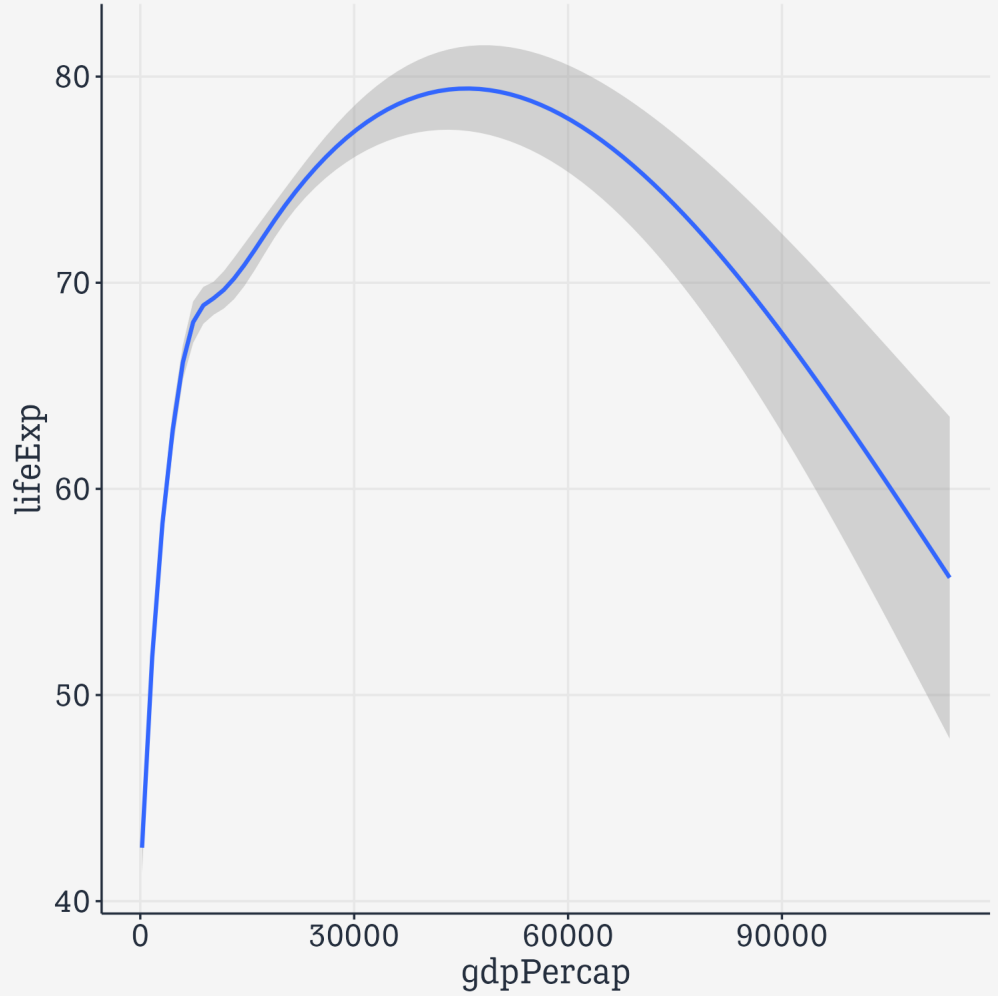
Life Expectancy vs GDP, using a smoother.

This process is additive

```
p ← ggplot(data = gapminder,
           mapping = aes(x = gdpPercap,
                         y=lifeExp))
```

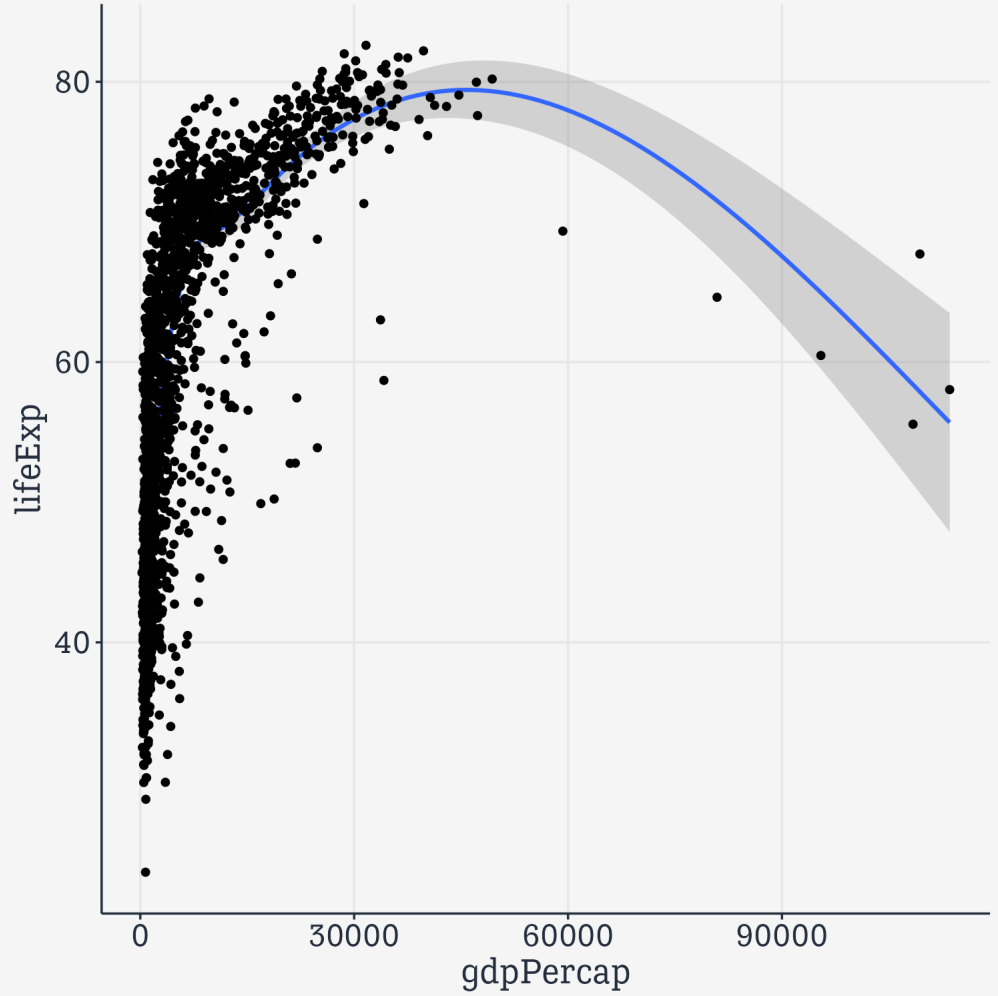
This process is additive

```
p ← ggplot(data = gapminder,
           mapping = aes(x = gdpPercap,
                         y=lifeExp))
p + geom_smooth()
```



This process is additive

```
p ← ggplot(data = gapminder,
           mapping = aes(x = gdpPercap,
                         y=lifeExp))
p + geom_smooth() +
  geom_point()
```



Every geom is a function

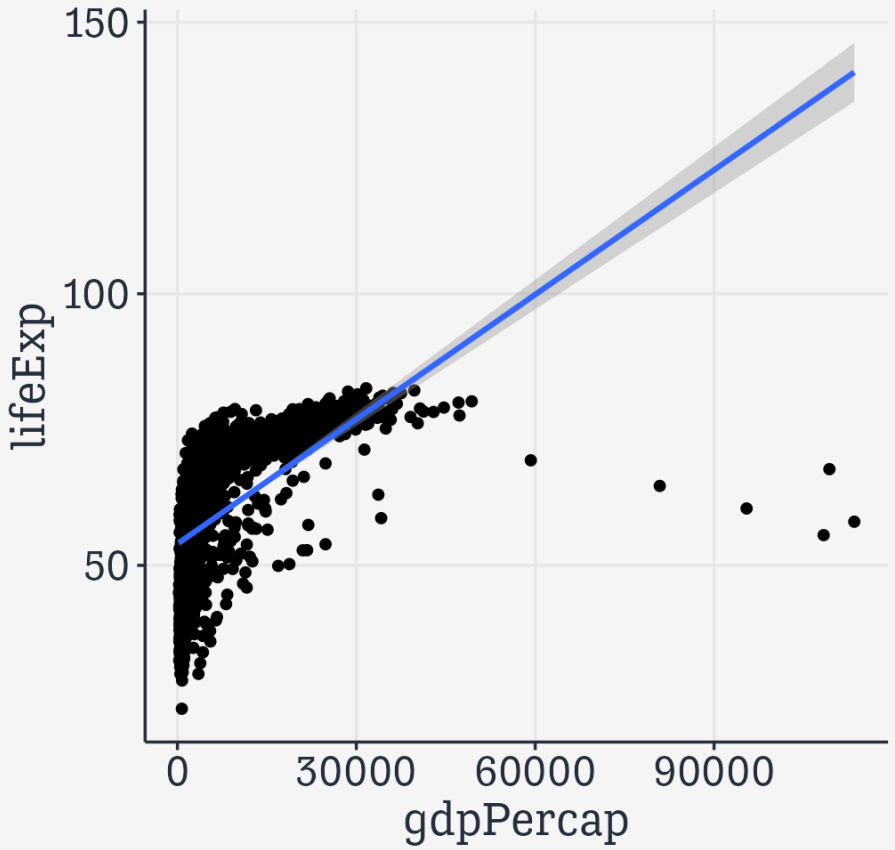
Functions take arguments

```
p ← ggplot(data = gapminder,
           mapping = aes(x = gdpPercap,
                         y = lifeExp))
p + geom_point() +
  geom_smooth(method = "lm")
```

Every geom is a function

Functions take arguments

```
p ← ggplot(data = gapminder,
           mapping = aes(x = gdpPercap,
                         y = lifeExp))
p + geom_point() +
  geom_smooth(method = "lm")
```

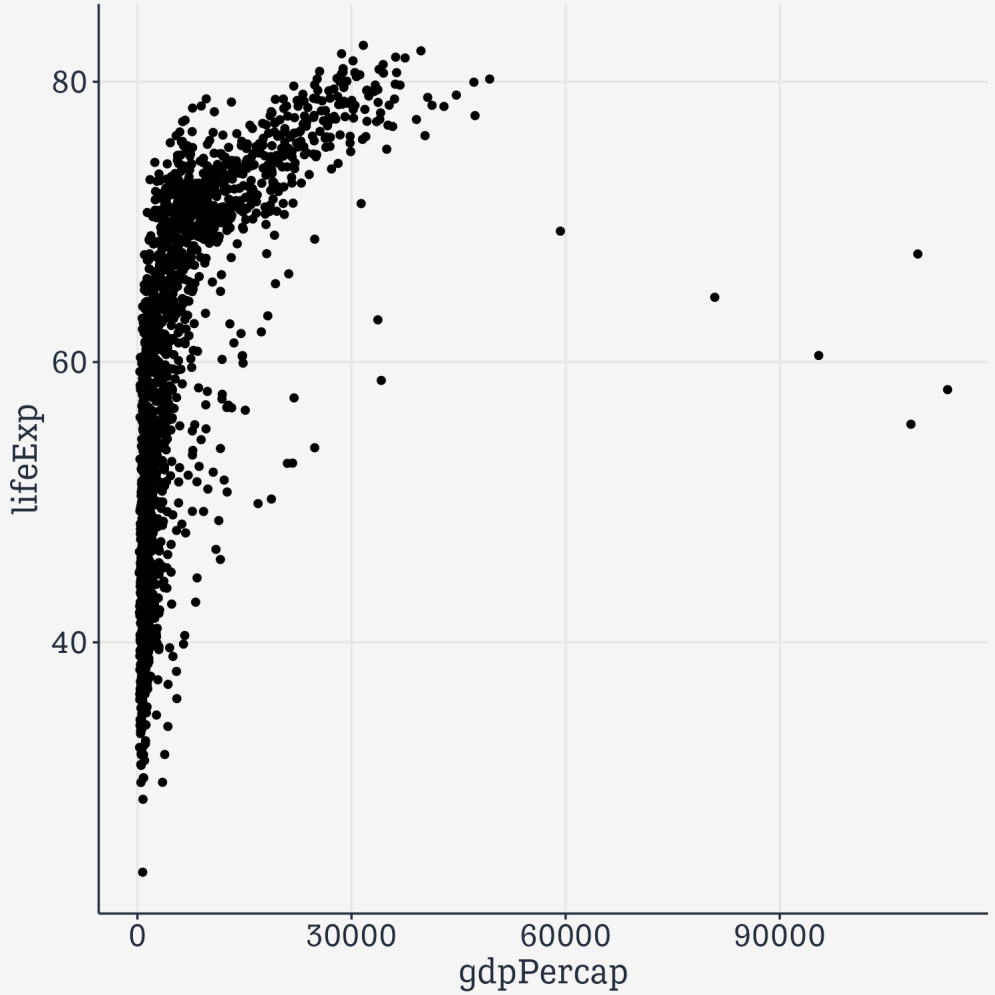


Keep Layering

```
p ← ggplot(data = gapminder,
             mapping = aes(x = gdpPercap,
                           y=lifeExp))
```

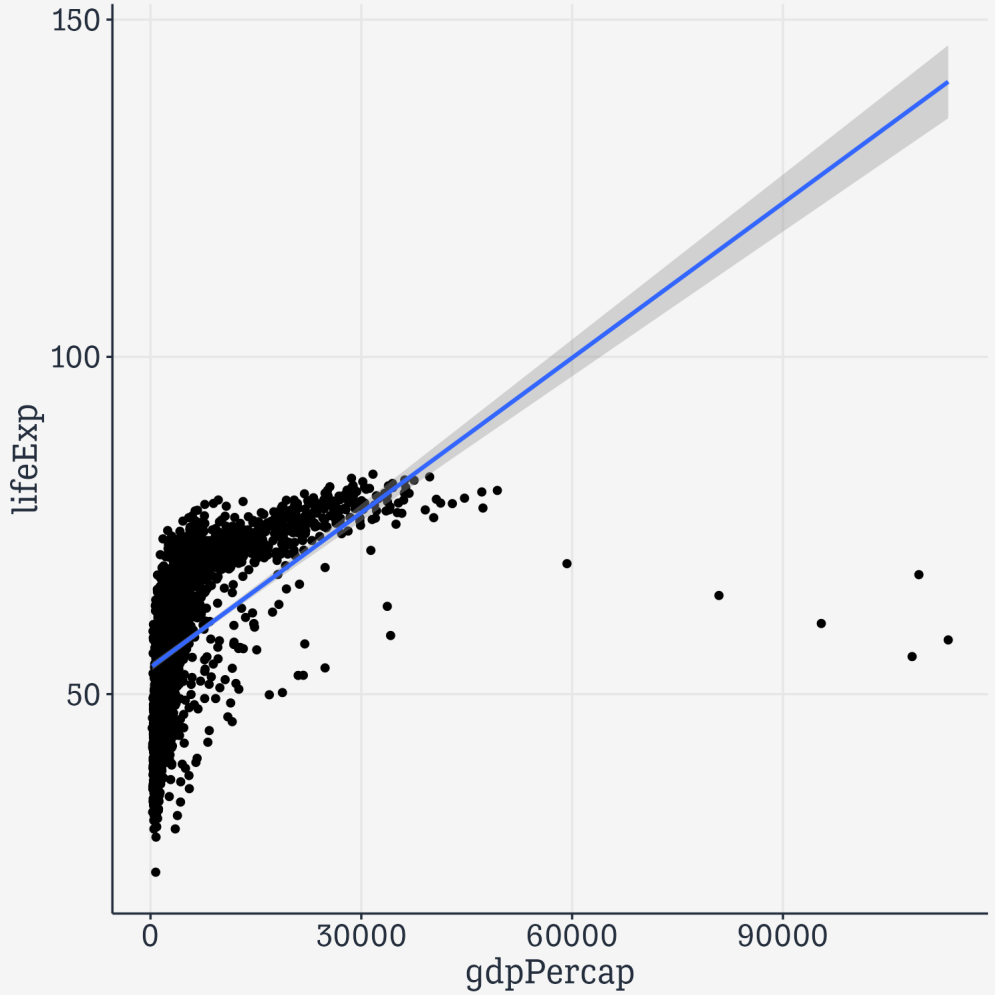
Keep Layering

```
p ← ggplot(data = gapminder,
           mapping = aes(x = gdpPercap,
                         y=lifeExp))
p + geom_point()
```



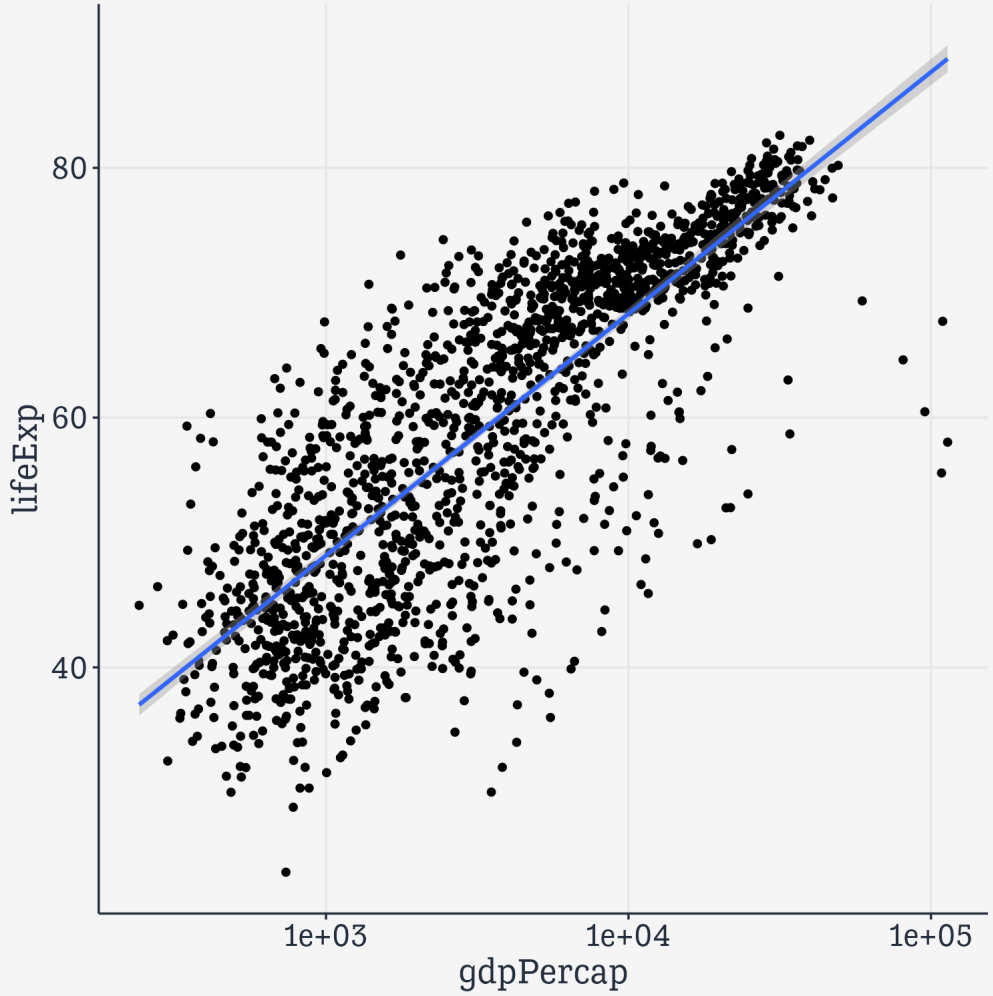
Keep Layering

```
 p ← ggplot(data = gapminder,
            mapping = aes(x = gdpPercap,
                          y=lifeExp))
p + geom_point() +
    geom_smooth(method = "lm")
```



Keep Layering

```
 p ← ggplot(data = gapminder,
            mapping = aes(x = gdpPercap,
                          y=lifeExp))
p + geom_point() +
    geom_smooth(method = "lm") +
    scale_x_log10()
```

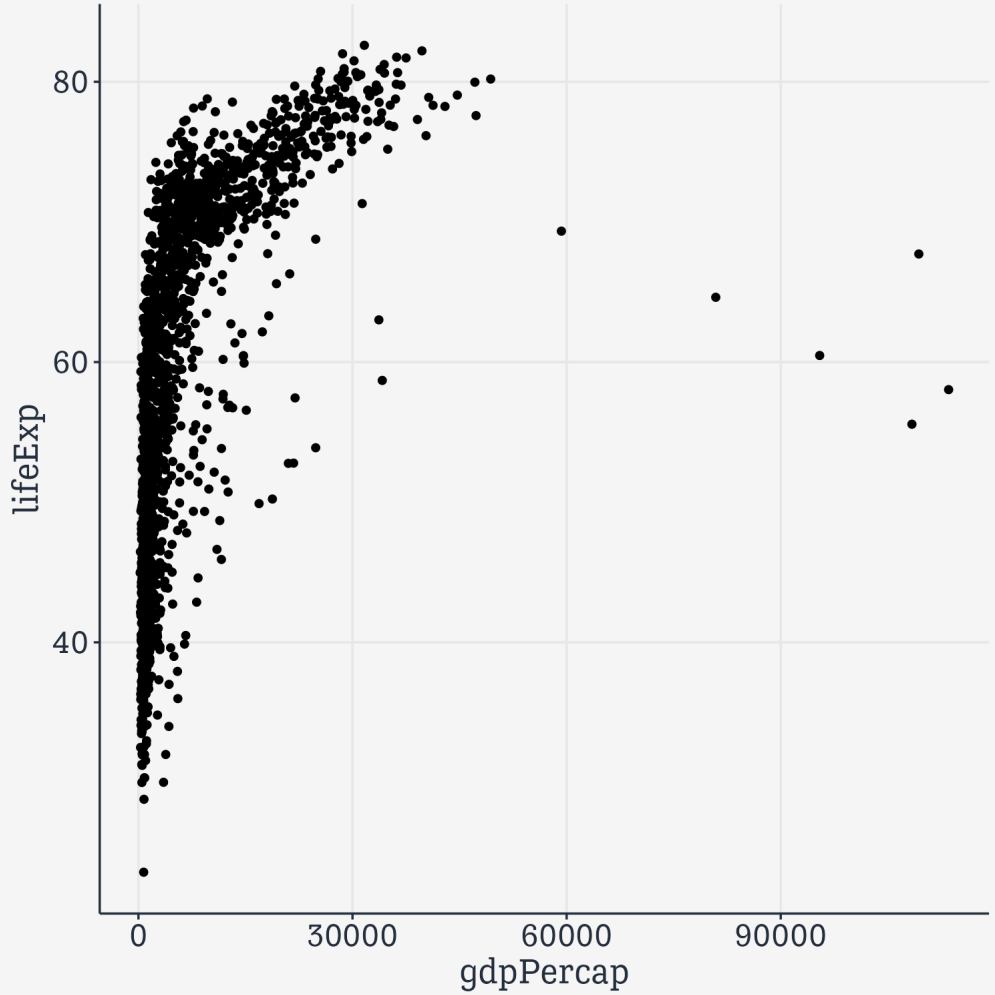


Fix the labels

```
p ← ggplot(data = gapminder,
           mapping = aes(x = gdpPercap,
                         y=lifeExp))
```

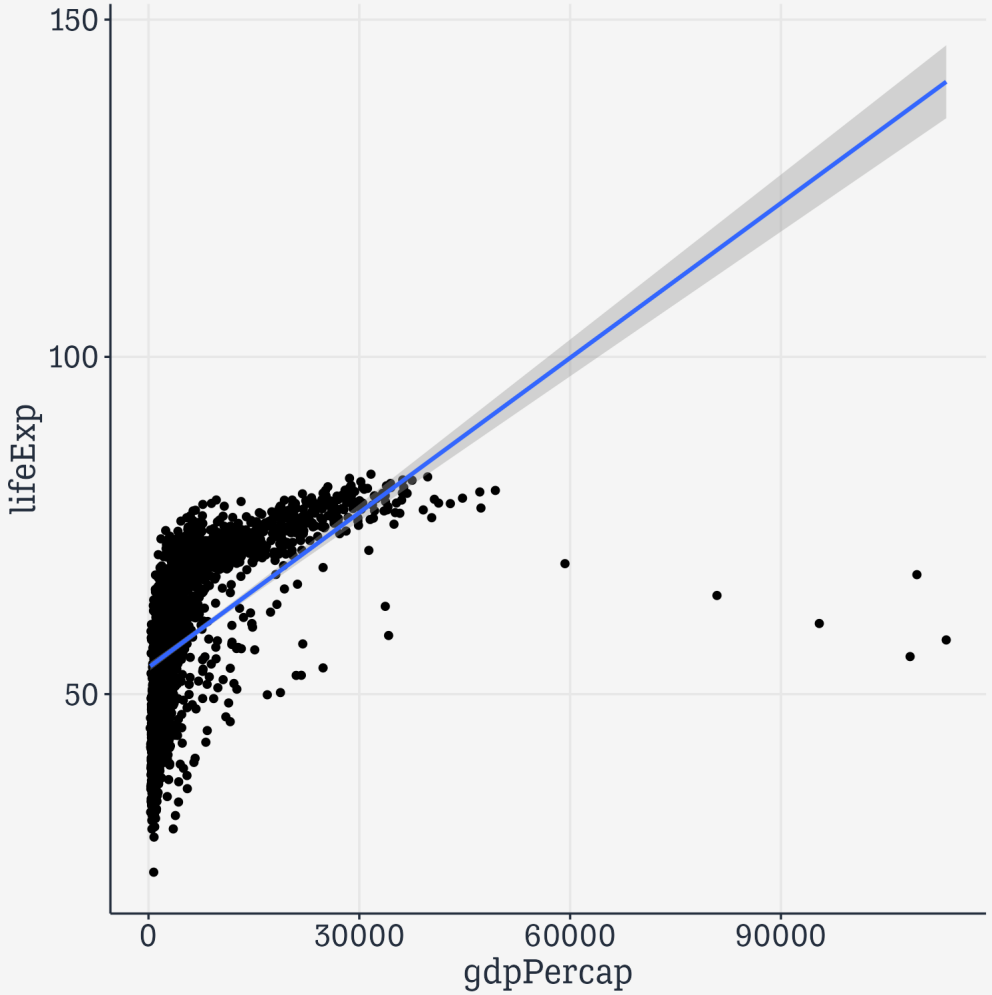
Fix the labels

```
p ← ggplot(data = gapminder,
           mapping = aes(x = gdpPercap,
                         y=lifeExp))
p + geom_point()
```



Fix the labels

```
p ← ggplot(data = gapminder,
           mapping = aes(x = gdpPercap,
                         y=lifeExp))
p + geom_point() +
    geom_smooth(method = "lm")
```



Fix the labels

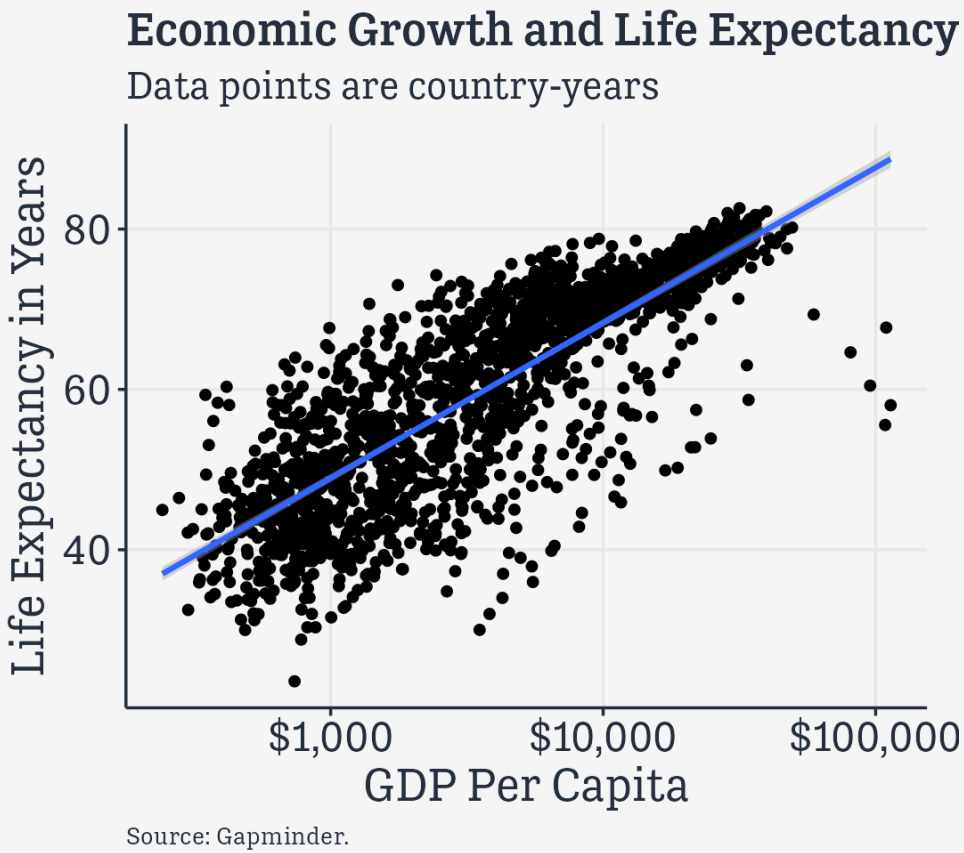
```
p ← ggplot(data = gapminder,
           mapping = aes(x = gdpPercap,
                         y=lifeExp))
p + geom_point() +
    geom_smooth(method = "lm") +
    scale_x_log10(labels = scales::label_dollar())
```

Add labels, title, and caption

```
p ← ggplot(data = gapminder,
           mapping = aes(x = gdpPercap,
                         y = lifeExp))
p + geom_point() +
  geom_smooth(method = "lm") +
    scale_x_log10(labels = scales::label_dollar()) +
    labs(x = "GDP Per Capita",
         y = "Life Expectancy in Years",
         title = "Economic Growth and Life Expectancy",
         subtitle = "Data points are country-years",
         caption = "Source: Gapminder.")
```

Add labels, title, and caption

```
p ← ggplot(data = gapminder,
           mapping = aes(x = gdpPercap,
                         y = lifeExp))
p + geom_point() +
  geom_smooth(method = "lm") +
    scale_x_log10(labels = scales::label_dollar()) +
    labs(x = "GDP Per Capita",
         y = "Life Expectancy in Years",
         title = "Economic Growth and Life Expectancy",
         subtitle = "Data points are country-years",
         caption = "Source: Gapminder.")
```



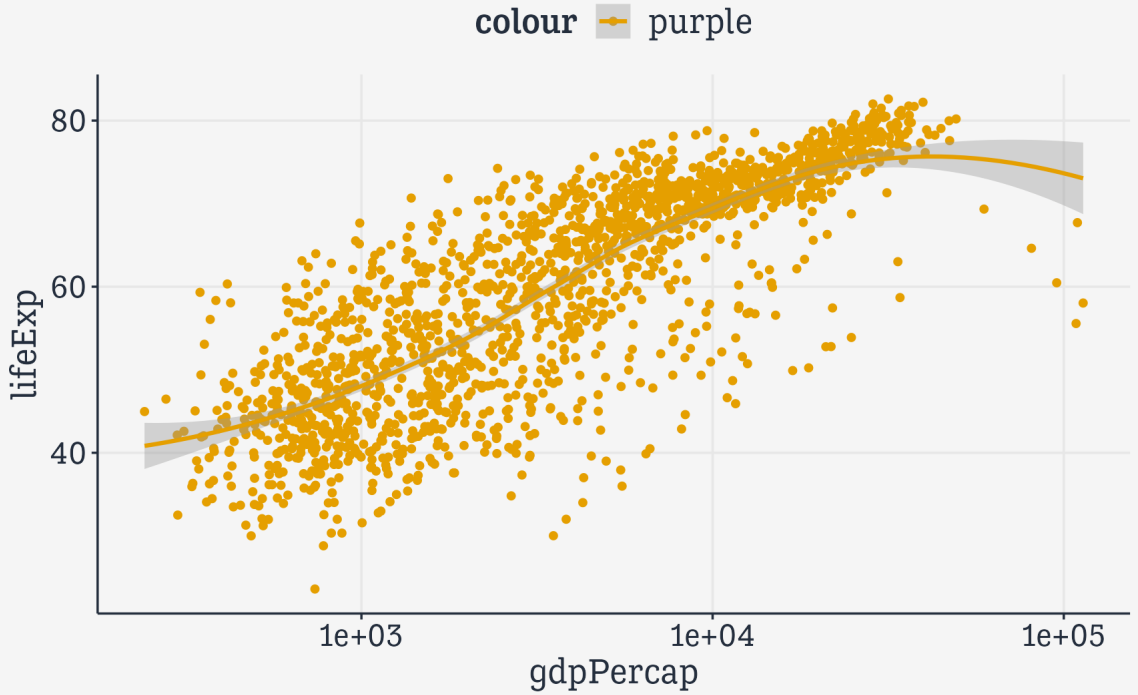
Mapping vs Setting your plot's aesthetics

"Can I change the color of the points?"

```
p ← ggplot(data = gapminder,
           mapping = aes(x = gdpPercap,
                         y = lifeExp,
                         color = "purple"))

## Put in an object for convenience
p_out ← p + geom_point() +
    geom_smooth(method = "loess") +
    scale_x_log10()
```

What has gone wrong here?

```
p_out
```

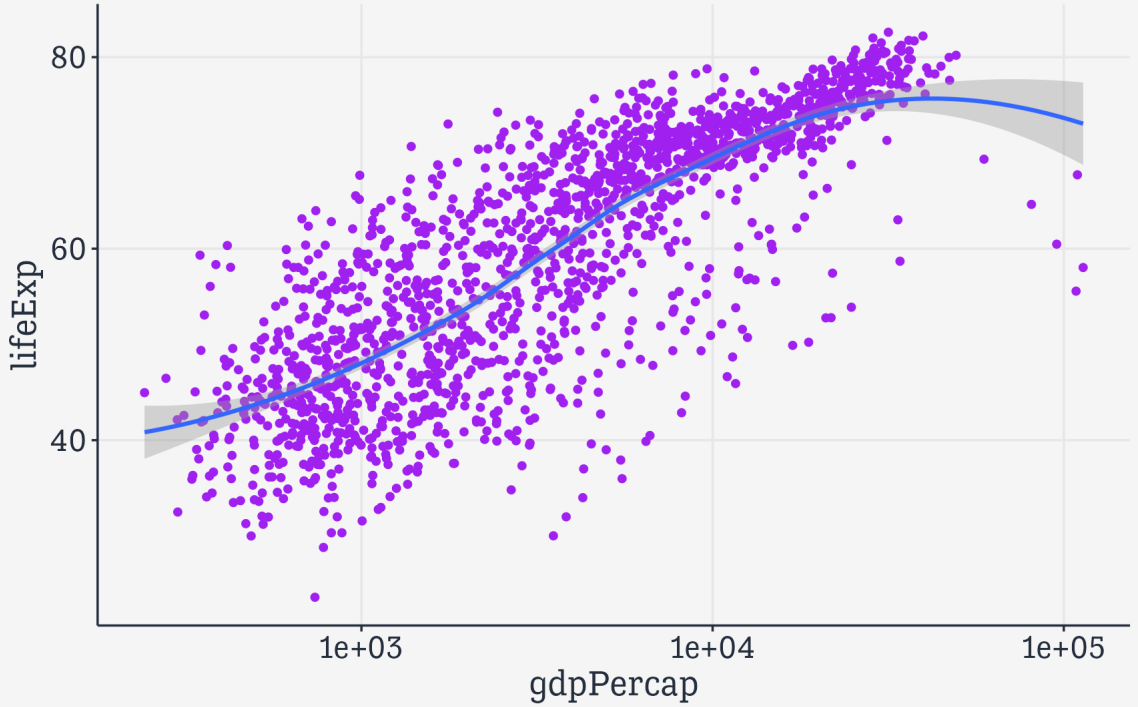


Try again

```
p ← ggplot(data = gapminder,
           mapping = aes(x = gdpPercap,
                         y = lifeExp))

## Put in an object for convenience
p_out ← p + geom_point(color = "purple") +
    geom_smooth(method = "loess") +
    scale_x_log10()
```

Try again

```
p_out
```



Geoms can take many arguments

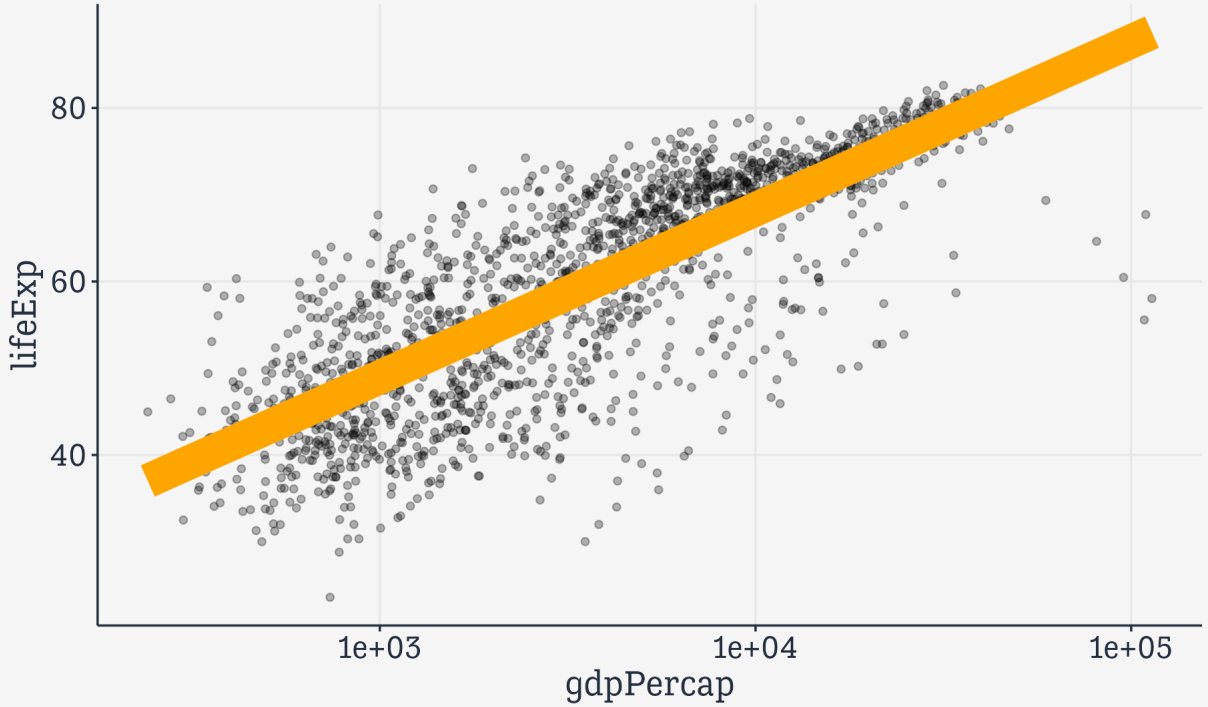
Here we set `color`, `size`, and `alpha`. Meanwhile `x` and `y` are mapped.

We also give non-default values to some other arguments

```
p ← ggplot(data = gapminder,
           mapping = aes(x = gdpPercap,
                         y = lifeExp))
p_out ← p + geom_point(alpha = 0.3) +
    geom_smooth(color = "orange",
                se = FALSE,
                linewidth = 8,
                method = "lm") +
    scale_x_log10()
```

Geoms can take many arguments

```
p_out
```

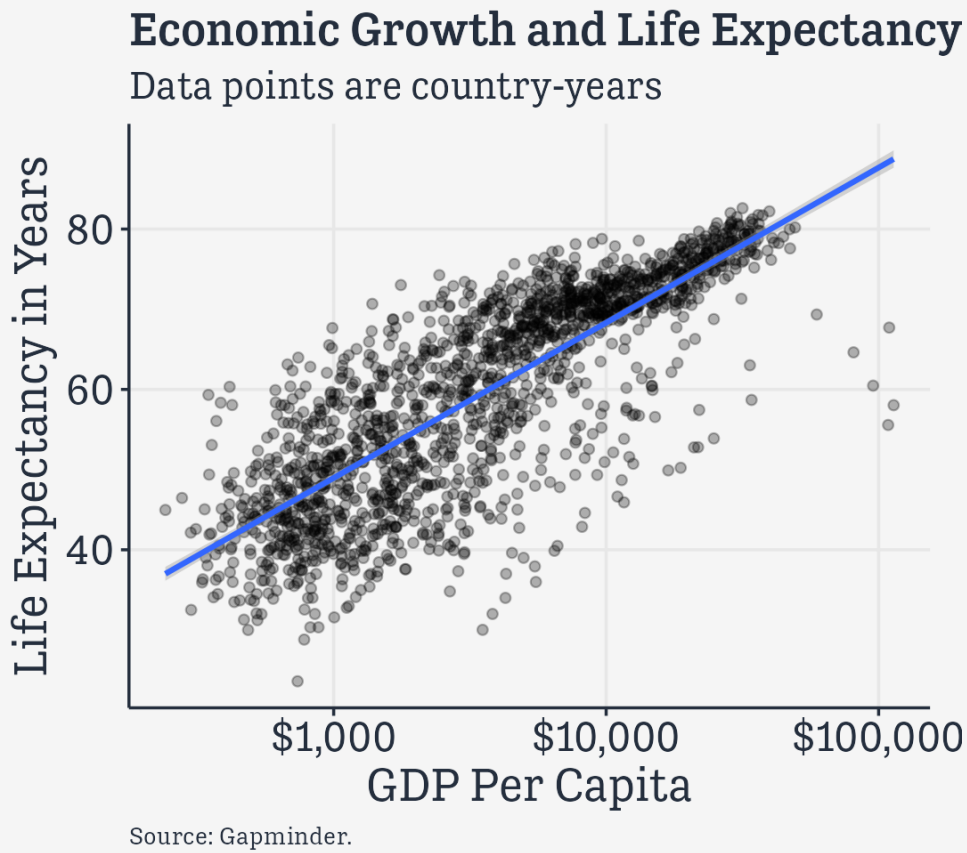


Setting `alpha` is handy for overplotted data

```
p ← ggplot(data = gapminder,
           mapping = aes(x = gdpPercap,
                         y = lifeExp))
p + geom_point(alpha = 0.3) +
  geom_smooth(method = "lm") +
    scale_x_log10(labels = scales::label_dollar()) +
    labs(x = "GDP Per Capita",
         y = "Life Expectancy in Years",
         title = "Economic Growth and Life Expectancy",
         subtitle = "Data points are country-years",
         caption = "Source: Gapminder.")
```

Setting `alpha` is handy for overplotted data

```
p ← ggplot(data = gapminder,
           mapping = aes(x = gdpPercap,
                         y = lifeExp))
p + geom_point(alpha = 0.3) +
  geom_smooth(method = "lm") +
    scale_x_log10(labels = scales::label_dollar()) +
    labs(x = "GDP Per Capita",
         y = "Life Expectancy in Years",
         title = "Economic Growth and Life Expectancy",
         subtitle = "Data points are country-years",
         caption = "Source: Gapminder.")
```



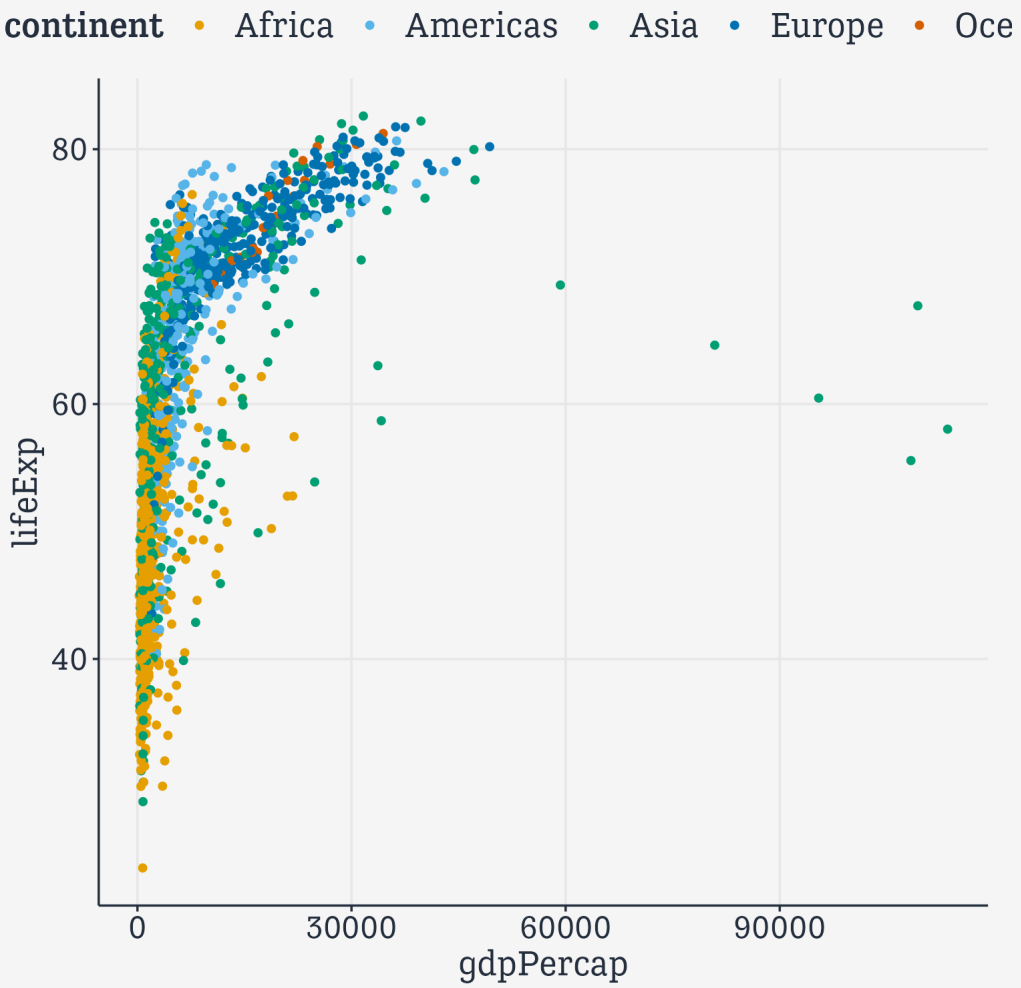
Map or Set values per geom

Geoms can take their own mappings

```
p ← ggplot(data = gapminder,
           mapping = aes(x = gdpPercap,
                         y = lifeExp,
                         color = continent,
                         fill = continent))
```

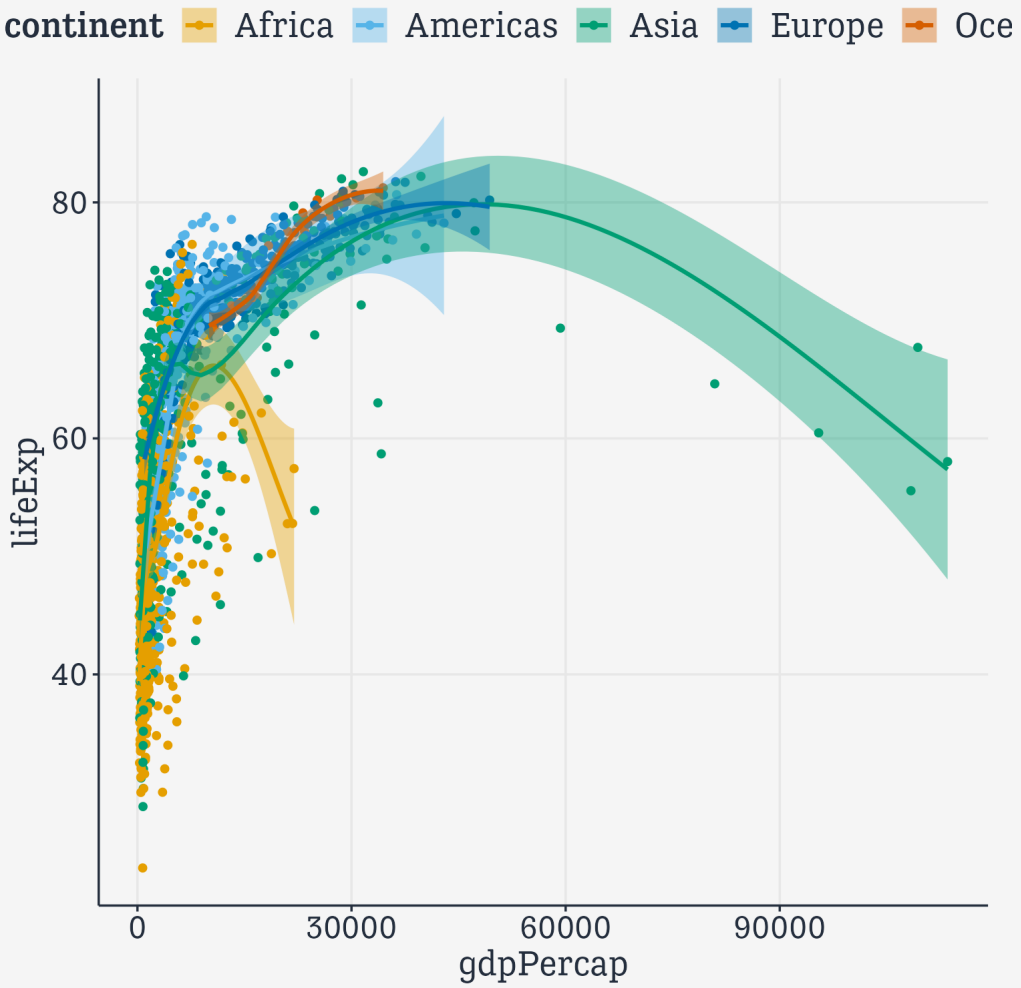
Geoms can take their own mappings

```
p ← ggplot(data = gapminder,
           mapping = aes(x = gdpPercap,
                         y = lifeExp,
                         color = continent,
                         fill = continent))
p + geom_point()
```



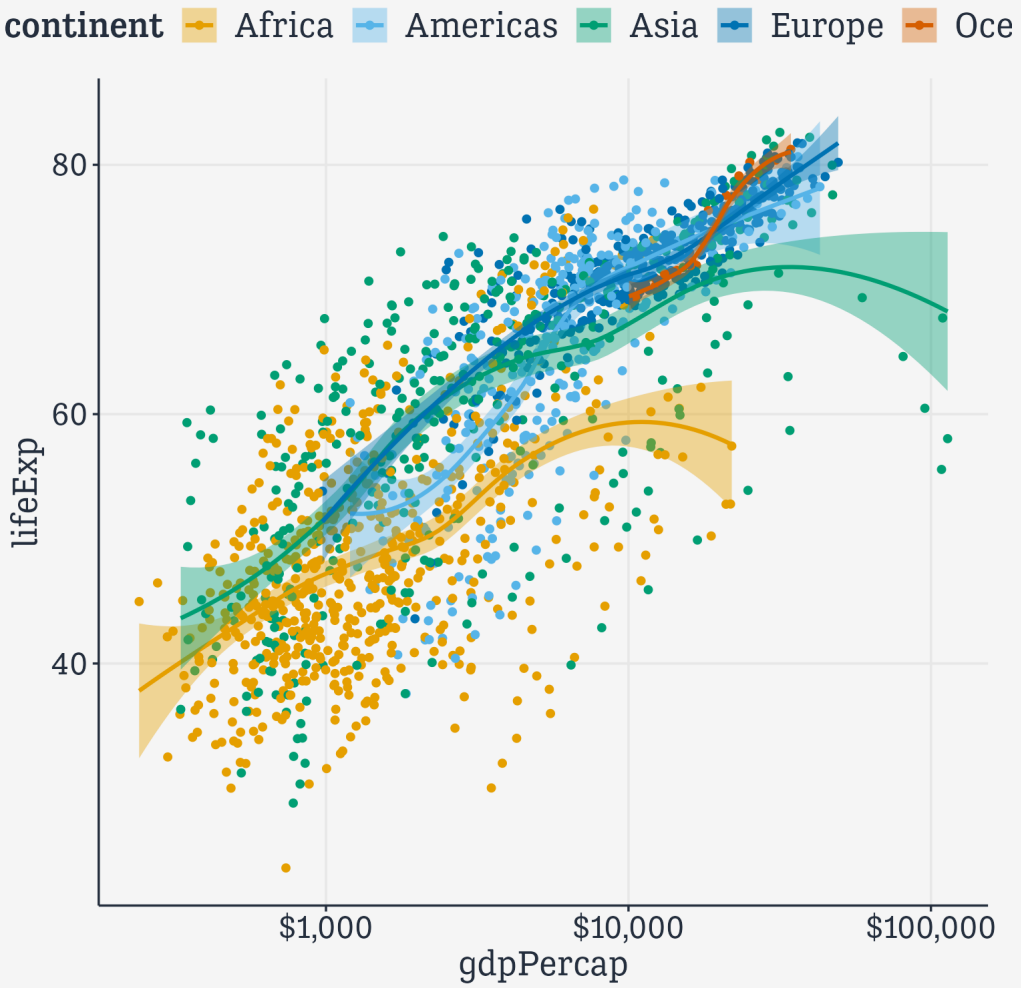
Geoms can take their own mappings

```
p ← ggplot(data = gapminder,
           mapping = aes(x = gdpPercap,
                         y = lifeExp,
                         color = continent,
                         fill = continent))
p + geom_point() +
    geom_smooth(method = "loess")
```



Geoms can take their own mappings

```
p ← ggplot(data = gapminder,
           mapping = aes(x = gdpPercap,
                         y = lifeExp,
                         color = continent,
                         fill = continent))
p + geom_point() +
    geom_smooth(method = "loess") +
    scale_x_log10(labels = scales::label_dollar())
```

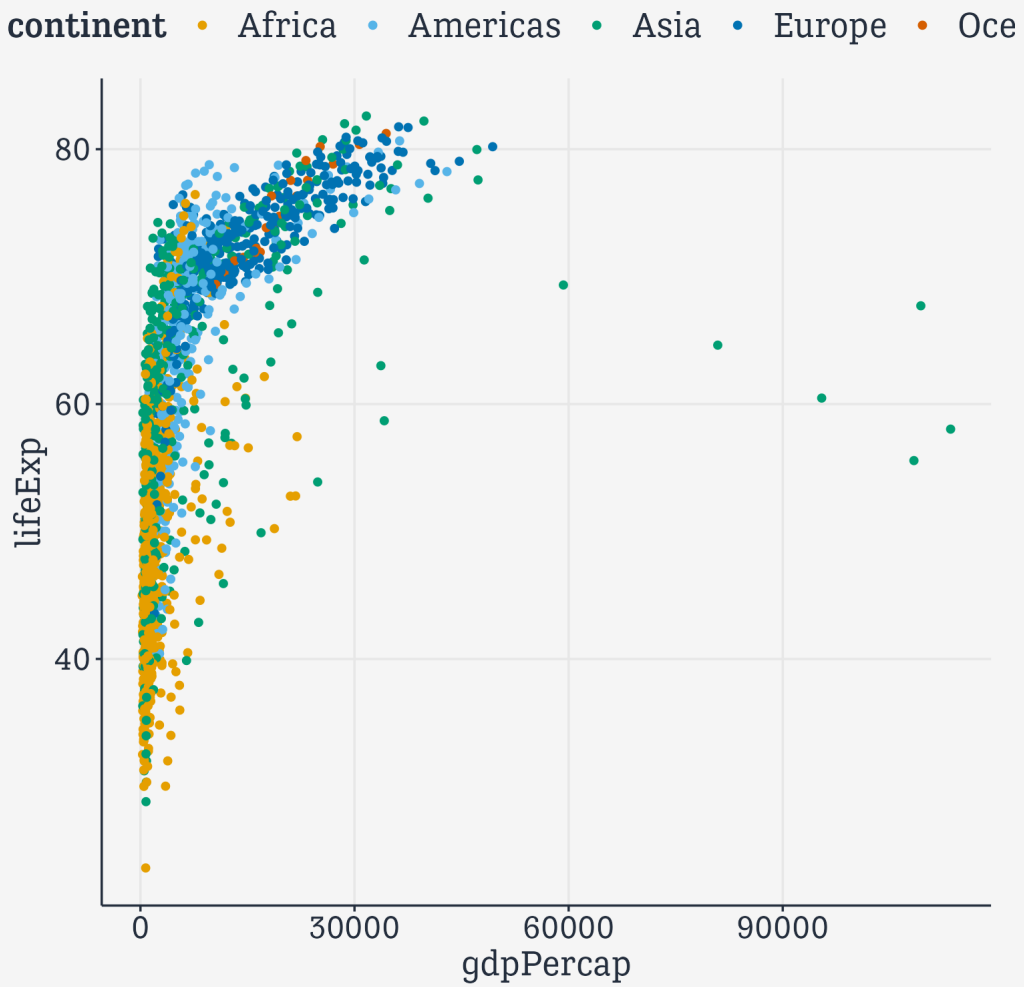


Geoms can take their own mappings

```
p ← ggplot(data = gapminder,
           mapping = aes(x = gdpPercap,
                         y = lifeExp))
```

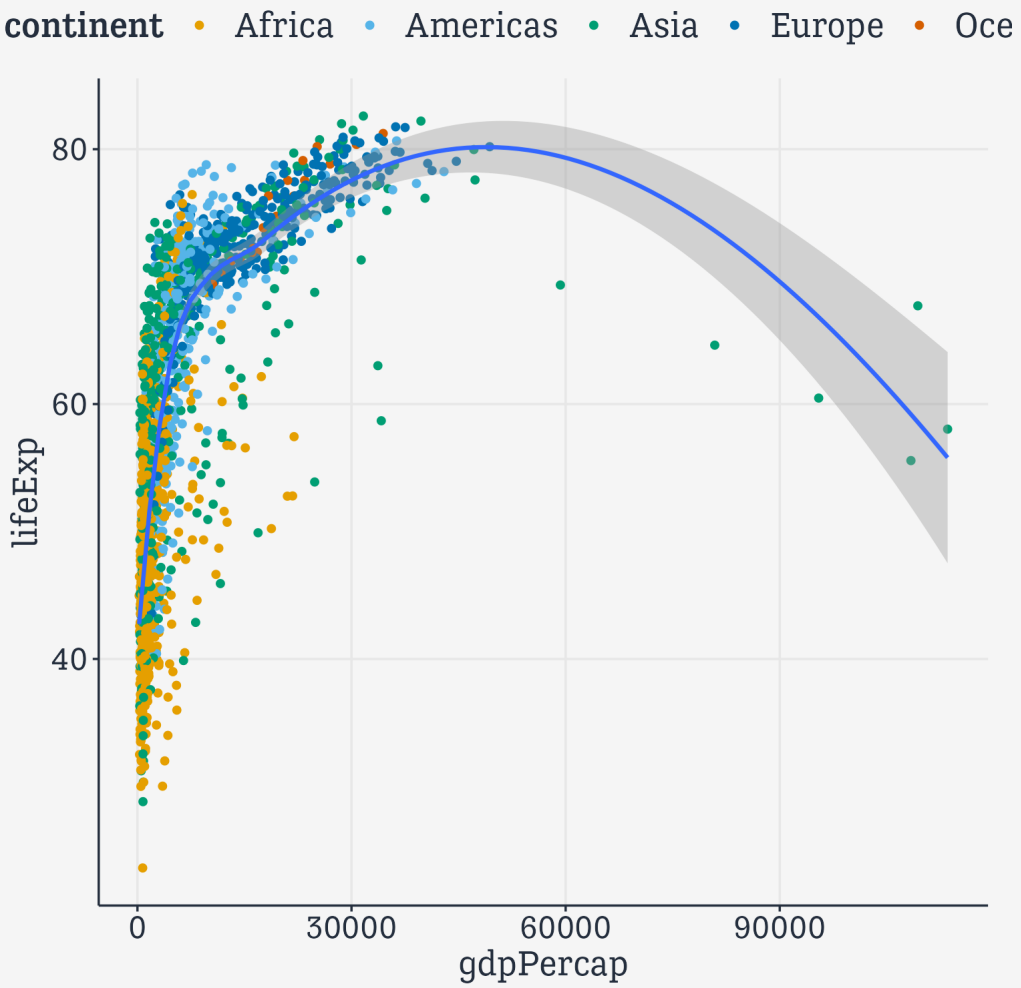
Geoms can take their own mappings

```
p ← ggplot(data = gapminder,
           mapping = aes(x = gdpPercap,
                         y = lifeExp))
p + geom_point(mapping = aes(color = continent))
```



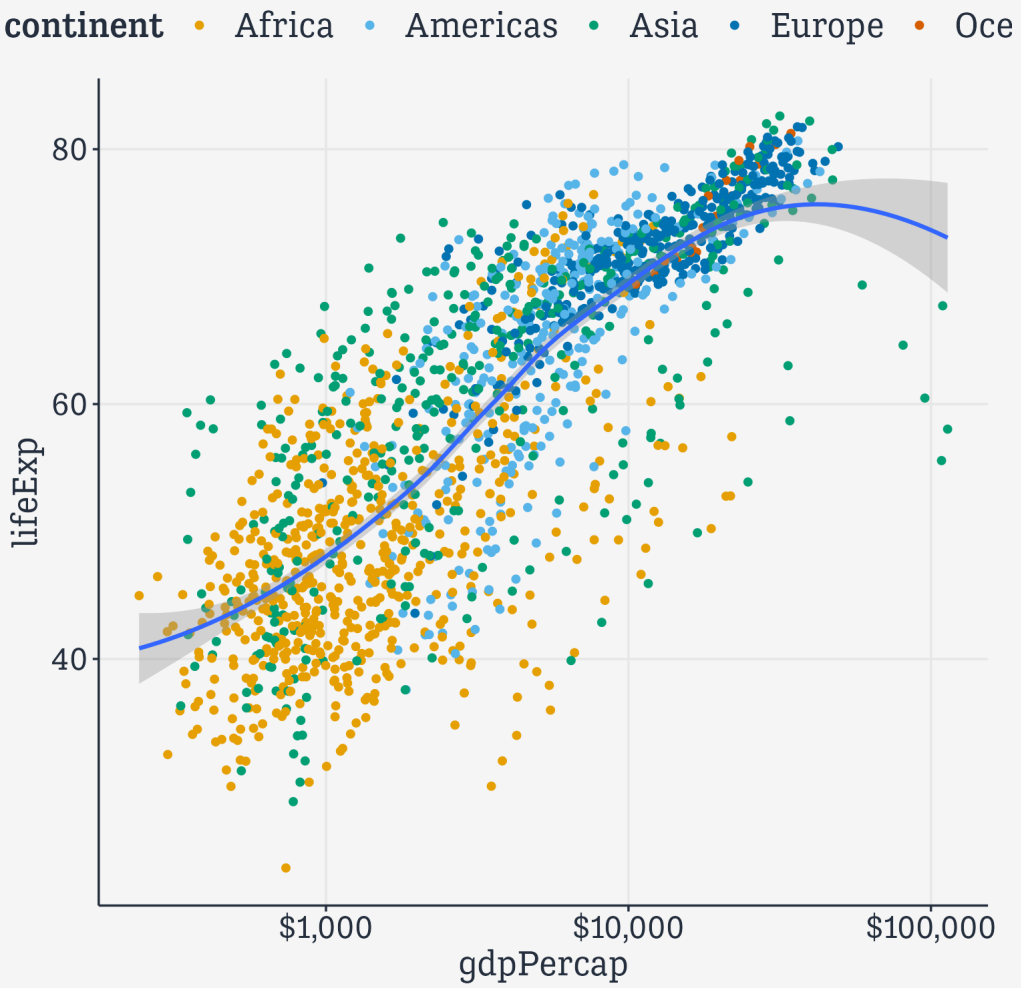
Geoms can take their own mappings

```
p ← ggplot(data = gapminder,
           mapping = aes(x = gdpPercap,
                         y = lifeExp))
p + geom_point(mapping = aes(color = continent)) +
    geom_smooth(method = "loess")
```



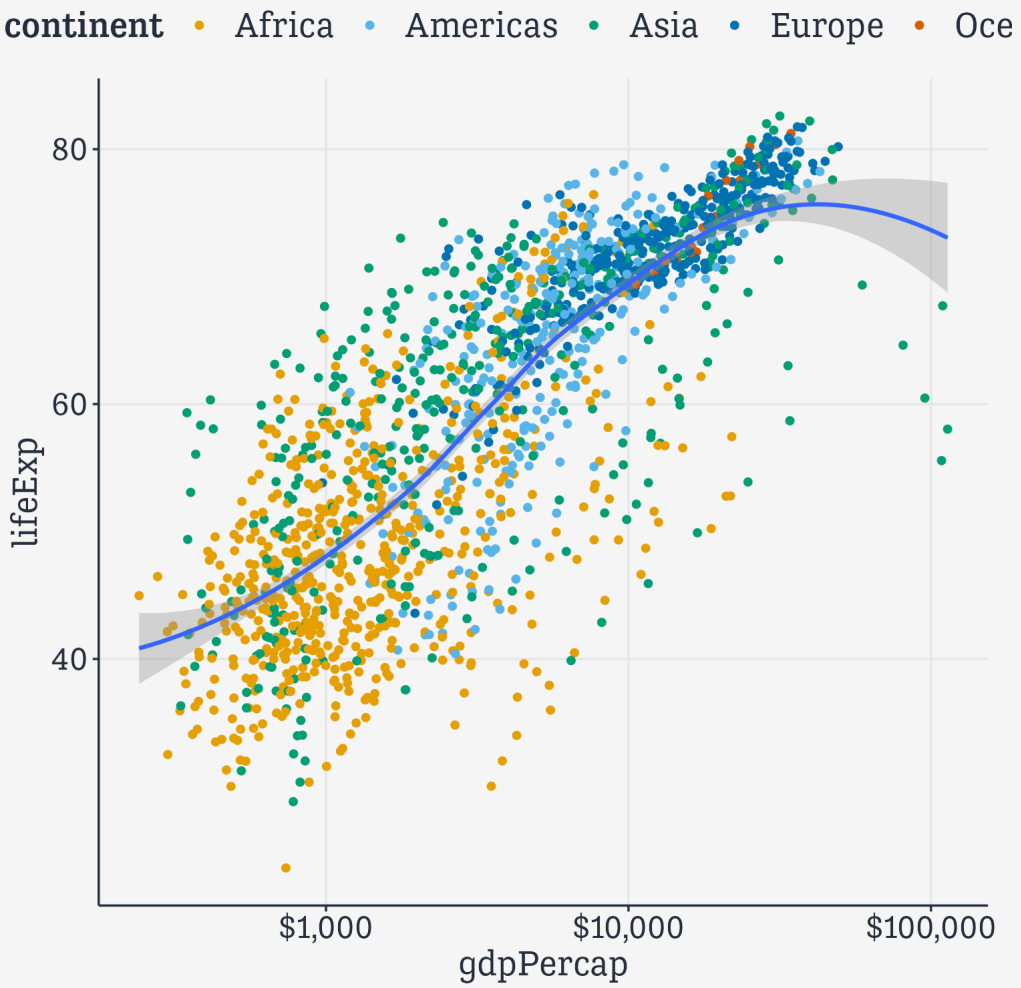
Geoms can take their own mappings

```
p ← ggplot(data = gapminder,
           mapping = aes(x = gdpPercap,
                         y = lifeExp))
p + geom_point(mapping = aes(color = continent)) +
    geom_smooth(method = "loess") +
    scale_x_log10(labels = scales::label_dollar())
```



Geoms can take their own mappings

```
p ← ggplot(data = gapminder,
           mapping = aes(x = gdpPercap,
                         y = lifeExp))
p + geom_point(mapping = aes(color = continent)) +
    geom_smooth(method = "loess") +
    scale_x_log10(labels = scales::label_dollar())
```



Pay attention to which scales and guides are drawn, and why

Guides and scales reflect `aes()` mappings

```
mapping = aes(color =
continent, fill = continent)
```

Guides and scales reflect `aes()` mappings

```
mapping = aes(color =
continent, fill = continent)
```


```
mapping = aes(color =
continent)
```

Remember: Every mapped variable has a scale

ggplot's FLOW OF ACTION

1. Tidy Data

gdp	lifexp	pop	continent
340	65	31	Euro
227	51	200	Amer
909	81	80	Euro
126	40	20	Asia

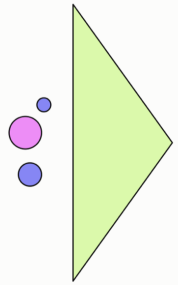
ggplot(data = gapminder,

2. Mapping

x=gdp
y=lifexp
color=continent
size=pop

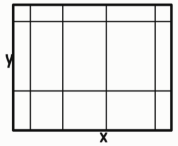
mapping =
aes(x = gdp,
y = lifespan,
color = continent,
size = pop))

3. Geom



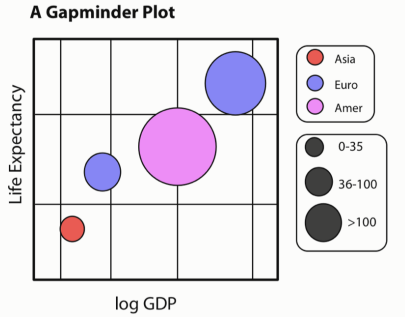
geom_point()

4. Co-Ordinates, Scales



coord_cartesian()
scale_x_log10()

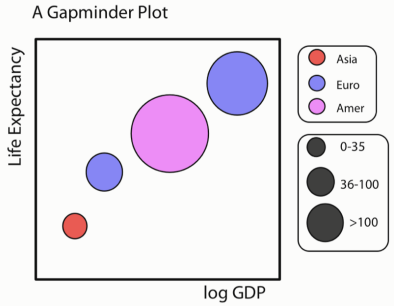
5. Labels & Guides



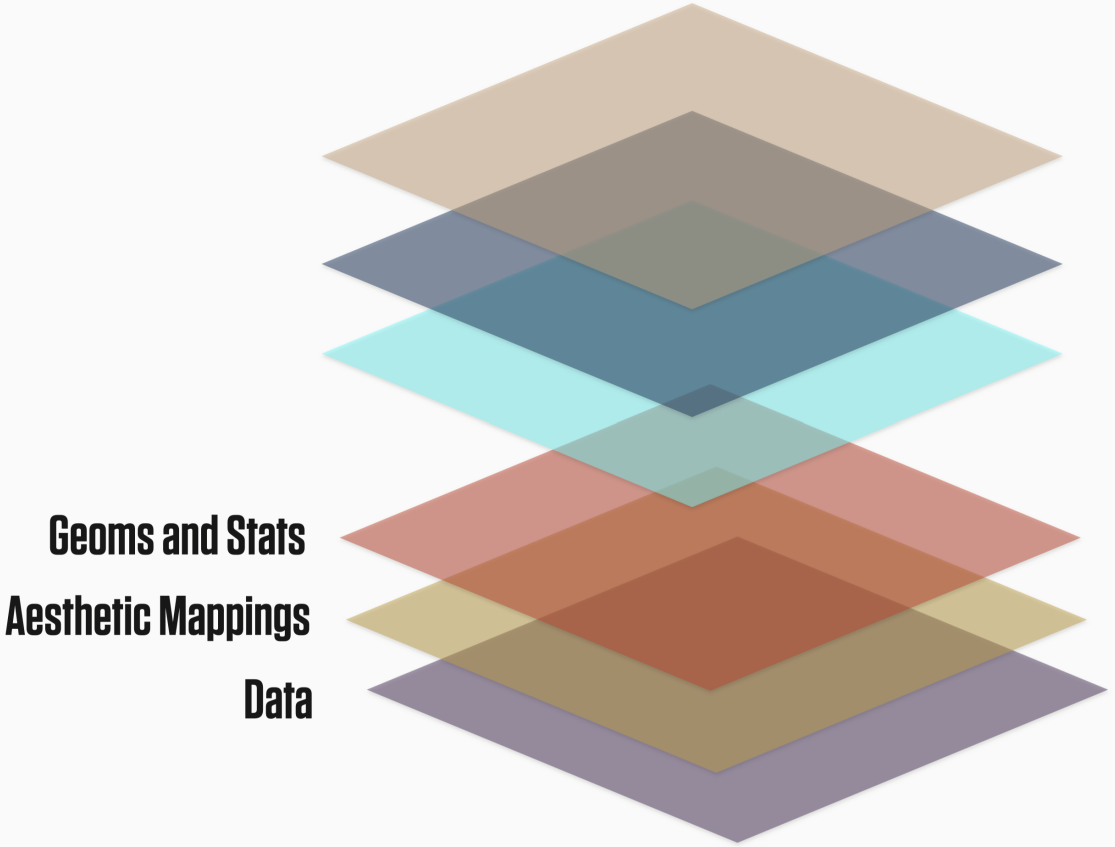
labs()

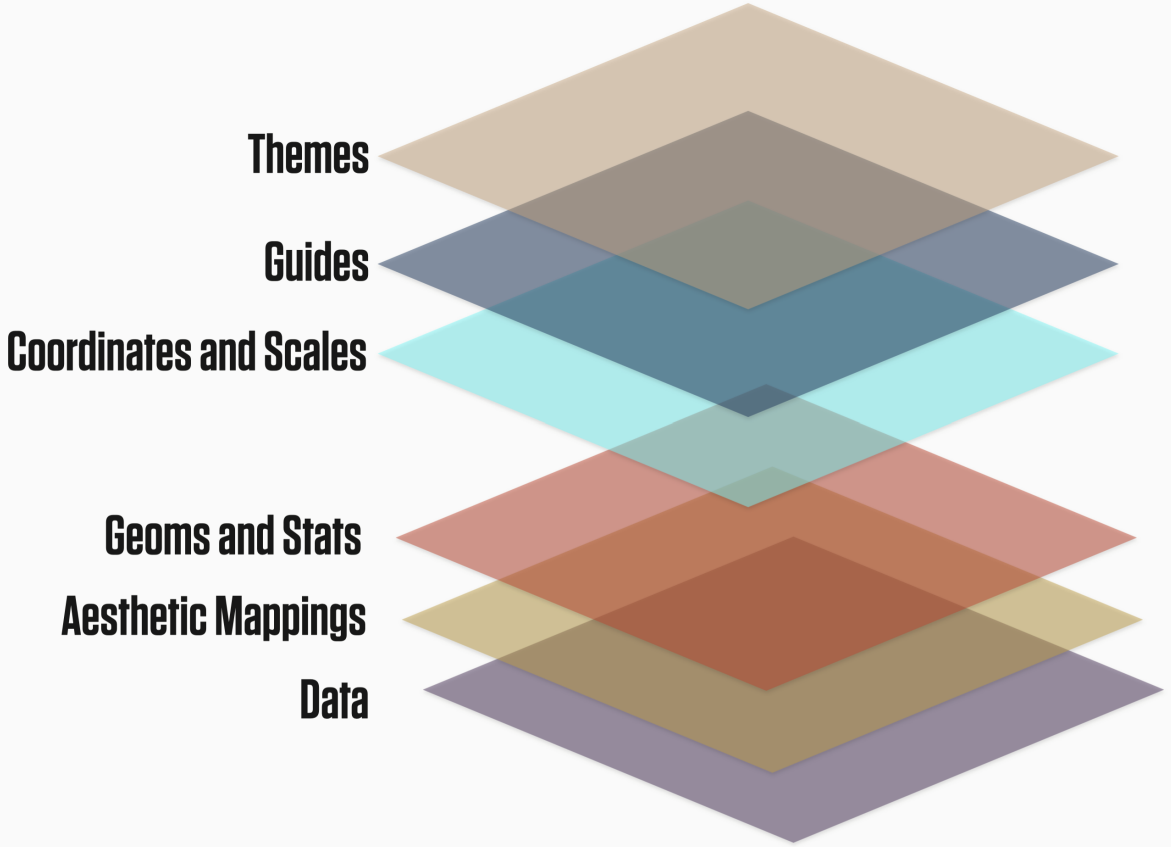
guides()

6. Themes



theme_minimal()


Geoms and Stats
Aesthetic Mappings
Data


Themes
Guides
Coordinates and Scales
Geoms and Stats
Aesthetic Mappings
Data

Feeding data to ggplot

Transform and summarize first. Then send your clean tables to ggplot.

Summarizing a Table

Here's what we're going to do:

1. Individual-Level GSS Data on Region and Religion

id	bigregion	religion
1014	Midwest	Protestant
1544	South	Protestant
665	Northeast	None
1618	South	None
2115	West	Catholic
417	South	Protestant
2045	West	Protestant
1863	Northeast	Other
1884	Midwest	Christian
1628	South	Protestant

2. Summary Count of Religious Preferences by Census Region

bigregion	religion	N
Northeast	Protestant	123
Northeast	Catholic	149
Northeast	Jewish	15
Northeast	None	97
Northeast	Christian	14
Northeast	Other	31

3. Percent Religious Preferences by Census Region

bigregion	religion	N	pct
Northeast	Protestant	123	28.3
Northeast	Catholic	149	34.3
Northeast	Jewish	15	3.4
Northeast	None	97	22.3
Northeast	Christian	14	3.2
Northeast	Other	31	7.1

Summarizing a Table

We're just taking a look at the relevant columns here. We don't need to narrow it like this to do our summary, though.

```
gss_sm ▷
  select(id, bigregion, religion)
```

```
## # A tibble: 2,867 × 3
##       id bigregion religion
##    <dbl> <fct>     <fct>
##  1     1 Northeast None
##  2     2 Northeast None
##  3     3 Northeast Catholic
##  4     4 Northeast Catholic
##  5     5 Northeast None
##  6     6 Northeast None
##  7     7 Northeast None
##  8     8 Northeast Catholic
##  9     9 Northeast Protestant
## 10    10 Northeast None
## # ℹ 2,857 more rows
```

Count up by *one* column or variable

```
gss_sm
```

```
## # A tibble: 2,867 × 32
##     year    id ballot       age childs sibs   degree race  sex   region income16
##    <dbl> <dbl> <labelled> <dbl>  <dbl> <labe> <fct>  <fct> <fct> <fct>  <fct>
##  1  2016     1 1             47      3 2      Bache… White Male  New E… $170000…
##  2  2016     2 2             61      0 3      High … White Male  New E… $50000 …
##  3  2016     3 3             72      2 3      Bache… White Male  New E… $75000 …
##  4  2016     4 1             43      4 3      High … White Fema… New E… $170000…
##  5  2016     5 3             55      2 2      Gradu… White Fema… New E… $170000…
##  6  2016     6 2             53      2 2      Junio… White Fema… New E… $60000 …
##  7  2016     7 1             50      2 2      High … White Male  New E… $170000…
##  8  2016     8 3             23      3 6      High … Other Fema… Middl… $30000 …
##  9  2016     9 1             45      3 5      High … Black Male  Middl… $60000 …
## 10  2016    10 3             71      4 1      Junio… White Male  Middl… $60000 …
## # ℹ 2,857 more rows
## # ℹ 21 more variables: relig <fct>, marital <fct>, padeg <fct>, madeg <fct>,
## #   partyid <fct>, polviews <fct>, happy <fct>, partners <fct>, grass <fct>,
## #   zodiac <fct>, pres12 <labelled>, wtssall <dbl>, income_rc <fct>,
## #   agegrp <fct>, ageq <fct>, siblings <fct>, kids <fct>, religion <fct>,
## #   bigregion <fct>, partners_rc <fct>, obama <dbl>
```

Count up by *one* column or variable

```
gss_sm ▷
  group_by(bigregion)   #<<
```

```
## # A tibble: 2,867 × 32
## # Groups:   bigregion [4]
##     year    id ballot       age childs sibs   degree race  sex   region income16
##    <dbl> <dbl> <labelled> <dbl>  <dbl> <labe> <fct>  <fct> <fct> <fct>  <fct>
##  1  2016     1 1             47      3 2      Bache… White Male  New E… $170000…
##  2  2016     2 2             61      0 3      High … White Male  New E… $50000 …
##  3  2016     3 3             72      2 3      Bache… White Male  New E… $75000 …
##  4  2016     4 1             43      4 3      High … White Fema… New E… $170000…
##  5  2016     5 3             55      2 2      Gradu… White Fema… New E… $170000…
##  6  2016     6 2             53      2 2      Junio… White Fema… New E… $60000 …
##  7  2016     7 1             50      2 2      High … White Male  New E… $170000…
##  8  2016     8 3             23      3 6      High … Other Fema… Middl… $30000 …
##  9  2016     9 1             45      3 5      High … Black Male  Middl… $60000 …
## 10  2016    10 3             71      4 1      Junio… White Male  Middl… $60000 …
## # ℹ 2,857 more rows
## # ℹ 21 more variables: relig <fct>, marital <fct>, padeg <fct>, madeg <fct>,
## #   partyid <fct>, polviews <fct>, happy <fct>, partners <fct>, grass <fct>,
## #   zodiac <fct>, pres12 <labelled>, wtssall <dbl>, income_rc <fct>,
## #   agegrp <fct>, ageq <fct>, siblings <fct>, kids <fct>, religion <fct>,
## #   bigregion <fct>, partners_rc <fct>, obama <dbl>
```

Count up by *one* column or variable

```
gss_sm ▷
  group_by(bigregion) ▷
  summarize(total = n())
```

```
## # A tibble: 4 × 2
##   bigregion total
##   <fct>     <int>
## 1 Northeast   488
## 2 Midwest     695
## 3 South      1052
## 4 West        632
```

Grouping changes the *logical* structure of the tibble. It tells you what subsets of rows the next mutate or summary operation will be carried out within.

Summarize by region and religion

```
gss_sm
```

```
## # A tibble: 2,867 × 32
##     year    id ballot       age childs sibs   degree race  sex   region income16
##    <dbl> <dbl> <labelled> <dbl>  <dbl> <labe> <fct>  <fct> <fct> <fct>  <fct>
##  1  2016     1 1             47      3 2      Bache… White Male  New E… $170000…
##  2  2016     2 2             61      0 3      High … White Male  New E… $50000 …
##  3  2016     3 3             72      2 3      Bache… White Male  New E… $75000 …
##  4  2016     4 1             43      4 3      High … White Fema… New E… $170000…
##  5  2016     5 3             55      2 2      Gradu… White Fema… New E… $170000…
##  6  2016     6 2             53      2 2      Junio… White Fema… New E… $60000 …
##  7  2016     7 1             50      2 2      High … White Male  New E… $170000…
##  8  2016     8 3             23      3 6      High … Other Fema… Middl… $30000 …
##  9  2016     9 1             45      3 5      High … Black Male  Middl… $60000 …
## 10  2016    10 3             71      4 1      Junio… White Male  Middl… $60000 …
## # ℹ 2,857 more rows
## # ℹ 21 more variables: relig <fct>, marital <fct>, padeg <fct>, madeg <fct>,
## #   partyid <fct>, polviews <fct>, happy <fct>, partners <fct>, grass <fct>,
## #   zodiac <fct>, pres12 <labelled>, wtssall <dbl>, income_rc <fct>,
## #   agegrp <fct>, ageq <fct>, siblings <fct>, kids <fct>, religion <fct>,
## #   bigregion <fct>, partners_rc <fct>, obama <dbl>
```

Summarize by region and religion

```
gss_sm ▷
  group_by(bigregion, religion)
```

```
## # A tibble: 2,867 × 32
## # Groups:   bigregion, religion [24]
##     year    id ballot       age childs sibs   degree race  sex   region income16
##    <dbl> <dbl> <labelled> <dbl>  <dbl> <labe> <fct>  <fct> <fct> <fct>  <fct>
##  1  2016     1 1             47      3 2      Bache… White Male  New E… $170000…
##  2  2016     2 2             61      0 3      High … White Male  New E… $50000 …
##  3  2016     3 3             72      2 3      Bache… White Male  New E… $75000 …
##  4  2016     4 1             43      4 3      High … White Fema… New E… $170000…
##  5  2016     5 3             55      2 2      Gradu… White Fema… New E… $170000…
##  6  2016     6 2             53      2 2      Junio… White Fema… New E… $60000 …
##  7  2016     7 1             50      2 2      High … White Male  New E… $170000…
##  8  2016     8 3             23      3 6      High … Other Fema… Middl… $30000 …
##  9  2016     9 1             45      3 5      High … Black Male  Middl… $60000 …
## 10  2016    10 3             71      4 1      Junio… White Male  Middl… $60000 …
## # ℹ 2,857 more rows
## # ℹ 21 more variables: relig <fct>, marital <fct>, padeg <fct>, madeg <fct>,
## #   partyid <fct>, polviews <fct>, happy <fct>, partners <fct>, grass <fct>,
## #   zodiac <fct>, pres12 <labelled>, wtssall <dbl>, income_rc <fct>,
## #   agegrp <fct>, ageq <fct>, siblings <fct>, kids <fct>, religion <fct>,
## #   bigregion <fct>, partners_rc <fct>, obama <dbl>
```

Summarize by region and religion

```
gss_sm ▷
  group_by(bigregion, religion) ▷
  summarize(total = n())
```

```
## # A tibble: 24 × 3
## # Groups:   bigregion [4]
##    bigregion religion   total
##    <fct>     <fct>      <int>
##  1 Northeast Protestant   158
##  2 Northeast Catholic     162
##  3 Northeast Jewish        27
##  4 Northeast None         112
##  5 Northeast Other         28
##  6 Northeast <NA>           1
##  7 Midwest   Protestant   325
##  8 Midwest   Catholic     172
##  9 Midwest   Jewish         3
## 10 Midwest   None         157
## # ℹ 14 more rows
```

Summarize by region and religion

```
gss_sm ▷
  group_by(bigregion, religion) ▷
  summarize(total = n()) ▷
  mutate(freq = total / sum(total),
           pct = round((freq*100), 1))
```

```
## # A tibble: 24 × 5
## # Groups:   bigregion [4]
##    bigregion religion   total    freq   pct
##    <fct>     <fct>      <int>   <dbl> <dbl>
##  1 Northeast Protestant   158 0.324    32.4
##  2 Northeast Catholic     162 0.332    33.2
##  3 Northeast Jewish        27 0.0553    5.5
##  4 Northeast None         112 0.230    23
##  5 Northeast Other         28 0.0574    5.7
##  6 Northeast <NA>           1 0.00205   0.2
##  7 Midwest   Protestant   325 0.468    46.8
##  8 Midwest   Catholic     172 0.247    24.7
##  9 Midwest   Jewish         3 0.00432   0.4
## 10 Midwest   None         157 0.226    22.6
## # ℹ 14 more rows
```

Summarize by region and religion

```
gss_sm ▷
  group_by(bigregion, religion) ▷
  summarize(total = n()) ▷
  mutate(freq = total / sum(total),
           pct = round((freq*100), 1))
```

```
## # A tibble: 24 × 5
## # Groups:   bigregion [4]
##    bigregion religion   total    freq   pct
##    <fct>     <fct>      <int>   <dbl> <dbl>
##  1 Northeast Protestant   158 0.324    32.4
##  2 Northeast Catholic     162 0.332    33.2
##  3 Northeast Jewish        27 0.0553    5.5
##  4 Northeast None         112 0.230    23
##  5 Northeast Other         28 0.0574    5.7
##  6 Northeast <NA>           1 0.00205   0.2
##  7 Midwest   Protestant   325 0.468    46.8
##  8 Midwest   Catholic     172 0.247    24.7
##  9 Midwest   Jewish         3 0.00432   0.4
## 10 Midwest   None         157 0.226    22.6
## # ℹ 14 more rows
```

Pipelines carry assumptions forward

```
gss_sm ▷
  group_by(bigregion, religion) ▷
  summarize(total = n()) ▷
  mutate(freq = total / sum(total),
           pct = round((freq*100), 1))
```

```
## # A tibble: 24 × 5
## # Groups:   bigregion [4]
##    bigregion religion   total    freq   pct
##    <fct>     <fct>      <int>   <dbl> <dbl>
##  1 Northeast Protestant   158 0.324    32.4
##  2 Northeast Catholic     162 0.332    33.2
##  3 Northeast Jewish        27 0.0553    5.5
##  4 Northeast None         112 0.230    23
##  5 Northeast Other         28 0.0574    5.7
##  6 Northeast <NA>           1 0.00205   0.2
##  7 Midwest   Protestant   325 0.468    46.8
##  8 Midwest   Catholic     172 0.247    24.7
##  9 Midwest   Jewish         3 0.00432   0.4
## 10 Midwest   None         157 0.226    22.6
## # ℹ 14 more rows
```

Groups are carried forward till summarized over, or explicitly ungrouped with `ungroup()`.

Pipelines carry assumptions forward

```
gss_sm ▷
  group_by(bigregion, religion) ▷
  summarize(total = n()) ▷
  mutate(freq = total / sum(total),
           pct = round((freq*100), 1))
```

```
## # A tibble: 24 × 5
## # Groups:   bigregion [4]
##    bigregion religion   total    freq   pct
##    <fct>     <fct>      <int>   <dbl> <dbl>
##  1 Northeast Protestant   158 0.324    32.4
##  2 Northeast Catholic     162 0.332    33.2
##  3 Northeast Jewish        27 0.0553    5.5
##  4 Northeast None         112 0.230    23
##  5 Northeast Other         28 0.0574    5.7
##  6 Northeast <NA>           1 0.00205   0.2
##  7 Midwest   Protestant   325 0.468    46.8
##  8 Midwest   Catholic     172 0.247    24.7
##  9 Midwest   Jewish         3 0.00432   0.4
## 10 Midwest   None         157 0.226    22.6
## # ℹ 14 more rows
```

Groups are carried forward till summarized over, or explicitly ungrouped with `ungroup()`.

Summary calculations are done on the innermost group, which then "disappears". (Notice how `religion` is no longer a group in the output, but `bigregion` is.)

Pipelines carry assumptions forward

```
gss_sm ▷
  group_by(bigregion, religion) ▷
  summarize(total = n()) ▷
  mutate(freq = total / sum(total),
         pct = round((freq*100), 1))
```

```
## # A tibble: 24 × 5
## # Groups:   bigregion [4]
##    bigregion religion   total    freq   pct
##    <fct>     <fct>      <int>   <dbl> <dbl>
##  1 Northeast Protestant   158 0.324    32.4
##  2 Northeast Catholic     162 0.332    33.2
##  3 Northeast Jewish        27 0.0553    5.5
##  4 Northeast None         112 0.230    23
##  5 Northeast Other         28 0.0574    5.7
##  6 Northeast <NA>           1 0.00205   0.2
##  7 Midwest   Protestant   325 0.468    46.8
##  8 Midwest   Catholic     172 0.247    24.7
##  9 Midwest   Jewish         3 0.00432   0.4
## 10 Midwest   None         157 0.226    22.6
## # ℹ 14 more rows
```

`mutate()` is quite clever. See how we can immediately use `freq` to calculate `pct`, even though we are creating them both in the same `mutate()` expression.

Convenience functions

```
gss_sm ▷
  group_by(bigregion, religion) ▷
  summarize(total = n()) ▷
  mutate(freq = total / sum(total),
           pct = round((freq*100), 1))
```

```
## # A tibble: 24 × 5
## # Groups:   bigregion [4]
##    bigregion religion   total    freq   pct
##    <fct>     <fct>      <int>   <dbl> <dbl>
##  1 Northeast Protestant   158 0.324    32.4
##  2 Northeast Catholic     162 0.332    33.2
##  3 Northeast Jewish        27 0.0553    5.5
##  4 Northeast None         112 0.230    23
##  5 Northeast Other         28 0.0574    5.7
##  6 Northeast <NA>           1 0.00205   0.2
##  7 Midwest   Protestant   325 0.468    46.8
##  8 Midwest   Catholic     172 0.247    24.7
##  9 Midwest   Jewish         3 0.00432   0.4
## 10 Midwest   None         157 0.226    22.6
## # ℹ 14 more rows
```

We're going to be doing this `group_by()` … `n()` step a lot. Some shorthand for it would be useful.

Three options for counting up rows

Do it yourself with `n()`

```
gss_sm ▷
  group_by(bigregion, religion) ▷
  summarize(n = n())
```

```
## # A tibble: 24 × 3
## # Groups:   bigregion [4]
##    bigregion religion       n
##    <fct>     <fct>      <int>
##  1 Northeast Protestant   158
##  2 Northeast Catholic     162
##  3 Northeast Jewish        27
##  4 Northeast None         112
##  5 Northeast Other         28
##  6 Northeast <NA>           1
##  7 Midwest   Protestant   325
##  8 Midwest   Catholic     172
##  9 Midwest   Jewish         3
## 10 Midwest   None         157
## # ℹ 14 more rows
```

*The result is **grouped**.*

Three options for counting up rows

Do it yourself with `n()`

```
gss_sm ▷
  group_by(bigregion, religion) ▷
  summarize(n = n())
```

```
## # A tibble: 24 × 3
## # Groups:   bigregion [4]
##    bigregion religion       n
##    <fct>     <fct>      <int>
##  1 Northeast Protestant   158
##  2 Northeast Catholic     162
##  3 Northeast Jewish        27
##  4 Northeast None         112
##  5 Northeast Other         28
##  6 Northeast <NA>           1
##  7 Midwest   Protestant   325
##  8 Midwest   Catholic     172
##  9 Midwest   Jewish         3
## 10 Midwest   None         157
## # ℹ 14 more rows
```

*The result is **grouped**.*

Use `tally()`

```
gss_sm ▷
  group_by(bigregion, religion) ▷
  tally()
```

```
## # A tibble: 24 × 3
## # Groups:   bigregion [4]
##    bigregion religion       n
##    <fct>     <fct>      <int>
##  1 Northeast Protestant   158
##  2 Northeast Catholic     162
##  3 Northeast Jewish        27
##  4 Northeast None         112
##  5 Northeast Other         28
##  6 Northeast <NA>           1
##  7 Midwest   Protestant   325
##  8 Midwest   Catholic     172
##  9 Midwest   Jewish         3
## 10 Midwest   None         157
## # ℹ 14 more rows
```

*The result is **grouped**.*

Three options for counting up rows

Do it yourself with `n()`

```
gss_sm ▷
  group_by(bigregion, religion) ▷
  summarize(n = n())
```

```
## # A tibble: 24 × 3
## # Groups:   bigregion [4]
##    bigregion religion       n
##    <fct>     <fct>      <int>
##  1 Northeast Protestant   158
##  2 Northeast Catholic     162
##  3 Northeast Jewish        27
##  4 Northeast None         112
##  5 Northeast Other         28
##  6 Northeast <NA>           1
##  7 Midwest   Protestant   325
##  8 Midwest   Catholic     172
##  9 Midwest   Jewish         3
## 10 Midwest   None         157
## # ℹ 14 more rows
```

*The result is **grouped**.*

Use `tally()`

```
gss_sm ▷
  group_by(bigregion, religion) ▷
  tally()
```

```
## # A tibble: 24 × 3
## # Groups:   bigregion [4]
##    bigregion religion       n
##    <fct>     <fct>      <int>
##  1 Northeast Protestant   158
##  2 Northeast Catholic     162
##  3 Northeast Jewish        27
##  4 Northeast None         112
##  5 Northeast Other         28
##  6 Northeast <NA>           1
##  7 Midwest   Protestant   325
##  8 Midwest   Catholic     172
##  9 Midwest   Jewish         3
## 10 Midwest   None         157
## # ℹ 14 more rows
```

*The result is **grouped**.*

use `count()`

```
gss_sm ▷
  count(bigregion, religion)
```

```
## # A tibble: 24 × 3
##    bigregion religion       n
##    <fct>     <fct>      <int>
##  1 Northeast Protestant   158
##  2 Northeast Catholic     162
##  3 Northeast Jewish        27
##  4 Northeast None         112
##  5 Northeast Other         28
##  6 Northeast <NA>           1
##  7 Midwest   Protestant   325
##  8 Midwest   Catholic     172
##  9 Midwest   Jewish         3
## 10 Midwest   None         157
## # ℹ 14 more rows
```

*One step; the result is **not grouped**.*

Pipelined tables can be quickly checked

```
## Calculate pct religion within region?
rel_by_region ← gss_sm ▷
  count(bigregion, religion) ▷
  mutate(pct = round((n/sum(n))*100, 1))

rel_by_region
```

```
## # A tibble: 24 × 4
##    bigregion religion       n   pct
##    <fct>     <fct>      <int> <dbl>
##  1 Northeast Protestant   158   5.5
##  2 Northeast Catholic     162   5.7
##  3 Northeast Jewish        27   0.9
##  4 Northeast None         112   3.9
##  5 Northeast Other         28   1
##  6 Northeast <NA>           1   0
##  7 Midwest   Protestant   325  11.3
##  8 Midwest   Catholic     172   6
##  9 Midwest   Jewish         3   0.1
## 10 Midwest   None         157   5.5
## # ℹ 14 more rows
```

Hm, did I sum over right group?

Pipelined tables can be quickly checked

```
## Calculate pct religion within region?
rel_by_region ← gss_sm ▷
  count(bigregion, religion) ▷
  mutate(pct = round((n/sum(n))*100, 1))

rel_by_region
```

```
## # A tibble: 24 × 4
##    bigregion religion       n   pct
##    <fct>     <fct>      <int> <dbl>
##  1 Northeast Protestant   158   5.5
##  2 Northeast Catholic     162   5.7
##  3 Northeast Jewish        27   0.9
##  4 Northeast None         112   3.9
##  5 Northeast Other         28   1
##  6 Northeast <NA>           1   0
##  7 Midwest   Protestant   325  11.3
##  8 Midwest   Catholic     172   6
##  9 Midwest   Jewish         3   0.1
## 10 Midwest   None         157   5.5
## # ℹ 14 more rows
```

Hm, did I sum over right group?

```
## Each region should sum to ~100
rel_by_region ▷
  group_by(bigregion) ▷
  summarize(total = sum(pct))
```

```
## # A tibble: 4 × 2
##   bigregion total
##   <fct>     <dbl>
## 1 Northeast  17
## 2 Midwest    24.3
## 3 South      36.7
## 4 West       22
```

No! What has gone wrong here?

Pipelined tables can be quickly checked

```
rel_by_region ← gss_sm ▷
  count(bigregion, religion) ▷
  mutate(pct = round((n/sum(n))*100, 1))
```

`count()` returns ungrouped results, so there are no groups carry forward to the `mutate()` step.

```
rel_by_region ▷
  summarize(total = sum(pct))
```

```
## # A tibble: 1 × 1
##   total
##   <dbl>
## 1   100
```

With `count()`, the `pct` values in this case are the marginals for the whole table.

Pipelined tables can be quickly checked

```
rel_by_region ← gss_sm ▷
  count(bigregion, religion) ▷
  mutate(pct = round((n/sum(n))*100, 1))
```

`count()` returns ungrouped results, so there are no groups carry forward to the `mutate()` step.

```
rel_by_region ▷
  summarize(total = sum(pct))
```

```
## # A tibble: 1 × 1
##   total
##   <dbl>
## 1   100
```

With `count()`, the `pct` values in this case are the marginals for the whole table.

```
rel_by_region ← gss_sm ▷
  group_by(bigregion, religion) ▷
  tally() ▷
  mutate(pct = round((n/sum(n))*100, 1))
```

```
# Check
rel_by_region ▷
  group_by(bigregion) ▷
  summarize(total = sum(pct))
```

```
## # A tibble: 4 × 2
##   bigregion total
##   <fct>     <dbl>
## 1 Northeast 100
## 2 Midwest    99.9
## 3 South     100
## 4 West      100.
```

`group_by()` and `tally()` both return a grouped result. We get some rounding error because we used `round()` after summing originally.

Two lessons

Check your tables!

Pipelines feed their content forward, so you need to make sure your results are not incorrect.

Two lessons

Check your tables!

Pipelines feed their content forward, so you need to make sure your results are not incorrect.

Often, complex tables and graphs can be disturbingly plausible even when wrong.

Two lessons

Check your tables!

Pipelines feed their content forward, so you need to make sure your results are not incorrect.

Often, complex tables and graphs can be disturbingly plausible even when wrong.

So, figure out what the result should be and test it!

Two lessons

Check your tables!

Pipelines feed their content forward, so you need to make sure your results are not incorrect.

Often, complex tables and graphs can be disturbingly plausible even when wrong.

So, figure out what the result should be and test it!

Starting with simple or toy cases can help with this process.

Two lessons

Inspect your pipes!

Understand pipelines by running them forward or peeling them back a step at a time.

This is a *very* effective way to understand your own and other people's code.

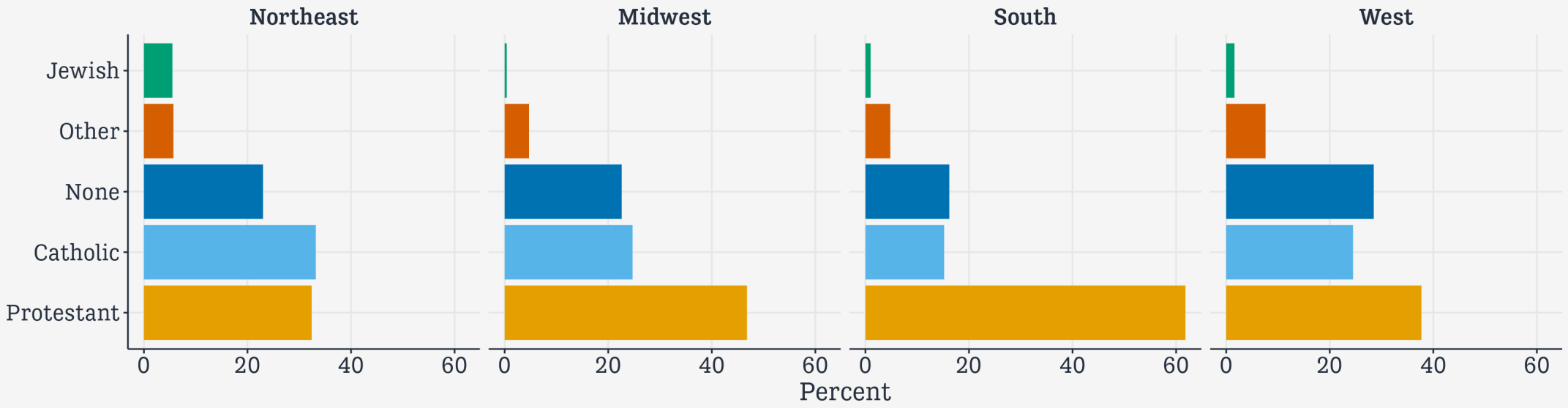
Pass your pipeline on to a table

```
gss_sm ▷
  count(bigregion, religion) ▷
  pivot_wider(names_from = bigregion, values_from = n) ▷
  kable()
```

religion	Northeast	Midwest	South	West
Protestant	158	325	650	238
Catholic	162	172	160	155
Jewish	27	3	11	10
None	112	157	170	180
Other	28	33	50	48
NA	1	5	11	1

Pass your pipeline on to a graph

```
gss_sm ▷
  group_by(bigregion, religion) ▷
  tally() ▷
  mutate(pct = round((n/sum(n))*100, 1)) ▷
  drop_na() ▷
  ggplot(mapping = aes(x = pct, y = reorder(religion, -pct), fill = religion)) +
  geom_col() +
    labs(x = "Percent", y = NULL) +
    guides(fill = "none") +
    facet_wrap(~ bigregion, nrow = 1)
```



Use `dplyr` pipelines to create summary tables. Then send your clean tables to `ggplot`.

Facets are often better than Guides

Let's put that table in an object

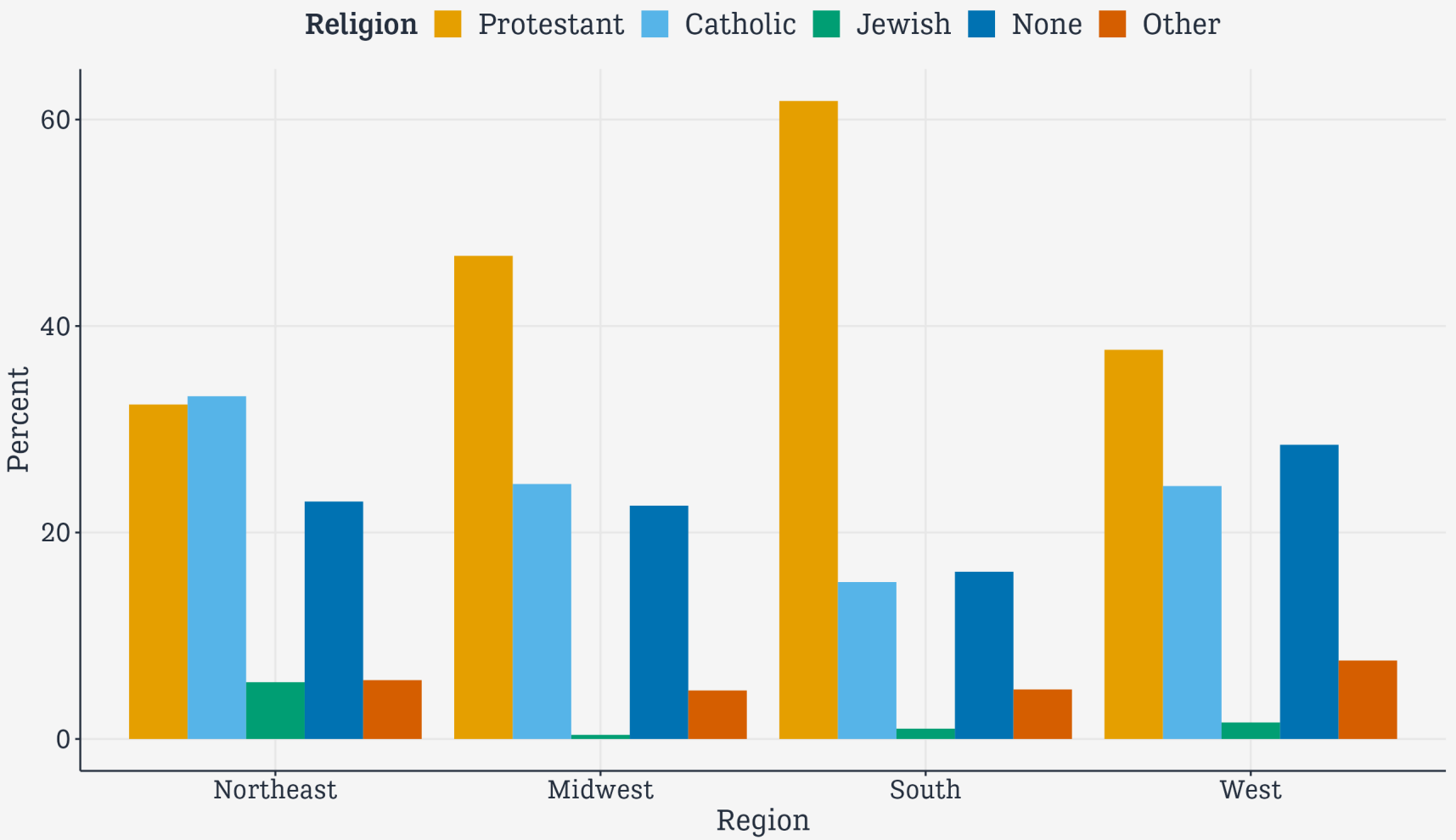
```
rel_by_region ← gss_sm ▷
  group_by(bigregion, religion) ▷
  tally() ▷
  mutate(pct = round((n/sum(n))*100, 1)) ▷
  drop_na()


head(rel_by_region)
```

```
## # A tibble: 6 × 4
## # Groups:   bigregion [2]
##   bigregion religion       n   pct
##   <fct>     <fct>      <int> <dbl>
## 1 Northeast Protestant   158  32.4
## 2 Northeast Catholic     162  33.2
## 3 Northeast Jewish        27   5.5
## 4 Northeast None         112  23
## 5 Northeast Other         28   5.7
## 6 Midwest   Protestant   325  46.8
```

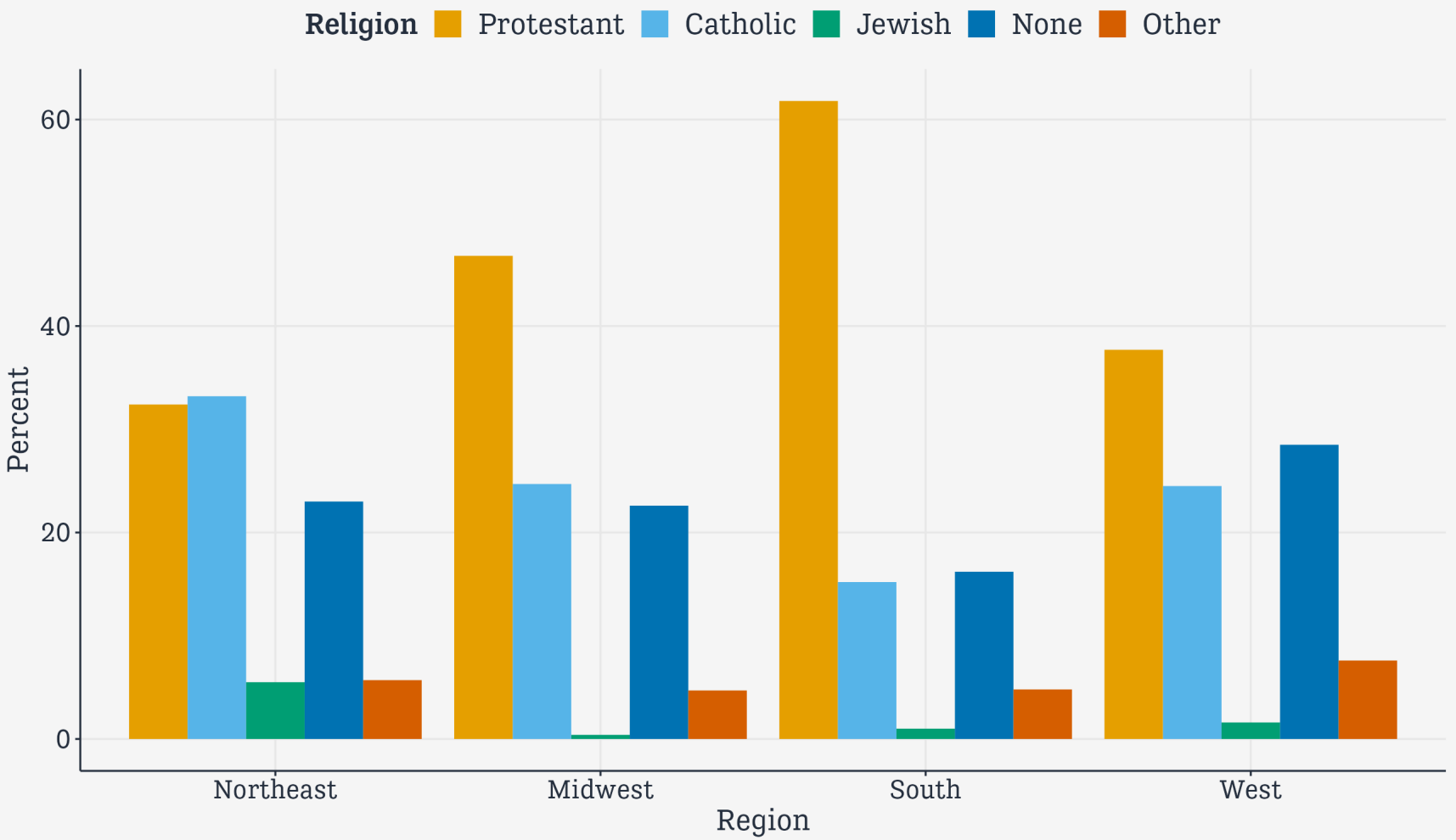
We might write ...

```
p ← ggplot(data = rel_by_region,
                mapping = aes(x = bigregion,
                              y = pct,
                              fill = religion))
p_out ← p + geom_col(position = "dodge") +
    labs(x = "Region",
         y = "Percent",
         fill = "Religion")
```

We might write ...



Is this an effective graph? Not really!



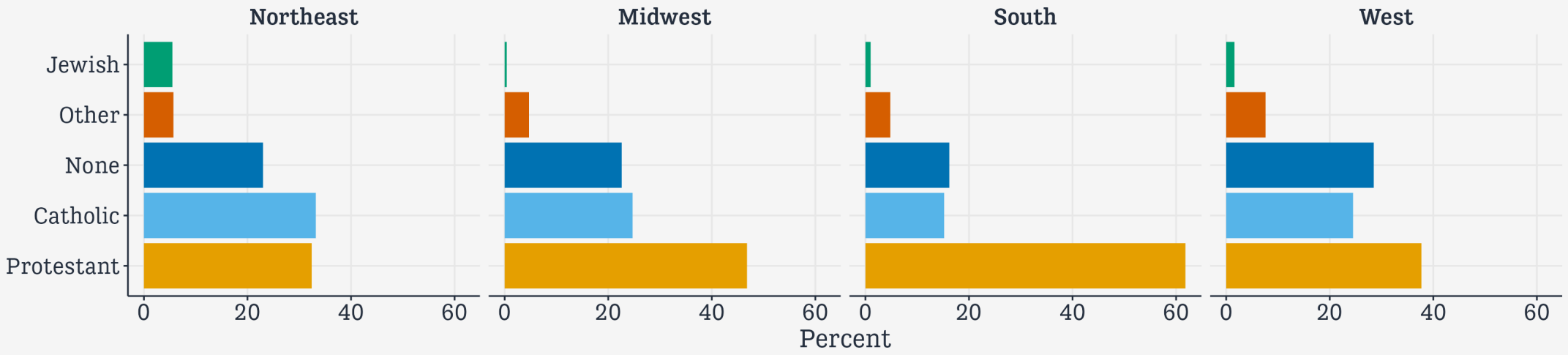
Try faceting instead

```
p ← ggplot(data = rel_by_region,
                mapping = aes(x = pct,
                              y = reorder(religion, -pct),
                              fill = religion))
p_out_facet ← p + geom_col() +
  guides(fill = "none") +
  facet_wrap(~ bigregion, nrow = 1) +
  labs(x = "Percent",
       y = NULL)
```

Putting categories on the y-axis is a very useful trick.

Faceting reduces the number of guides the viewer needs to consult.

Try faceting instead



Try putting categories on the y-axis. (And reorder them by x.)

Try faceting variables instead of mapping them to color or shape.

Try to minimize the need for guides and legends.

Some data on Organ Donation

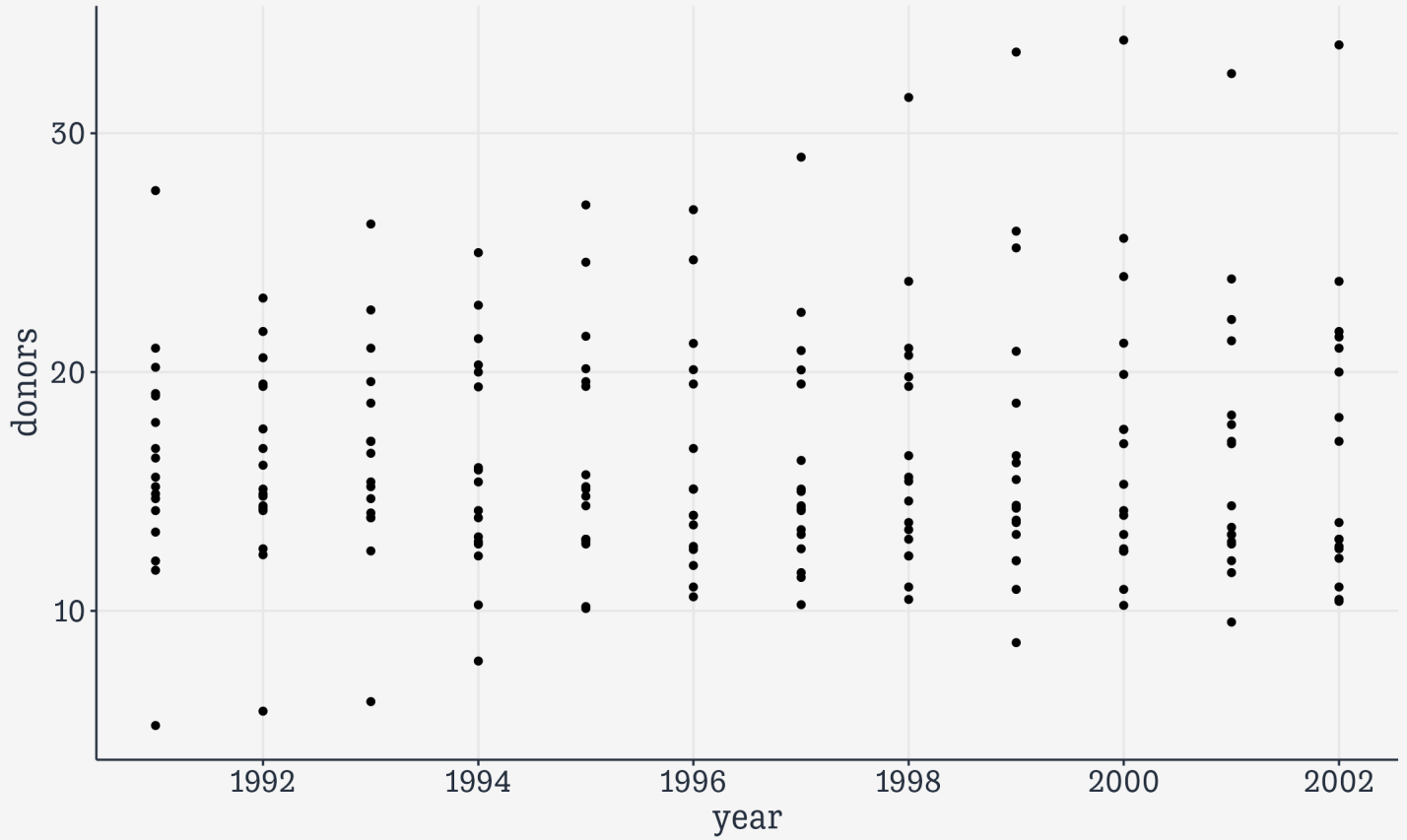
`organdata` is in the `socviz` package

```
organdata
```

```
## # A tibble: 238 × 21
##    country   year       donors   pop pop_dens   gdp gdp_lag health health_lag
##    <chr>     <date>      <dbl> <int>    <dbl> <int>   <int>  <dbl>      <dbl>
##  1 Australia NA           NA    17065    0.220 16774   16591   1300       1224
##  2 Australia 1991-01-01  12.1   17284    0.223 17171   16774   1379       1300
##  3 Australia 1992-01-01  12.4   17495    0.226 17914   17171   1455       1379
##  4 Australia 1993-01-01  12.5   17667    0.228 18883   17914   1540       1455
##  5 Australia 1994-01-01  10.2   17855    0.231 19849   18883   1626       1540
##  6 Australia 1995-01-01  10.2   18072    0.233 21079   19849   1737       1626
##  7 Australia 1996-01-01  10.6   18311    0.237 21923   21079   1846       1737
##  8 Australia 1997-01-01  10.3   18518    0.239 22961   21923   1948       1846
##  9 Australia 1998-01-01  10.5   18711    0.242 24148   22961   2077       1948
## 10 Australia 1999-01-01   8.67  18926    0.244 25445   24148   2231       2077
## # ℹ 228 more rows
## # ℹ 12 more variables: pubhealth <dbl>, roads <dbl>, cerebvas <int>,
## #   assault <int>, external <int>, txp_pop <dbl>, world <chr>, opt <chr>,
## #   consent_law <chr>, consent_practice <chr>, consistent <chr>, ccode <chr>
```

First looks

```
p ← ggplot(data = organdata,
           mapping = aes(x = year, y = donors))
p + geom_point()
```

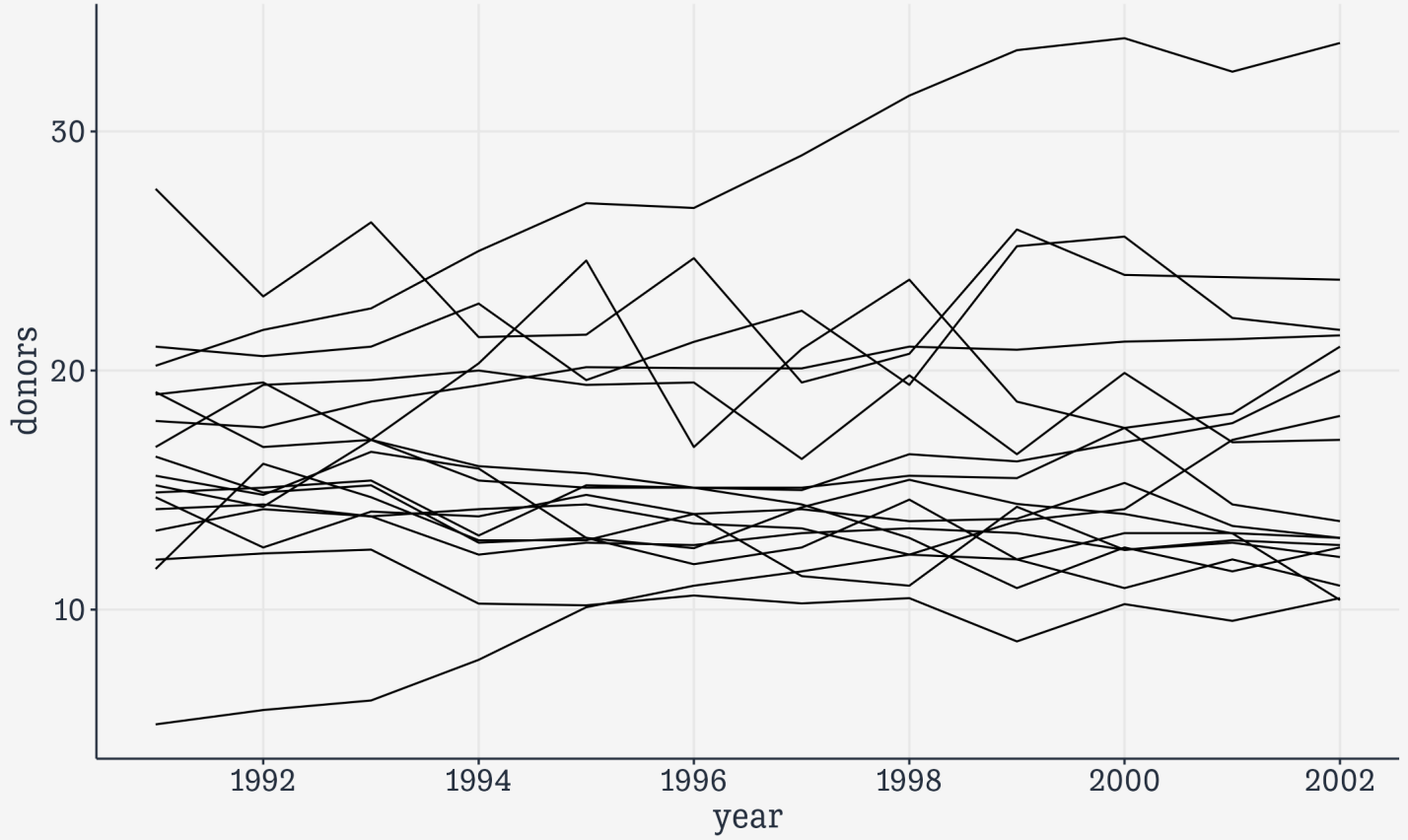


First looks

```
p ← ggplot(data = organdata,
           mapping = aes(x = year, y = donors))
p + geom_line()
```

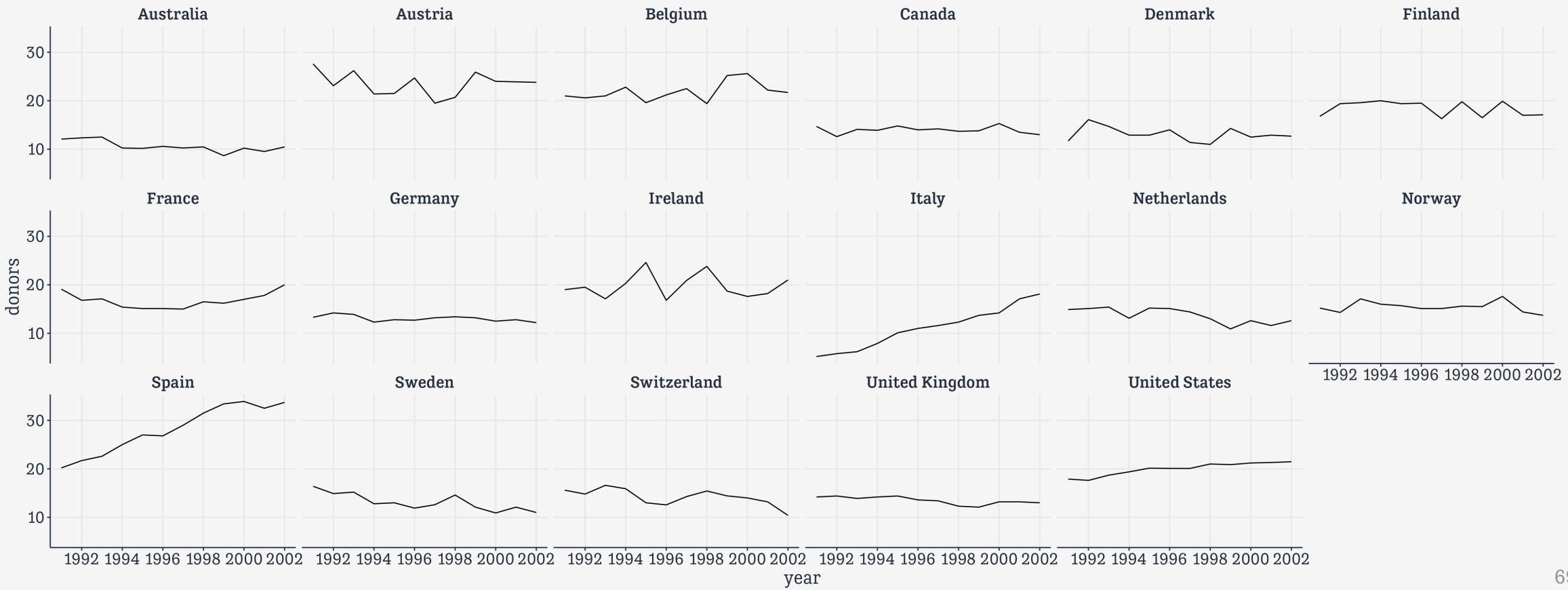
First looks

```
p ← ggplot(data = organdata,
           mapping = aes(x = year, y = donors))
p + geom_line(aes(group = country))
```



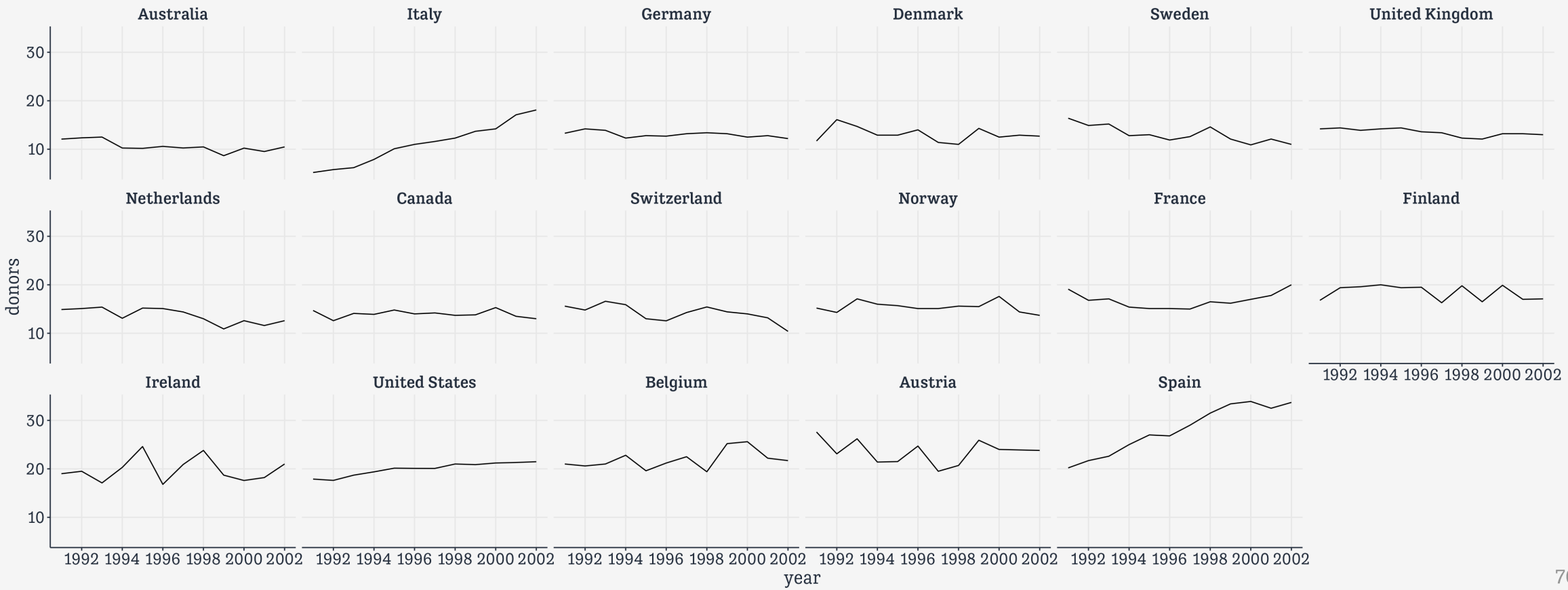
First looks

```
p ← ggplot(data = organdata,
           mapping = aes(x = year, y = donors))
p + geom_line() +
  facet_wrap(~ country, nrow = 3)
```



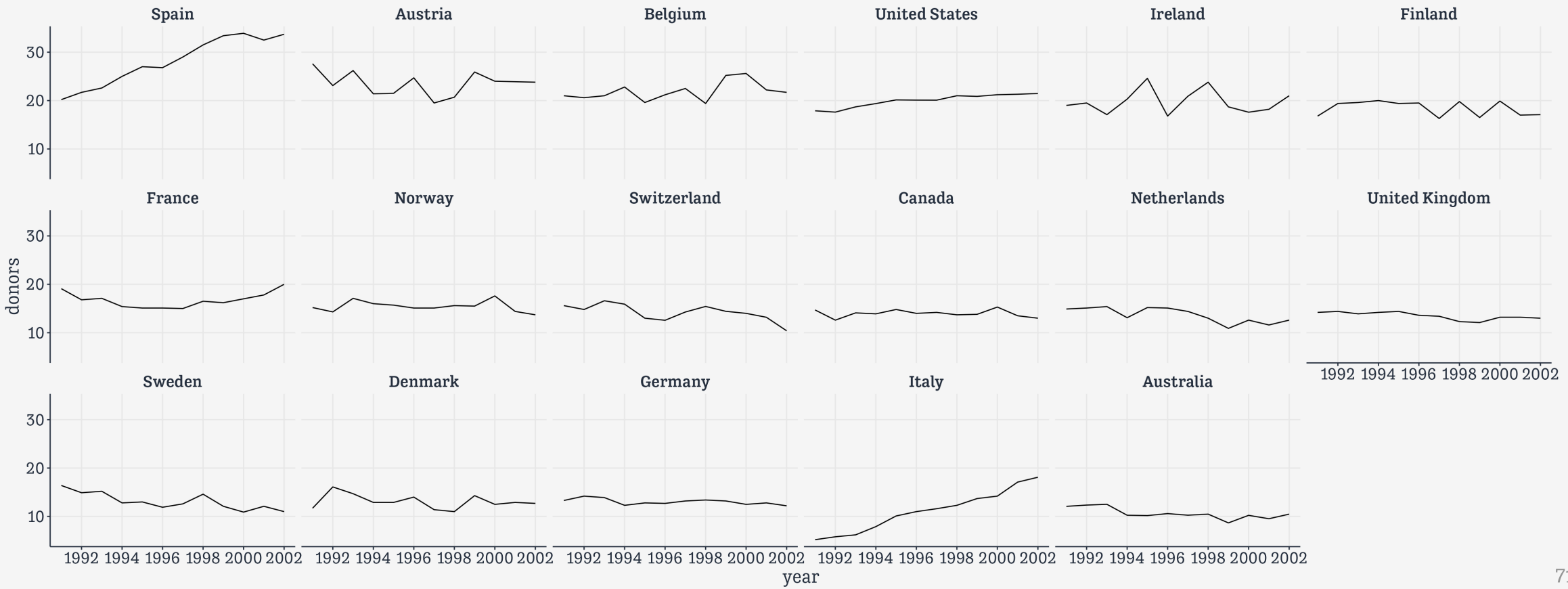
First looks

```
p ← ggplot(data = organdata,
           mapping = aes(x = year, y = donors))
p + geom_line() +
  facet_wrap(~ reorder(country, donors, na.rm = TRUE), nrow = 3)
```



First looks

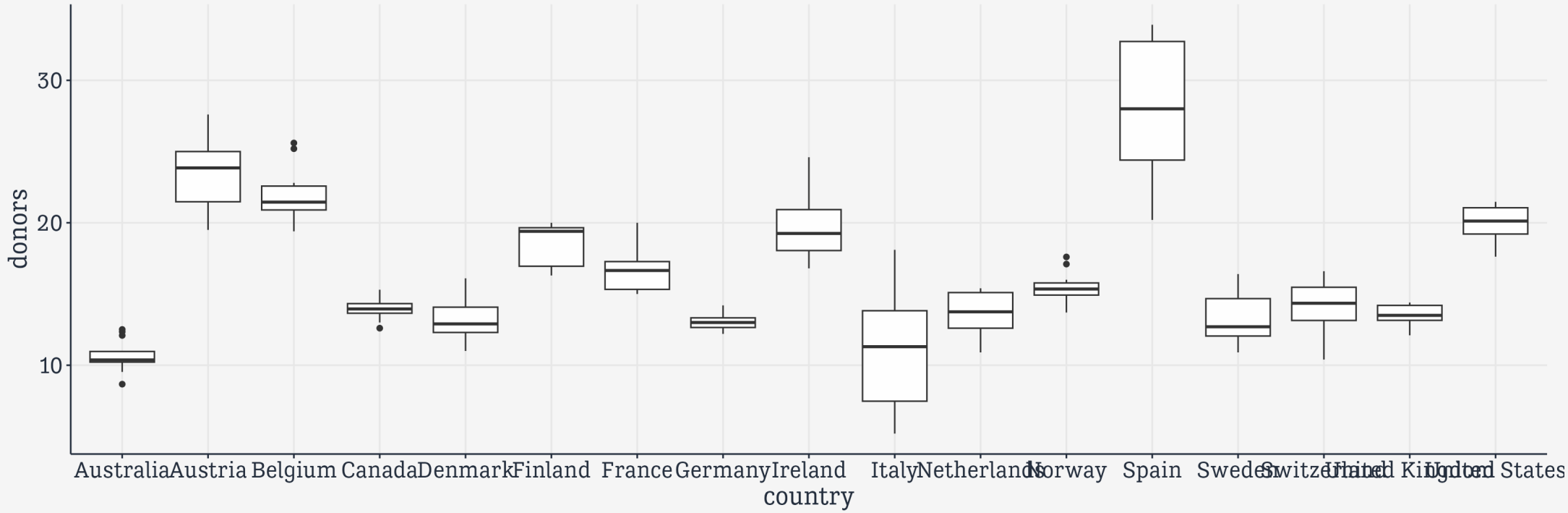
```
p ← ggplot(data = organdata,
           mapping = aes(x = year, y = donors))
p + geom_line() +
  facet_wrap(~ reorder(country, -donors, na.rm = TRUE), nrow = 3)
```



Showing continuous measures by category

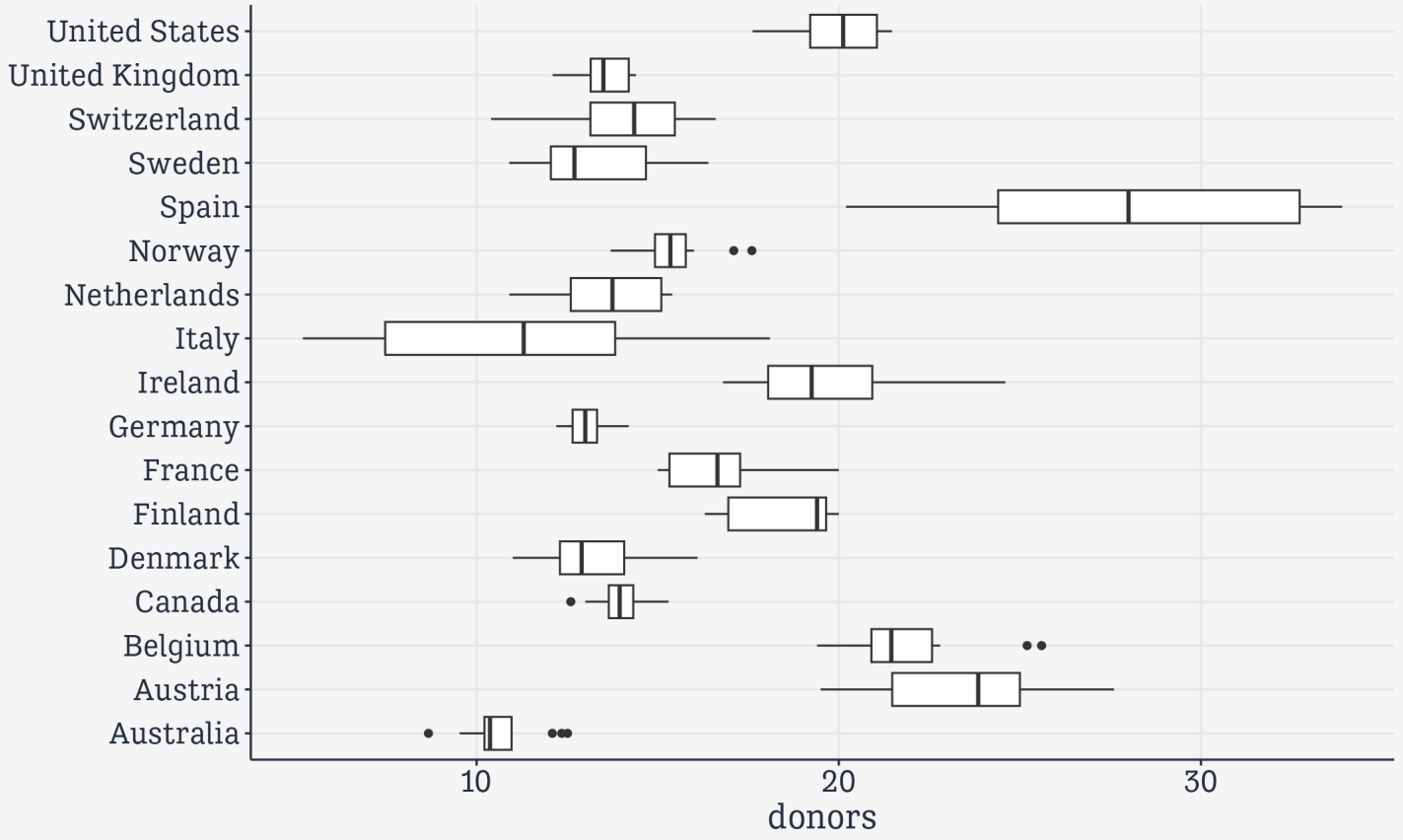
Boxplots: geom_boxplot()

```
## Pipeline the data directly; then it's implicitly the first argument to `ggplot()`
organdata ▷
  ggplot(mapping = aes(x = country, y = donors)) +
  geom_boxplot()
```



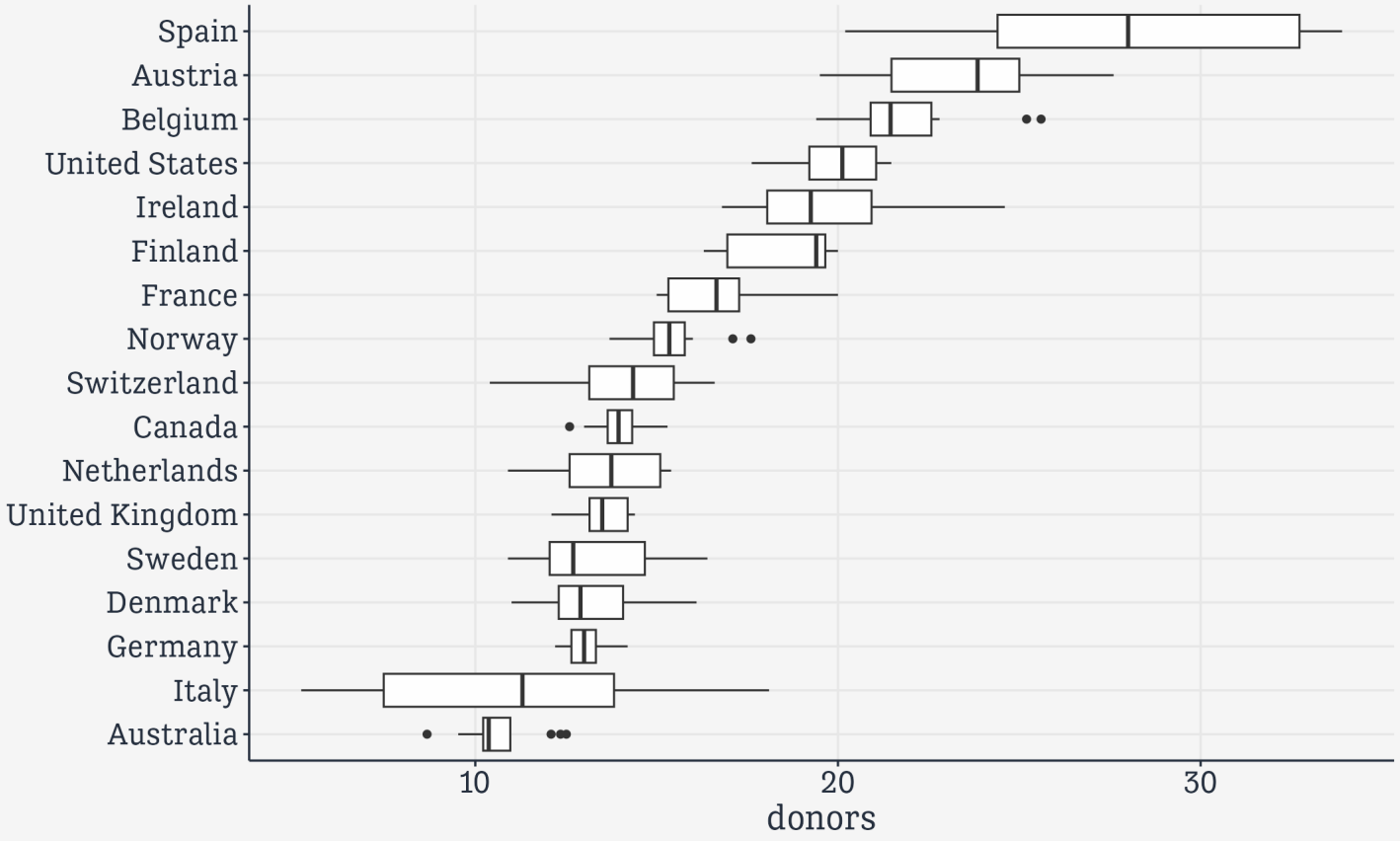
Put categories on the y-axis!

```
organdata ▷
  ggplot(mapping = aes(x = donors, y = country)) +
  geom_boxplot() +
  labs(y = NULL)
```



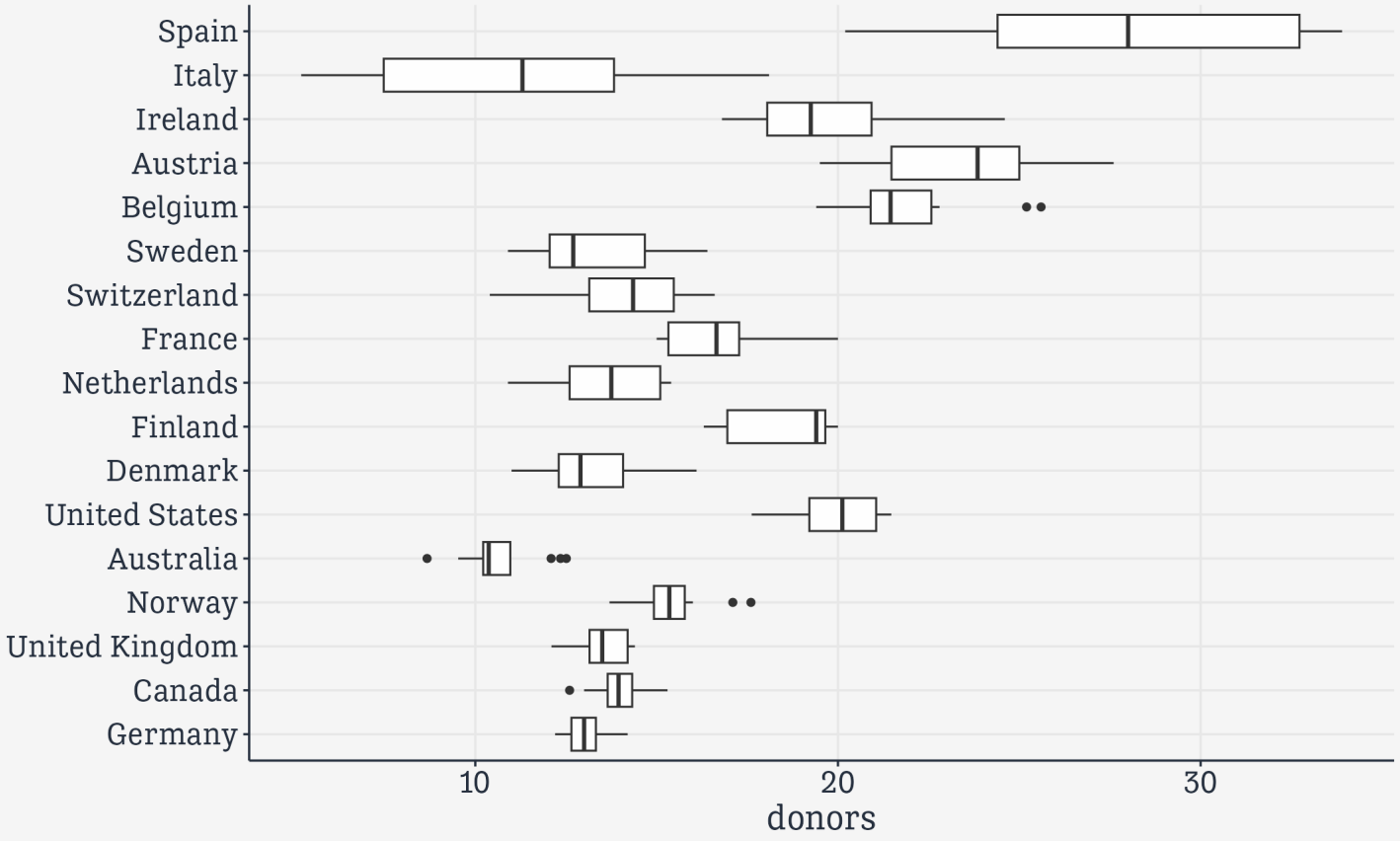
Reorder y by the mean of x

```
organdata ▷
  ggplot(mapping = aes(x = donors, y = reorder(country, donors, na.rm = TRUE))) +
  geom_boxplot() +
  labs(y = NULL)
```



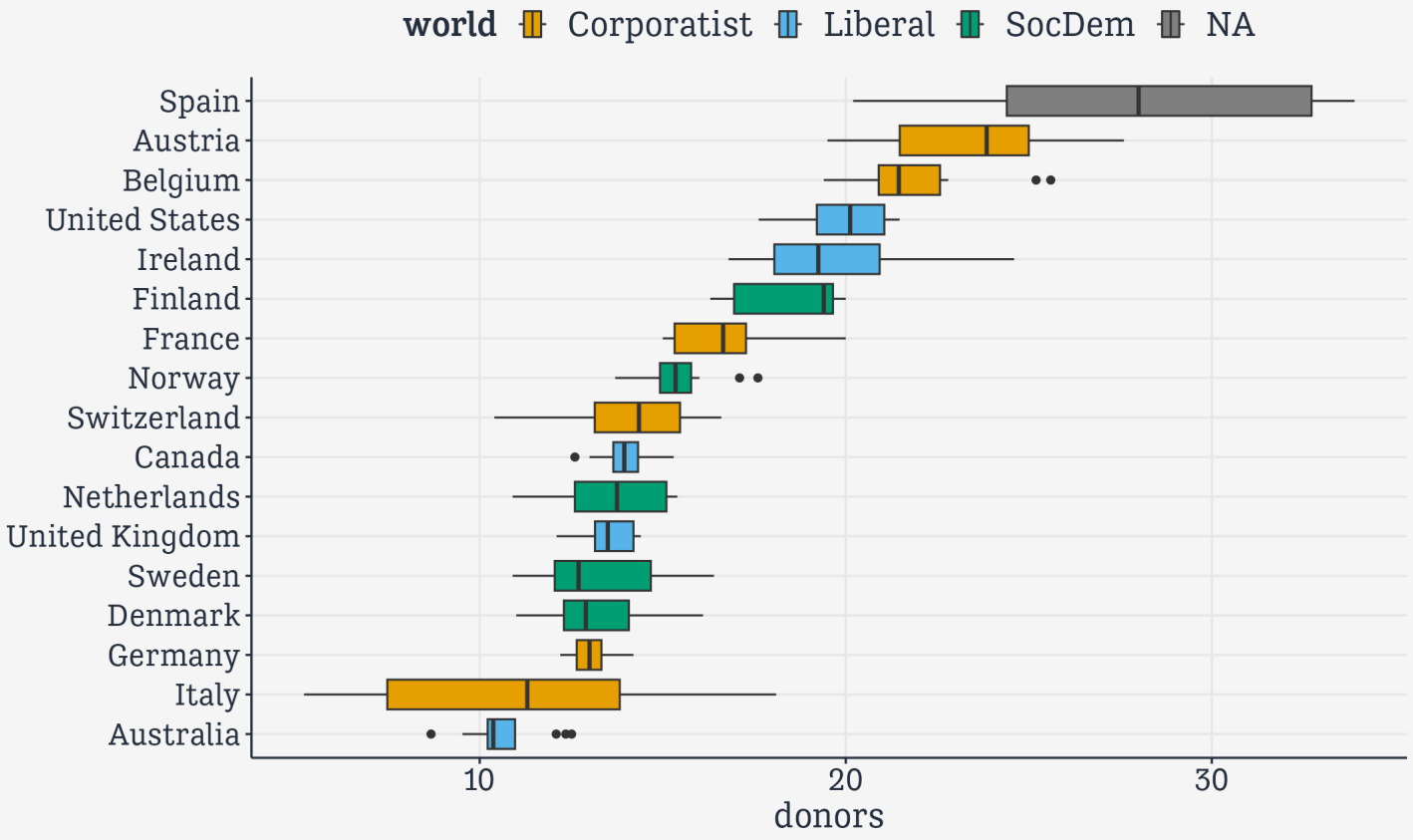
(Reorder y by any statistic you like)

```
organdata ▷
  ggplot(mapping = aes(x = donors, y = reorder(country, donors, sd, na.rm = TRUE))) +
  geom_boxplot() +
  labs(y = NULL)
```



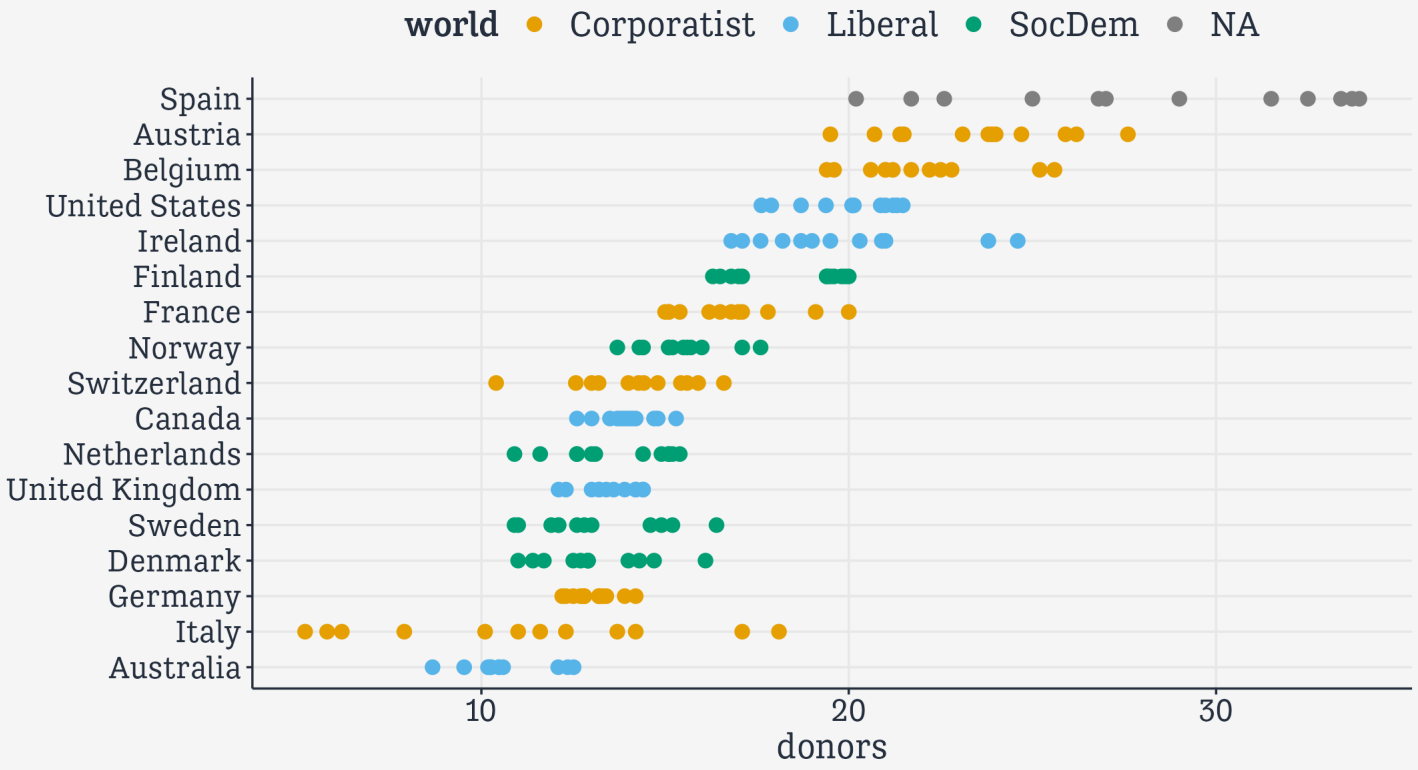
geom_boxplot() knows color and fill

```
organdata ▷
  ggplot(mapping = aes(x = donors, y = reorder(country, donors, na.rm = TRUE), fill = world)) +
  geom_boxplot() +
  labs(y = NULL)
```



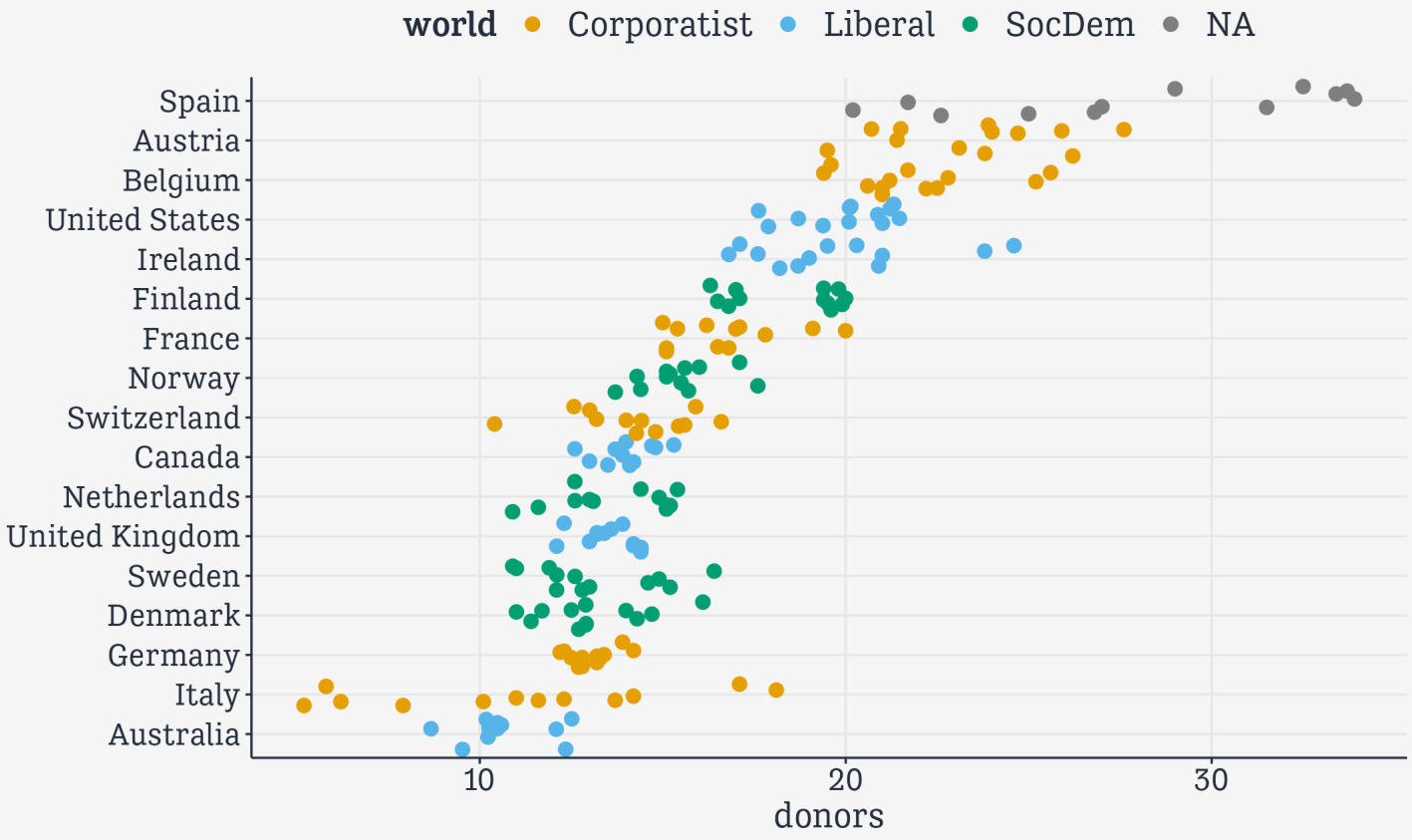
These strategies are quite general

```
organdata ▷
  ggplot(mapping = aes(x = donors, y = reorder(country, donors, na.rm = TRUE), color = world)) +
  geom_point(size = rel(3)) +
  labs(y = NULL)
```



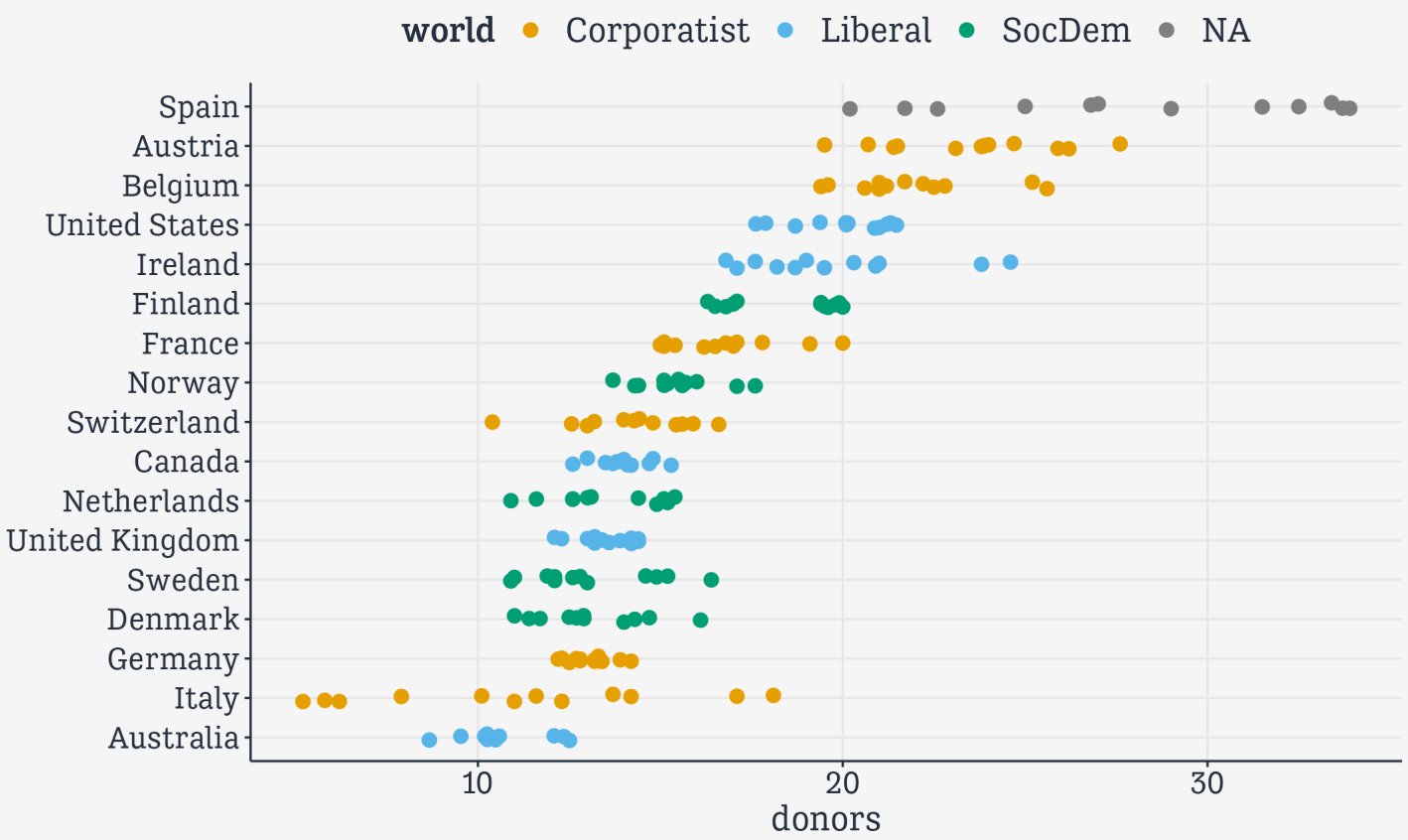
geom-jitter() can help with overplotting

```
organdata ▷
  ggplot(mapping = aes(x = donors, y = reorder(country, donors, na.rm = TRUE), color = world)) +
  geom_jitter(size = rel(3)) +
  labs(y = NULL)
```



Adjust with a `position` argument

```
organdata ▷
  ggplot(mapping = aes(x = donors, y = reorder(country, donors, na.rm = TRUE),
                       color = world)) +
  geom_jitter(size = rel(3), position = position_jitter(height = 0.1)) +
  labs(y = NULL)
```



Summarize better with `dplyr`

Summarize a bunch of variables

```
by_country ← organdata ▷
  group_by(consent_law, country)  ▷
    summarize(donors_mean= mean(donors, na.rm = TRUE),
              donors_sd = sd(donors, na.rm = TRUE),
              gdp_mean = mean(gdp, na.rm = TRUE),
              health_mean = mean(health, na.rm = TRUE),
              roads_mean = mean(roads, na.rm = TRUE),
              cerebvas_mean = mean(cerebvas, na.rm = TRUE))

head(by_country)
```

```
## # A tibble: 6 × 8
## # Groups:   consent_law [1]
##   consent_law country     donors_mean donors_sd gdp_mean health_mean roads_mean
##   <chr>       <chr>             <dbl>     <dbl>    <dbl>       <dbl>      <dbl>
## 1 Informed    Australia          10.6     1.14    22179.       1958.      105.
## 2 Informed    Canada             14.0     0.751   23711.       2272.      109.
## 3 Informed    Denmark            13.1     1.47    23722.       2054.      102.
## 4 Informed    Germany            13.0     0.611   22163.       2349.      113.
## 5 Informed    Ireland            19.8     2.48    20824.       1480.      118.
## 6 Informed    Netherlands        13.7     1.55    23013.       1993.       76.1
## # ℹ 1 more variable: cerebvas_mean <dbl>
```

This works, but there's so much repetition! It's an open invitation to make mistakes copying and pasting.

DRY:

Don't Repeat Yourself

Use `across()` and `where()` instead

```
by_country ← organdata ▷
  group_by(consent_law, country) ▷
    summarize(across(where(is.numeric),
                     list(mean = mean,
                          sd = sd),
                      na.rm = TRUE))
head(by_country)
```

```
## # A tibble: 6 × 28
## # Groups:   consent_law [1]
##   consent_law country     donors_mean donors_sd pop_mean pop_sd pop_dens_mean
##   <chr>       <chr>             <dbl>     <dbl>    <dbl>  <dbl>         <dbl>
## 1 Informed    Australia          10.6     1.14    18318.  831.          0.237
## 2 Informed    Canada             14.0     0.751   29608. 1193.          0.297
## 3 Informed    Denmark            13.1     1.47     5257.   80.6        12.2
## 4 Informed    Germany            13.0     0.611   80255. 5158.         22.5
## 5 Informed    Ireland            19.8     2.48     3674.  132.          5.23
## 6 Informed    Netherlands        13.7     1.55    15548.  373.         37.4
## # ℹ 21 more variables: pop_dens_sd <dbl>, gdp_mean <dbl>, gdp_sd <dbl>,
## #   gdp_lag_mean <dbl>, gdp_lag_sd <dbl>, health_mean <dbl>, health_sd <dbl>,
## #   health_lag_mean <dbl>, health_lag_sd <dbl>, pubhealth_mean <dbl>,
## #   pubhealth_sd <dbl>, roads_mean <dbl>, roads_sd <dbl>, cerebvas_mean <dbl>,
## #   cerebvas_sd <dbl>, assault_mean <dbl>, assault_sd <dbl>,
## #   external_mean <dbl>, external_sd <dbl>, txp_pop_mean <dbl>,
## #   txp_pop_sd <dbl>
```

Use `across()` and `where()` instead

```
by_country ← organdata ▷
  group_by(consent_law, country) ▷
    summarize(across(where(is.numeric),
                     list(mean = mean,
                          sd = sd),
                      na.rm = TRUE),
              .groups = "drop")
head(by_country)
```

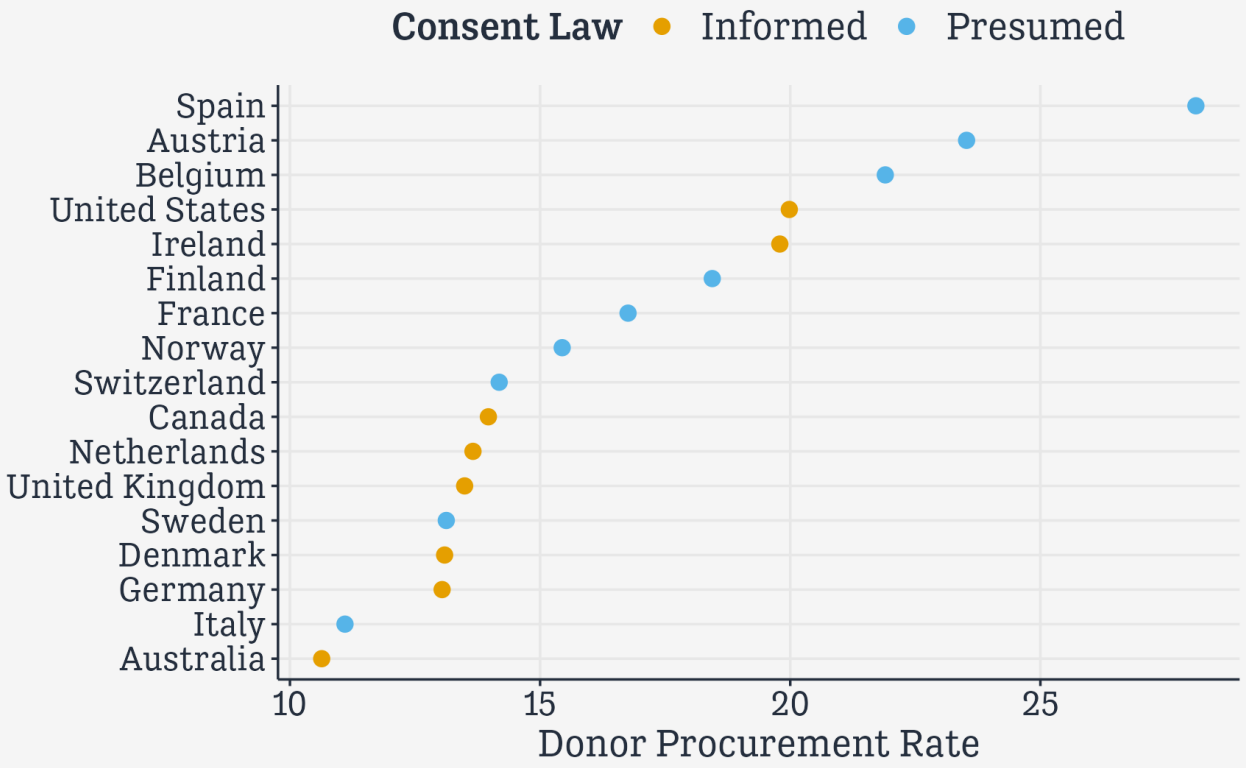
```
## # A tibble: 6 × 28
##   consent_law country     donors_mean donors_sd pop_mean pop_sd pop_dens_mean
##   <chr>       <chr>             <dbl>     <dbl>    <dbl>  <dbl>         <dbl>
## 1 Informed    Australia          10.6     1.14    18318.  831.          0.237
## 2 Informed    Canada             14.0     0.751   29608. 1193.          0.297
## 3 Informed    Denmark            13.1     1.47     5257.   80.6        12.2
## 4 Informed    Germany            13.0     0.611   80255. 5158.         22.5
## 5 Informed    Ireland            19.8     2.48     3674.  132.          5.23
## 6 Informed    Netherlands        13.7     1.55    15548.  373.         37.4
## # ℹ 21 more variables: pop_dens_sd <dbl>, gdp_mean <dbl>, gdp_sd <dbl>,
## #   gdp_lag_mean <dbl>, gdp_lag_sd <dbl>, health_mean <dbl>, health_sd <dbl>,
## #   health_lag_mean <dbl>, health_lag_sd <dbl>, pubhealth_mean <dbl>,
## #   pubhealth_sd <dbl>, roads_mean <dbl>, roads_sd <dbl>, cerebvas_mean <dbl>,
## #   cerebvas_sd <dbl>, assault_mean <dbl>, assault_sd <dbl>,
## #   external_mean <dbl>, external_sd <dbl>, txp_pop_mean <dbl>,
## #   txp_pop_sd <dbl>
```

Plot our summary data

```
by_country ▷
  ggplot(mapping =
           aes(x = donors_mean,
               y = reorder(country, donors_mean),
               color = consent_law)) +
  geom_point(size=3) +
  labs(x = "Donor Procurement Rate",
       y = NULL,
       color = "Consent Law")
```

Plot our summary data

```
by_country ▷
  ggplot(mapping =
           aes(x = donors_mean,
               y = reorder(country, donors_mean),
               color = consent_law)) +
  geom_point(size=3) +
  labs(x = "Donor Procurement Rate",
       y = NULL,
       color = "Consent Law")
```



What about faceting it instead?

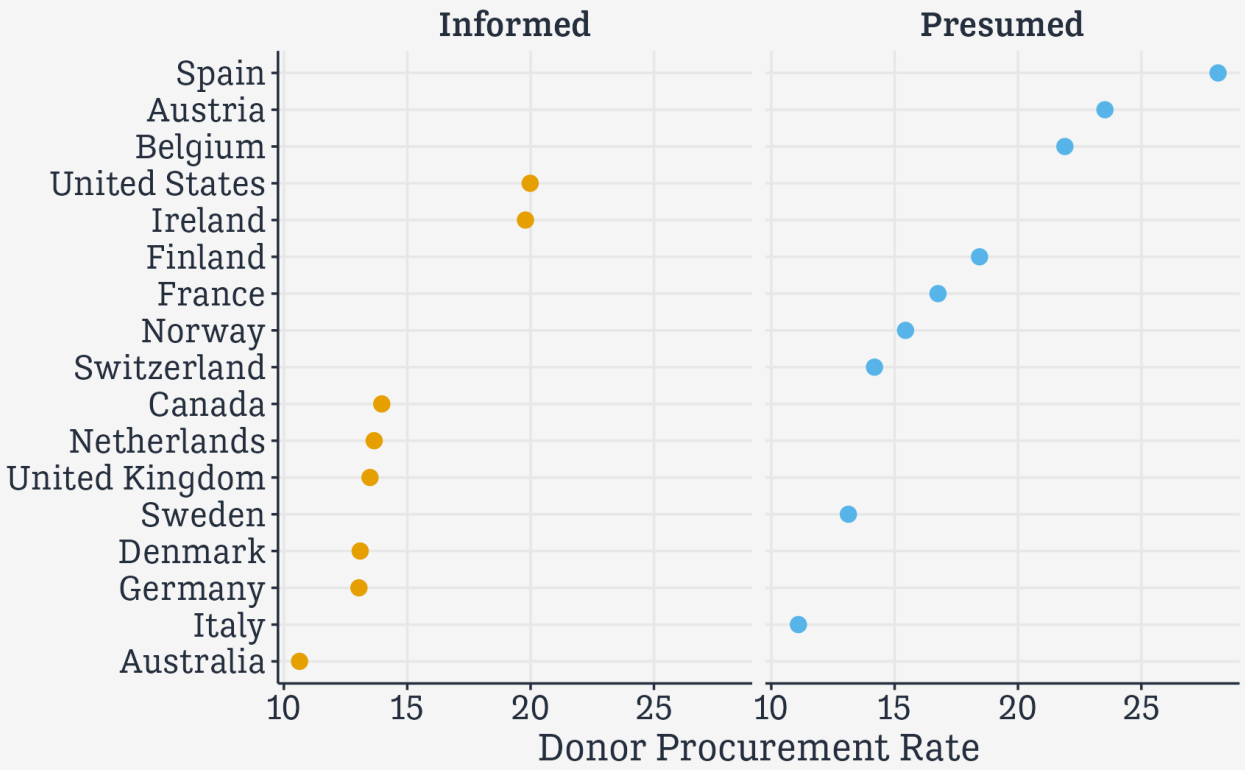
```
by_country ▷
  ggplot(mapping =
           aes(x = donors_mean,
               y = reorder(country, donors_mean),
               color = consent_law)) +
  geom_point(size=3) +
  guides(color = "none") +
  facet_wrap(~ consent_law) +
  labs(x = "Donor Procurement Rate",
       y = NULL,
       color = "Consent Law")
```

The problem is that countries can only be in one Consent Law category.

What about faceting it instead?

```
by_country ▷
  ggplot(mapping =
           aes(x = donors_mean,
               y = reorder(country, donors_mean),
               color = consent_law)) +
  geom_point(size=3) +
  guides(color = "none") +
  facet_wrap(~ consent_law) +
  labs(x = "Donor Procurement Rate",
       y = NULL,
       color = "Consent Law")
```

The problem is that countries can only be in one Consent Law category.



What about faceting it instead?

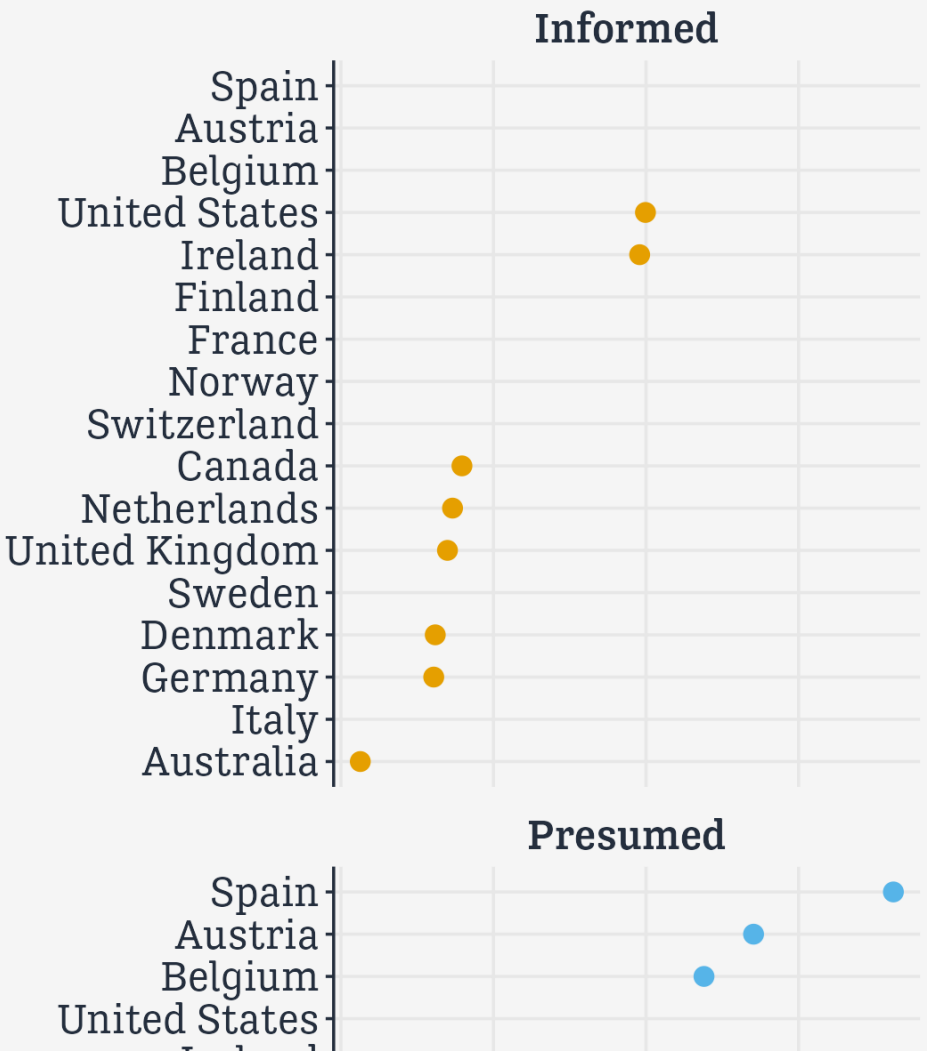
```
by_country ▷
  ggplot(mapping =
           aes(x = donors_mean,
               y = reorder(country, donors_mean),
               color = consent_law)) +
  geom_point(size=3) +
  guides(color = "none") +
  facet_wrap(~ consent_law, ncol = 1) +
  labs(x = "Donor Procurement Rate",
       y = NULL,
       color = "Consent Law")
```

Restricting to one column doesn't fix it.

What about faceting it instead?

```
by_country ▷
  ggplot(mapping =
           aes(x = donors_mean,
               y = reorder(country, donors_mean),
               color = consent_law)) +
  geom_point(size=3) +
  guides(color = "none") +
  facet_wrap(~ consent_law, ncol = 1) +
  labs(x = "Donor Procurement Rate",
       y = NULL,
       color = "Consent Law")
```

Restricting to one column doesn't fix it.



Allow the y-scale to vary

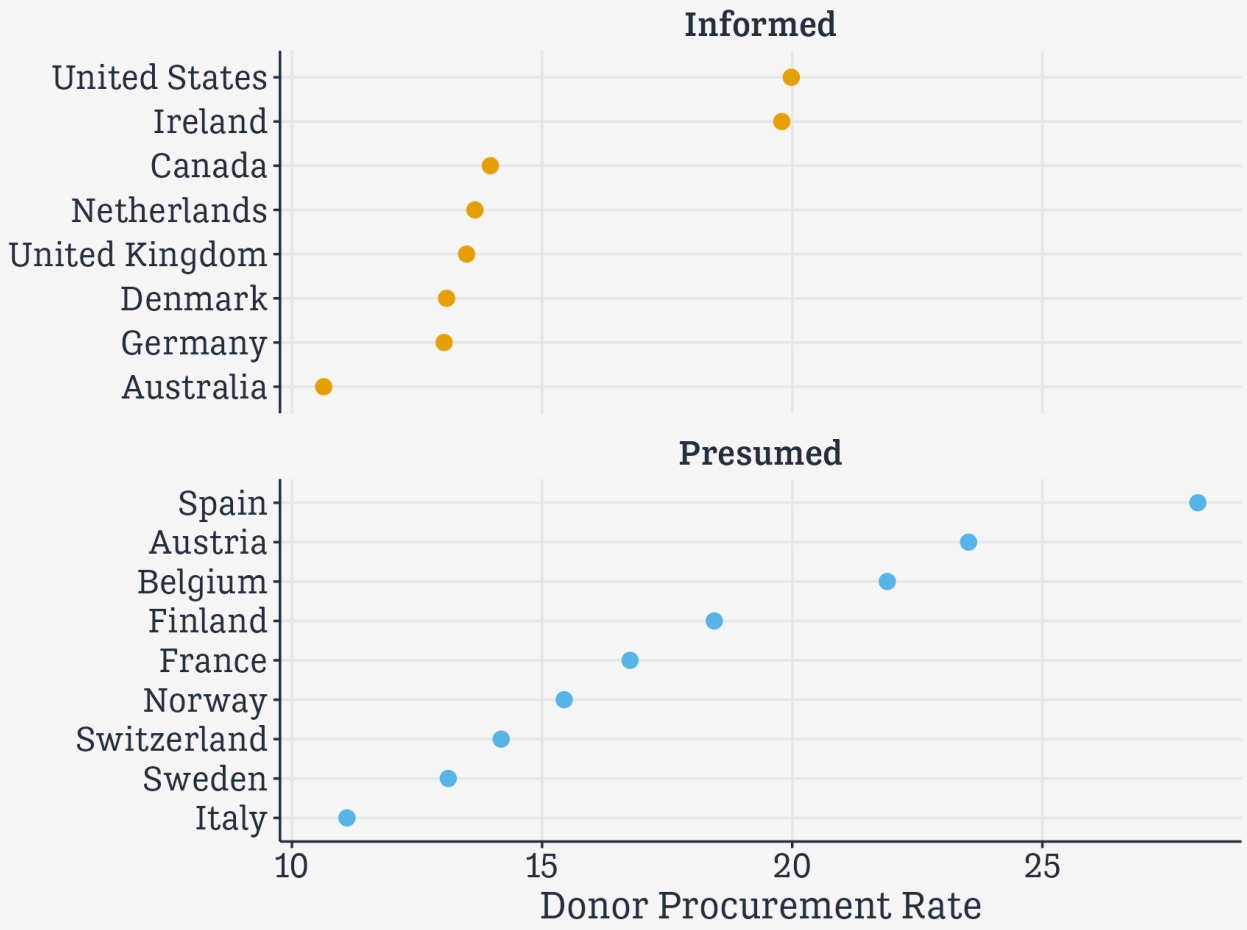
```
by_country ▷
  ggplot(mapping =
           aes(x = donors_mean,
               y = reorder(country, donors_mean),
               color = consent_law)) +
  geom_point(size=3) +
  guides(color = "none") +
  facet_wrap(~ consent_law,
             ncol = 1,
             scales = "free_y") +
  labs(x = "Donor Procurement Rate",
       y = NULL,
       color = "Consent Law")
```

Normally the point of a facet is to preserve comparability between panels by not allowing the scales to vary. But for categorical measures it can be useful to allow this.

Allow the y-scale to vary

```
by_country ▷
  ggplot(mapping =
           aes(x = donors_mean,
               y = reorder(country, donors_mean),
               color = consent_law)) +
  geom_point(size=3) +
  guides(color = "none") +
  facet_wrap(~ consent_law,
             ncol = 1,
             scales = "free_y") +
  labs(x = "Donor Procurement Rate",
       y = NULL,
       color = "Consent Law")
```

Normally the point of a facet is to preserve comparability between panels by not allowing the scales to vary. But for categorical measures it can be useful to allow this.

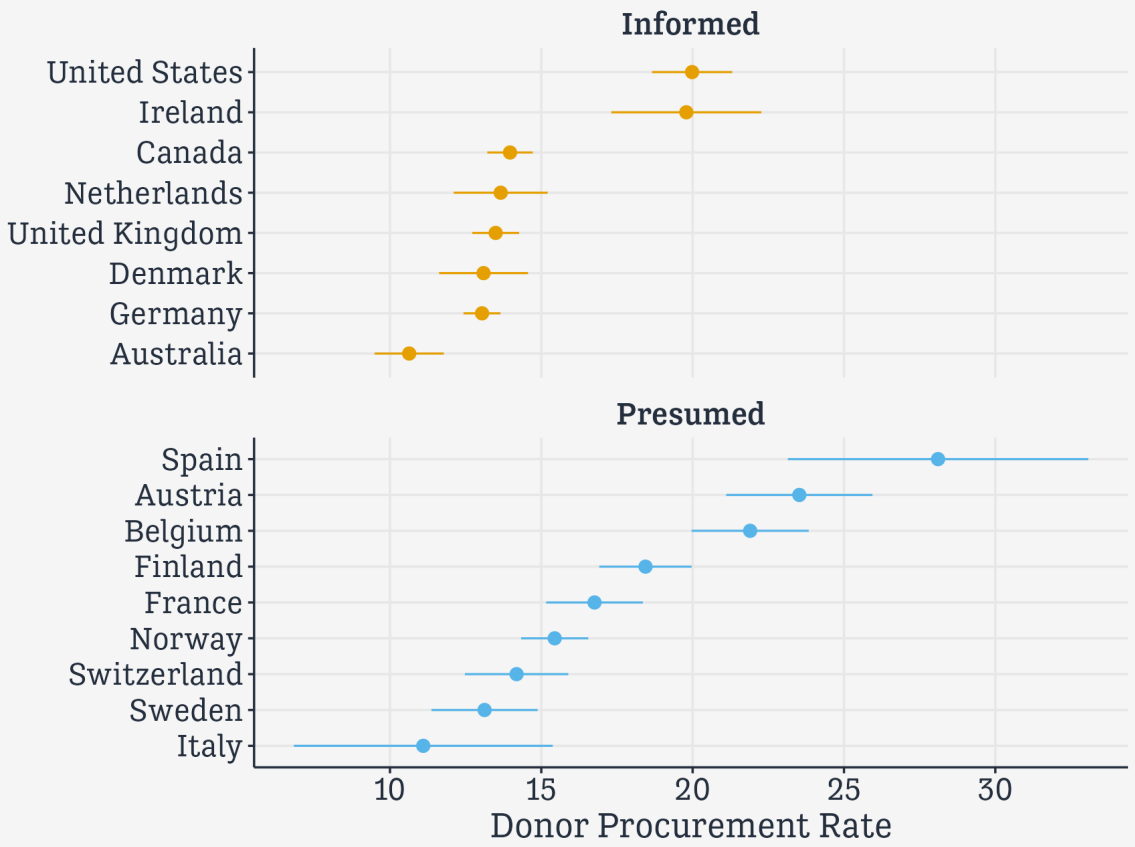


Again, these methods are general

```
by_country ▷
  ggplot(mapping =
           aes(x = donors_mean,
               y = reorder(country, donors_mean),
               color = consent_law)) +
  geom_pointrange(mapping =
                    aes(xmin = donors_mean - donors_sd,
                        xmax = donors_mean + donors_sd)) +
  guides(color = "none") +
  facet_wrap(~ consent_law,
             ncol = 1,
             scales = "free_y") +
  labs(x = "Donor Procurement Rate",
       y = NULL,
       color = "Consent Law")
```

Again, these methods are general

```
by_country ▷
  ggplot(mapping =
           aes(x = donors_mean,
               y = reorder(country, donors_mean),
               color = consent_law)) +
  geom_pointrange(mapping =
                    aes(xmin = donors_mean - donors_sd,
                        xmax = donors_mean + donors_sd)) +
  guides(color = "none") +
  facet_wrap(~ consent_law,
             ncol = 1,
             scales = "free_y") +
  labs(x = "Donor Procurement Rate",
       y = NULL,
       color = "Consent Law")
```



Plot text directly

`geom_text()` for basic labels is very limited

We'll use `ggrepel` instead

The `ggrepel` package provides `geom_text_repel()` and `geom_label_repel()`

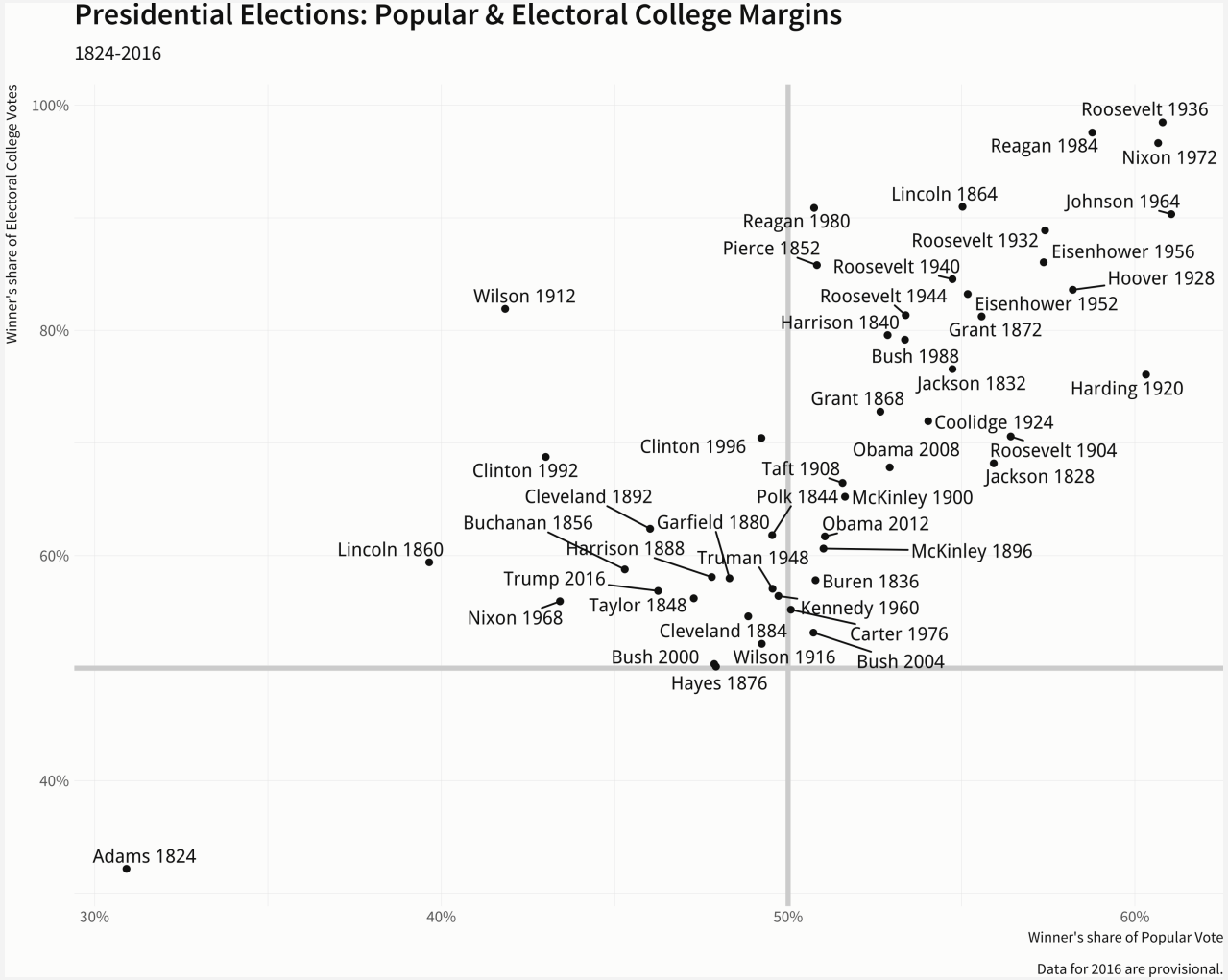
U.S. Historic Presidential Elections

elections_historic is in socviz

```
elections_historic
```

```
## # A tibble: 49 × 19
##    election  year winner      win_party ec_pct popular_pct popular_margin  votes
##       <int> <int> <chr>       <chr>      <dbl>       <dbl>          <dbl>  <int>
##  1       10  1824 John Quinc… D.-R.      0.322       0.309         -0.104  1.13e5
##  2       11  1828 Andrew Jac… Dem.       0.682       0.559          0.122  6.43e5
##  3       12  1832 Andrew Jac… Dem.       0.766       0.547          0.178  7.03e5
##  4       13  1836 Martin Van… Dem.       0.578       0.508          0.142  7.63e5
##  5       14  1840 William He… Whig       0.796       0.529          0.0605 1.28e6
##  6       15  1844 James Polk  Dem.       0.618       0.495          0.0145 1.34e6
##  7       16  1848 Zachary Ta… Whig       0.562       0.473          0.0479 1.36e6
##  8       17  1852 Franklin P… Dem.       0.858       0.508          0.0695 1.61e6
##  9       18  1856 James Buch… Dem.       0.588       0.453          0.122  1.84e6
## 10       19  1860 Abraham Li… Rep.       0.594       0.396          0.101  1.86e6
## # ℹ 39 more rows
## # ℹ 11 more variables: margin <int>, runner_up <chr>, ru_part <chr>,
## #   turnout_pct <dbl>, winner_lname <chr>, winner_label <chr>, ru_lname <chr>,
## #   ru_label <chr>, two_term <lgl>, ec_votes <dbl>, ec_denom <dbl>
```

We'll draw a plot like this



Keep things neat

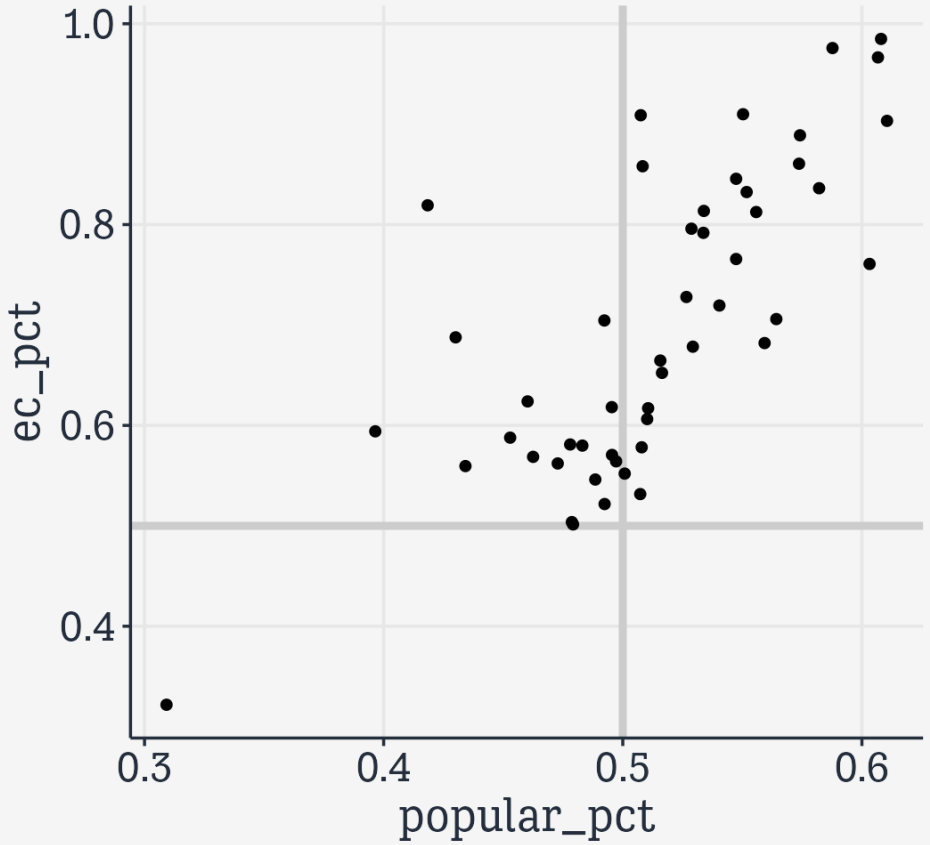
```
## The packages we'll use in addition to ggplot
library(ggrepel)
library(scales)

p_title ← "Presidential Elections: Popular & Electoral College Margins"
p_subtitle ← "1824-2016"
p_caption ← "Data for 2016 are provisional."
x_label ← "Winner's share of Popular Vote"
y_label ← "Winner's share of Electoral College Votes"
```

Base Layer, Lines, Points

```
p ← ggplot(data = elections_historic,
           mapping = aes(x = popular_pct,
                         y = ec_pct,
                         label = winner_label))

p + geom_hline(yintercept = 0.5,
               linewidth = 1.4,
               color = "gray80") +
    geom_vline(xintercept = 0.5,
               linewidth = 1.4,
               color = "gray80") +
    geom_point()
```

Base Layer, Lines, Points

```
p ← ggplot(data = elections_historic,
           mapping = aes(x = popular_pct,
                         y = ec_pct,
                         label = winner_label))

p + geom_hline(yintercept = 0.5,
               linewidth = 1.4,
               color = "gray80") +
    geom_vline(xintercept = 0.5,
               linewidth = 1.4,
               color = "gray80") +
    geom_point()
```



Add the labels

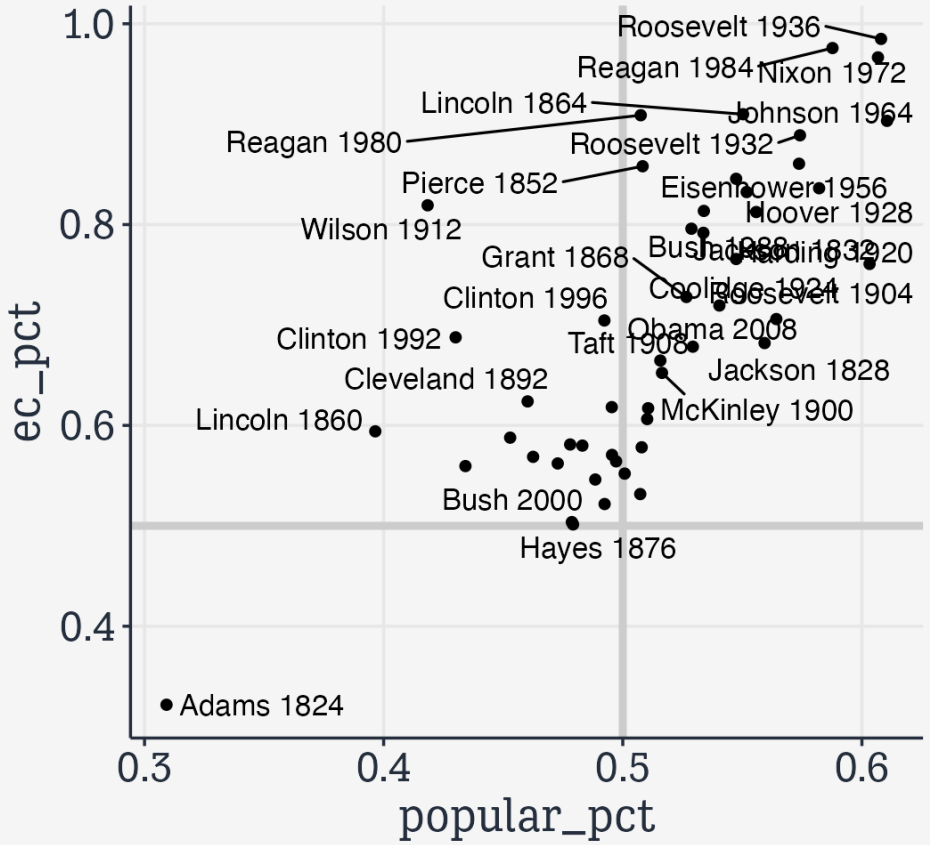
```
p ← ggplot(data = elections_historic,
           mapping = aes(x = popular_pct,
                         y = ec_pct,
                         label = winner_label))

p + geom_hline(yintercept = 0.5,
               linewidth = 1.4, color = "gray80") +
  geom_vline(xintercept = 0.5,
             linewidth = 1.4, color = "gray80") +
  geom_point() +
  geom_text_repel()
```

This looks messy because `geom_text_repel()` uses the dimensions of the available graphics device to iteratively figure out the labels. Let's allow it to draw on the whole slide.

Add the labels

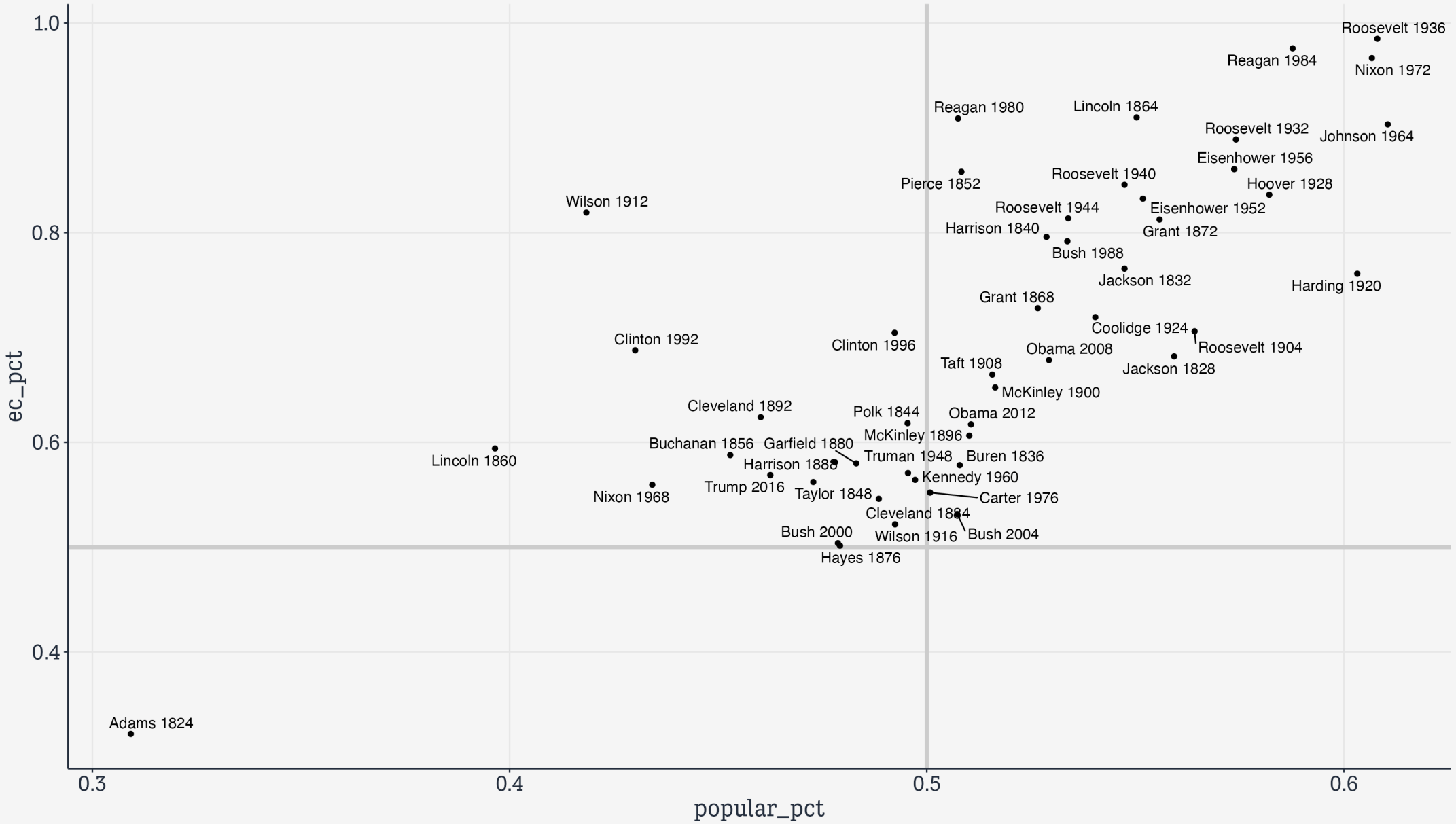
```
p ← ggplot(data = elections_historic,
           mapping = aes(x = popular_pct,
                         y = ec_pct,
                         label = winner_label))

p + geom_hline(yintercept = 0.5,
               linewidth = 1.4, color = "gray80") +
  geom_vline(xintercept = 0.5,
             linewidth = 1.4, color = "gray80") +
  geom_point() +
  geom_text_repel()
```

This looks messy because `geom_text_repel()` uses the dimensions of the available graphics device to iteratively figure out the labels. Let's allow it to draw on the whole slide.



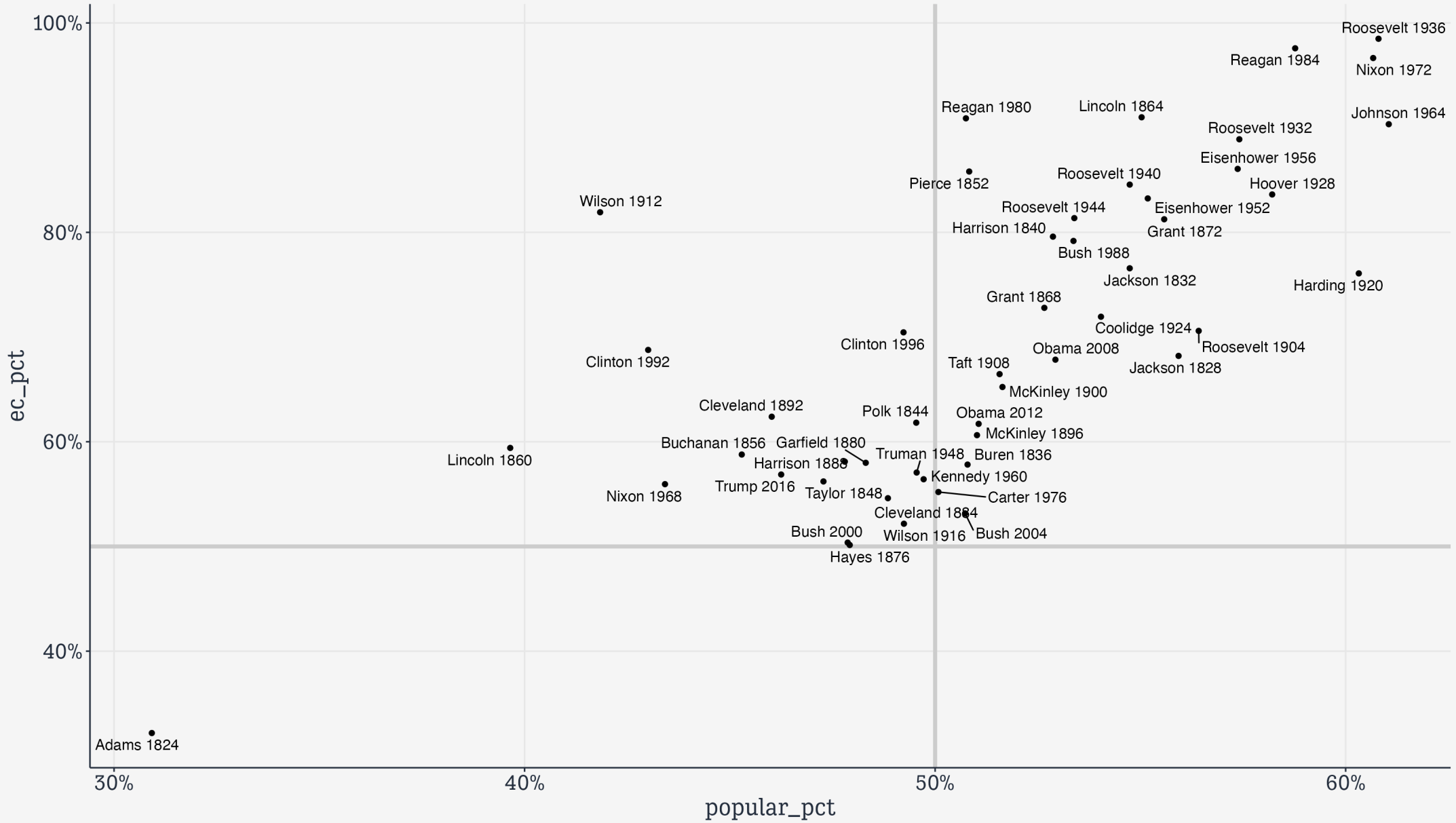
The labeling is with respect to the plot size

```
p ← ggplot(data = elections_historic,
           mapping  = aes(x = popular_pct,
                          y = ec_pct,
                          label = winner_label))

p_out ← p +
  geom_hline(yintercept = 0.5,
             linewidth = 1.4,
             color = "gray80") +
  geom_vline(xintercept = 0.5,
             linewidth = 1.4,
             color = "gray80") +
  geom_point() +
  geom_text_repel()
```


1.0
0.8
0.6
0.4
ec_pct
0.3
0.4
0.5
0.6
popular_pct
Roosevelt 1936
Reagan 1984
Nixon 1972
Reagan 1980
Lincoln 1864
Roosevelt 1932
Johnson 1964
Eisenhower 1956
Pierce 1852
Roosevelt 1940
Hoover 1928
Wilson 1912
Roosevelt 1944
Eisenhower 1952
Harrison 1840
Grant 1872
Bush 1988
Jackson 1832
Harding 1920
Grant 1868
Coolidge 1924
Clinton 1992
Clinton 1996
Obama 2008
Roosevelt 1904
Taft 1908
Jackson 1828
McKinley 1900
Cleveland 1892
Polk 1844
Obama 2012
McKinley 1896
Buchanan 1856
Garfield 1880
Truman 1948
Buren 1836
Lincoln 1860
Harrison 1888
Kennedy 1960
Trump 2016
Taylor 1848
Carter 1976
Nixon 1968
Cleveland 1884
Bush 2000
Wilson 1916
Bush 2004
Hayes 1876
Adams 1824

Adjust the Scales

```
p ← ggplot(data = elections_historic,
           mapping  = aes(x = popular_pct,
                          y = ec_pct,
                          label = winner_label))
p_out ← p + geom_hline(yintercept = 0.5,
                       linewidth = 1.4,
                       color = "gray80") +
    geom_vline(xintercept = 0.5,
               linewidth = 1.4,
               color = "gray80") +
    geom_point() +
    geom_text_repel() +
    scale_x_continuous(labels = label_percent()) +
    scale_y_continuous(labels = label_percent())
```

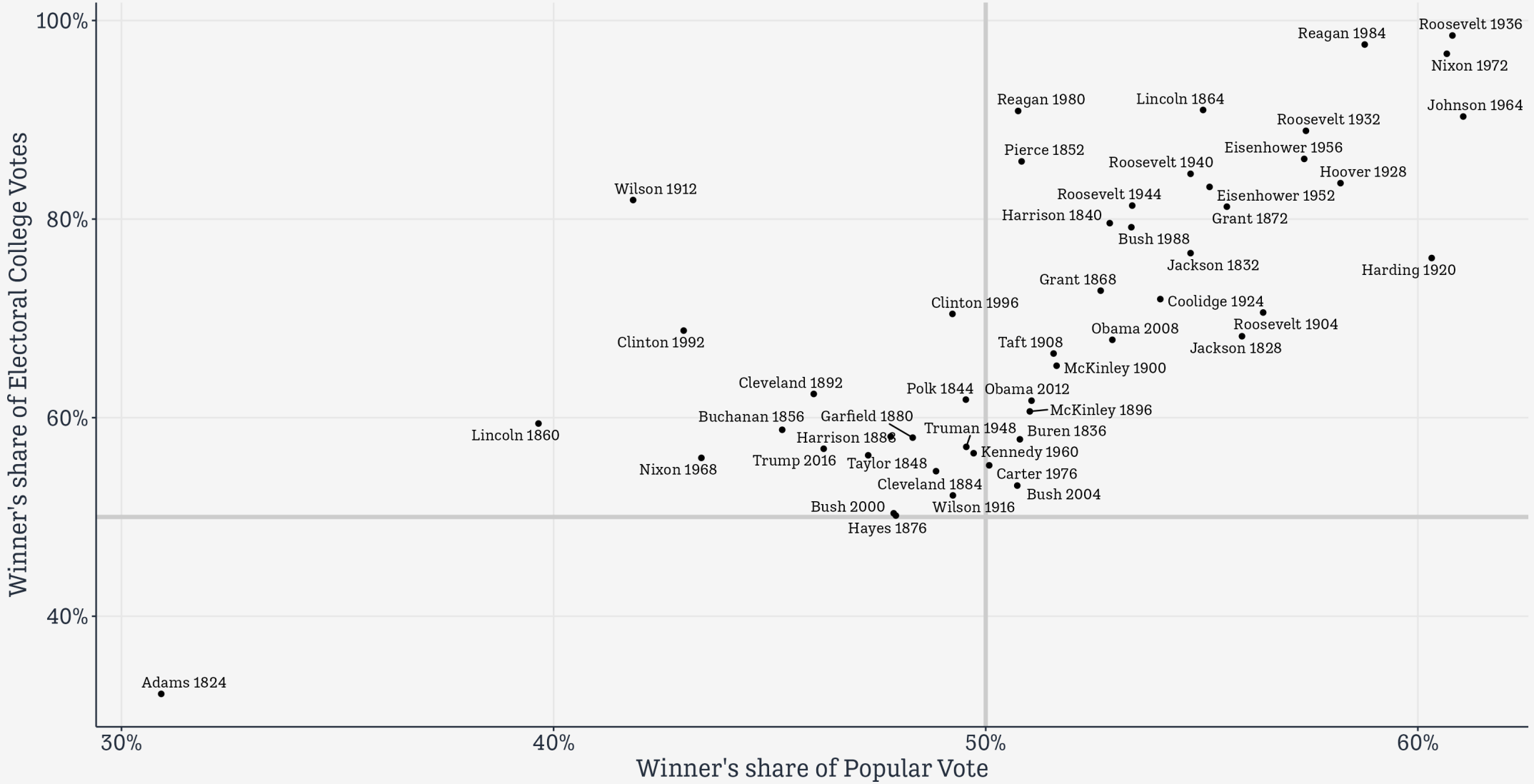

100%
80%
60%
40%
ec_pct
30%
40%
50%
60%
popular_pct
Roosevelt 1936
Reagan 1984
Nixon 1972
Reagan 1980
Lincoln 1864
Johnson 1964
Roosevelt 1932
Eisenhower 1956
Pierce 1852
Roosevelt 1940
Hoover 1928
Wilson 1912
Roosevelt 1944
Eisenhower 1952
Harrison 1840
Grant 1872
Bush 1988
Jackson 1832
Harding 1920
Grant 1868
Coolidge 1924
Clinton 1996
Roosevelt 1904
Obama 2008
Clinton 1992
Taft 1908
Jackson 1828
McKinley 1900
Cleveland 1892
Polk 1844
Obama 2012
McKinley 1896
Buchanan 1856
Garfield 1880
Truman 1948
Buren 1836
Lincoln 1860
Harrison 1888
Kennedy 1960
Trump 2016
Taylor 1848
Carter 1976
Nixon 1968
Cleveland 1884
Bush 2000
Wilson 1916
Bush 2004
Hayes 1876
Adams 1824

Add the labels

```
p ← ggplot(data = elections_historic,
           mapping  = aes(x = popular_pct,
                          y = ec_pct,
                          label = winner_label))
p_out ← p + geom_hline(yintercept = 0.5,
                       linewidth = 1.4,
                       color = "gray80") +
  geom_vline(xintercept = 0.5,
             linewidth = 1.4,
             color = "gray80") +
  geom_point() +
  geom_text_repel(mapping = aes(family = "Tenso Slide")) +
  scale_x_continuous(labels = label_percent()) +
  scale_y_continuous(labels = label_percent()) +
  labs(x = x_label, y = y_label,
       title = p_title,
       subtitle = p_subtitle,
       caption = p_caption)
```

Presidential Elections: Popular & Electoral College Margins

1824-2016



Data for 2016 are provisional.