Dataviz course

Lasse Gullvåg Sætre

08-16-2017

Kieran explains Markdown

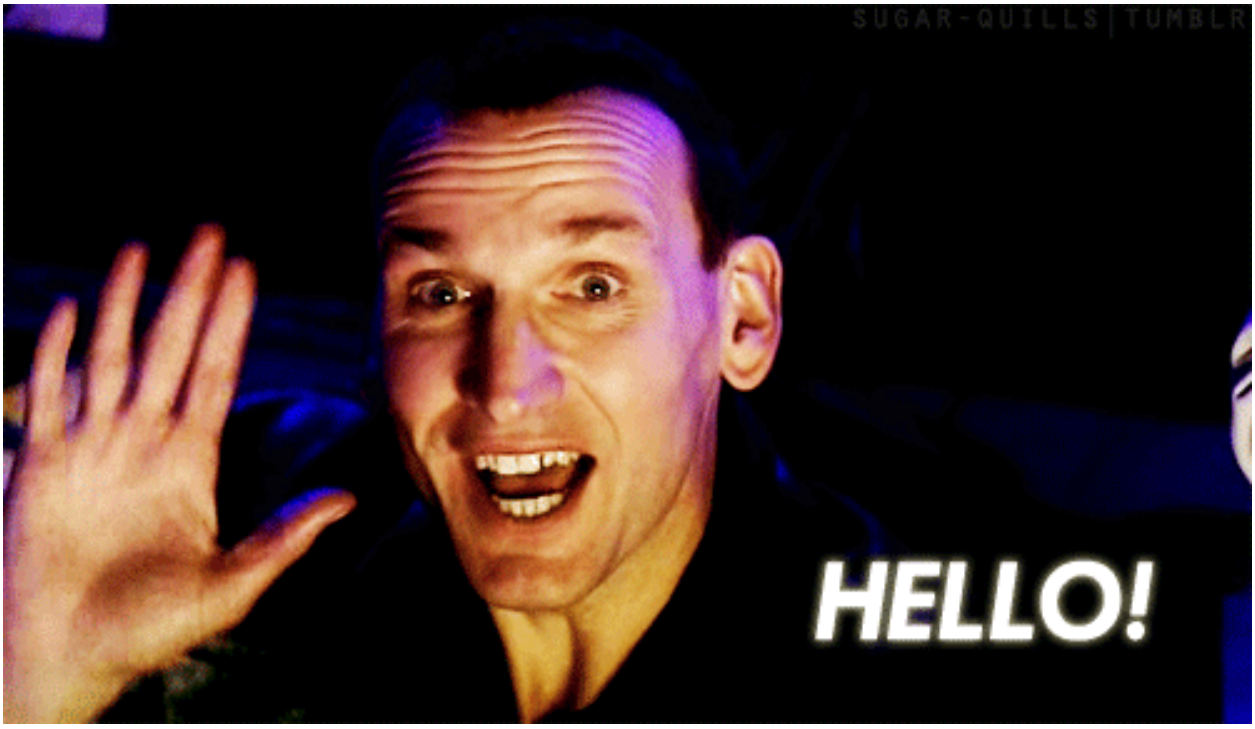


Figure 1: R Markdown, seems like regular old Markdown with R capabilities

Packages used for this course

```
my_packages <- c("tidyverse", "broom", "coefplot", "dotwhisker",

"gapminder", "GGally", "ggrepel", "gridExtra", "interplot",
"margins", "maps", "mapproj", "mapdata", "MASS",
"quantreg", "scales", "survey", "viridis",
"viridisLite", "devtools")

install.packages(my_packages,

repos = "http://cran.rstudio.com")

devtools::install_github("kjhealy/socviz")
```

So let's try to load the tidyverse

Tidyverse

```
  library("tidyverse")
```

```
## Loading tidyverse: ggplot2
## Loading tidyverse: tibble
## Loading tidyverse: tidyr
## Loading tidyverse: readr
## Loading tidyverse: purrr
## Loading tidyverse: dplyr
```

```
## Conflicts with tidy packages ----------------------------------------------
```

```
## filter(): dplyr, stats
## lag():    dplyr, stats
```

```
  library("socviz")
```

What R code looks like

R is a object oriented language. Everything has a name. Some names are forbidden. Things that are reserved for the core features of the language, like true or the plus symbol. Use naming conventions for the objects you're working with.

```
my_numbers <- c(1, 2, 3, 1, 3, 5, 25)
```

The <- thing is the assign parameter, interpret it as "gets". it performs the action of creating objects. Use the keyboard shortcut alt dash.

C is short for concoctanate. Takes comma-seperated numbers or strings and joins them together into a vector.

So let's do this. The mean function:

```
mean(x = my_numbers)
```

```
## [1] 5.714286
```

All functions have paranthesis. This is where the inputs go. The mean function takes one required argument. Their names are internal to their functions (objects you've created outside the function wont intefer).

Functions take inputs, perform actions, produce outputs.

The first argument of mean is X. If you don't specify X, it will work anyway?

```
mean(my_numbers)
```

```
## [1] 5.714286
```

You can assign a functiuons output to a named object

So these are all functions:

```
class(my_numbers)
```

```
## [1] "numeric"
```

```
str(my_numbers)
```

```
##  num [1:7] 1 2 3 1 3 5 25
```

```
table(my_numbers)
```

```
## my_numbers
##  1  2  3  5 25
##  2  1  2  1  1
```

```
x <- c(my_numbers, 5)
y <- c(my_numbers, "hello")

print(y)
```

```
## [1] "1"     "2"     "3"     "1"     "3"     "5"     "25"    "hello"
```

```
class(y)
```

```
## [1] "character"
```

Functions can also be nested. And will be ecaluated from the inside out. It starts with the innermost and gives it off to the next one outside of it.

```
mean(c(my_numbers, x))
```

```
## [1] 5.666667
```

Every object has at least one class.

Vectors are sequences of different data. Numeric, character, factor. Arrays are tables in a way: Matrix, data.frame, tibble. Functions are also their own class Models are also a class

Everything has a name, everything is a object. Every object has a class.

Get Working Directory is useful, at least outside R studio.

```
getwd()
```

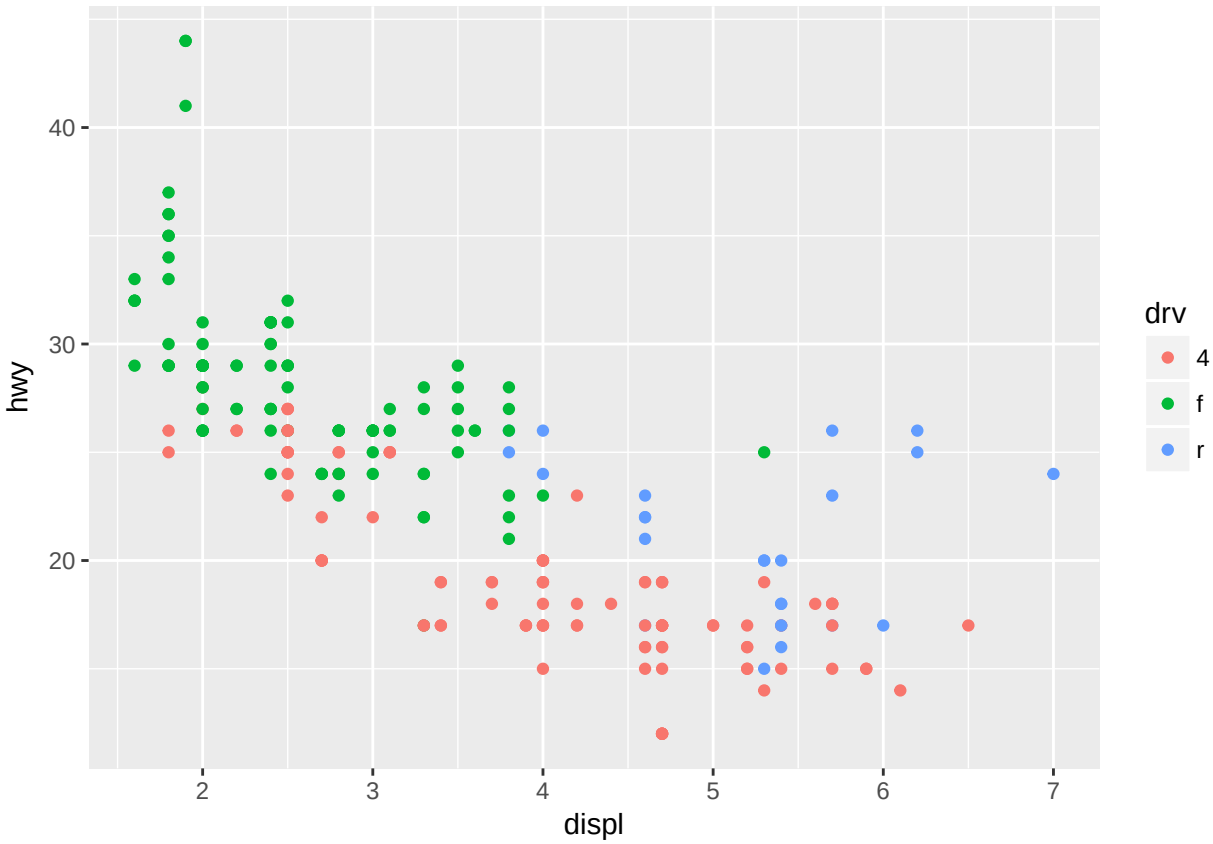
```
## [1] "/home/lassegs/Documents/sos9028/notes/rmd"
```

R will be frustrating

We're going to be adding a lot of objects together. When you type your codde out, you add objects to each other. Make sure that the + symbol is at the end of the line. It does not like it when you start lines with any arithmetic operator.

Let's do a ggplot

```
p <- ggplot(data = mpg,
            mapping = aes(x = displ,
                          y = hwy,
                          color = drv))
p + geom_point()
```



named things gets the output of this function with these arguments. ^

```
library(gapminder)
head(gapminder)
```

```
## # A tibble: 6 x 6
##       country continent  year lifeExp      pop gdpPercap
##        <fctr>    <fctr> <int>   <dbl>    <int>     <dbl>
## 1 Afghanistan      Asia  1952  28.801  8425333  779.4453
## 2 Afghanistan      Asia  1957  30.332  9240934  820.8530
## 3 Afghanistan      Asia  1962  31.997 10267083  853.1007
## 4 Afghanistan      Asia  1967  34.020 11537966  836.1971
## 5 Afghanistan      Asia  1972  36.088 13079460  739.9811
## 6 Afghanistan      Asia  1977  38.438 14880372  786.1134
```

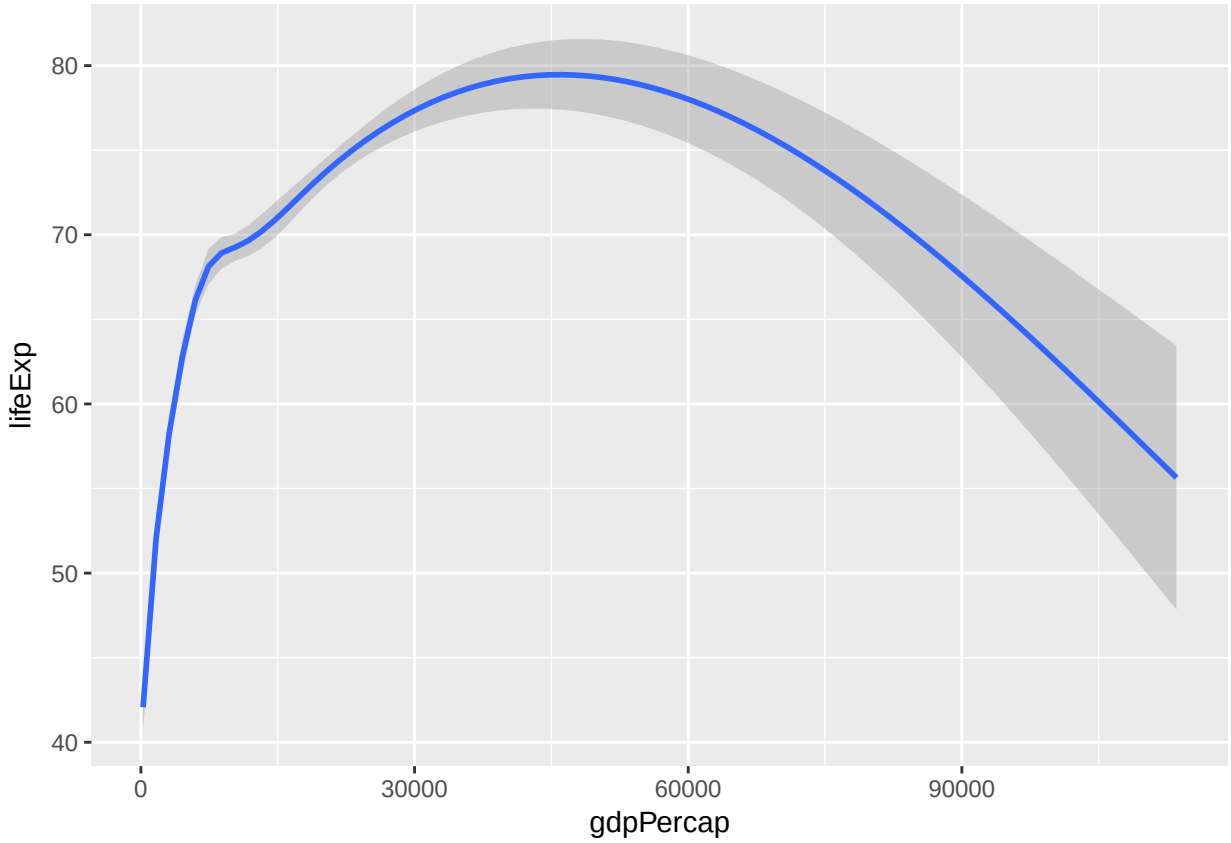
```
tail(gapminder)
```

```
## # A tibble: 6 x 6
##    country continent  year lifeExp      pop gdpPercap
##     <fctr>    <fctr> <int>   <dbl>    <int>     <dbl>
## 1 Zimbabwe    Africa  1982  60.363  7636524  788.8550
## 2 Zimbabwe    Africa  1987  62.351  9216418  706.1573
## 3 Zimbabwe    Africa  1992  60.377 10704340  693.4208
## 4 Zimbabwe    Africa  1997  46.809 11404948  792.4500
## 5 Zimbabwe    Africa  2002  39.989 11926563  672.0386
## 6 Zimbabwe    Africa  2007  43.487 12311143  469.7093
```

```
p <- ggplot(data = gapminder,
            mapping = aes(x = gdpPercap,
```

```
                          y = lifeExp))
p + geom_smooth()
```

```
## `geom_smooth()` using method = 'gam'
```



What data am I using? What variables am I plotting? What geometry do I wanna make?

Tidy Data

Data in long format vs data in wide format. GGPlot wants your data in long format. If the data is in the right shape, everything goes much more smoothly. wants two-dimensional data, with variables in the columns, observations in the rows.

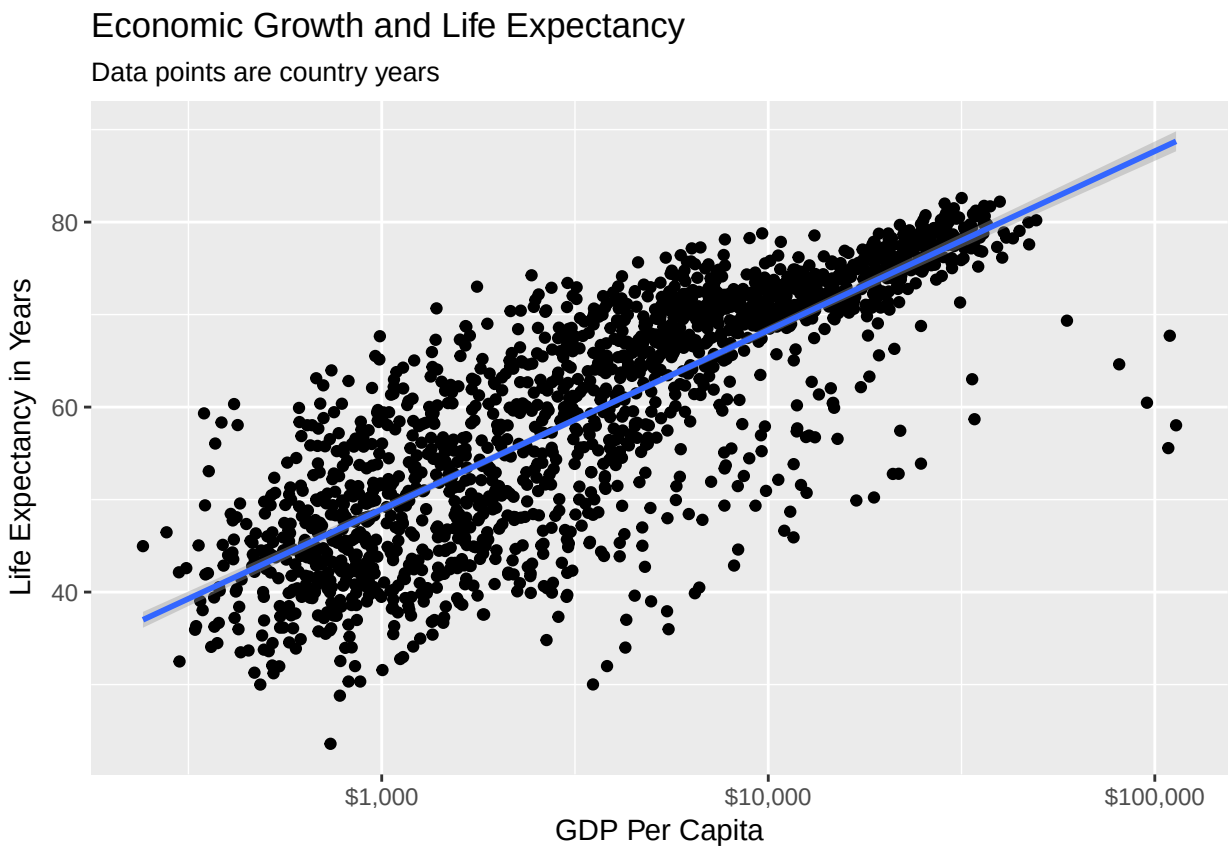
Loaded into R it will be object with a name. A data.frame is like a table. Same as matrix, except matrix only has numbers. A data.frame can contain variables of different types (numerical, characters, categorical, dates).

TidyR is the tool to try to detangle untidy data.

Mappings link data to things you see on the plot. This is in the aes mapping. X and Y are the most obvious. Other aesthetic mappings can include, eg., color, shape and size. Think about the logical relationship between the variable and the thing (e.g. color) representing it.

```
library(gapminder)
p <- ggplot(data=gapminder,
            mapping = aes( x = gdpPercap,
                           y = lifeExp))
p + geom_point() +
    geom_smooth(method='gam') +
```

```
    scale_x_log10(labels = scales::dollar) +
    labs(x = "GDP Per Capita",
         y = "Life Expectancy in Years",
         title = "Economic Growth and Life Expectancy",
         subtitle = "Data points are country years")
```



Mapping vs setting aesthetics

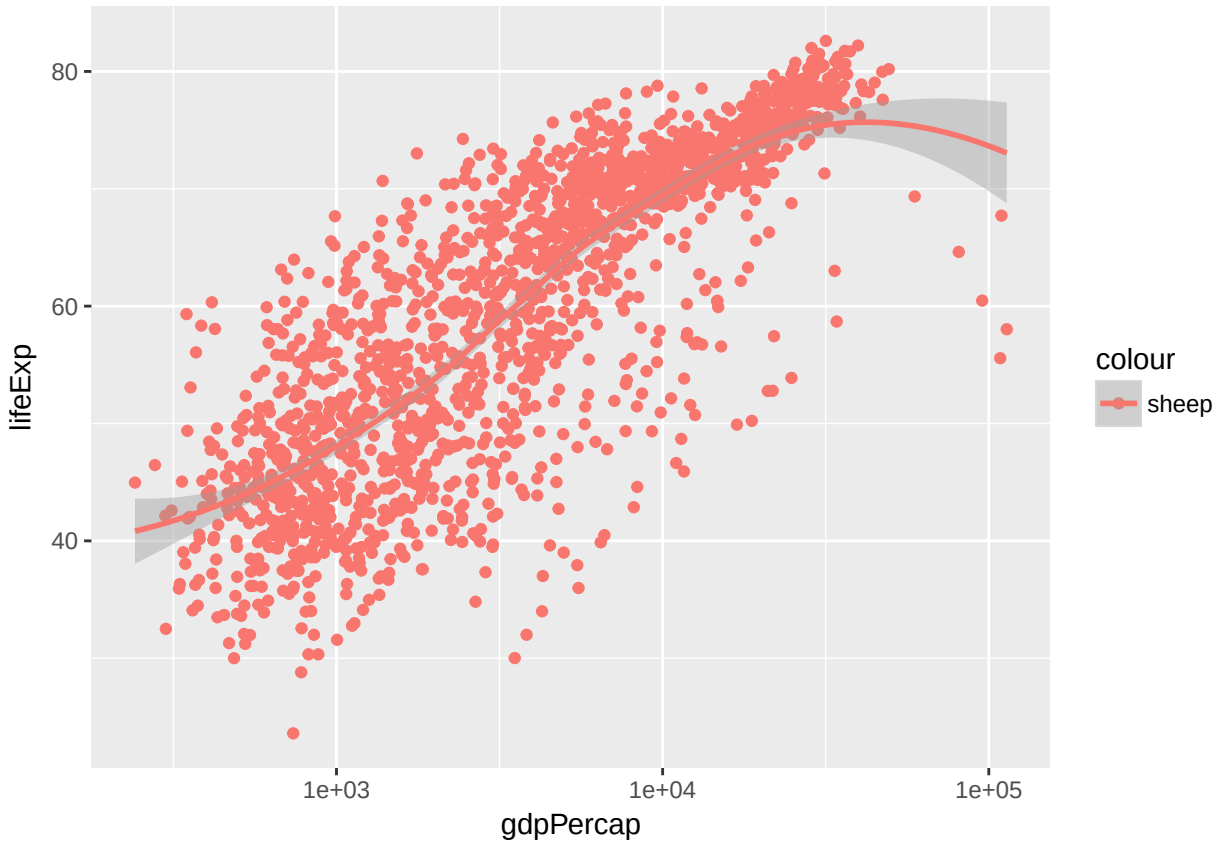
Crucial distinction. Makes everything clearer.

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

p + geom_point() +
    geom_smooth() +
    scale_x_log10()
```

```
## `geom_smooth()` using method = 'gam'
```

Mapping defines the relationship between a variable and the aesthetic property of the graph. Therefor mapping color = "purple" makes it try to map the variable purple to color, but there is no such variable. So it creates one on the fly, and what you get is a new variable with the only value is "purple". That's a biproduct of something which actually is useful: You can specify variables on the fly. This is the degenerate case of that.

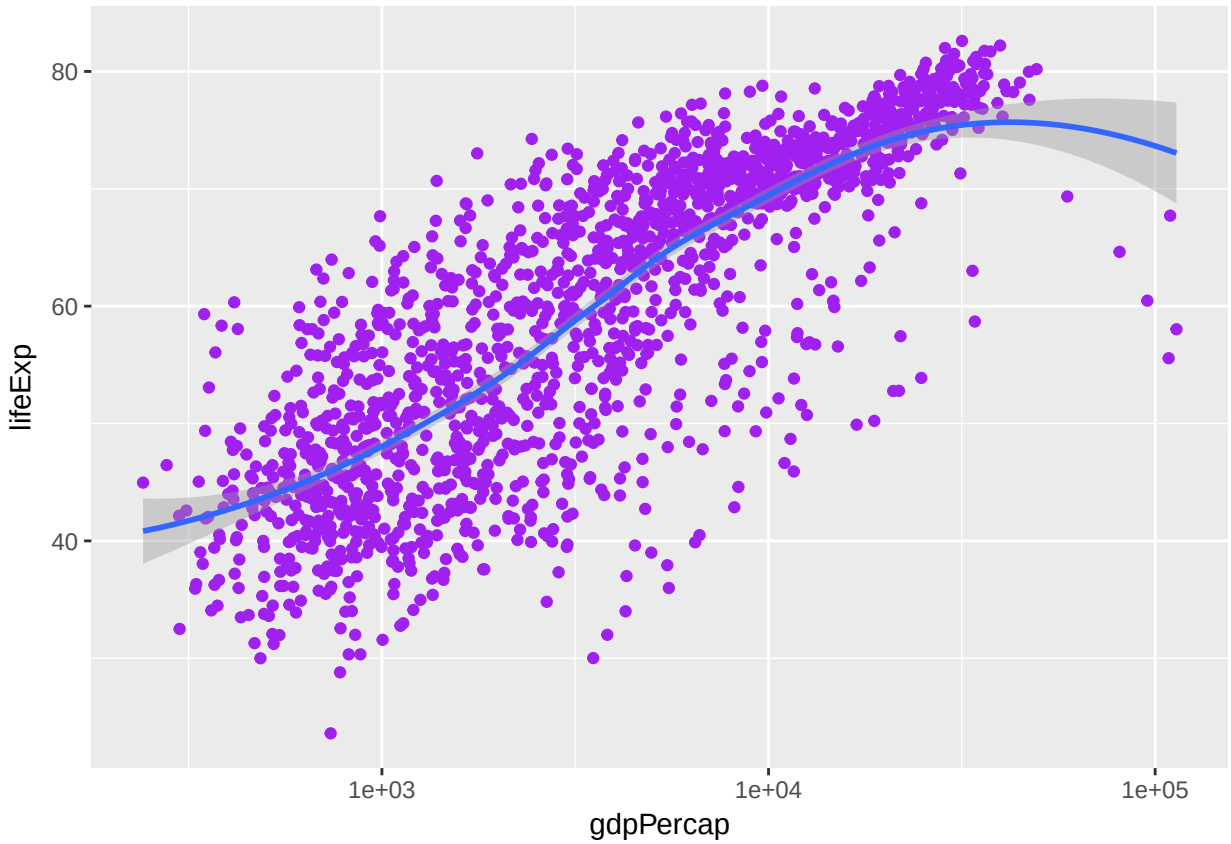
```
p <- ggplot(data = gapminder,
            mapping = aes(x = gdpPercap,
                          y = lifeExp,
                          color = "sheep"))
p + geom_point() +
  geom_smooth(method='loess') +
  scale_x_log10()
```



Doesn't matter what you call it, it will still be painted the first color: Red.

The way to do it is to **set** it:

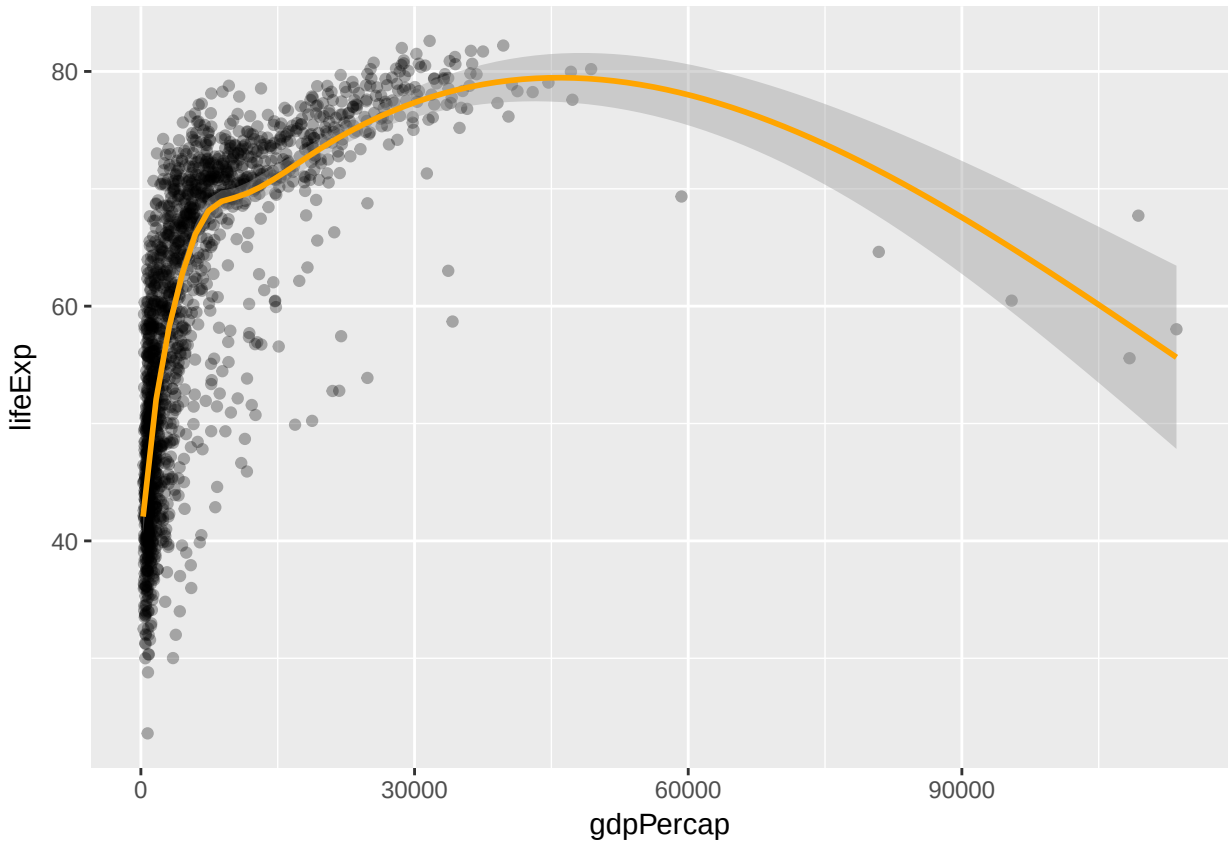
```
p <- ggplot(data = gapminder,
            mapping = aes(x = gdpPercap,
                          y = lifeExp))
p + geom_point(color = "purple") +
    geom_smooth(method='loess') +
    scale_x_log10()
```



This is not useful representationally. But we can do more.

```
p <- ggplot(data = gapminder,
            mapping = aes(x = gdpPercap,
                          y = lifeExp))
p + geom_point(alpha= '0.3') +
    geom_smooth(color = "orange")
```

```
## `geom_smooth()` using method = 'gam'
```

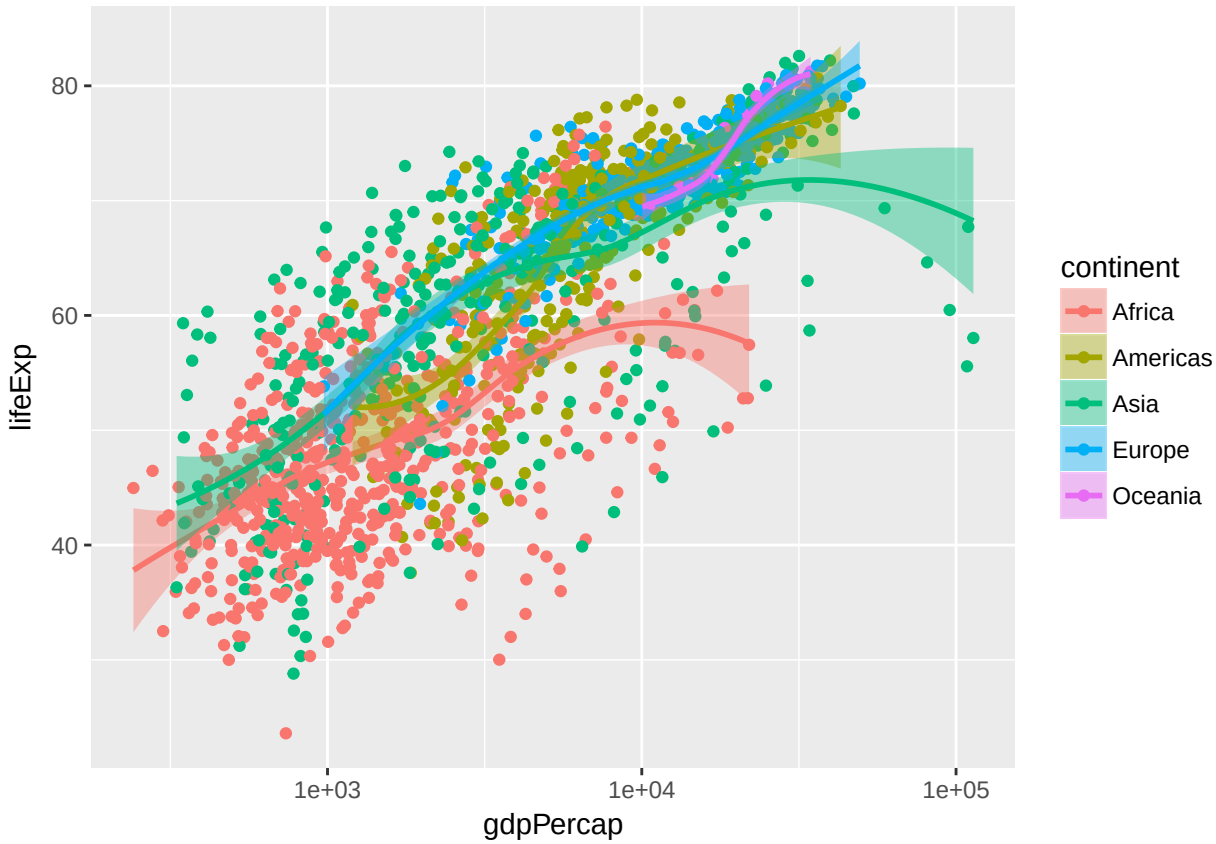



```
    scale_x_log10()
```

```
## <ScaleContinuousPosition>
##  Range:
##  Limits:    0 --    1
```

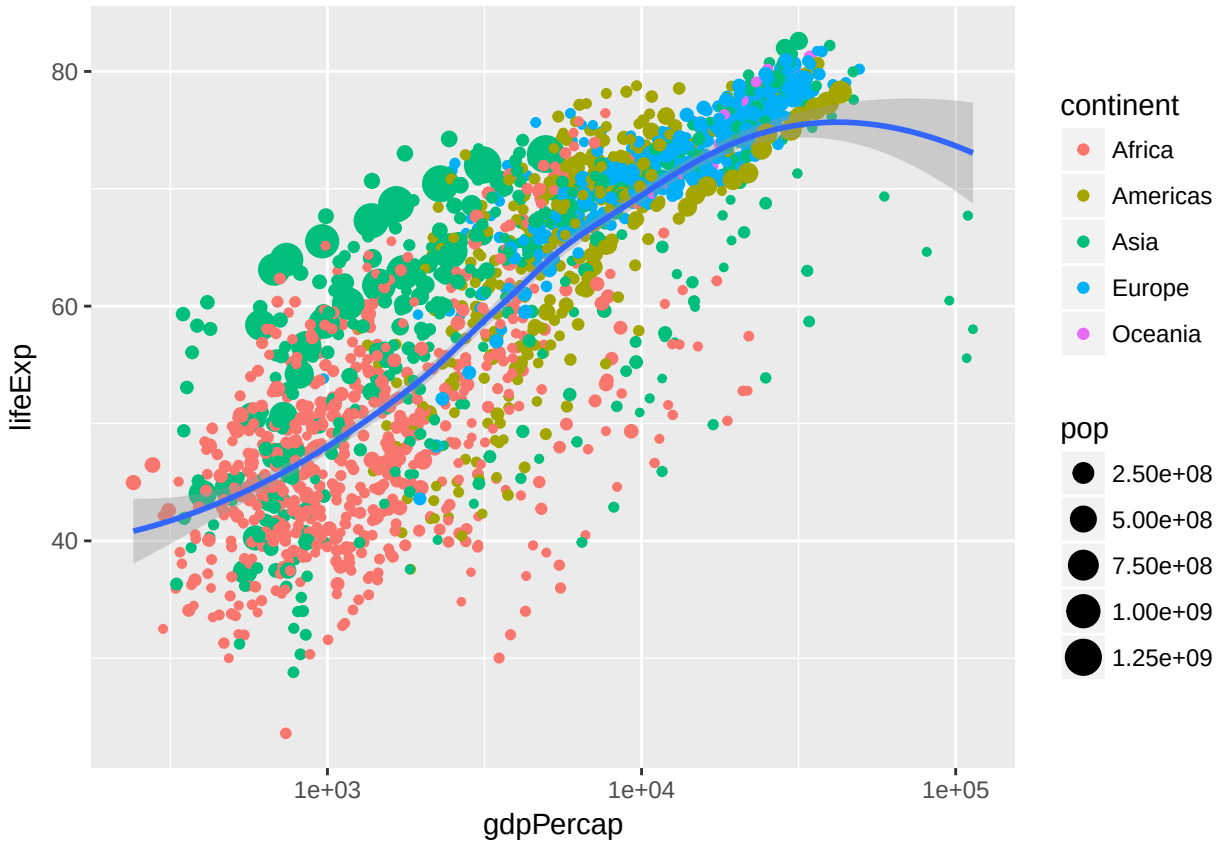
Now lets try to map it reasonably:

```
p <- ggplot(data = gapminder,
            mapping = aes(x = gdpPercap,
                          y = lifeExp,
                          color = continent,
                          fill = continent))
p + geom_point() +
    geom_smooth(method = 'loess') +
    scale_x_log10()
```



Mapping in the ggplot baselayer, it sets global mappings inherited by everything else. But you can set mapping geom per goem, by layers.

```
p <- ggplot(data = gapminder,
            mapping = aes(x = gdpPercap,
                          y = lifeExp))
p + geom_point(mapping =
                 aes(color = continent,
                     size = pop)) +
  geom_smooth(method = 'loess') +
  scale_x_log10()
```



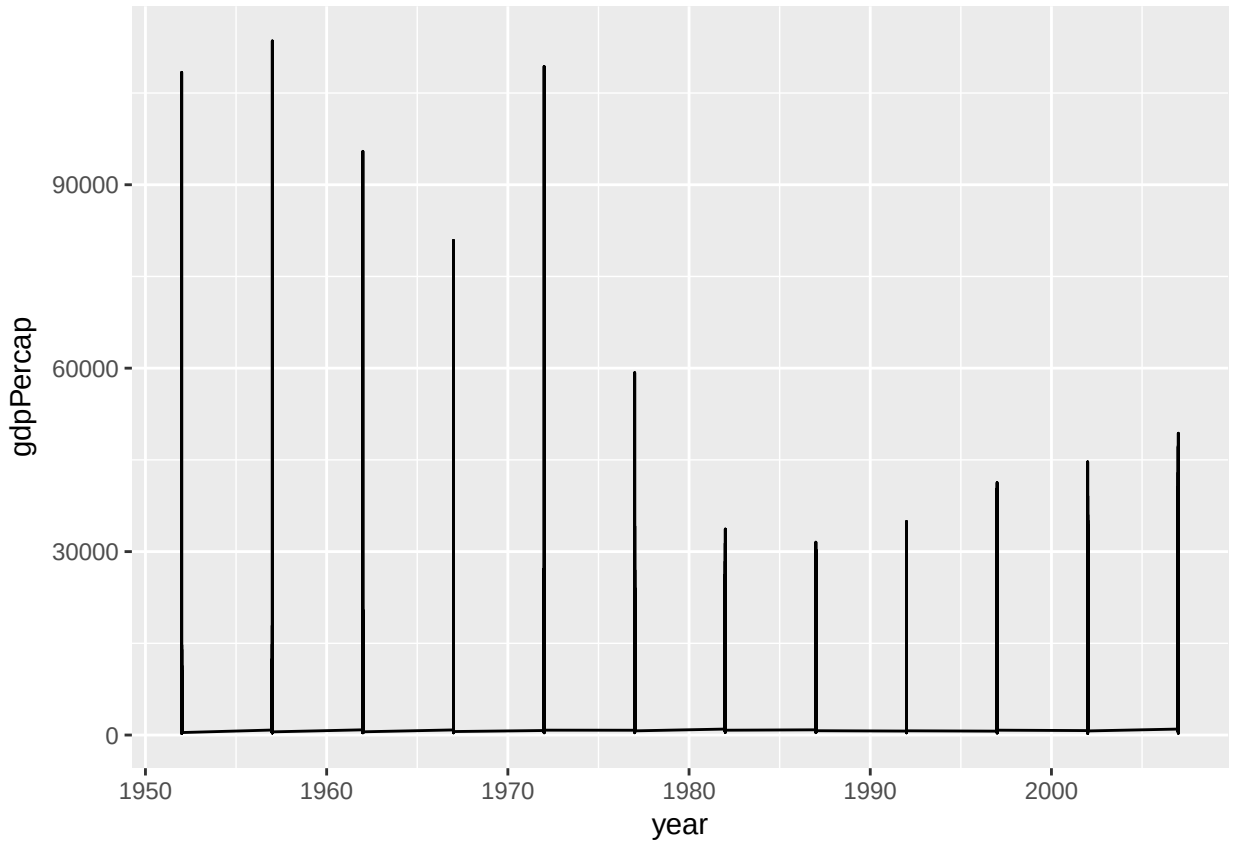
So both mappings can be set on both a base and layer basis.

Grammar

The grammar is a set of rules for how to produce graphics from data, taking pieces of data and mapping them to geometric objects (like points and lines) that have aesthetic qualities.

> Like other rules of synstax, the grammar limits what you can say, but doesn't make what you say sensible or meaningful. Noam Chomsky

```
p <- ggplot(data = gapminder,
            mapping = aes(x = year,
                          y = gdpPercap))
p + geom_line()
```



This is syntactically correct, but does not make sense. What is happening here? You know what the gapminder dataset looks like

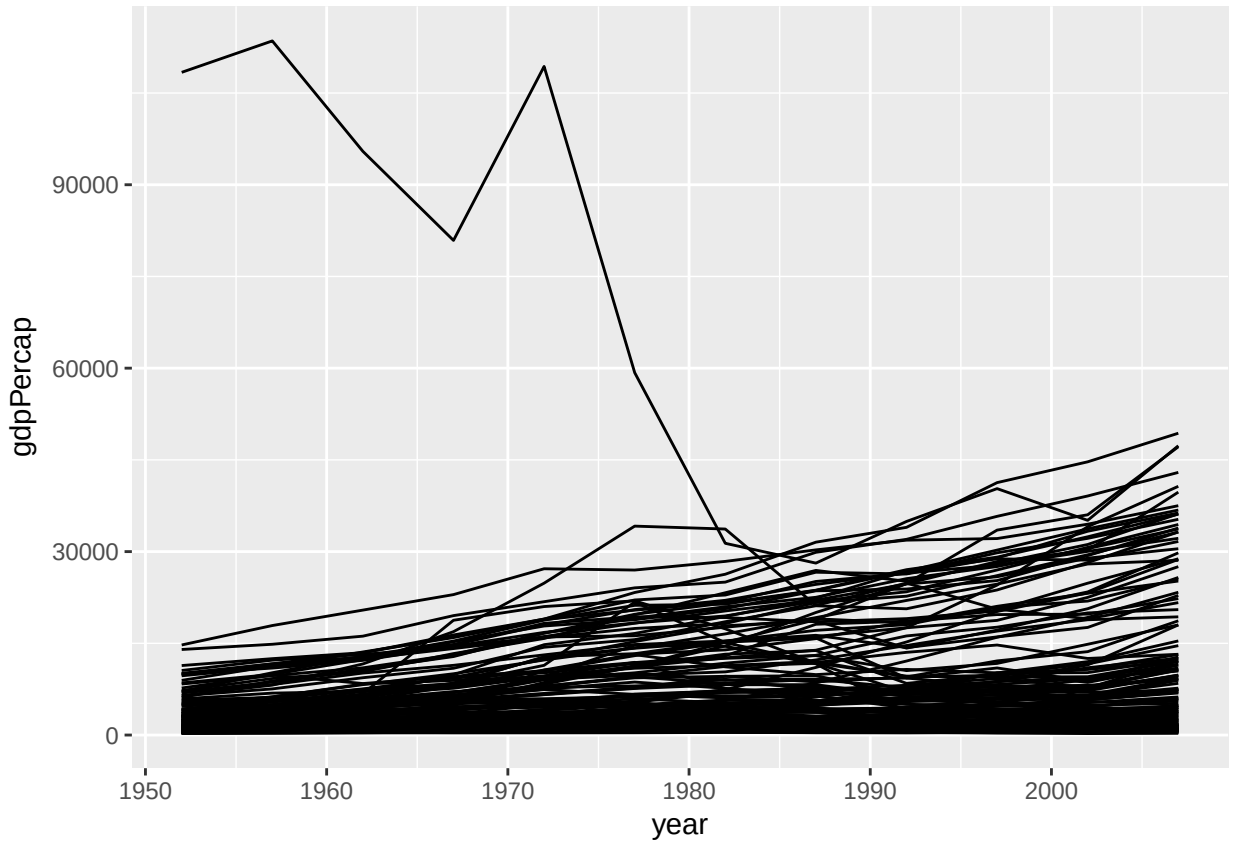
```
head(gapminder)
```

```
## # A tibble: 6 x 6
##       country continent  year lifeExp      pop gdpPercap
##        <fctr>    <fctr> <int>   <dbl>    <int>     <dbl>
## 1 Afghanistan      Asia  1952  28.801  8425333  779.4453
## 2 Afghanistan      Asia  1957  30.332  9240934  820.8530
## 3 Afghanistan      Asia  1962  31.997 10267083  853.1007
## 4 Afghanistan      Asia  1967  34.020 11537966  836.1971
## 5 Afghanistan      Asia  1972  36.088 13079460  739.9811
## 6 Afghanistan      Asia  1977  38.438 14880372  786.1134
```

ggplot does not infer anything else about the data. geom_line does not know about the structure of the data, and therefore does not know about the grouping of country from country. Does not know about the *groupiness*. Its faithfully joining up all the years, but does not take into account that the data is grouped into country.

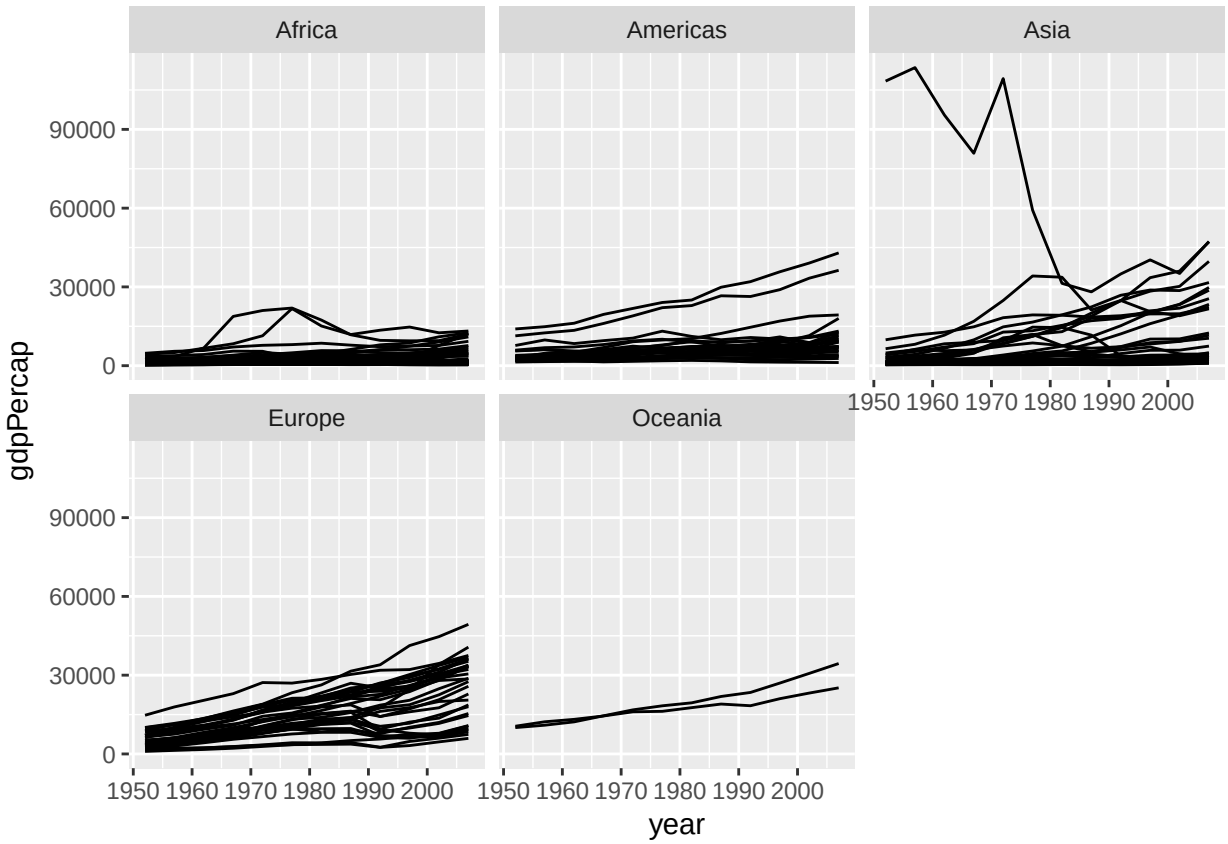
Can be told though:

```
p <- ggplot(data = gapminder,
            mapping = aes(x = year,
                          y = gdpPercap))
p + geom_line(aes(group = country))
```



Faceting your geoms

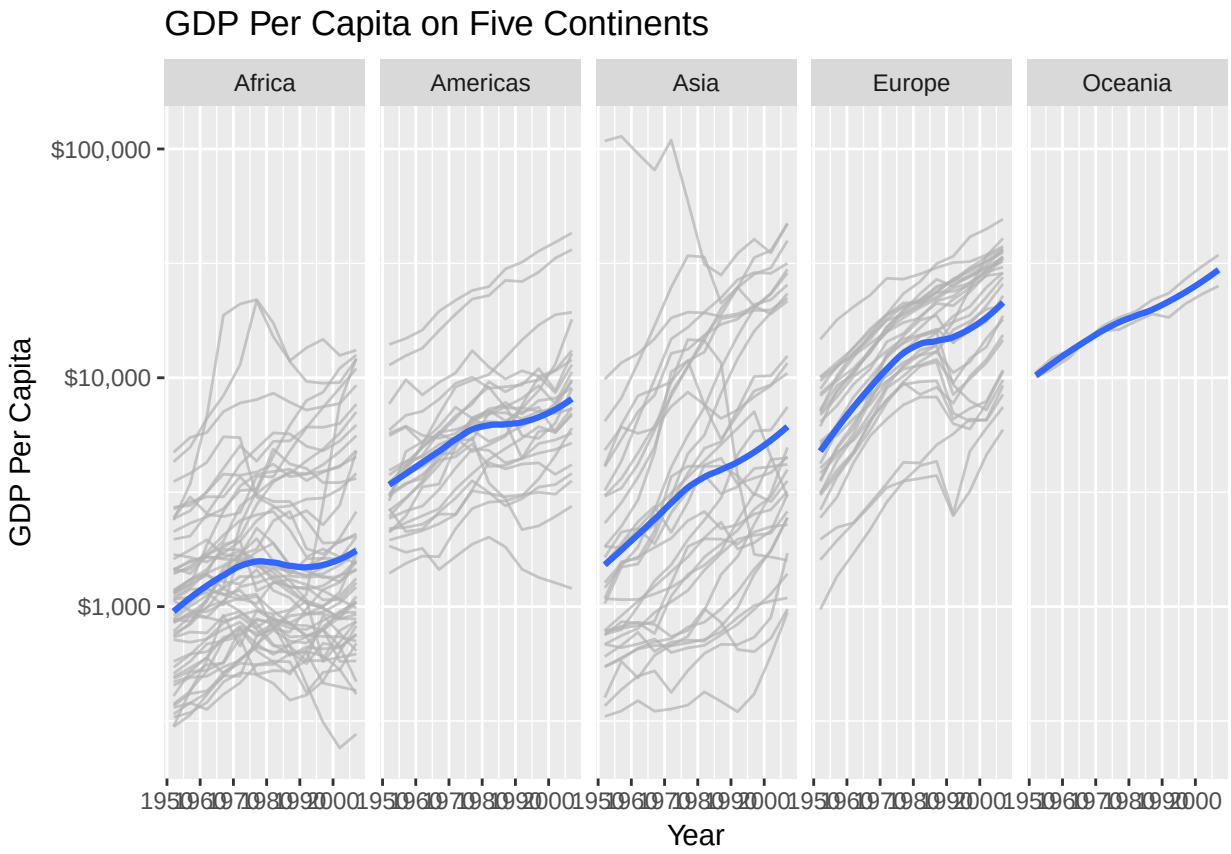
```
p <- ggplot(data = gapminder,
            mapping = aes(x = year,
                          y = gdpPercap))
p + geom_line(mapping = aes(group = country)) +
  facet_wrap(~ continent)
```



A facet is not a geom, it's a way of arranging geoms. Facets use R's 'formula' syntax. Read the ~ as "on" or "by".

Let's put it on a row, and insert some of the bells and whistles from before:

```
p + geom_line(color="gray70",
              alpha=0.7,
              aes(group = country)) +
  geom_smooth(size = 1.1,
              method = "loess",
              se = FALSE) +
  scale_y_log10(labels=scales::dollar) +
  facet_wrap(~ continent, ncol = 5) +
  labs(x = "Year",
       y = "GDP Per Capita",
       title = "GDP Per Capita on Five Continents")
```

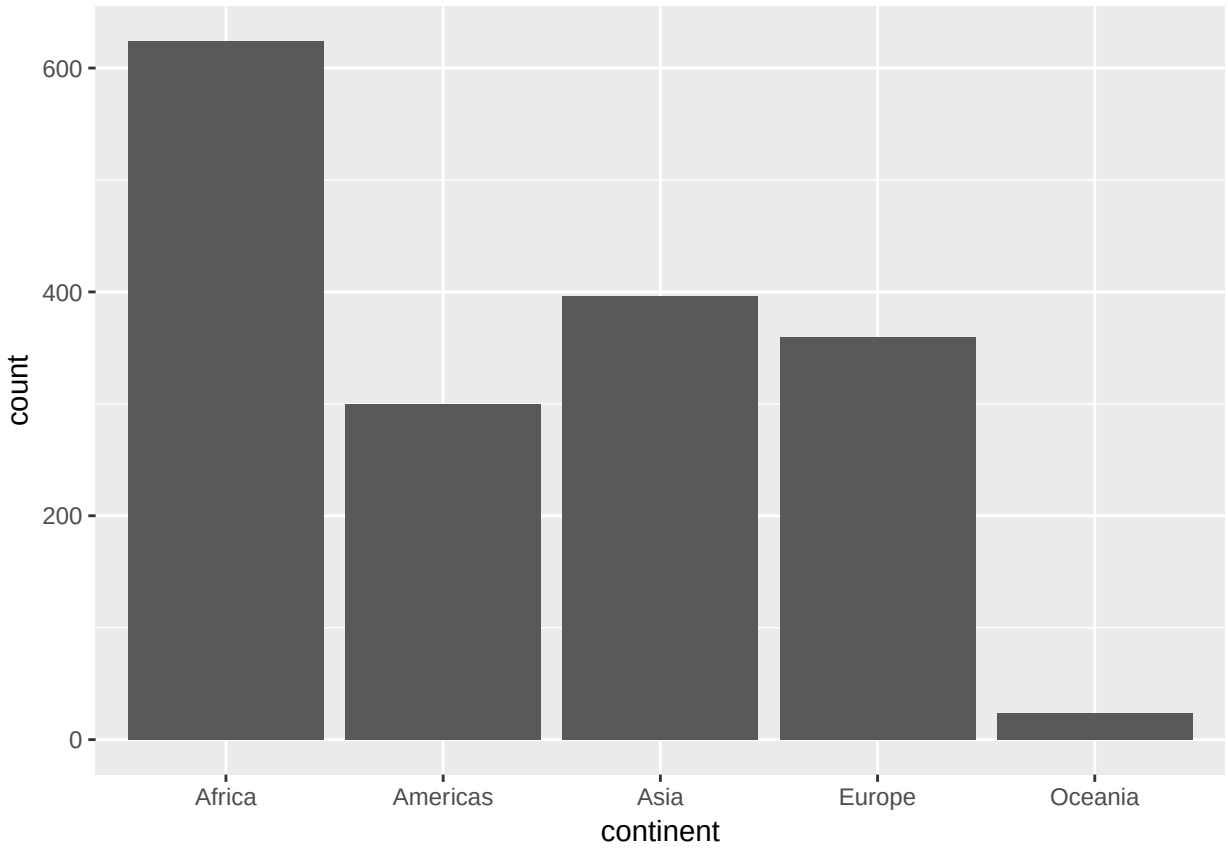


It did a lot of things behind the scenes: Nice logarithmic labels for dollars at 1000 10000 and 100000.

More about stats

What is going on behind the scenes, and how do we take advantage of that? New geom:

```
p <- ggplot(data = gapminder,
            mapping = aes( x = continent))

p + geom_bar()
```

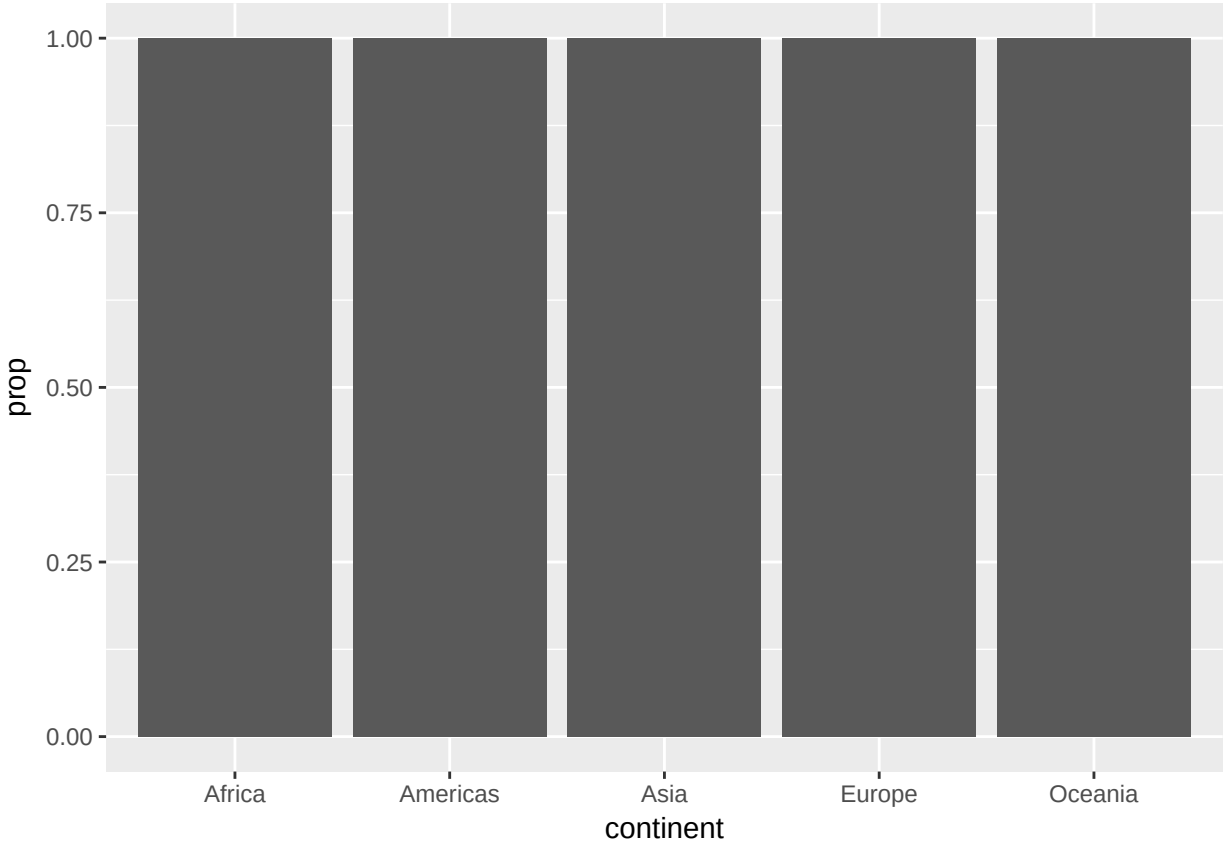


We only specified X, but also got Y. geom_bar calculated the count of observations, to display itself. So this graph lists the total number of country years in the dataset.

It does this using the default stat_ function associated with the geom_bar(), stat_count(). This function can compute two new variables, count, and prop (short for proportion). The count statistic is the default used.

To use the prop:

```
p <- ggplot(data = gapminder,
            mapping = aes(x = continent))
p + geom_bar(mapping = aes(y = ..prop..))
```

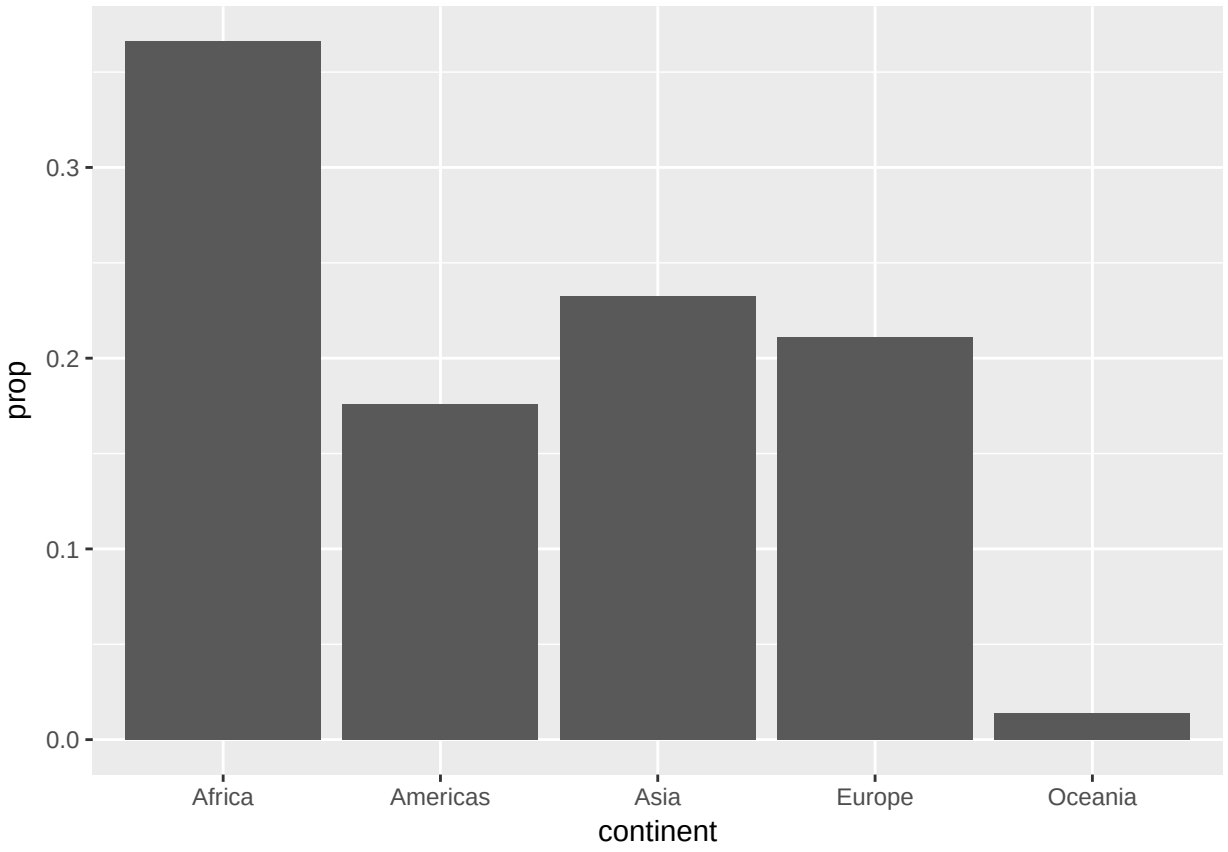


It doesn't do what we want it to, though. Not informative. We infer that this is a grouping issue.

The .. is the syntax for accessing statistics computed by the stat_ functions. Nothing syntactically magical.

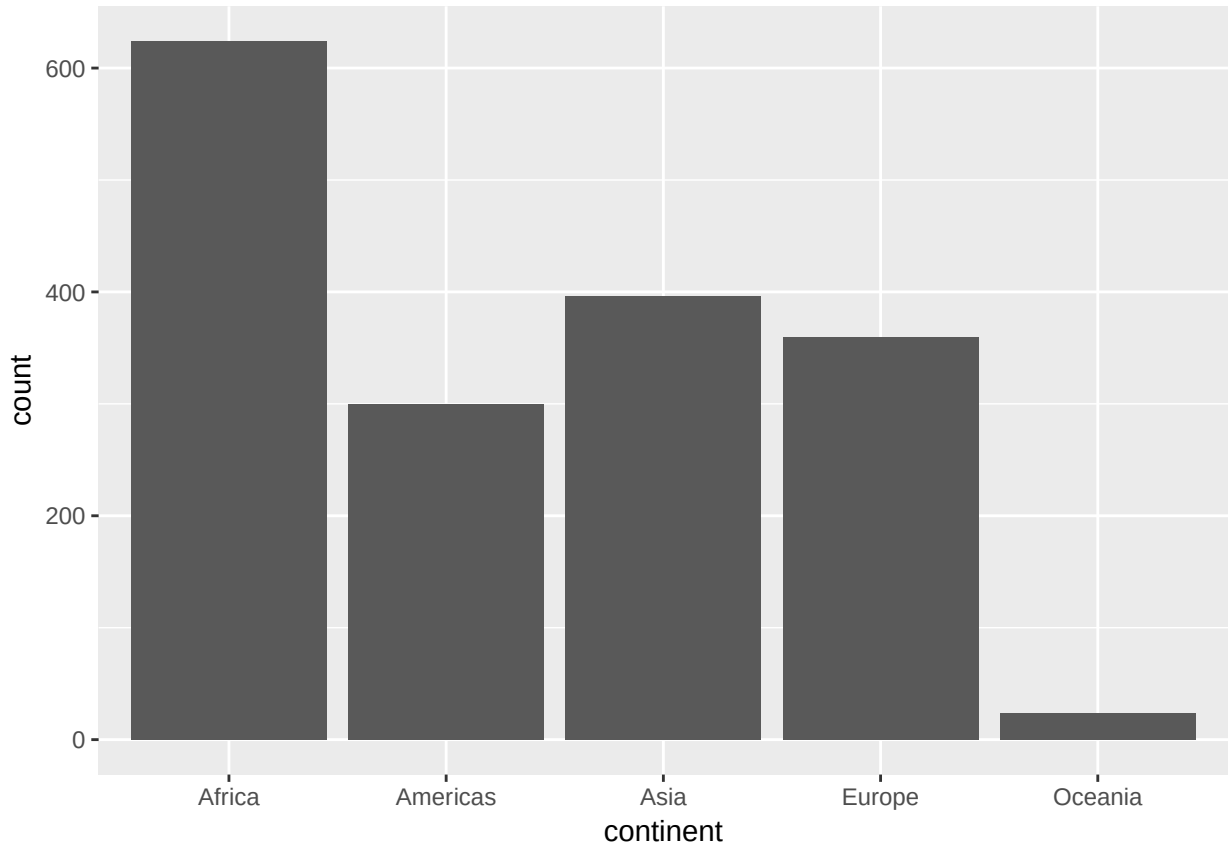
So let's make it sensible, by telling geom_bar to treat the whole dataset as one.

```
p <- ggplot(data = gapminder,
            mapping = aes(x = continent))
p + geom_bar(mapping = aes(y = ..prop..,
                           group = 1))
```

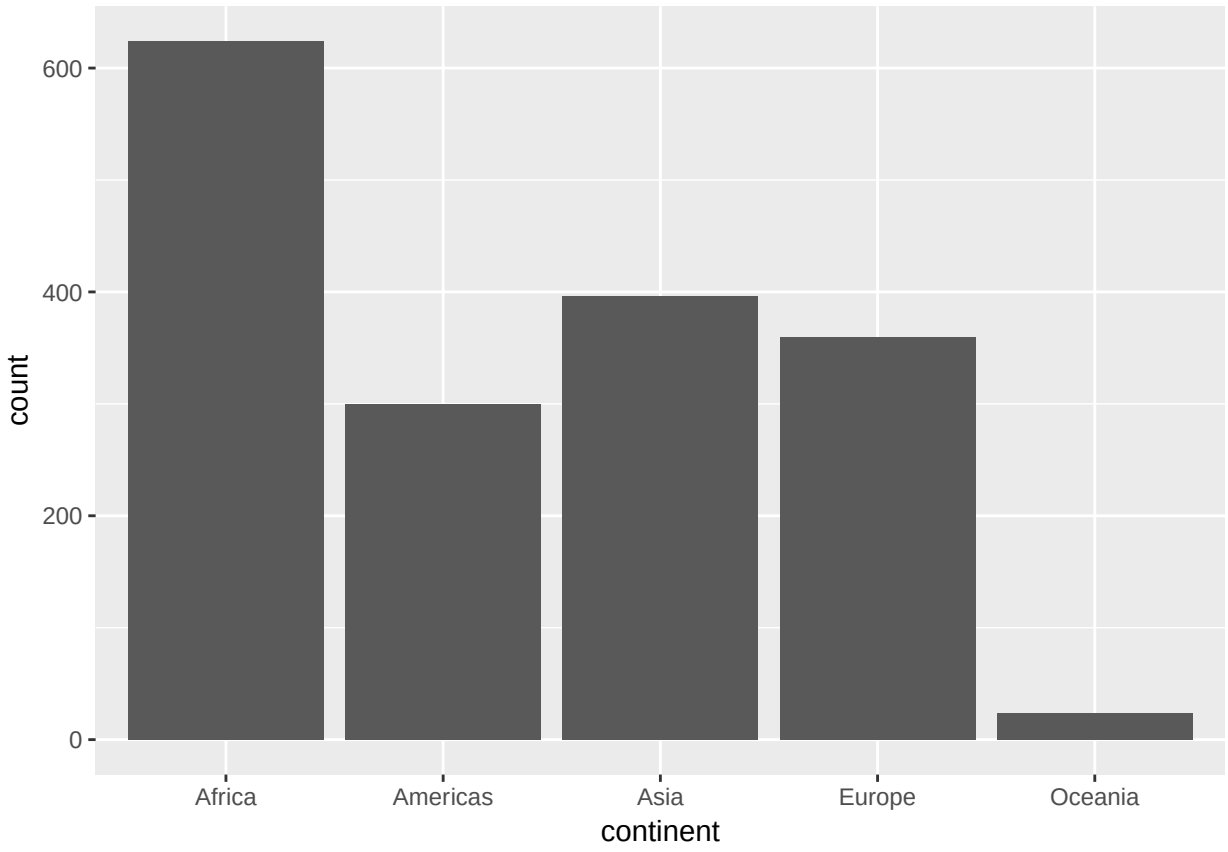


Geom functions call their default stat functions, and vice versa

```
p <- ggplot(data = gapminder,
            mapping = aes(x = continent))
p + geom_bar()
```

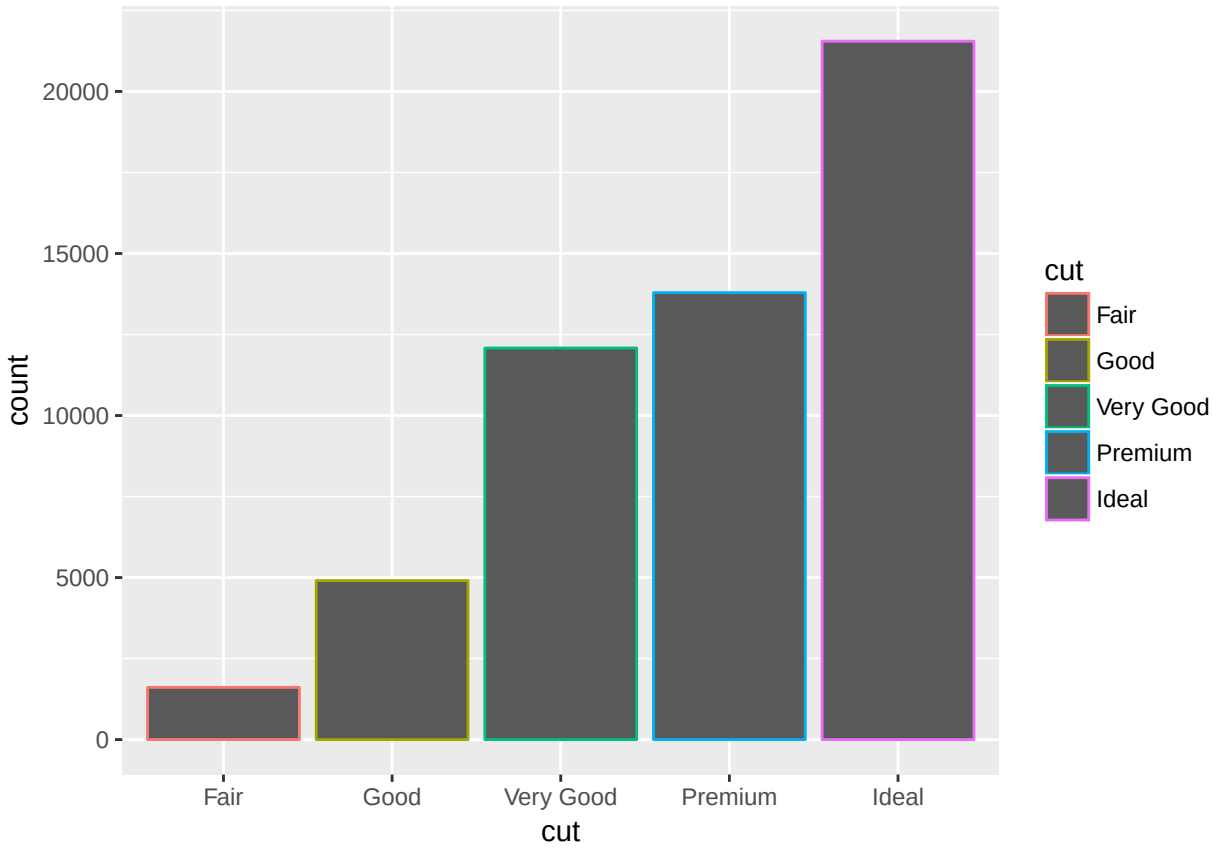



```
# is equal to
p + stat_count()
```



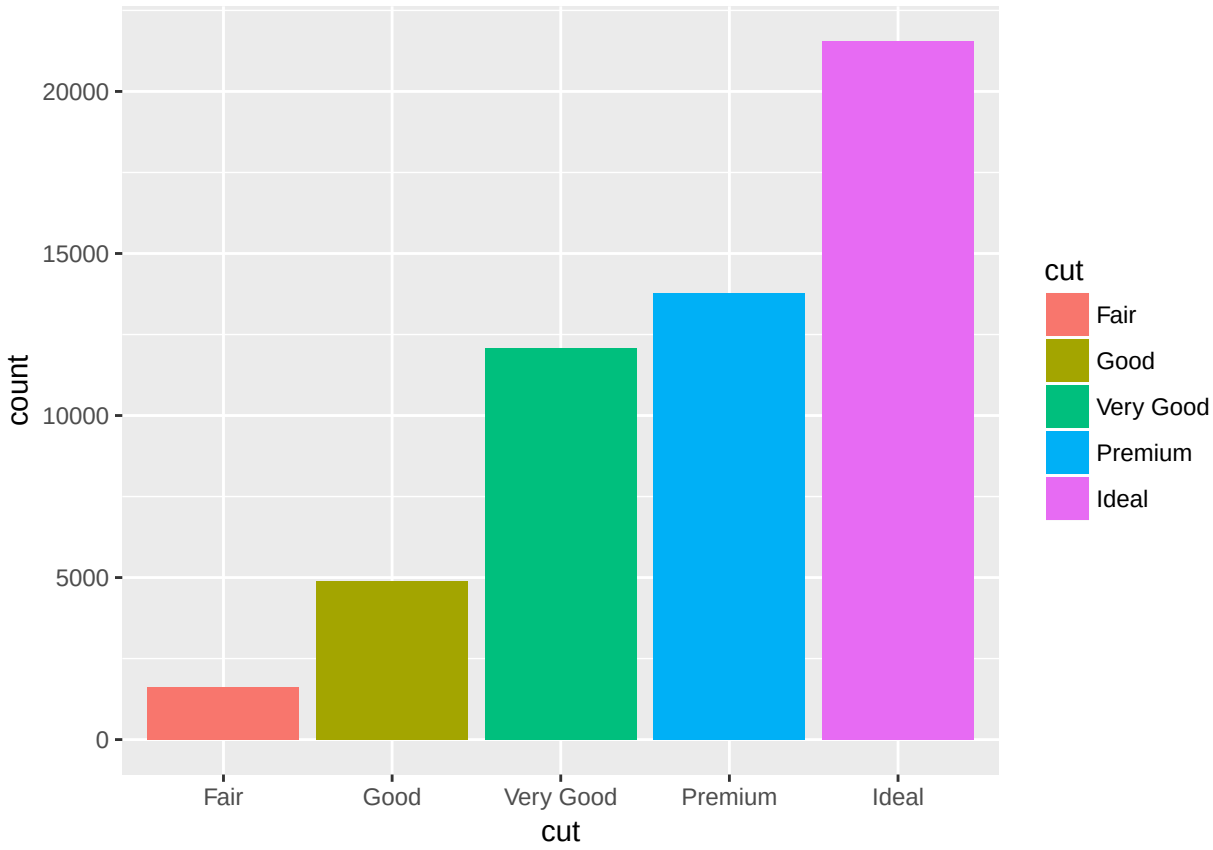
Position adjustments

```
p <- ggplot (data = diamonds,
             mapping = aes(x = cut,
                           color = cut))
p + geom_bar()
```



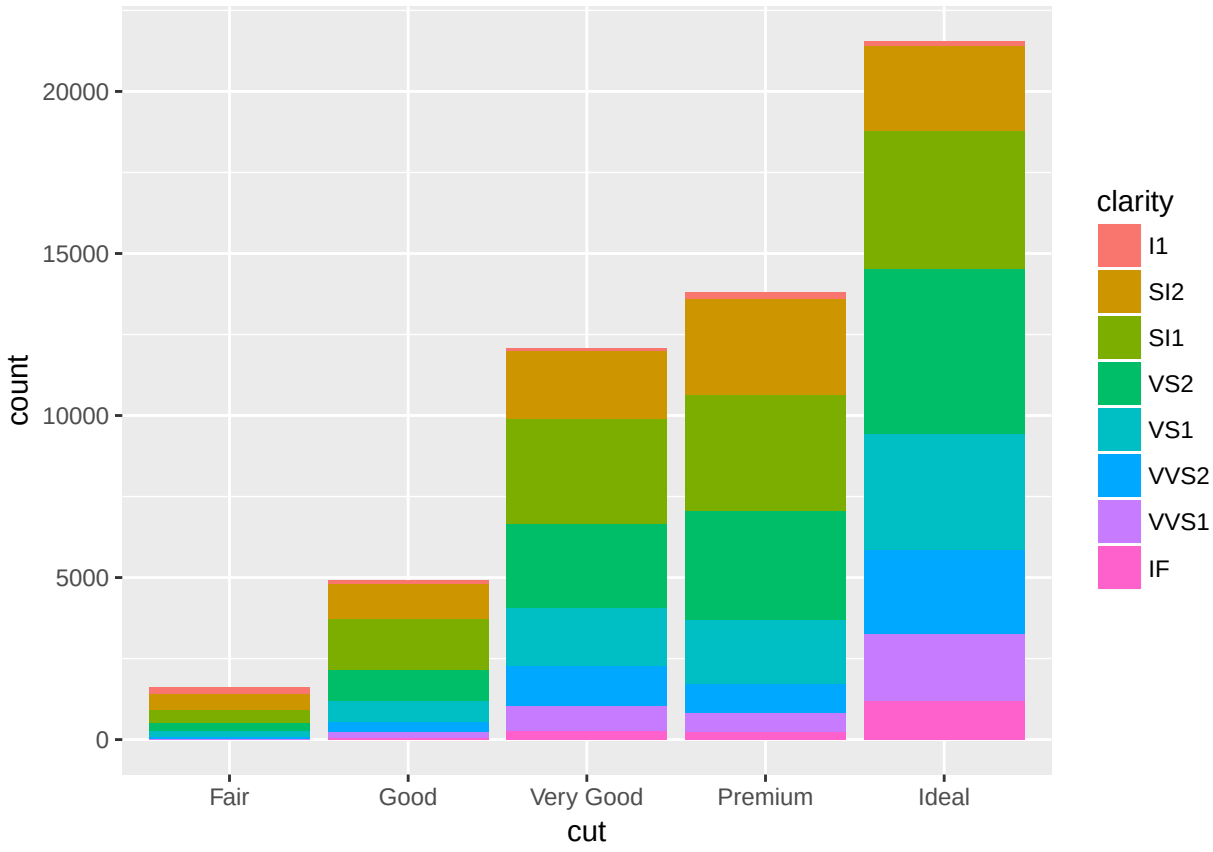
Color gets the outside bar. Fill takes the inside

```
p <- ggplot (data = diamonds,
             mapping = aes(x = cut,
                           fill = cut))
p + geom_bar()
```



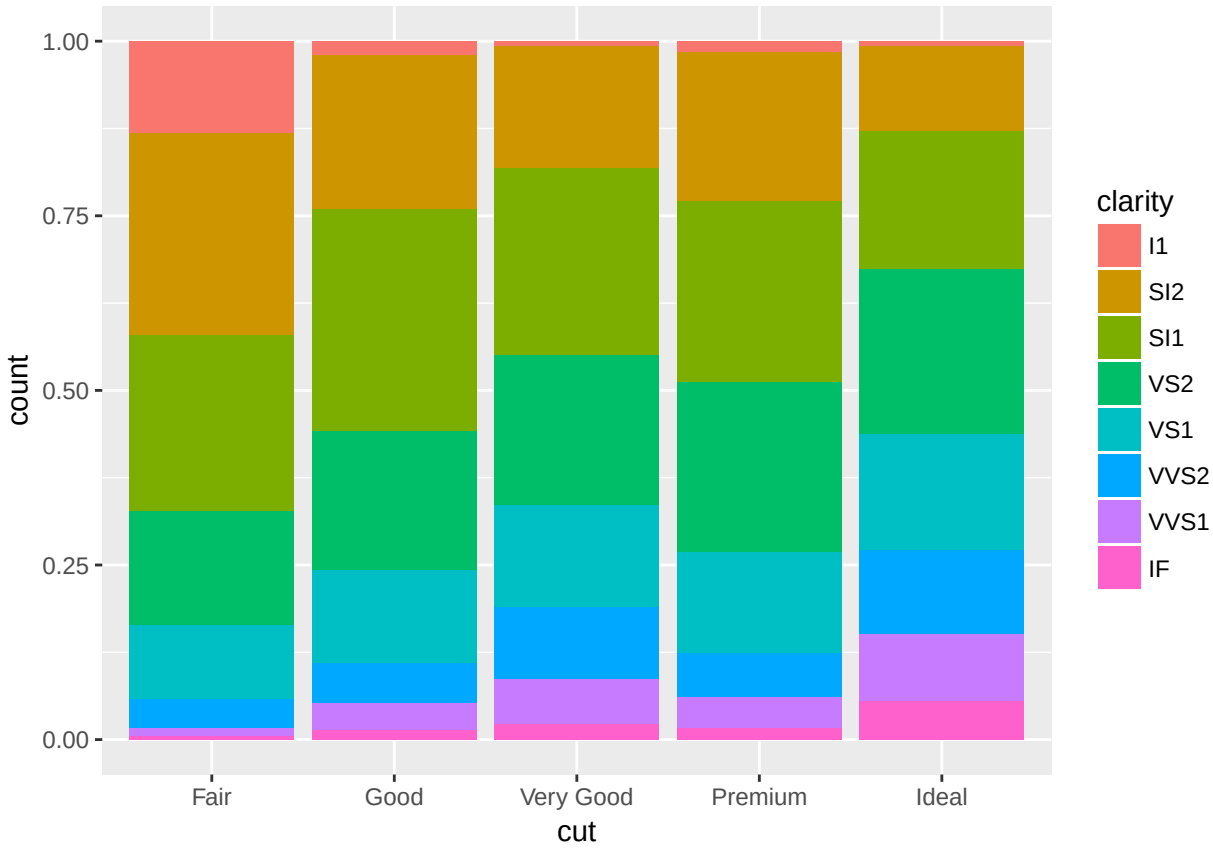
This legend is redundant. Let's cross-classify cut/clarity:

```
p <- ggplot(data = diamonds,
            mapping = aes(x = cut, fill = clarity))
p + geom_bar()
```



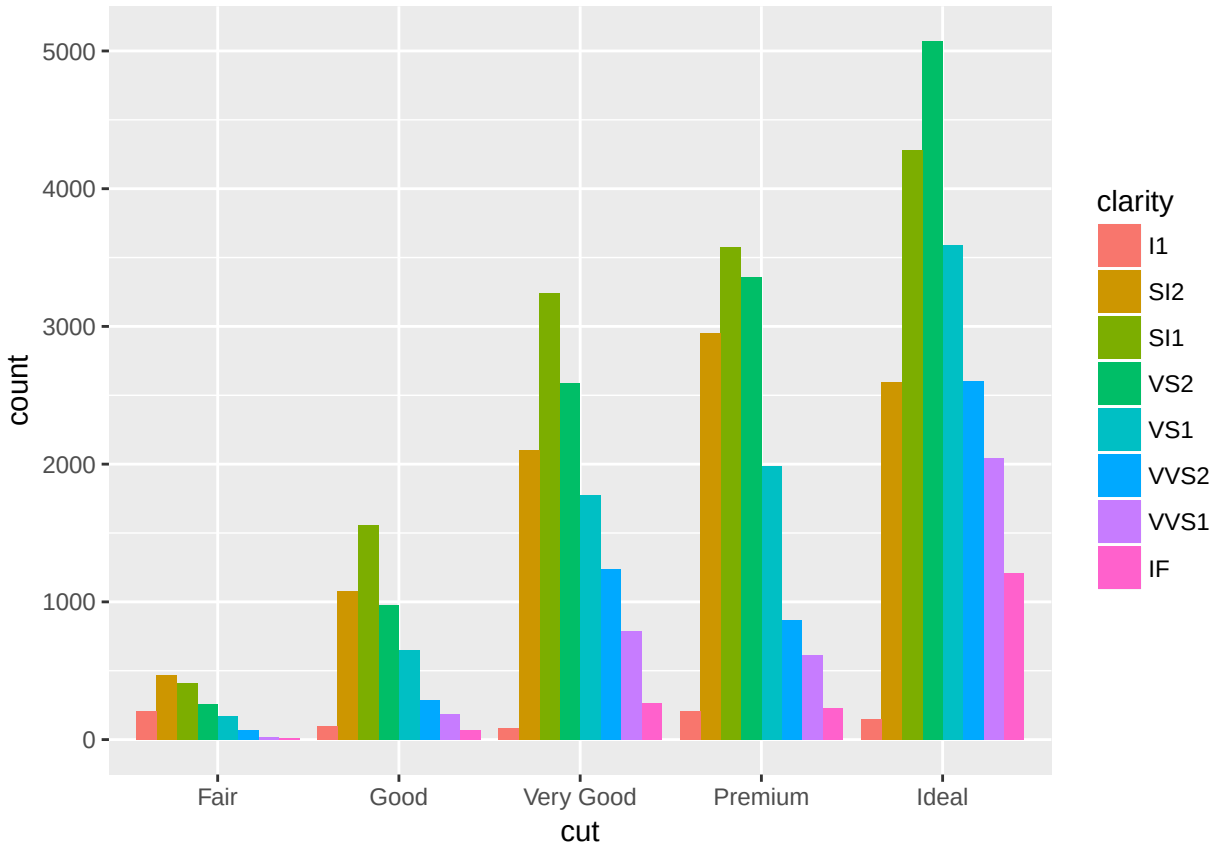
or to make more useful with fill

```
p <- ggplot(data = diamonds,
            mapping = aes(x = cut, fill = clarity))
p + geom_bar(position = 'fill')
```



or dodge

```
p <- ggplot(data = diamonds,
            mapping = aes(x = cut, fill = clarity))
p + geom_bar(position = 'dodge')
```



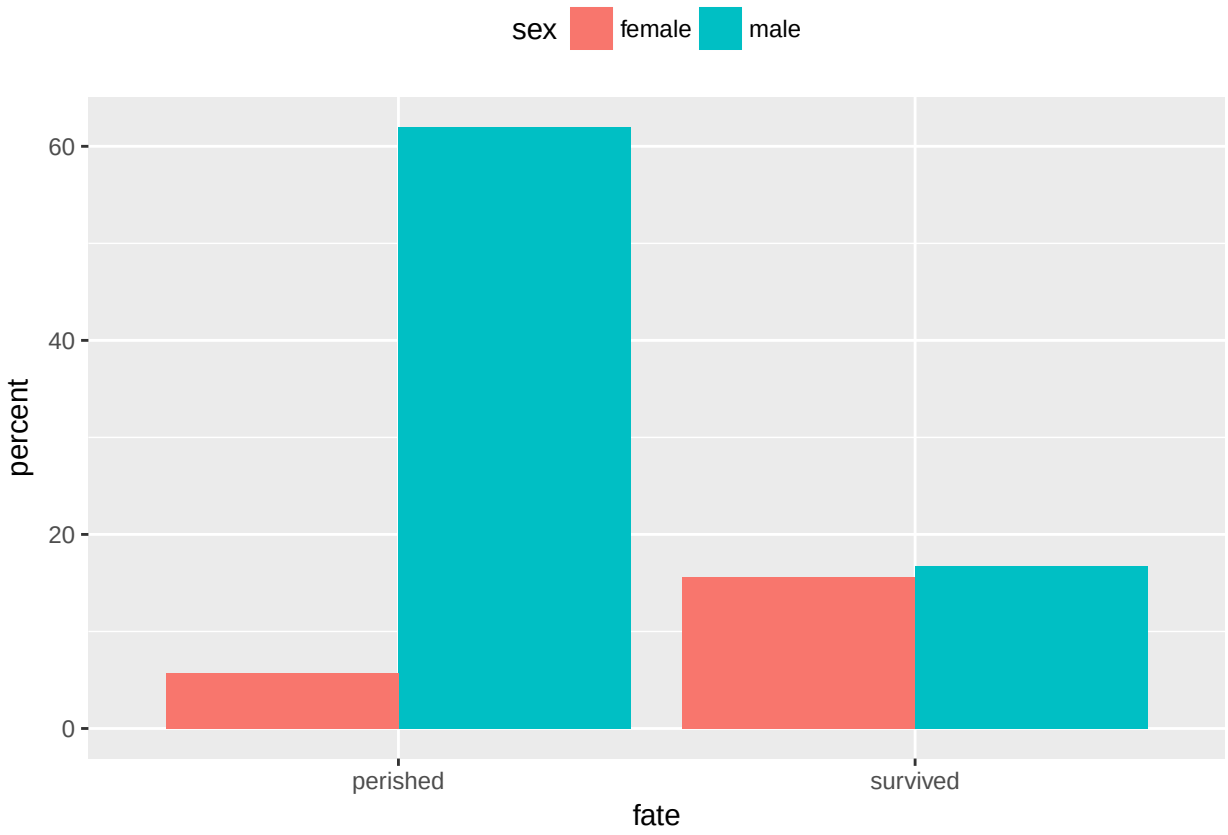
Plotting a finished table

```
head(titanic)
```

```
##       fate    sex    n percent
## 1 perished   male 1364    62.0
## 2 perished female  126     5.7
## 3 survived   male  367    16.7
## 4 survived female  344    15.6
```

When calculations are already done for us, we want to tell geom_bar to not calculate for us:

```
p <- ggplot(data = titanic,
            mapping = aes(x = fate,
                          y = percent,
                          fill = sex))
p + geom_bar(stat = "identity", position = "dodge") +
  theme(legend.position = "top")
```



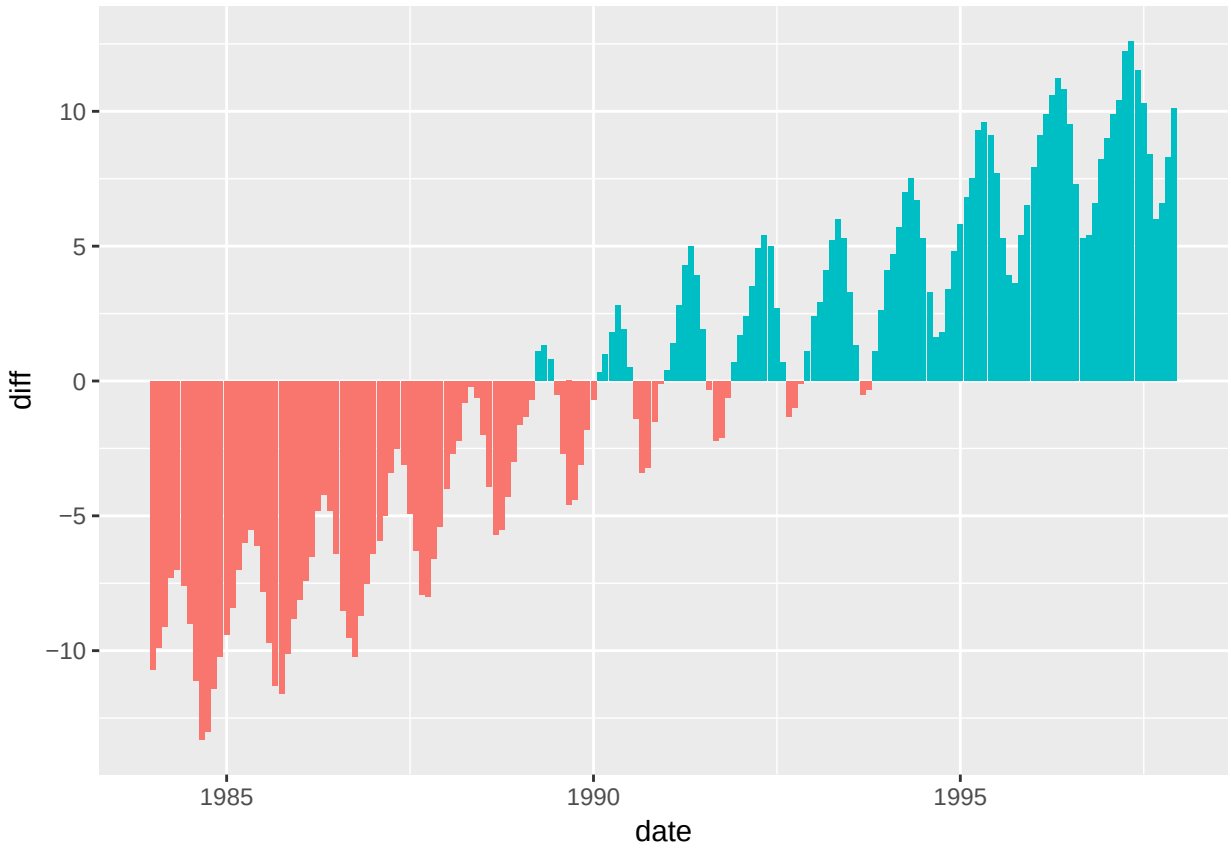
The theme() function controls parts of the plot that don't belong to its "grammatical" structure.

The position argument can also be identitiy: "plot it right here"

```
head(maunaloa)
```

```
##       conc       date  diff   pos
## 301 343.52 1984-01-01 -10.7 FALSE
## 302 344.33 1984-02-01  -9.9 FALSE
## 303 345.11 1984-03-01  -9.1 FALSE
## 304 346.88 1984-04-01  -7.3 FALSE
## 305 347.25 1984-05-01  -7.0 FALSE
## 306 346.62 1984-06-01  -7.6 FALSE
```

```
p <- ggplot(data = maunaloa,
            mapping = aes(x=date,
                          y=diff,
                          fill=pos))
p + geom_bar(stat="identity", position="identity") +
  guides(fill=FALSE)
```



Histograms and Kernel densities

1 new dataset

```
head(midwest)
```

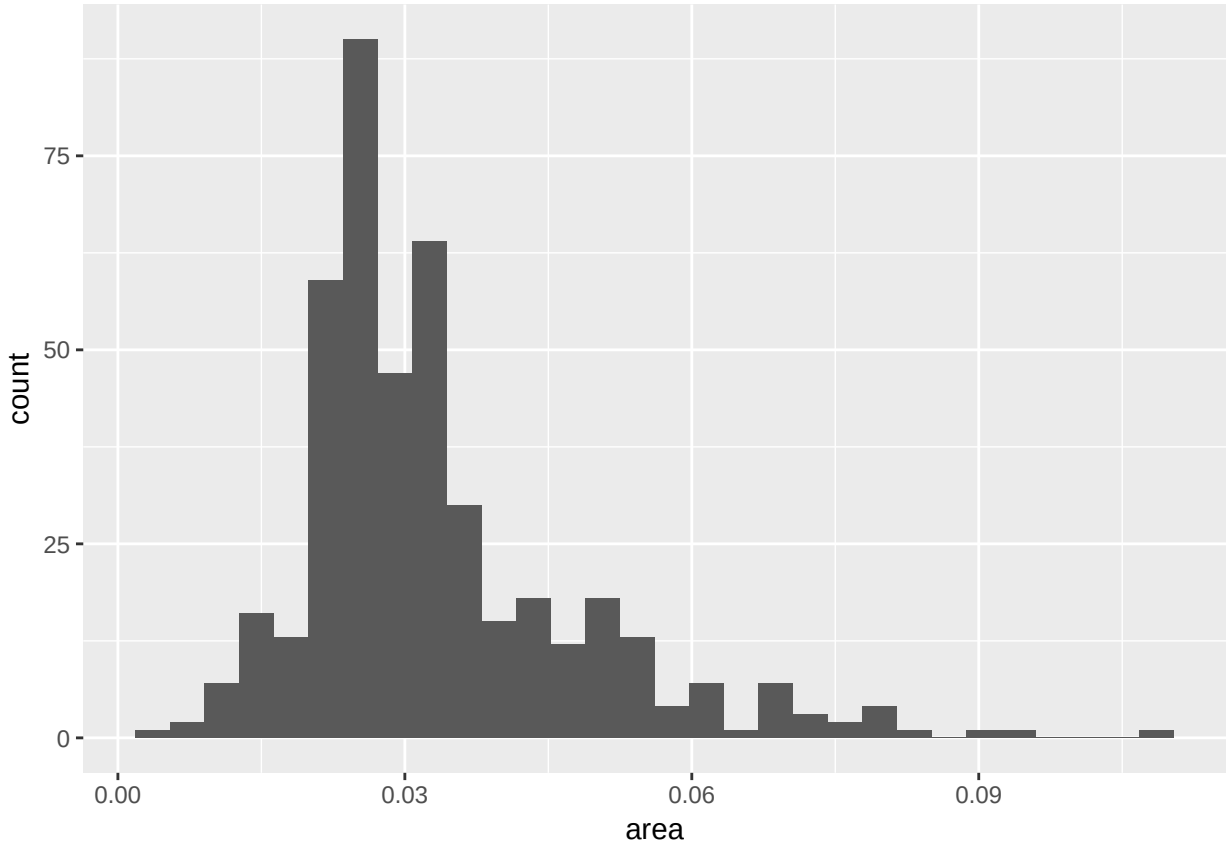
```
## # A tibble: 6 x 28
##     PID    county state  area poptotal popdensity popwhite popblack
##   <int>     <chr> <chr> <dbl>    <int>      <dbl>    <int>    <int>
## 1   561     ADAMS    IL 0.052    66090  1270.9615    63917     1702
## 2   562 ALEXANDER    IL 0.014    10626   759.0000     7054     3496
## 3   563      BOND    IL 0.022    14991   681.4091    14477      429
## 4   564     BOONE    IL 0.017    30806  1812.1176    29344      127
## 5   565     BROWN    IL 0.018     5836   324.2222     5264      547
## 6   566    BUREAU    IL 0.050    35688   713.7600    35157       50
## # ... with 20 more variables: popamerindian <int>, popasian <int>,
## #   popother <int>, percwhite <dbl>, percblack <dbl>, percamerindan <dbl>,
## #   percasian <dbl>, percother <dbl>, popadults <int>, perchsd <dbl>,
## #   percollege <dbl>, percprof <dbl>, poppovertyknown <int>,
## #   percpovertyknown <dbl>, percbelowpoverty <dbl>,
## #   percchildbelowpovert <dbl>, percadultpoverty <dbl>,
## #   percelderlypoverty <dbl>, inmetro <int>, category <chr>
```

2 new geoms: geom_histogram() and geom_density

```
p <- ggplot(data = midwest,
            mapping = aes(x = area))
```

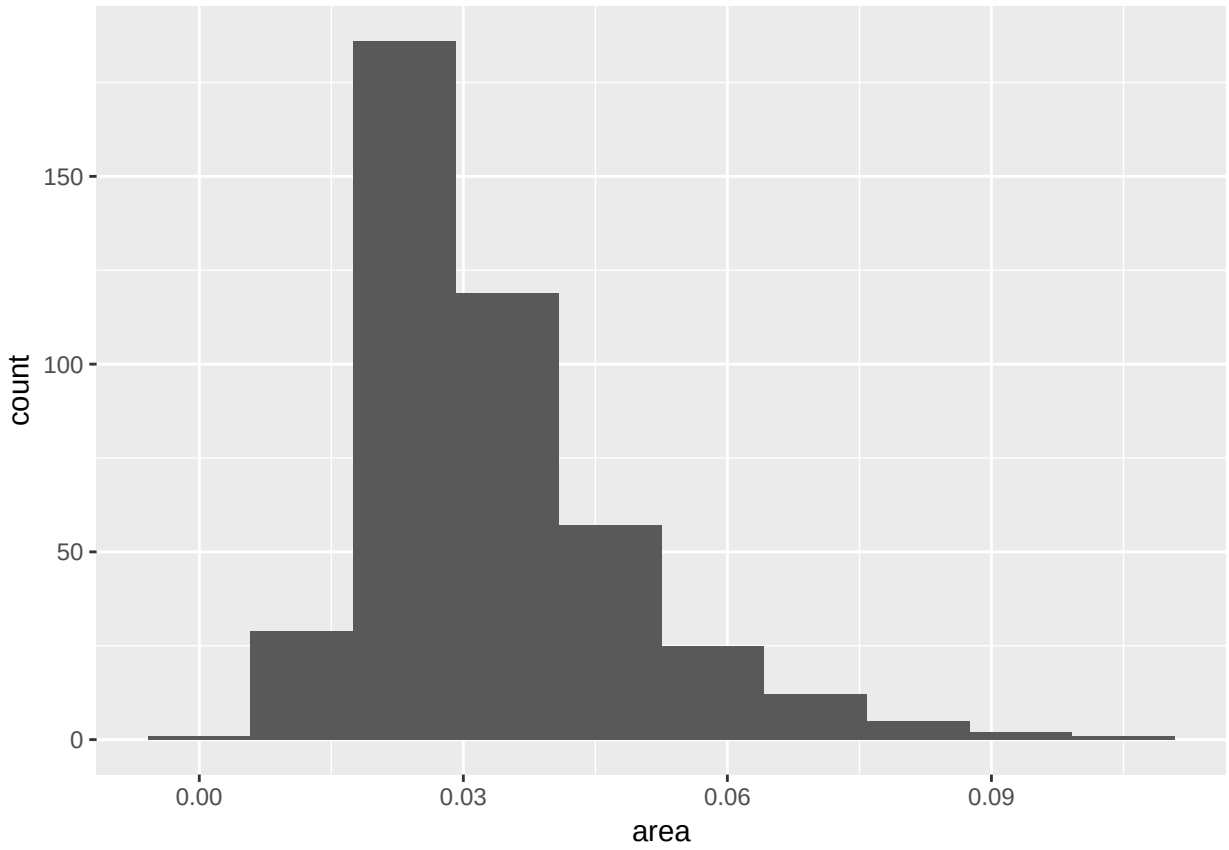
```
p + geom_histogram()
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



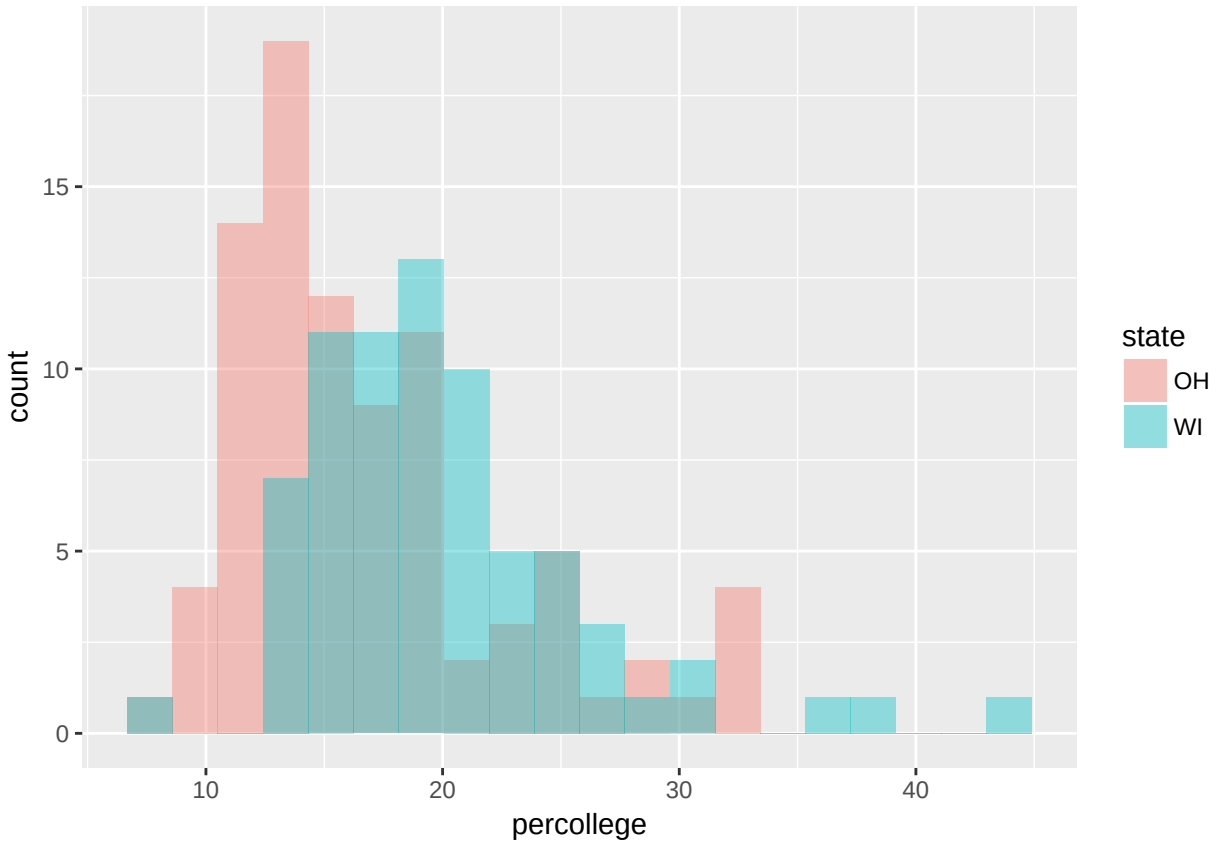
To follow its recommendation, lets set bins to 10 to make it coarser:

```
p <- ggplot(data = midwest,
            mapping = aes(x = area))
p + geom_histogram(bins = 10)
```



Let's mess around a bit. As a functional language, we can perform actions on the fly, as we go, without having to make a new dataset.

```
OH.WI <- c("OH", "WI")

# Subset our data on the fly

p <- ggplot(data = subset(midwest, state %in% OH.WI),
            mapping = aes(x = percollege, fill = state))
p + geom_histogram(position = "identity",
                   alpha = 0.4, bins = 20)
```

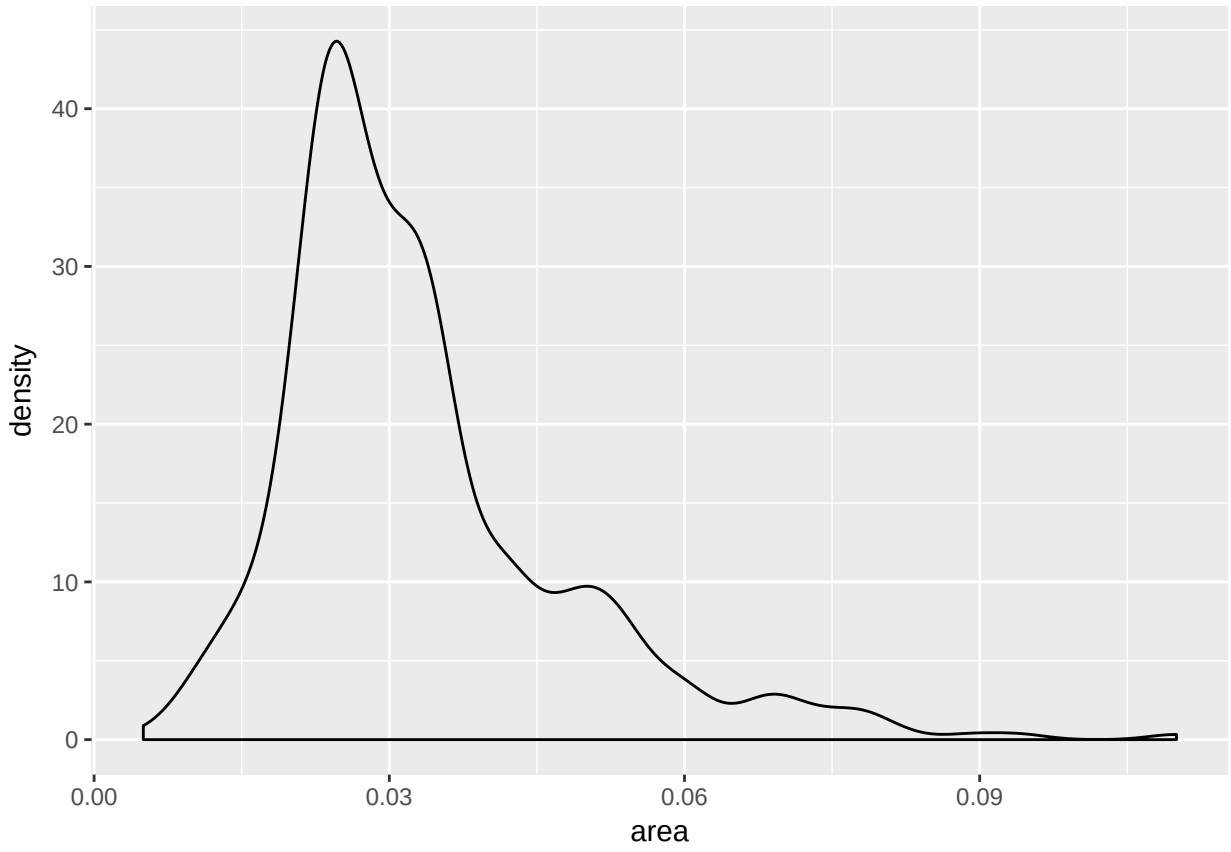


New operator as well %in%. A convinient built in operator that returns a true false value. Asks for a subset in the data WHERE the state is in OH.WI

geom_hist()'s continious counterpart, geom_density()

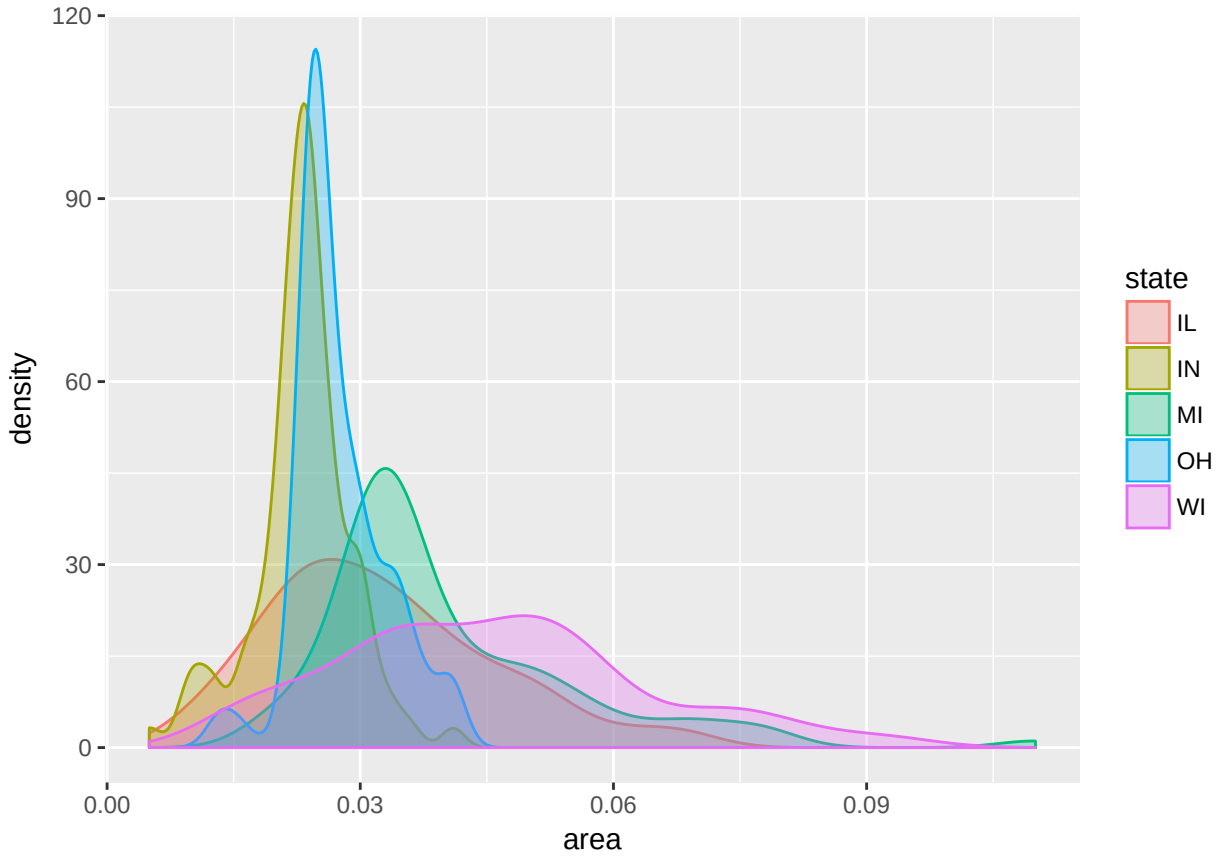
Newer, depend on the availability of quick computers. Estimates kernel density. Think of it as a one dimensional smoother. What sort of continious distribution does this look like?

```
p <- ggplot(data = midwest,
            mapping = aes(x = area))
p + geom_density()
```



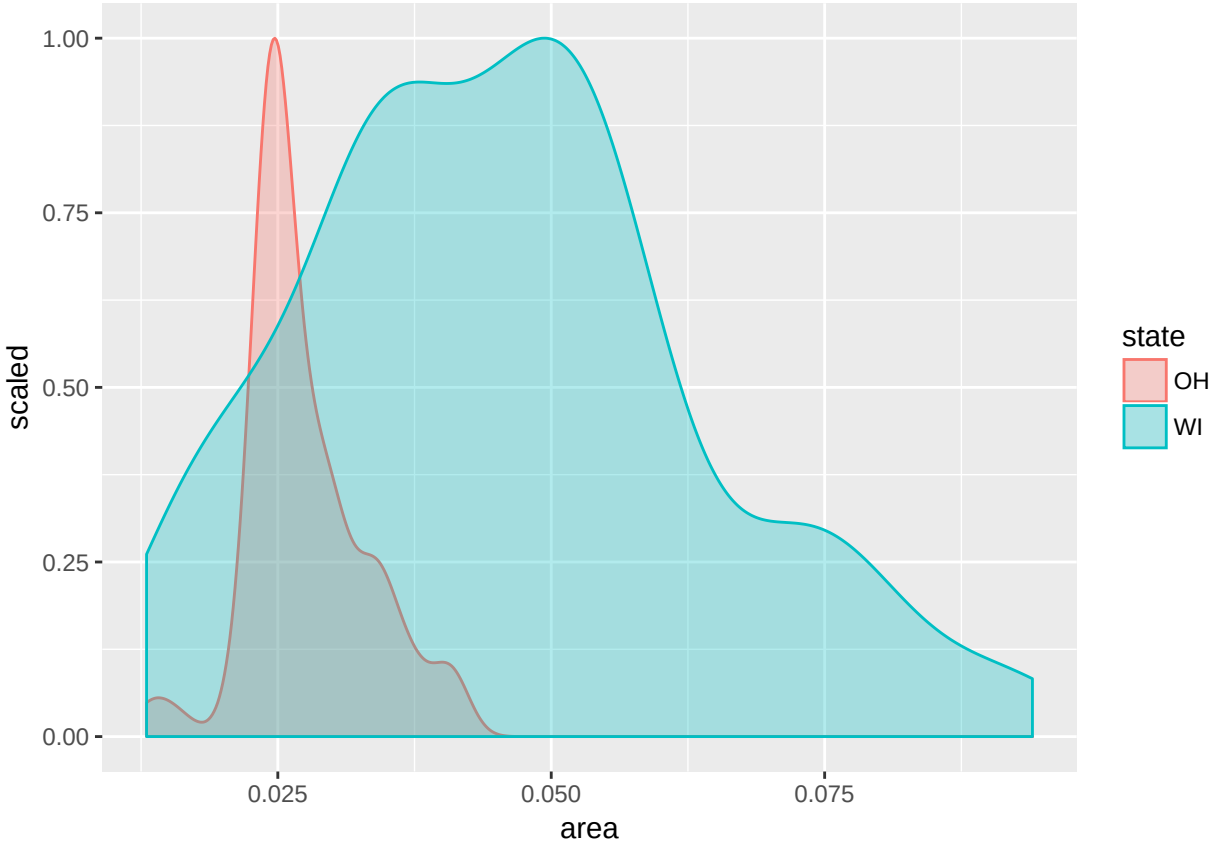
Make it nice:

```
p <- ggplot(data = midwest,
            mapping = aes(x = area,
                          fill = state,
                          color = state))
p + geom_density(alpha = 0.3)
```



We can also compute, if we want the percentage distances, scale it by

```
p <- ggplot(data = subset(midwest, subset = state %in% OH.WI),
            mapping = aes(x = area, fill = state, color = state))
p + geom_density(alpha = 0.3, mapping = (aes(y = ..scaled..)))
```



That's all for now folks.

Day 2: 08-17-2017

Group, Facet and Transform

Although you can do a lot of transformations on the fly in ggplot, the code can become complicated. It's better in the long run to do the data preparation first. Create an object that is your summary table first, and then plot it. One reason for this is for your own code to stay clean. The other reason is that the methods we can use to edit, rearrange and sort our data are very general. It is handled by the deplyr() library, which selects, slices and filters to quickly reshape your data. The third reason is to keep track of the results we are getting are actually correct. It's easier to do that when you have access to the numbers, objects, directly.

When using this, one has to go back and check all the time.

Frequency tables

A small subset of the US general social survey, the GSS. Going on since 1971. gss_s

```
table(gss_sm$religion)
```

```
##
## Protestant   Catholic     Jewish       None      Other
##       1371        649         51        619        159
```

or

```
head(gss_sm)
```

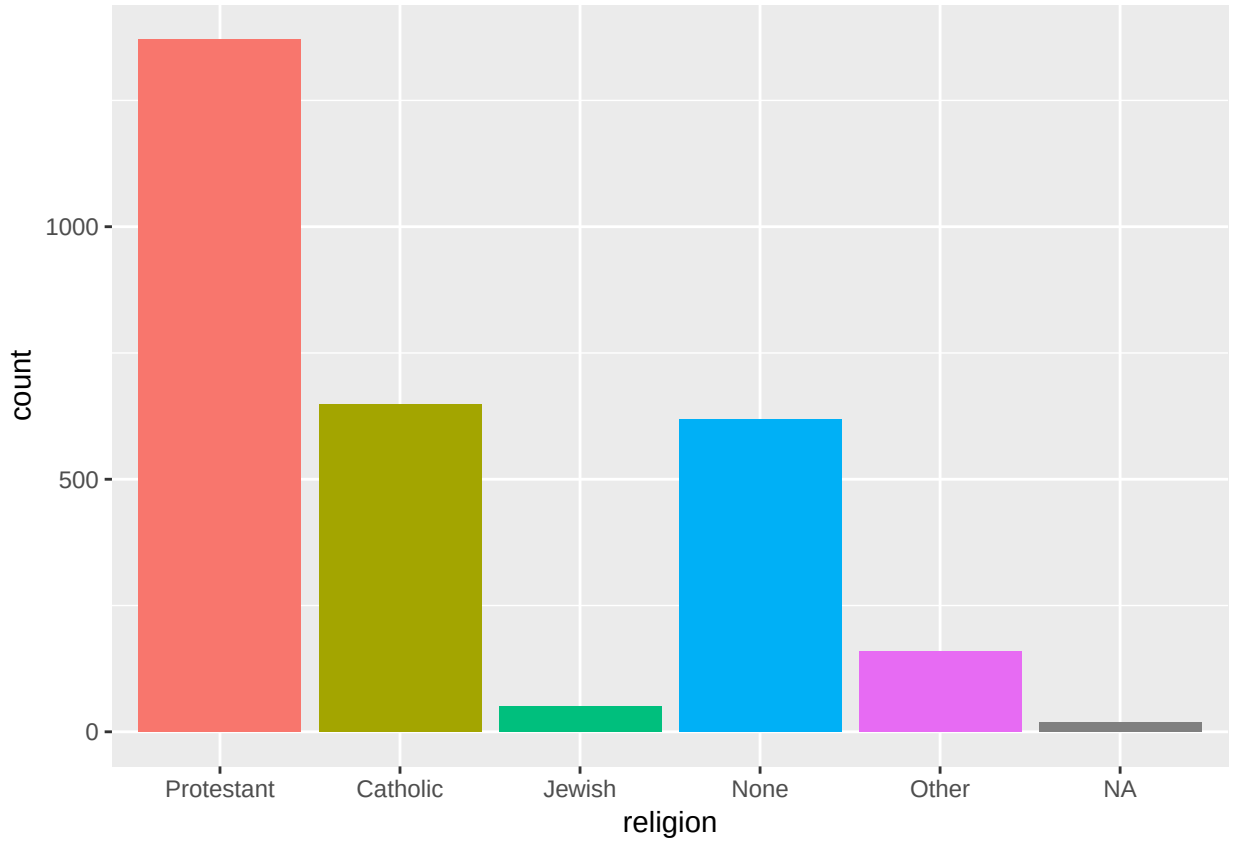
```
## # A tibble: 6 x 32
##    year    id    ballot   age childs      sibs         degree   race
##   <dbl> <dbl> <dbl+lbl> <dbl>  <dbl> <dbl+lbl>         <fctr> <fctr>
## 1  2016     1         1    47      3         2       Bachelor  White
## 2  2016     2         2    61      0         3    High School  White
## 3  2016     3         3    72      2         3       Bachelor  White
## 4  2016     4         1    43      4         3    High School  White
## 5  2016     5         3    55      2         2       Graduate  White
## 6  2016     6         2    53      2         2 Junior College  White
## # ... with 24 more variables: sex <fctr>, region <fctr>, income16 <fctr>,
## #   relig <fctr>, marital <fctr>, padeg <fctr>, madeg <fctr>,
## #   partyid <fctr>, polviews <fctr>, happy <fctr>, partners <fctr>,
## #   grass <fctr>, zodiac <fctr>, pres12 <dbl+lbl>, wtssall <dbl>,
## #   income_rc <fctr>, agegrp <fctr>, ageq <fctr>, siblings <fctr>,
## #   kids <fctr>, religion <fctr>, bigregion <fctr>, partners_rc <fctr>,
## #   obama <dbl>
```

or

```
dim(gss_sm)
```

```
## [1] 2867   32
```

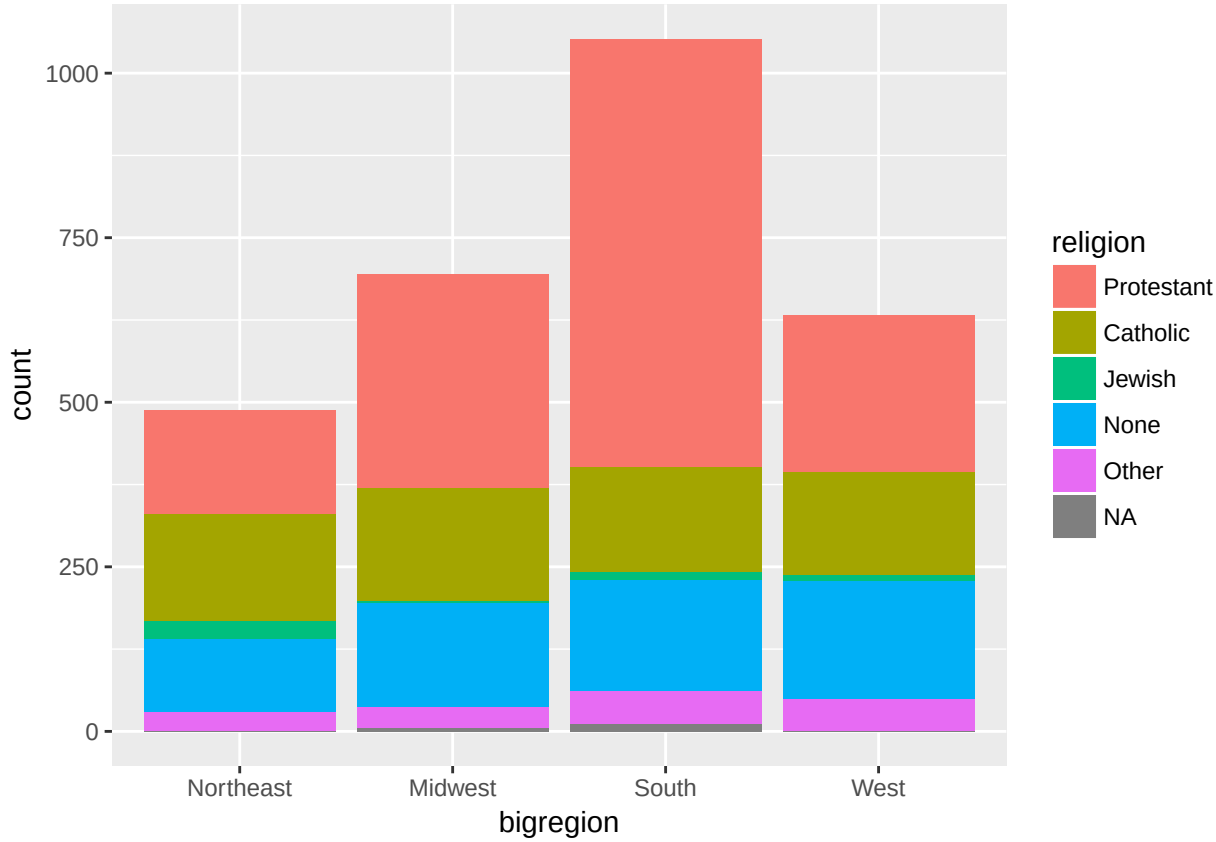
```
p <- ggplot(data = gss_sm,
            aes(religion, fill = religion))
p + geom_bar() + guides(fill = FALSE)
```



A

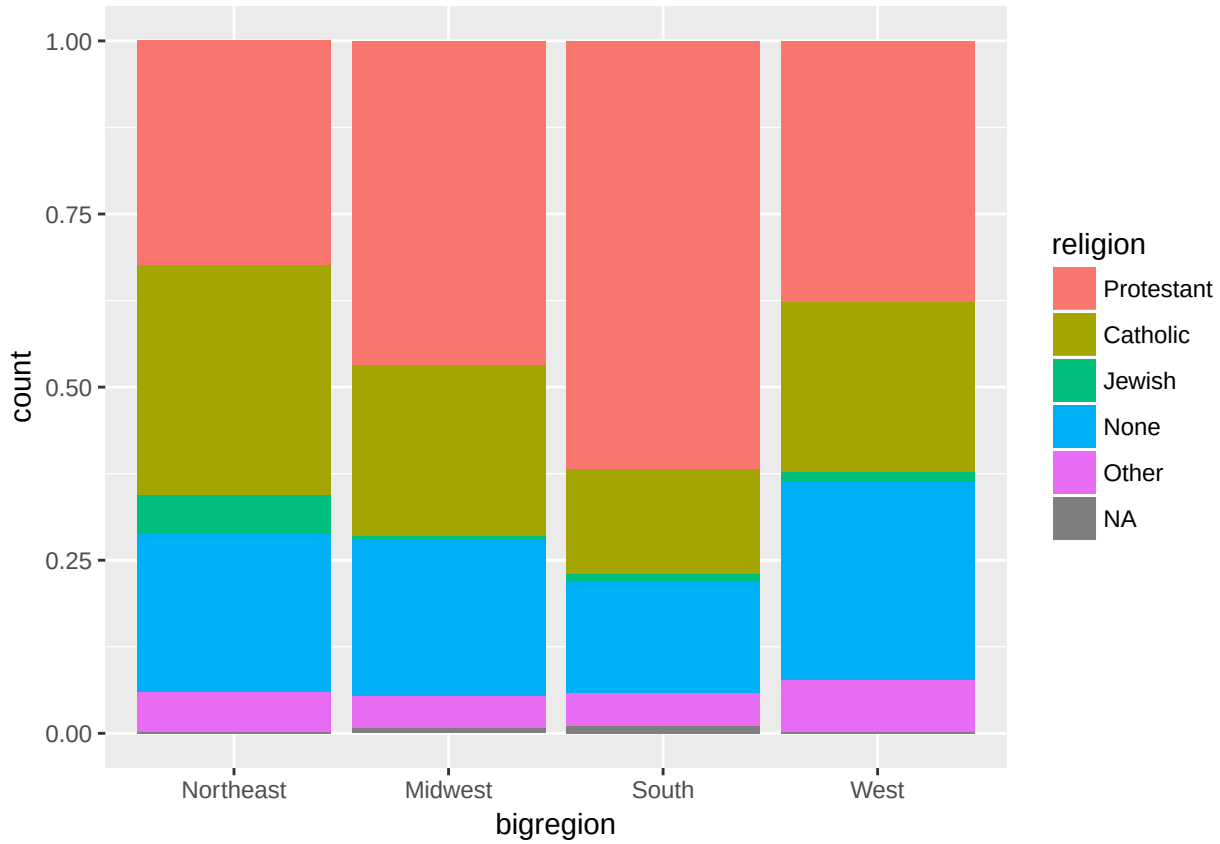
more interesting way to use fill is this:

```
p <- ggplot(data = gss_sm,
            aes(x = bigregion,
                fill = religion))
p + geom_bar()
```



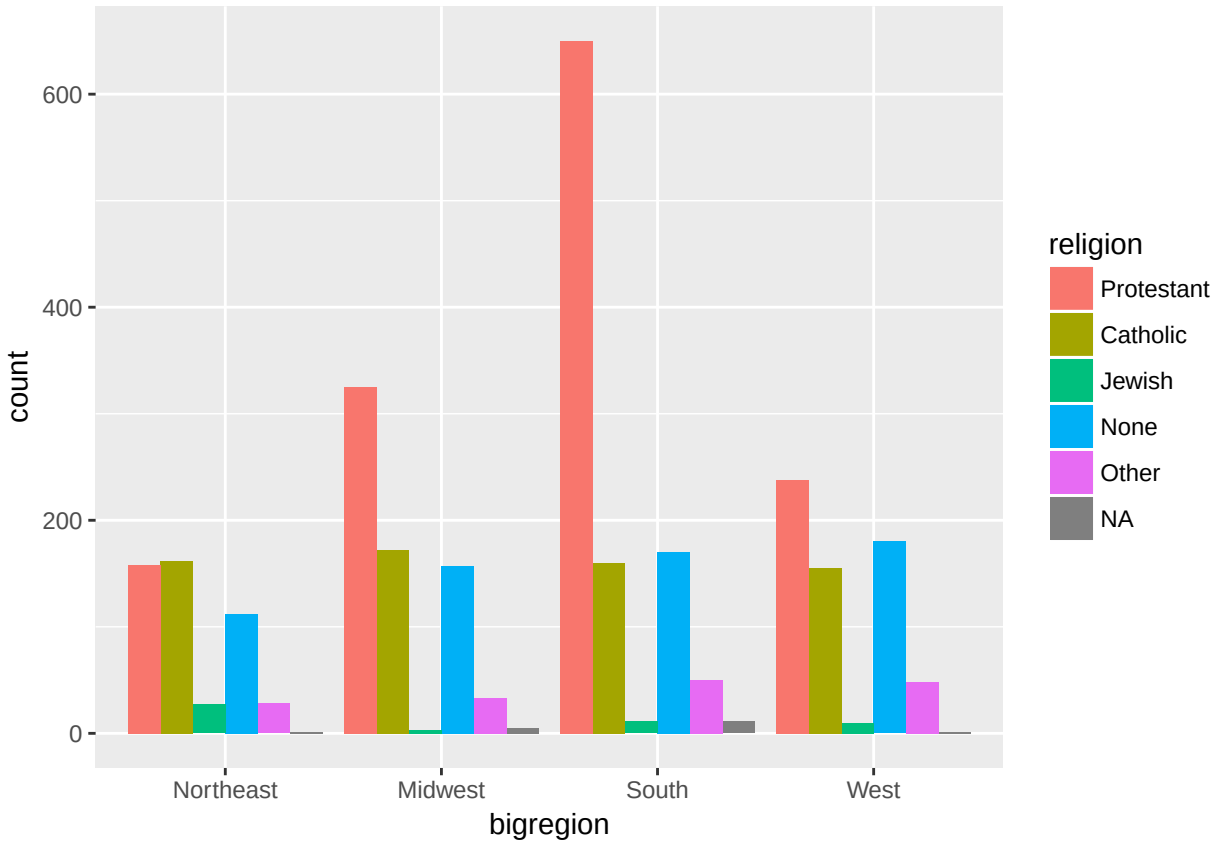
Better to show it like this maybe:

```
p <- ggplot(data = gss_sm,
            aes(x = bigregion,
                fill = religion))
p + geom_bar(position = "fill")
```



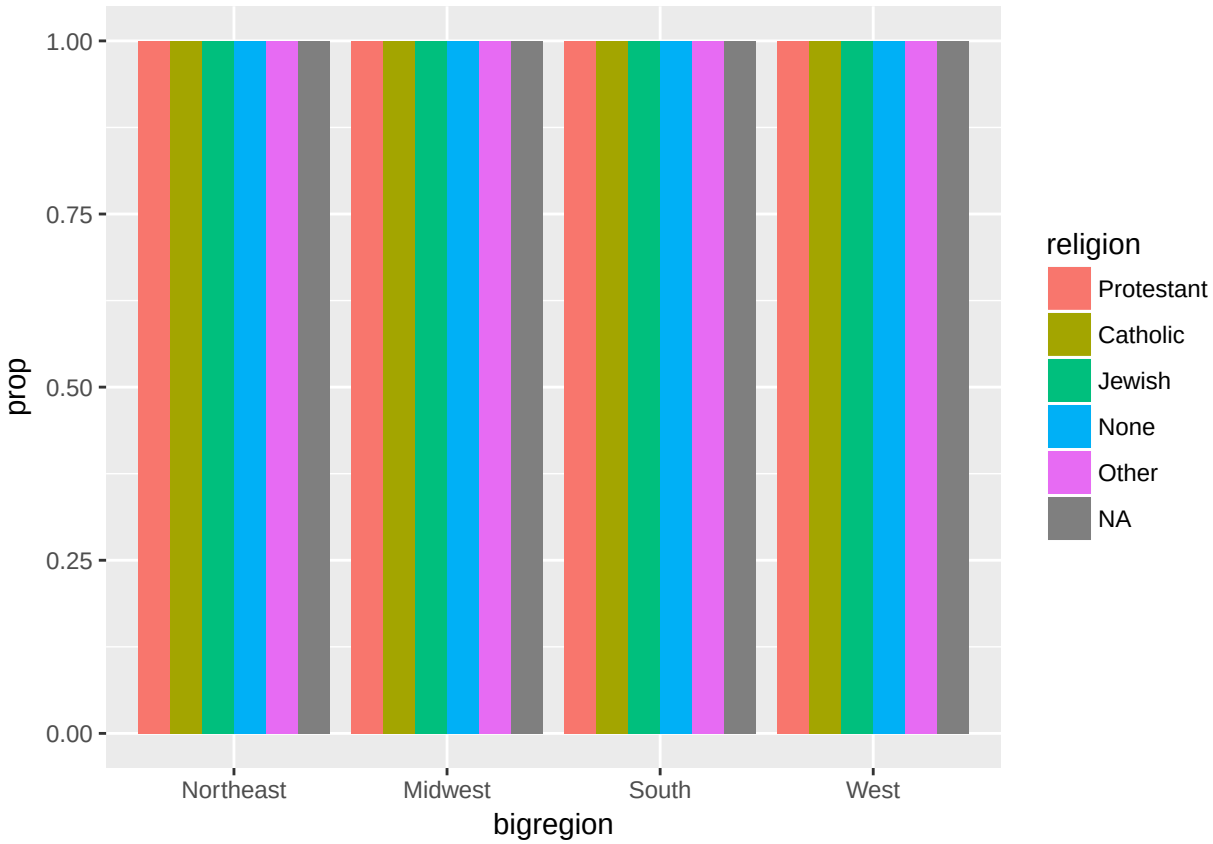
Or maybe with dodge

```
p <- ggplot(data = gss_sm,
            aes(x = bigregion,
                fill = religion))
p + geom_bar(position = "dodge")
```



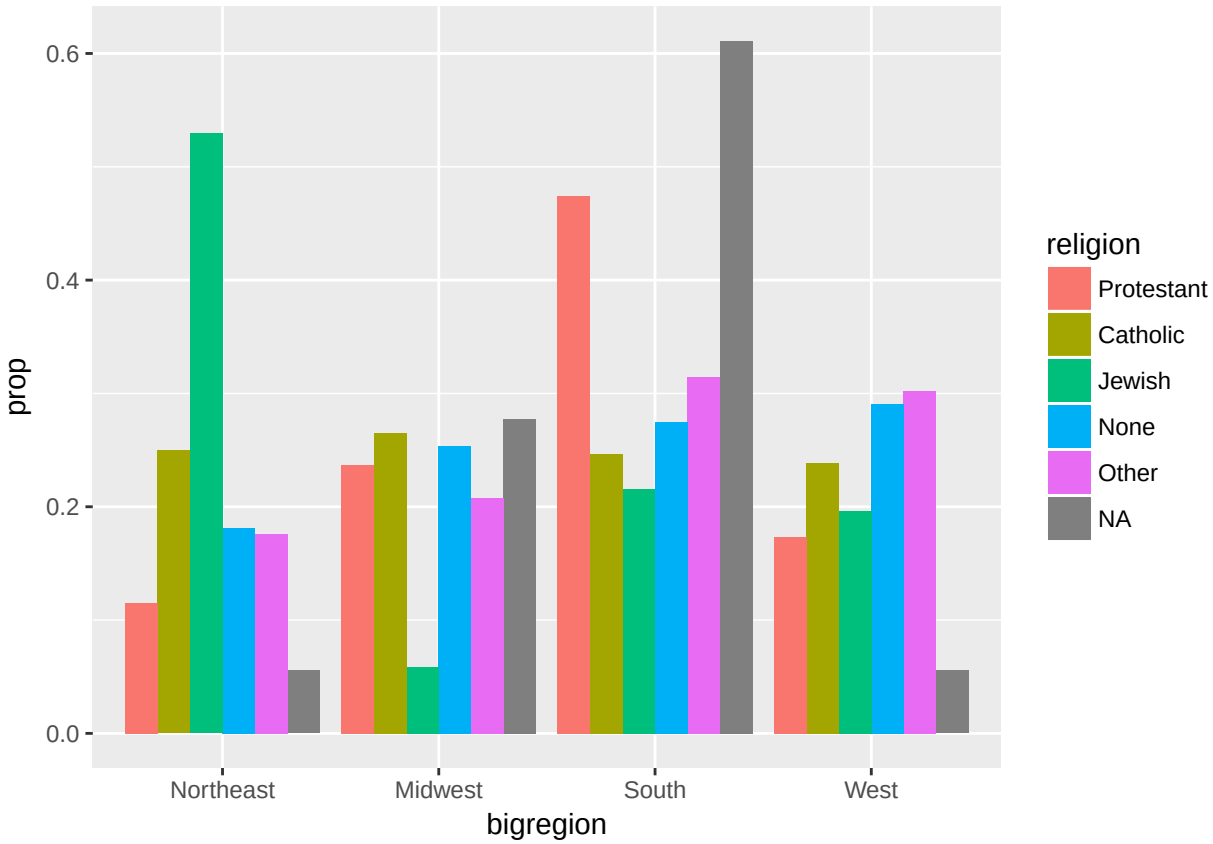
But if we want to properly show proportions within regions. How to?

```
p <- ggplot(data = gss_sm,
            aes(x = bigregion,
                fill = religion))
p + geom_bar(position = "dodge", mapping = aes(y = ..prop..))
```



That's not right. We want to group it, so that proportions are relative to the whole.

```
p <- ggplot(data = gss_sm,
            aes(x = bigregion,
                fill = religion))
p + geom_bar(position = "dodge",
             mapping = aes(y = ..prop..,
                           group = religion))
```



The bars sum to one across the regions, that is not correct. We need to understand the table

```
tail(gss_sm$religion)
```

```
## [1] Protestant Protestant Protestant Protestant Catholic   None
## Levels: Protestant Catholic Jewish None Other
```

We could fix this with complex ggplotting. But lets fix the data.frame.

Summarize and transfomr using pipes.

The operator is %>% . Kind of similar to %in%. Operator. Things that return a result. Special operators are formatted like this. Pipes are powerful. We want to get the stacked bar chart, but showing the bars side by side, with proportions. So percentages per row, in stead of per column.

Reorganizing tables with dplyr

We want to make summary counts of religious preference by census region. And then make percent religous preferences by census region.

The pipe lets data pass through with modifications. dplyr does its work through different verbs.

group_by() Group the data at the level we want, such as "religion by region" or "authors by publications by year"

filter() or **select()** Filter or select pieces of the data. This gets us the subset of the table we want to work on. filter() rows, select() columns.

mutate() Mutate the data by creating new variables at the current level of grouping. Mutating adds new columns to the table.

summarize() Summarize or aggregate the grouped data. This creates new variables at a higher level of grouping. For example we might calculate means with mean() or counts with n(). THis results in a smaller summary table, which we might do more things with if we want.

Create a pipeline of transformations with the pipe operator

You can read the pipeline operator as "and then" or in shell "|"

```
rel_by_region <- gss_sm %>%
    group_by(bigregion, religion) %>% # Nest inwards by going left to right. First the bigger, then the
    summarize(N = n()) %>% # Notice we're not making new objects, just piping it downwards. Summar
    mutate(freq = N / sum(N),
           pct = round((freq*100), 1)) # Round it off to one decimal place
```

Objects in a pipeline carry forward some assumption about context. We create variables on the fly.

Grouping with group_by() carries forward; summary calculations are applied to the innermost group, and returned for the next group up. Summarize peel off each level of the grouping and return the results to the group up.

Mutate did not change the grouping level, just created a new column / variable at the level of the grouping.

Use pipelines to create summary table objects, then graph them.

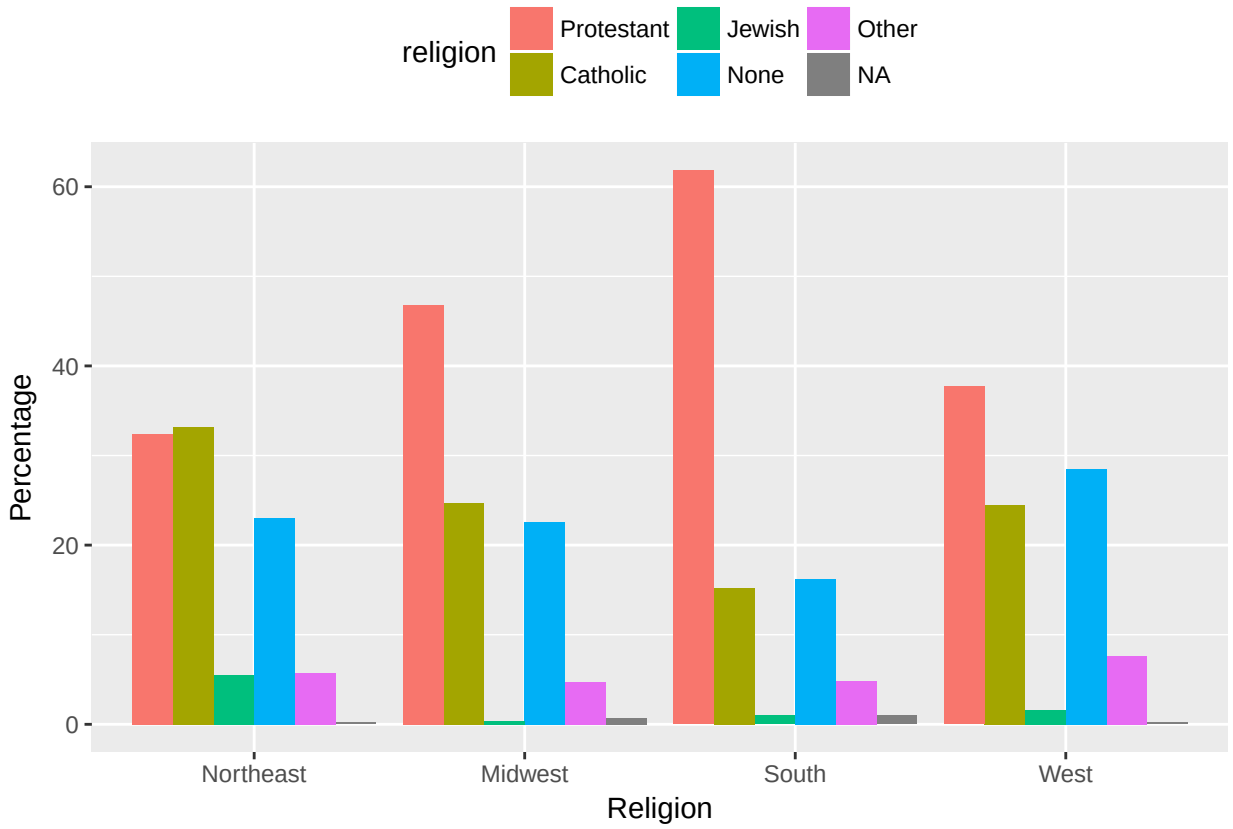
Lets do a sanity check:

```
rel_by_region %>%
  group_by(bigregion) %>%
  summarize(total = sum(pct))
```

```
## # A tibble: 4 x 2
##   bigregion total
##      <fctr> <dbl>
## 1 Northeast 100.0
## 2   Midwest  99.9
## 3     South 100.0
## 4      West 100.1
```

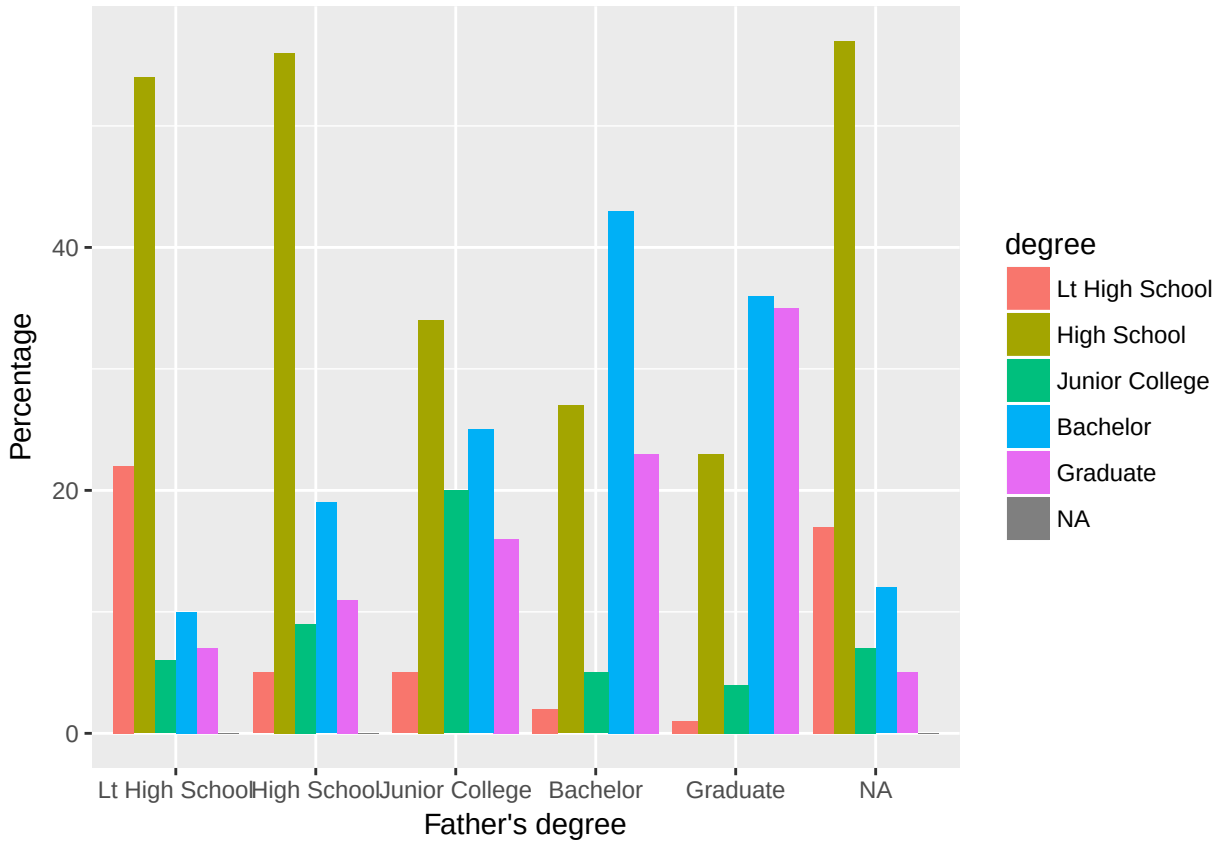
Perfect (or kinda: errors from summing up. ACCEPTABLE)

```
p <- ggplot(data = rel_by_region,
     mapping = aes(x = bigregion,
                   y = pct,
                   fill = religion))
p + geom_bar(position = "dodge",
             stat = "identity") +
    labs(x = "Religion",
         y = "Percentage",
         fill = "religion") +
    theme(legend.position = "top")
```



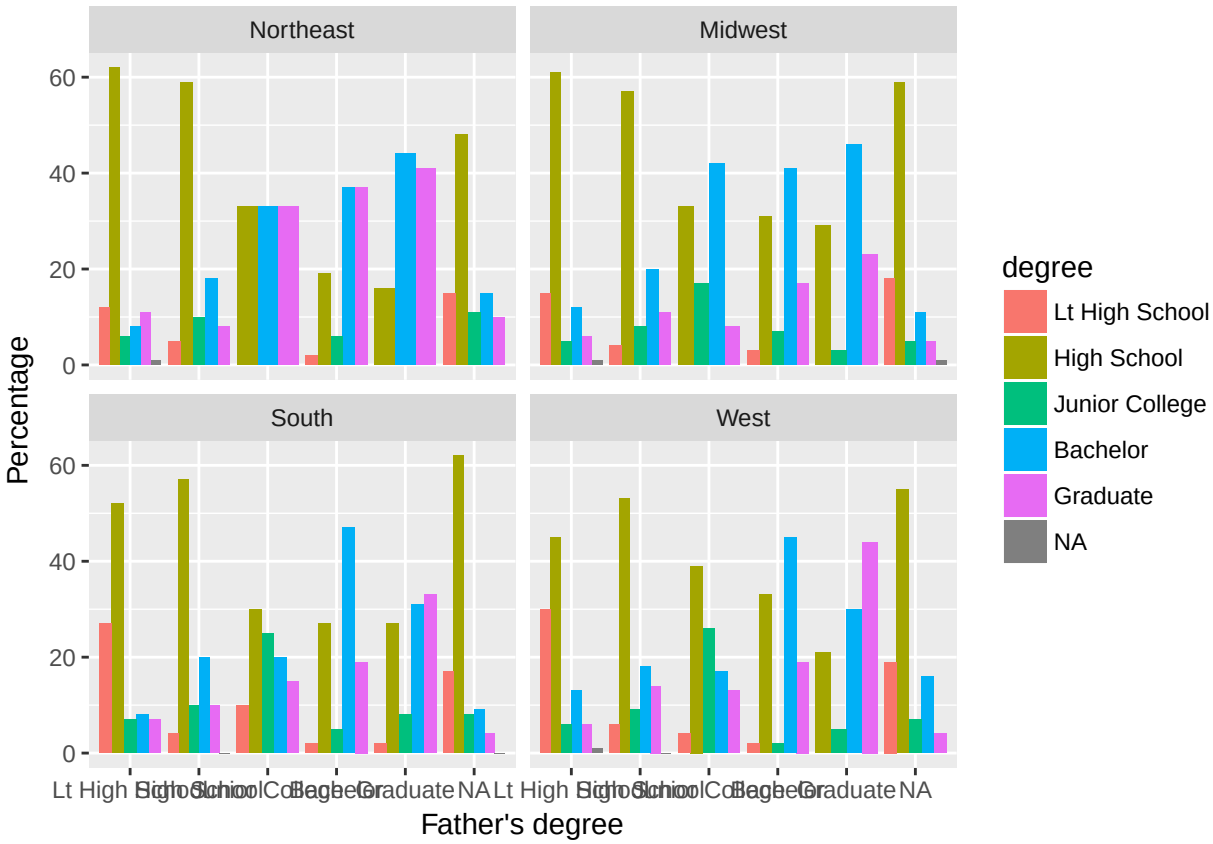
So lets look at education mobility:

```
degree_by_pa <- gss_sm %>%
  group_by(padeg, degree) %>%
  summarize(N = n()) %>%
  mutate(freq = N / sum(N),
         pct = round(freq*100), 1)

p <- ggplot(data = degree_by_pa,
            mapping = aes(x = padeg,
                          y = pct,
                          fill = degree))
p + geom_bar(position = "dodge",
             stat = "identity") +
    labs( x = "Father's degree",
          y = "Percentage")
```



And faceted by bigregion

```
degree_by_pa <- gss_sm %>%
  group_by(bigregion, padeg, degree) %>% # It needed to be grouped by bigregion as well
  summarize(N = n()) %>%
  mutate(freq = N / sum(N),
         pct = round(freq*100), 1)

p <- ggplot(data = degree_by_pa,
            mapping = aes(x = padeg,
                          y = pct,
                          fill = degree))
p + geom_bar(position = "dodge",
             stat = "identity") +
  facet_wrap(~ bigregion) + # here comes the faceting. Remember; tilde means "by"
    labs( x = "Father's degree",
          y = "Percentage")
```



Working with geoms

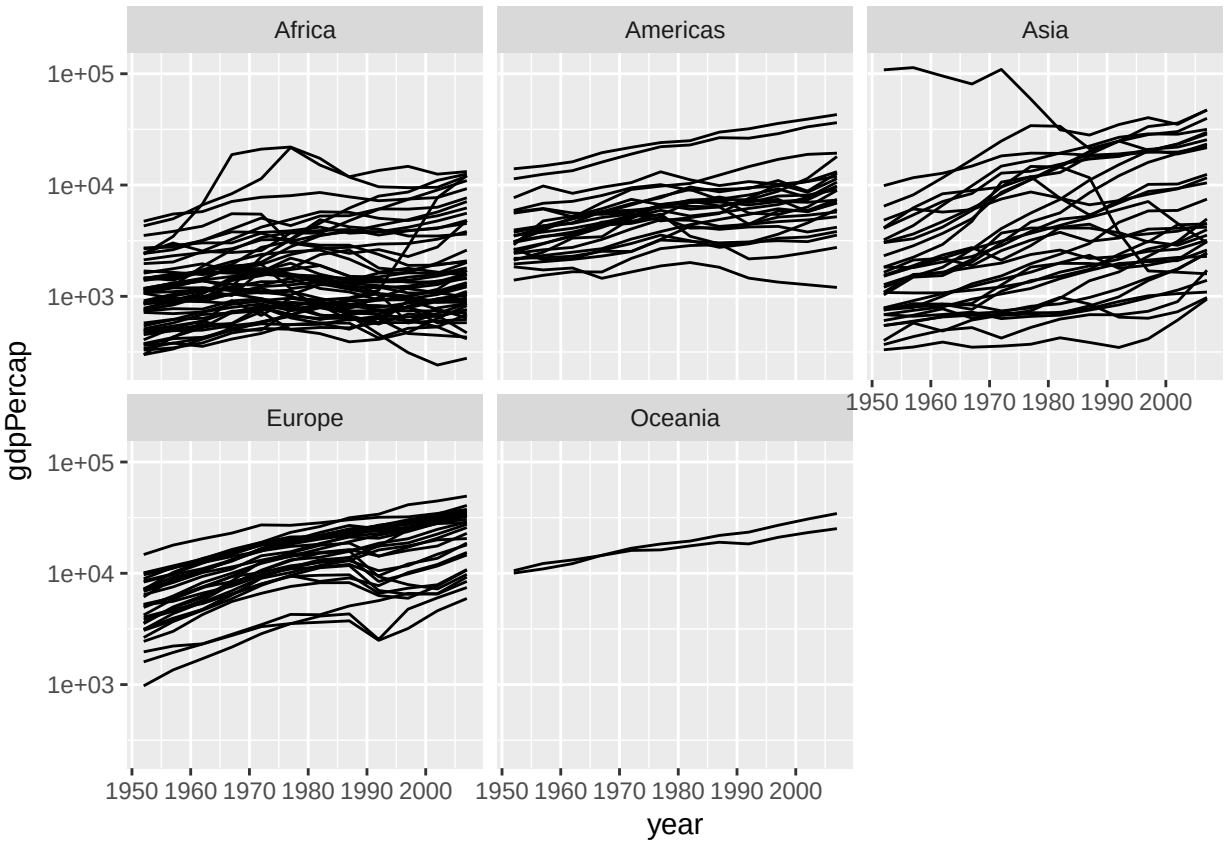
We're expanding our vocabulary. We'll also work with multiple layered plots.

ggplot is a graphing template. We are always starting with our table of tidy data. The steps we take : p <- ggplot (data = , mapping=aes((mapping = aes(), stat = , position =) +

+ +)

So using it with gapminder:

```
p <- ggplot(data = gapminder,
            mapping = aes(x = year,
                          y = gdpPercap))
p + geom_line(aes(group = country)) +
    scale_y_log10() +
    coord_cartesian() +
    facet_wrap(~ continent)
```



geom_point() geom_line() geom_smooth() etc...

The organ donation data

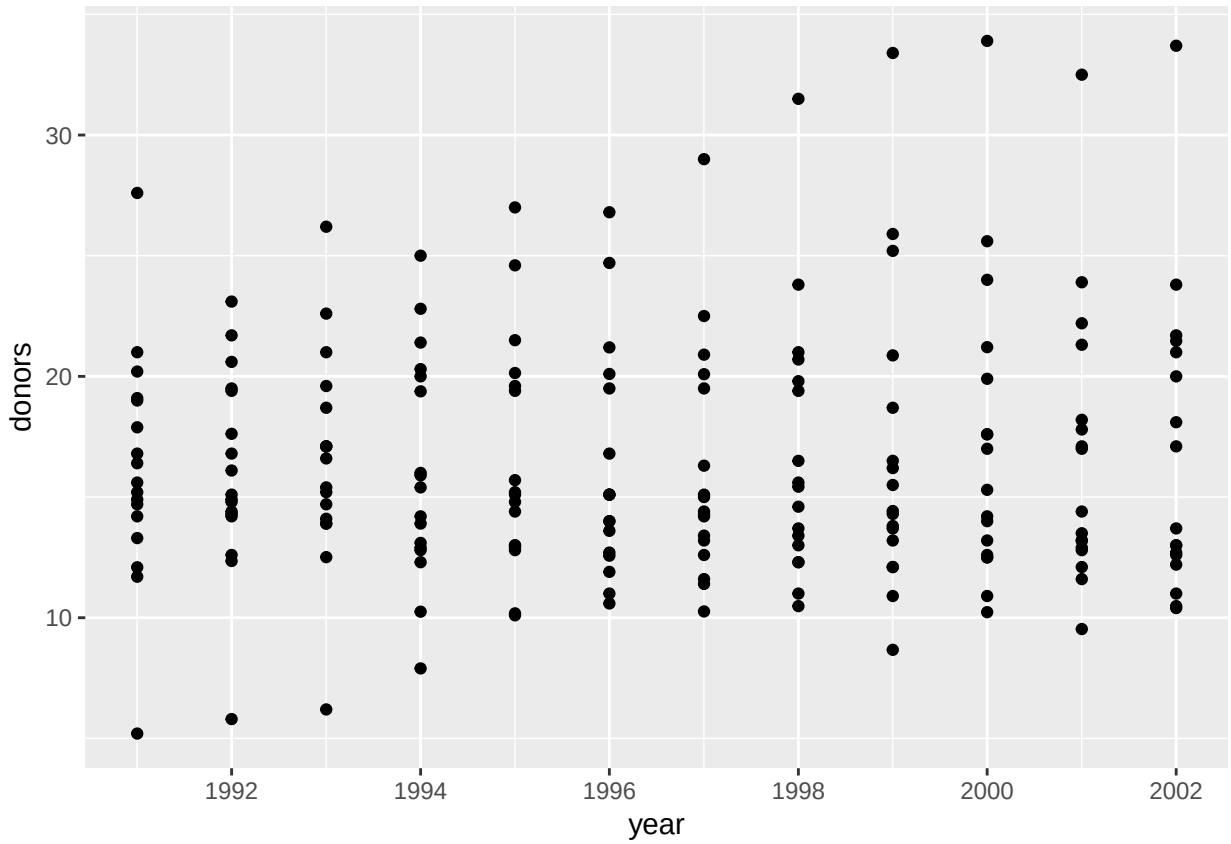
Collected by Kieran. Looking at it with pipelines:

```
organdata %>% select(1:6) %>% sample_n(size = 10)
```

```
## # A tibble: 10 x 6
##        country       year donors   pop   pop.dens   gdp
##          <chr>     <date>  <dbl> <int>      <dbl> <int>
##  1   Australia         NA     NA 17065  0.2204433 16774
##  2     Germany 1997-01-01   13.2 82035 22.9770608 22589
##  3 Switzerland 1993-01-01   16.6  6938 16.8031000 25316
##  4     Austria         NA     NA    NA         NA    NA
##  5       Spain         NA     NA    NA         NA    NA
##  6 Netherlands 2001-01-01   11.6 16046 38.6371298 28756
##  7      Norway 1997-01-01   15.1  4405  1.3600716 27784
##  8     Germany 1998-01-01   13.4 82047 22.9804218 23283
##  9     Germany         NA     NA 63254 17.7167185 20359
## 10     Germany 2002-01-01   12.2 82489 23.1042209 25843
```

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

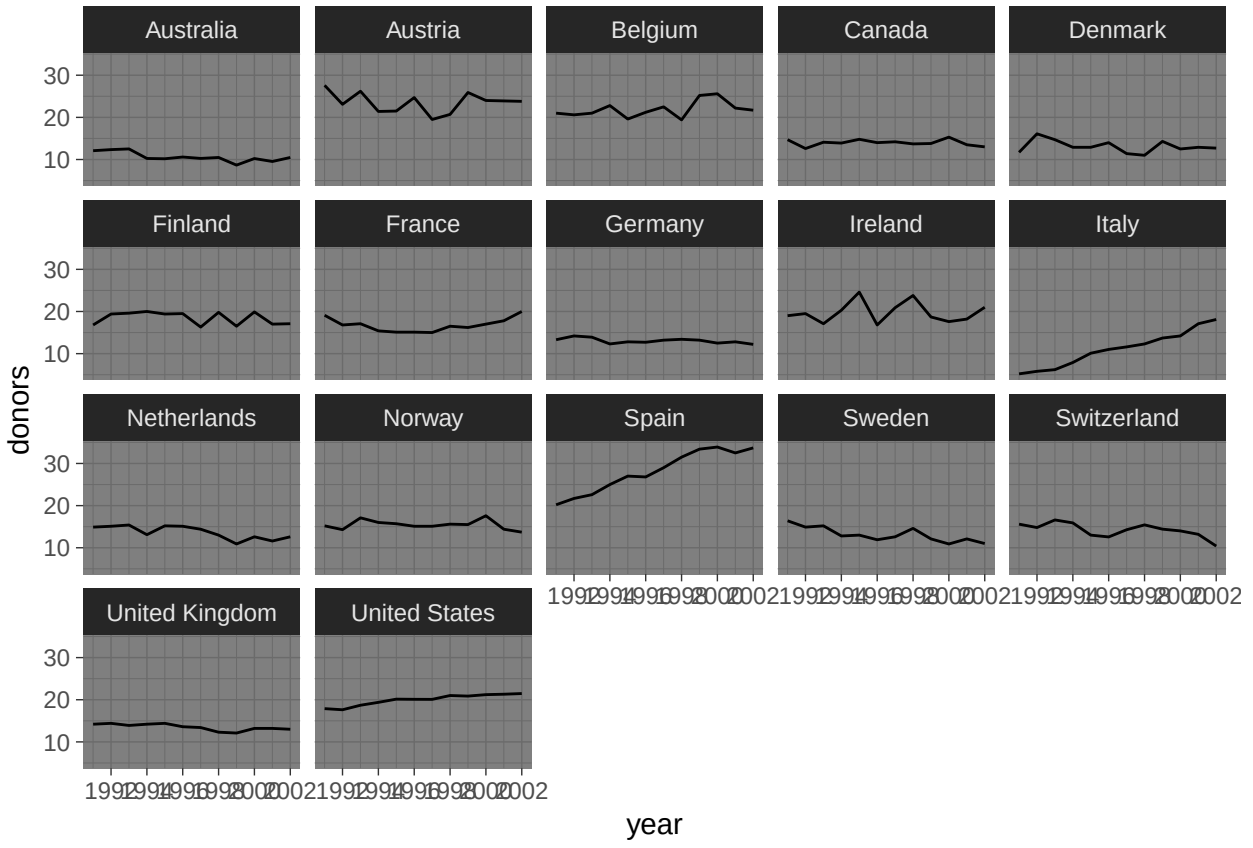
```
## Warning: Removed 34 rows containing missing values (geom_point).
```



The warning is not critical, but should be payed attention to.

```
p <- ggplot(data = organdata,
            aes(x = year,
                y = donors))

p + geom_line() +
    facet_wrap(~ country) +
    theme_dark()
```

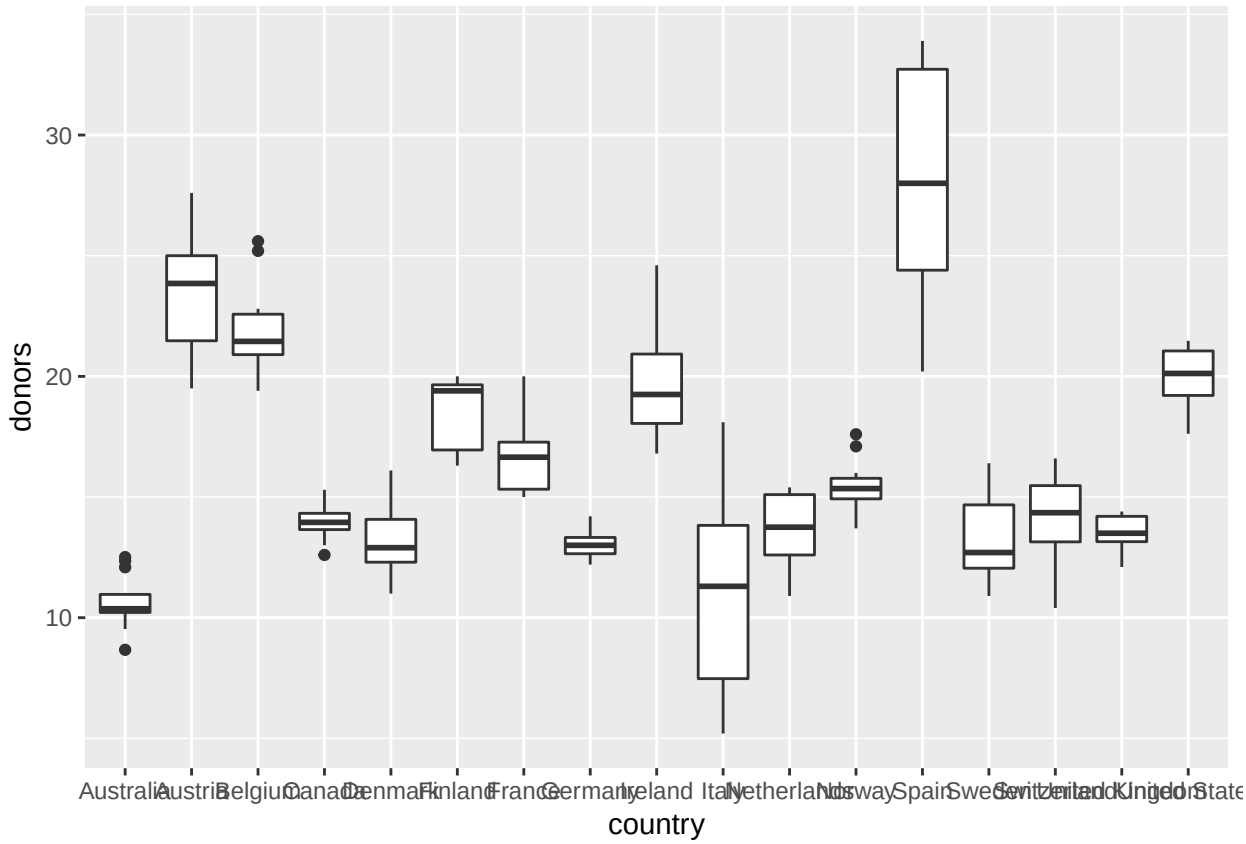
```
## Warning: Removed 2 rows containing missing values (geom_path).
```



Boxplots

```
p <- ggplot(data = organdata,
            mapping = aes(x = country, y = donors))
p + geom_boxplot() +
    theme_gray()
```

```
## Warning: Removed 34 rows containing non-finite values (stat_boxplot).
```

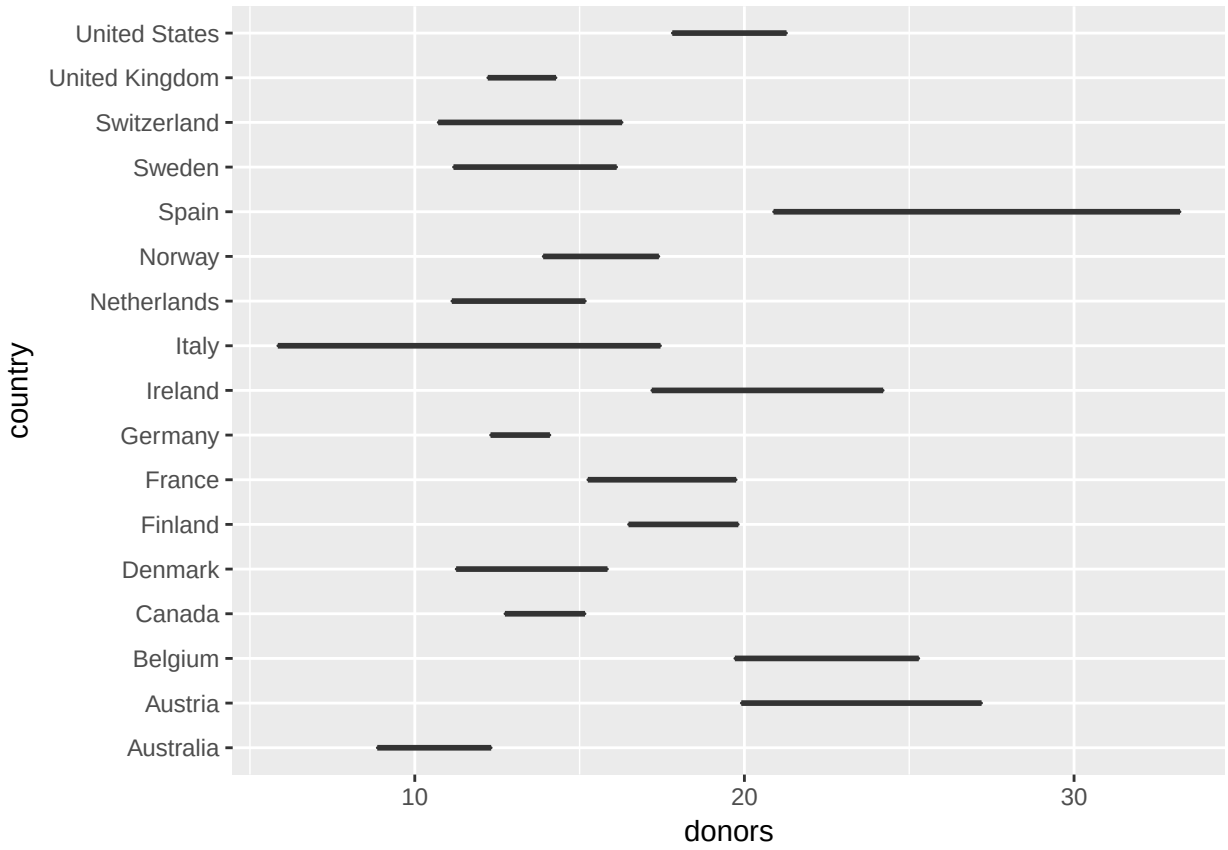


Boxplots are similar to scatterplots, they should probably have X and Y. But it's not showing us the crossregional distribution.

```
p <- ggplot(data = organdata,
            aes(x = donors, y = country))
p + geom_boxplot()
```

```
## Warning: Removed 34 rows containing non-finite values (stat_boxplot).
```

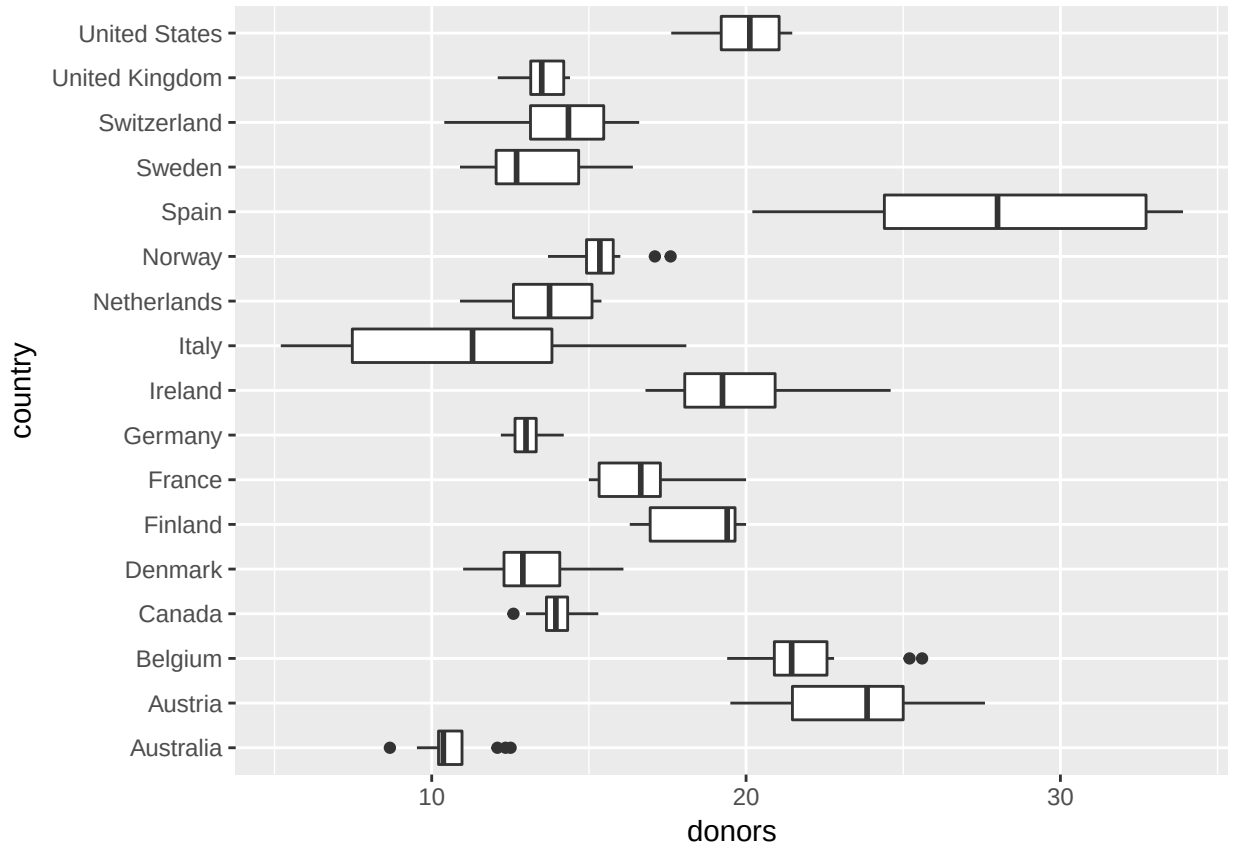
```
## Warning: position_dodge requires non-overlapping x intervals
```



Boxplot does not like it that way.

```
p <- ggplot(data = organdata,
            mapping = aes(x = country, y = donors))
p + geom_boxplot() +
    coord_flip()
```

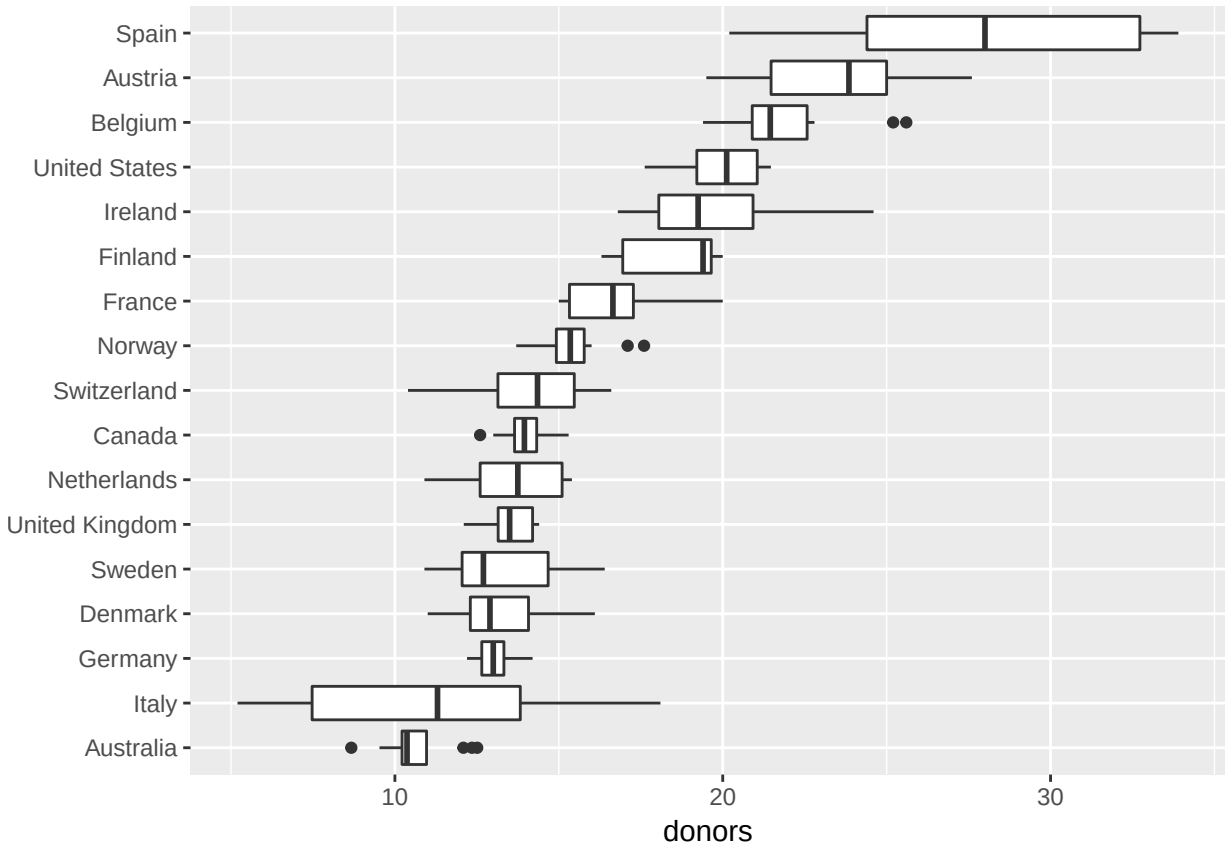
```
## Warning: Removed 34 rows containing non-finite values (stat_boxplot).
```



Lets order the categorical variabel by donor

```
p <- ggplot(data = organdata,
            mapping = aes(x = reorder(country, donors, na.rm=TRUE), y = donors))
p + geom_boxplot() +
  coord_flip() + # Flipping it makes it right
  labs(x = NULL)
```

```
## Warning: Removed 34 rows containing non-finite values (stat_boxplot).
```

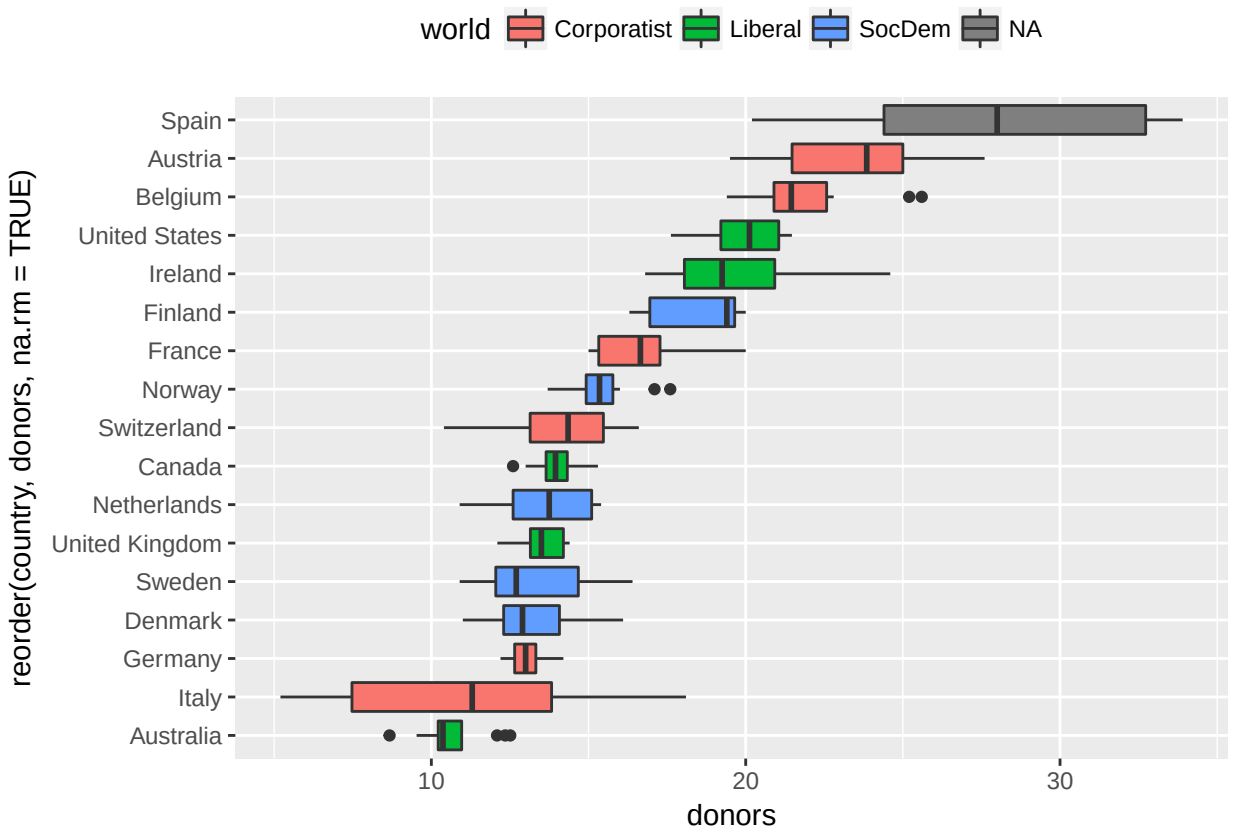


The reorder function takes variable, by-variable, the default function is mean() (but can also take median() std()). And we asked it to not calculate NA into the mean.

Another data set

```
p <- ggplot(data = organdata,
            aes(x = reorder(country, donors, na.rm=TRUE),
                y = donors,
                fill = world))
p + geom_boxplot() +
  coord_flip() +
  theme(legend.position = "top")
```

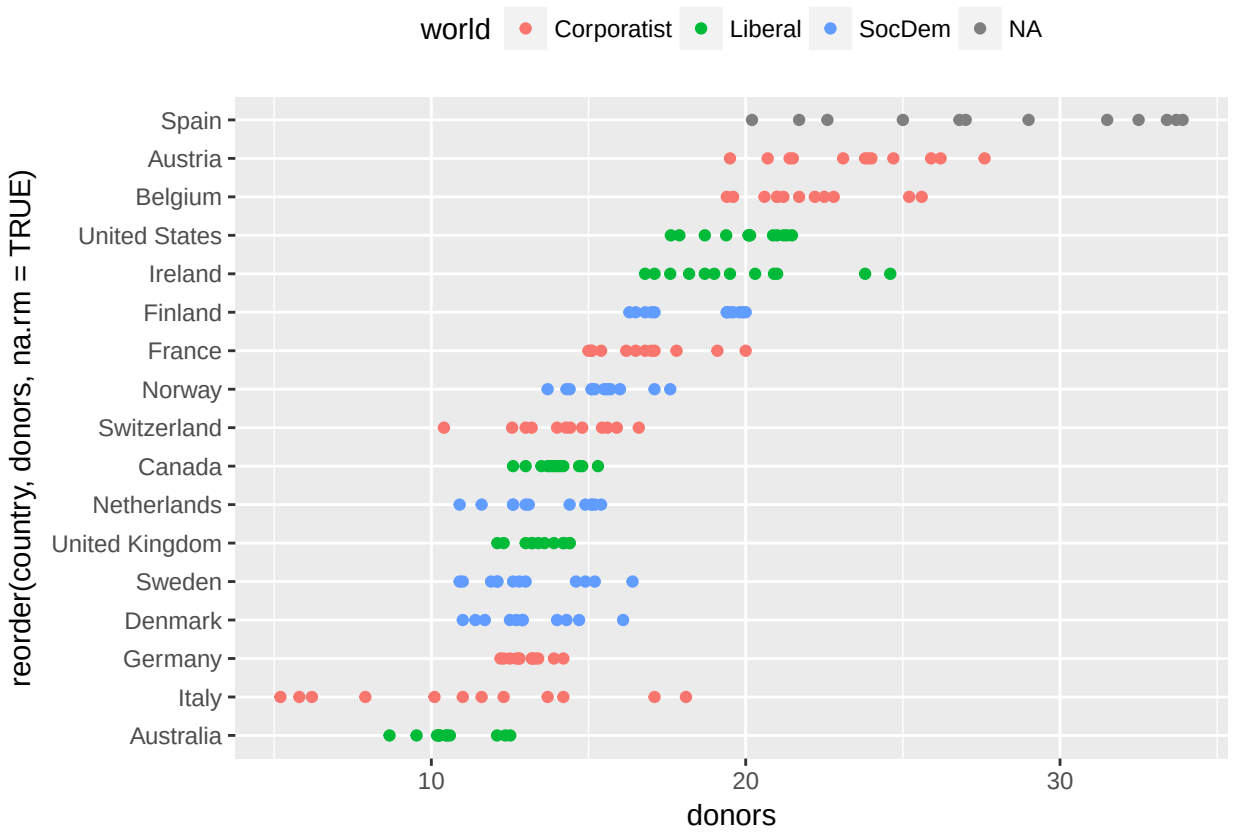
```
## Warning: Removed 34 rows containing non-finite values (stat_boxplot).
```



Lets try to make it a dotplot

```
p <- ggplot(data = organdata,
            aes(x = reorder(country, donors, na.rm=TRUE),
                y = donors,
                color = world))
p + geom_point() +
  coord_flip() +
  theme(legend.position = "top")
```

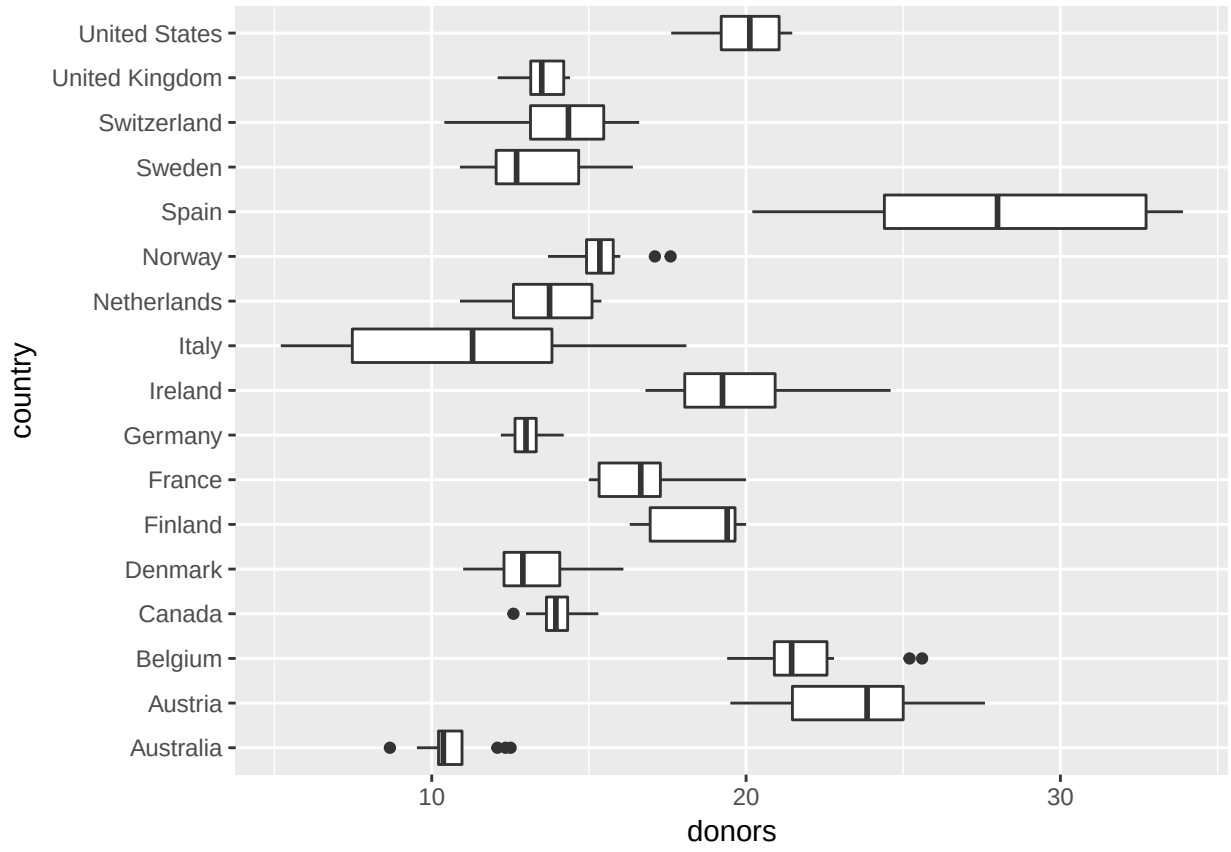
```
## Warning: Removed 34 rows containing missing values (geom_point).
```



geom_jitter can help with overplotting

```
p <- ggplot(data = organdata,
            mapping = aes(x = country, y = donors))
p + geom_boxplot() +
  coord_flip() # Flipping it makes it right
```

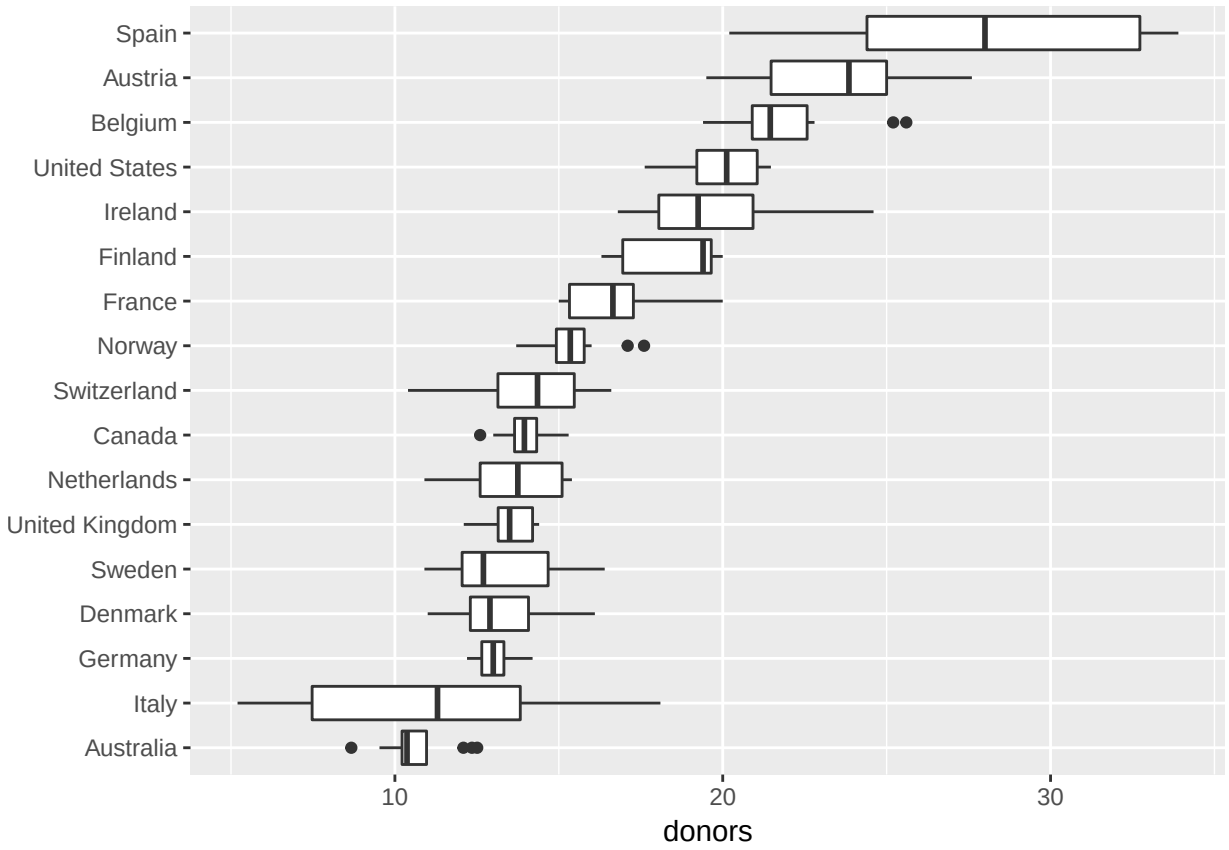
```
## Warning: Removed 34 rows containing non-finite values (stat_boxplot).
```



Lets order the categorical variabel by donor

```
p <- ggplot(data = organdata,
            mapping = aes(x = reorder(country, donors, na.rm=TRUE), y = donors))
p + geom_boxplot() +
  coord_flip() + # Flipping it makes it right
  labs(x = NULL)
```

```
## Warning: Removed 34 rows containing non-finite values (stat_boxplot).
```

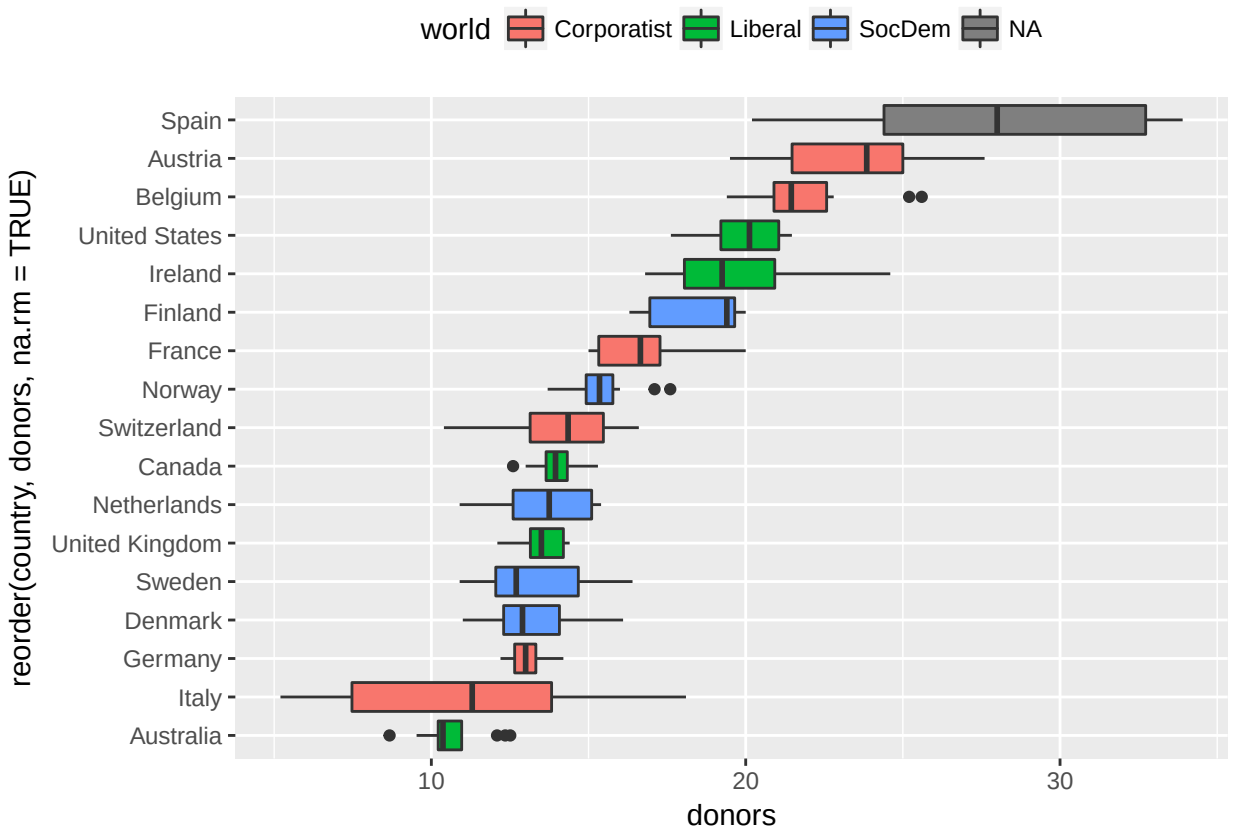


The reorder function takes variable, by-variable, the default function is mean() (but can also take median() std()). And we asked it to not calculate NA into the mean.

Another data set

```
p <- ggplot(data = organdata,
            aes(x = reorder(country, donors, na.rm=TRUE),
                y = donors,
                fill = world))
p + geom_boxplot() +
  coord_flip() +
  theme(legend.position = "top")
```

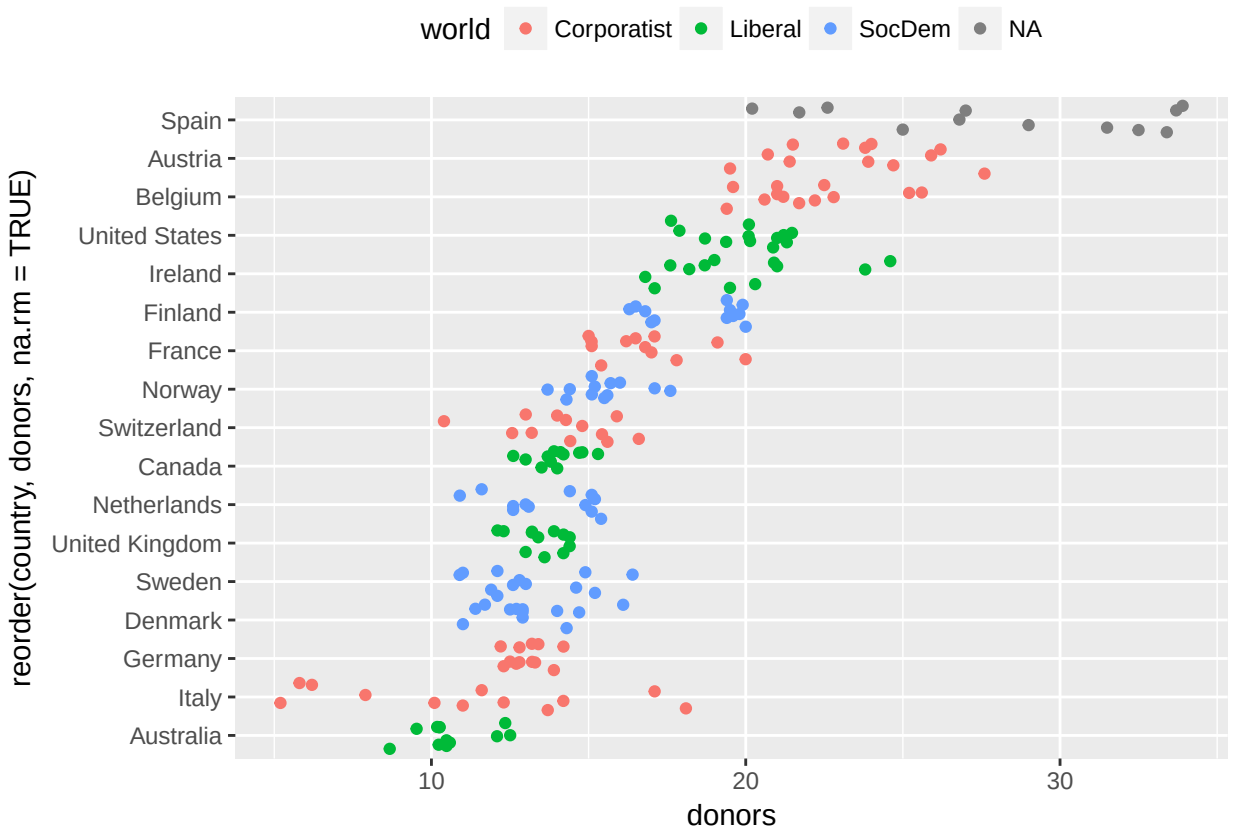
```
## Warning: Removed 34 rows containing non-finite values (stat_boxplot).
```



Lets try to make it a dotplot

```
p <- ggplot(data = organdata,
            aes(x = reorder(country, donors, na.rm=TRUE),
                y = donors,
                color = world))
p + geom_jitter() +
  coord_flip() +
  theme(legend.position = "top")
```

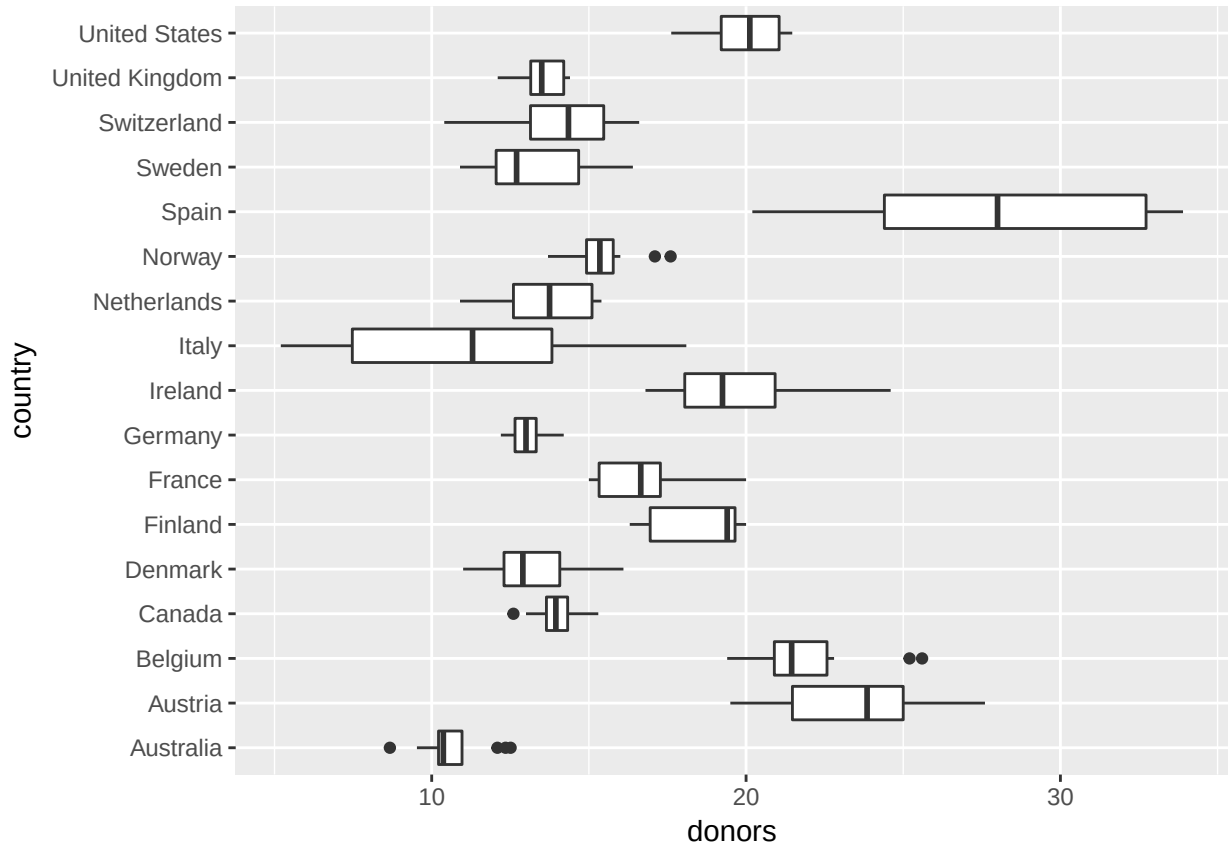
```
## Warning: Removed 34 rows containing missing values (geom_point).
```



But it becomes a little too much. Lets reduce the jitter

```
p <- ggplot(data = organdata,
            mapping = aes(x = country, y = donors))
p + geom_boxplot() +
  coord_flip() # Flipping it makes it right
```

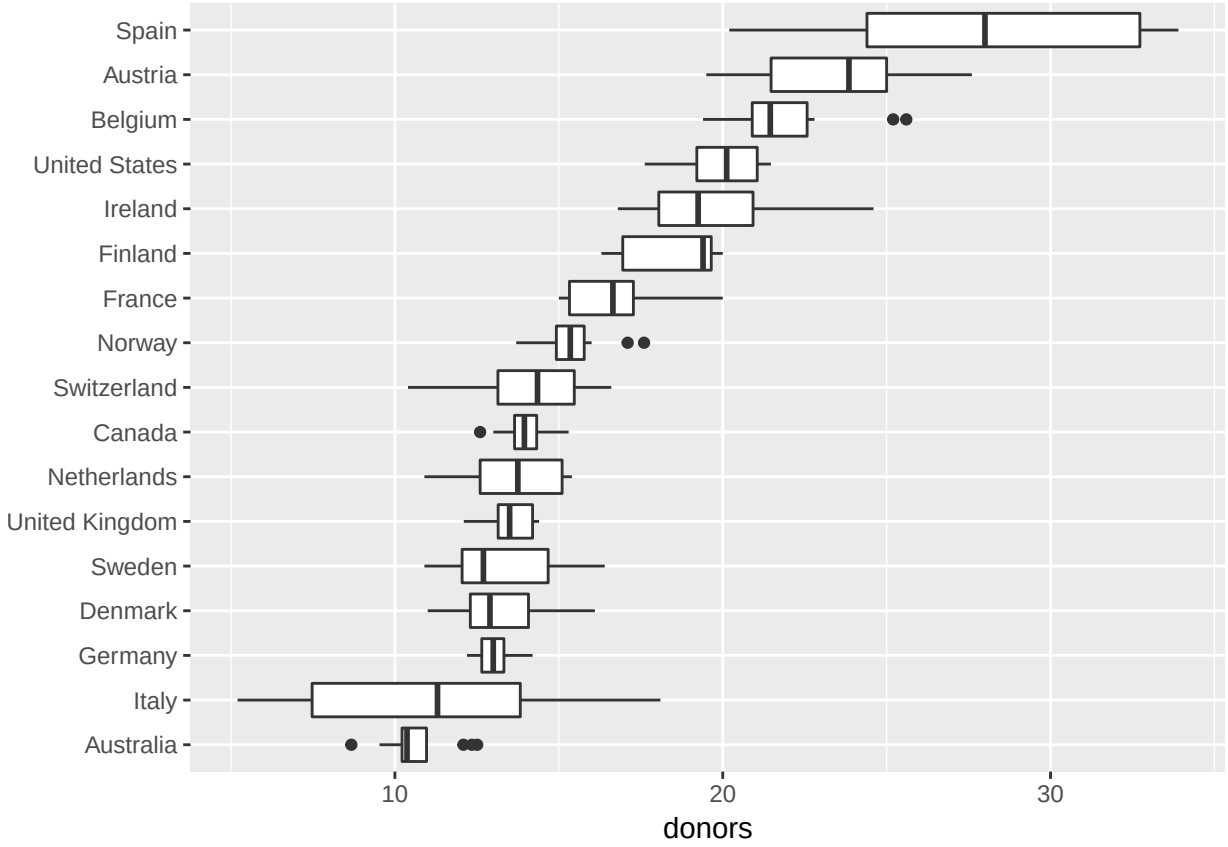
```
## Warning: Removed 34 rows containing non-finite values (stat_boxplot).
```



Lets order the categorical variabel by donor

```
p <- ggplot(data = organdata,
            mapping = aes(x = reorder(country, donors, na.rm=TRUE), y = donors))
p + geom_boxplot() +
  coord_flip() + # Flipping it makes it right
  labs(x = NULL)
```

```
## Warning: Removed 34 rows containing non-finite values (stat_boxplot).
```

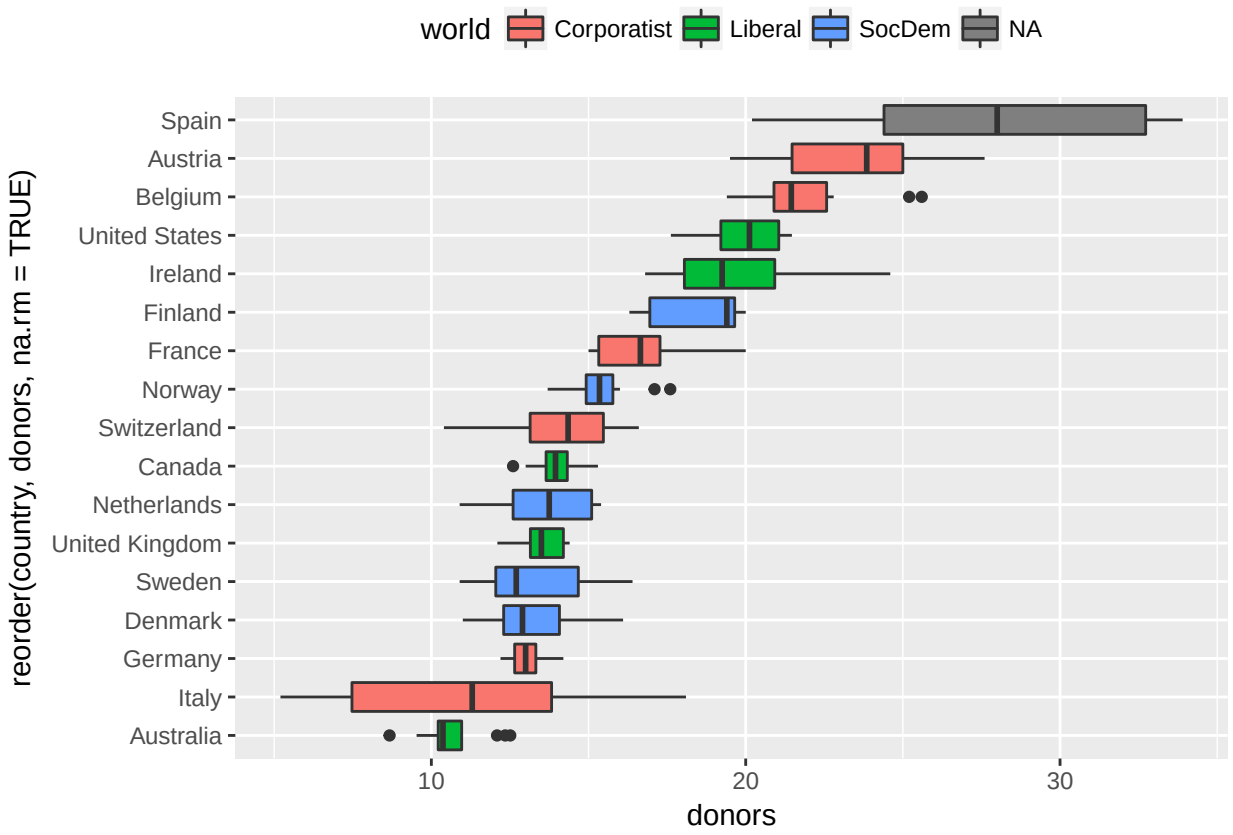


The reorder function takes variable, by-variable, the default function is mean() (but can also take median() std()). And we asked it to not calculate NA into the mean.

Another data set

```
p <- ggplot(data = organdata,
            aes(x = reorder(country, donors, na.rm=TRUE),
                y = donors,
                fill = world))
p + geom_boxplot() +
  coord_flip() +
  theme(legend.position = "top")
```

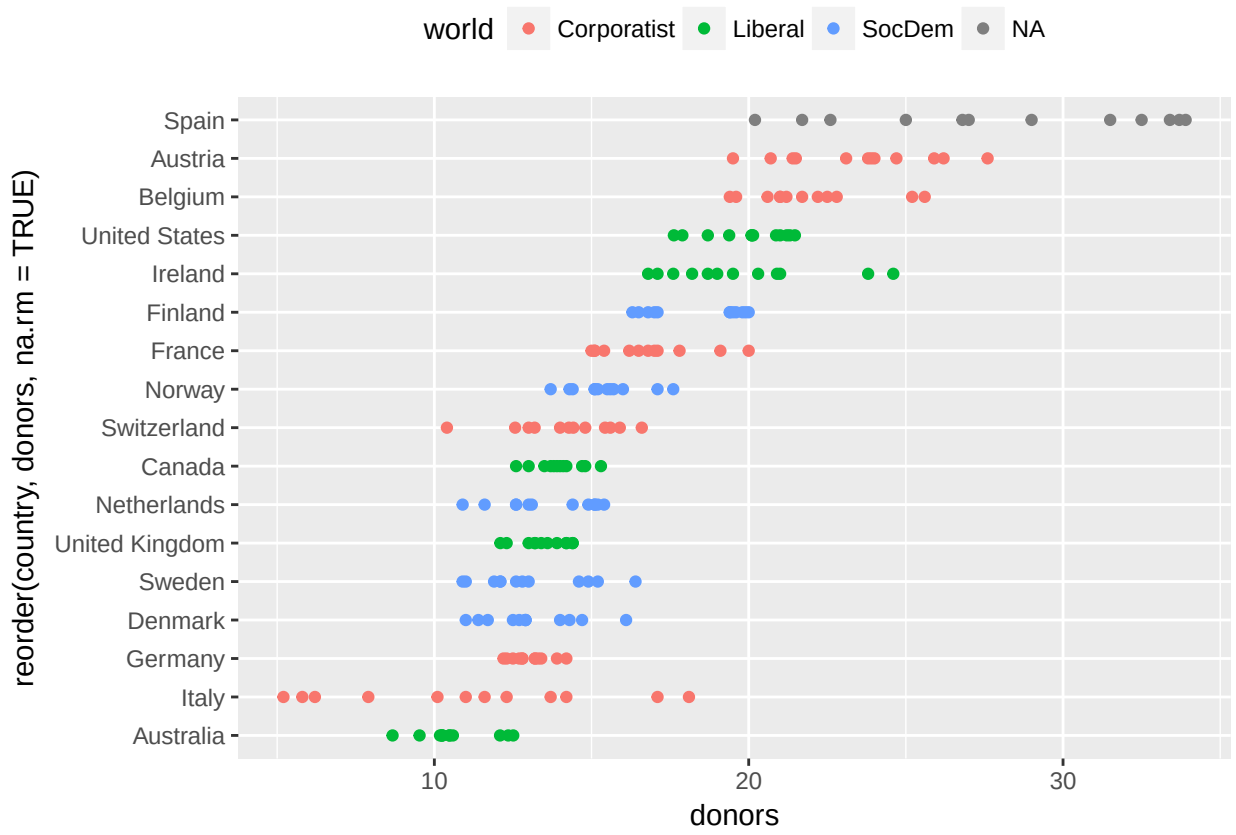
```
## Warning: Removed 34 rows containing non-finite values (stat_boxplot).
```



Lets try to make it a dotplot

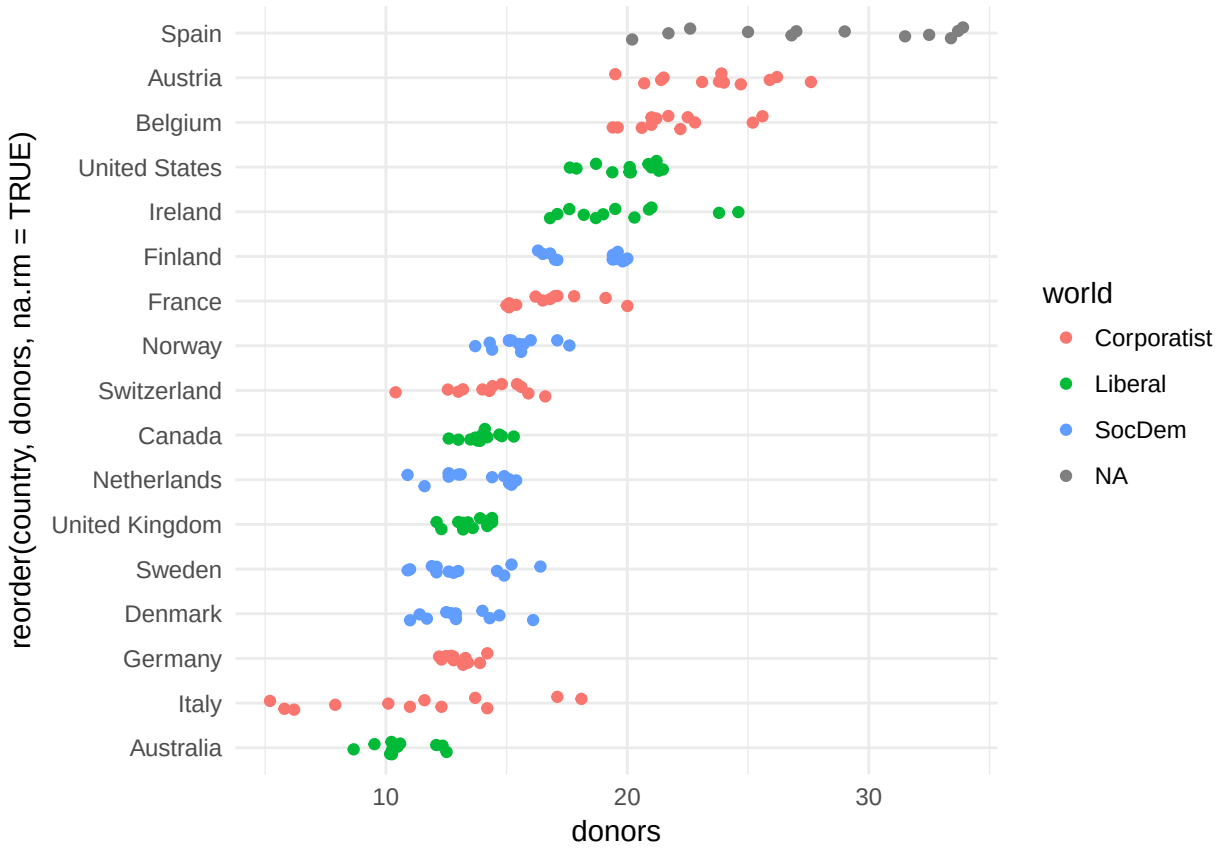
```
p <- ggplot(data = organdata,
            aes(x = reorder(country, donors, na.rm=TRUE),
                y = donors,
                color = world))
p + geom_point() +
  coord_flip() +
  theme(legend.position = "top")
```

```
## Warning: Removed 34 rows containing missing values (geom_point).
```




```
p <- ggplot(data = organdata,
            aes(x = reorder(country, donors, na.rm=TRUE),
                y = donors,
                color = world))
p + geom_jitter(position = position_jitter(width = 0.15)) + # reducing jitter a bit
  coord_flip() +
  theme(legend.position = "top") +
  theme_minimal()
```

```
## Warning: Removed 34 rows containing missing values (geom_point).
```



Summarize with dplyr

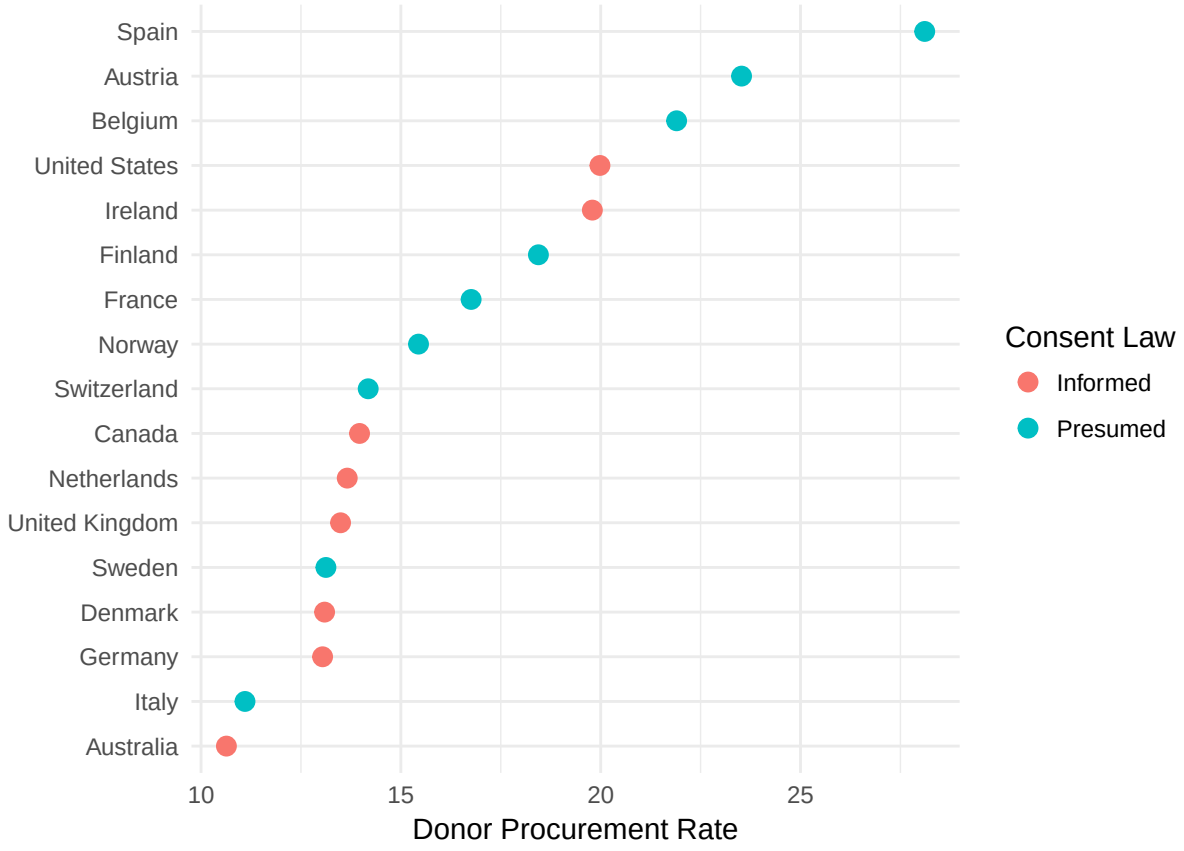
```
by_country <- organdata %>% group_by(consent.law, country) %>%
  summarize(don.rate = mean(donors, na.rm = TRUE),
            don.sd = sd(donors, na.rm = TRUE),
            gdp = mean(gdp, na.rm = TRUE),
            health = mean(health, na.rm = TRUE),
            roads = mean(roads, na.rm = TRUE),
            cerebvas = mean(cerebvas, na.rm = TRUE))
by_country
```

```
## Source: local data frame [17 x 8]
## Groups: consent.law [?]
##
## # A tibble: 17 x 8
##    consent.law        country don.rate    don.sd      gdp   health
##          <chr>          <chr>    <dbl>     <dbl>    <dbl>    <dbl>
##  1    Informed      Australia 10.63500 1.1428075 22178.54 1957.500
##  2    Informed         Canada 13.96667 0.7511607 23711.08 2271.929
##  3    Informed        Denmark 13.09167 1.4681208 23722.31 2054.071
##  4    Informed        Germany 13.04167 0.6111960 22163.23 2348.750
##  5    Informed        Ireland 19.79167 2.4784373 20824.38 1479.929
##  6    Informed    Netherlands 13.65833 1.5518074 23013.15 1992.786
##  7    Informed United Kingdom 13.49167 0.7751344 21359.31 1561.214
##  8    Informed  United States 19.98167 1.3253667 29211.77 3988.286
```

```
##  9    Presumed        Austria 23.52500 2.4159037 23875.85 1875.357
## 10    Presumed        Belgium 21.90000 1.9357874 22499.62 1958.357
## 11    Presumed        Finland 18.44167 1.5264089 21018.92 1615.286
## 12    Presumed         France 16.75833 1.5974174 22602.85 2159.643
## 13    Presumed          Italy 11.10000 4.2769998 21554.15 1757.000
## 14    Presumed         Norway 15.44167 1.1090195 26448.38 2217.214
## 15    Presumed          Spain 28.10833 4.9630376 16933.00 1289.071
## 16    Presumed         Sweden 13.12500 1.7535030 22415.46 1951.357
## 17    Presumed    Switzerland 14.18250 1.7090940 27233.00 2776.071
## # ... with 2 more variables: roads <dbl>, cerebvas <dbl>
```

Now that we've made a nice object, lets ggplot it

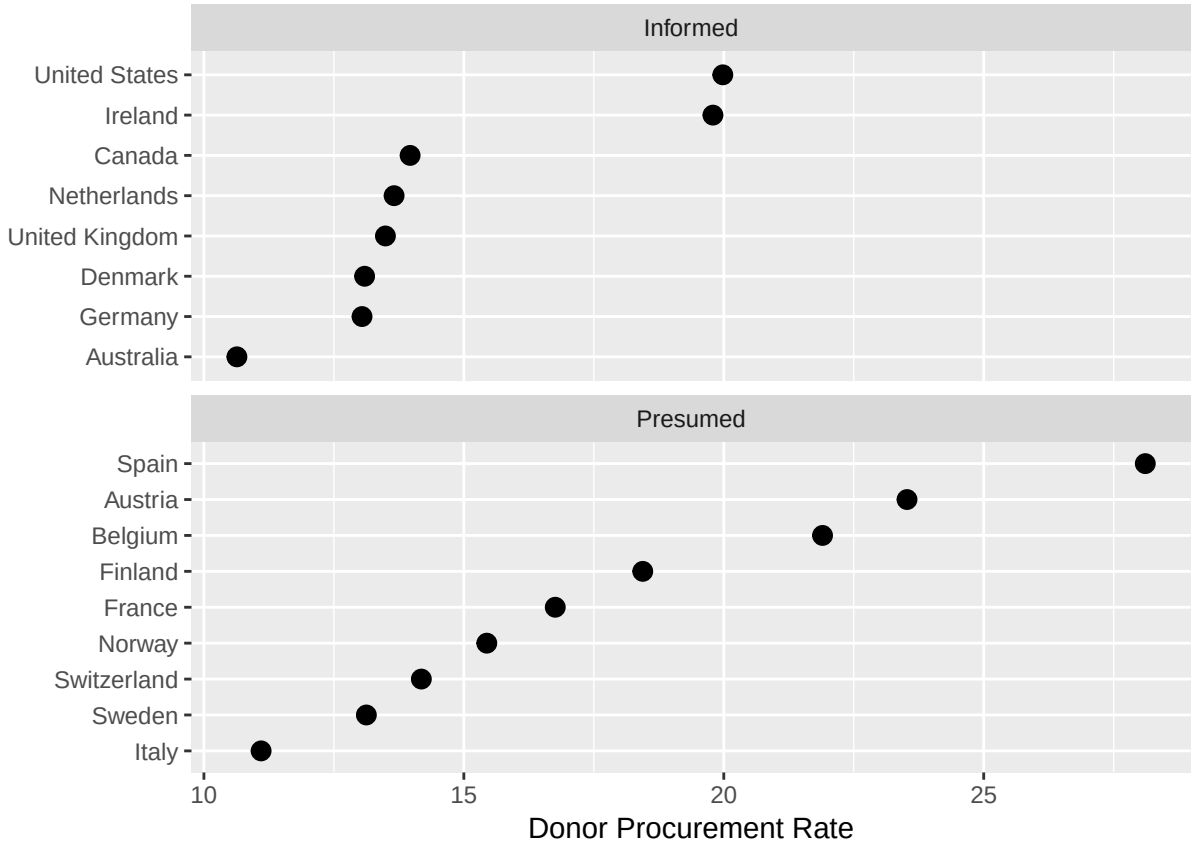
```
p <- ggplot(data = by_country,
            aes( x = don.rate,
                 y = reorder(country, don.rate),
                 color = consent.law))
p + geom_point(size=3) +
  labs(x="Donor Procurement Rate",
       y="",
       color="Consent Law") +
  theme(legend.position = "top") +
  theme_minimal()
```



Or we can facet it. See that we use free_y to get countries at each facet, and ncol=1 to only one column with 2 rows.

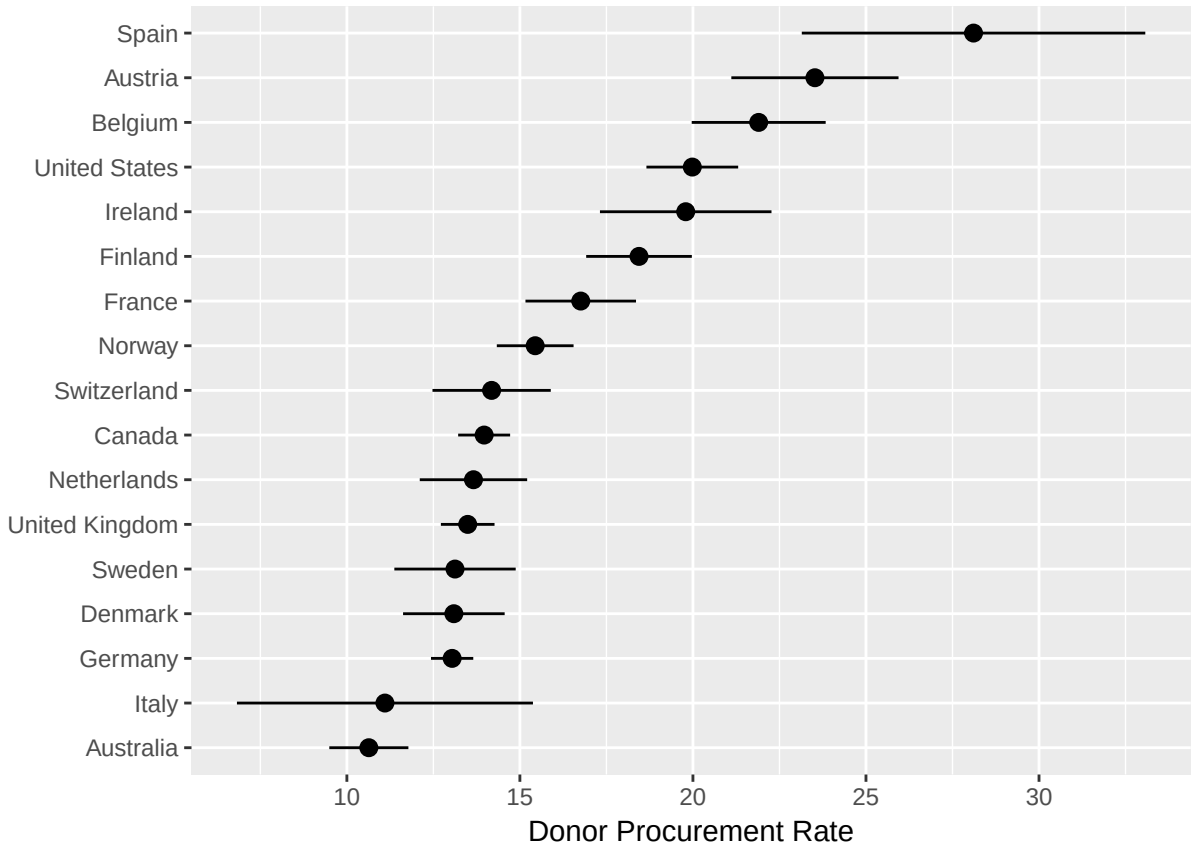
```
p <- ggplot(data = by_country,
```

```
            mapping = aes(x = don.rate,
                          y = reorder(country, don.rate)))
p + geom_point(size=3) +
  facet_wrap(~ consent.law, scale = "free_y", ncol=1) +
  labs(x="Donor Procurement Rate",
       y= "")
```



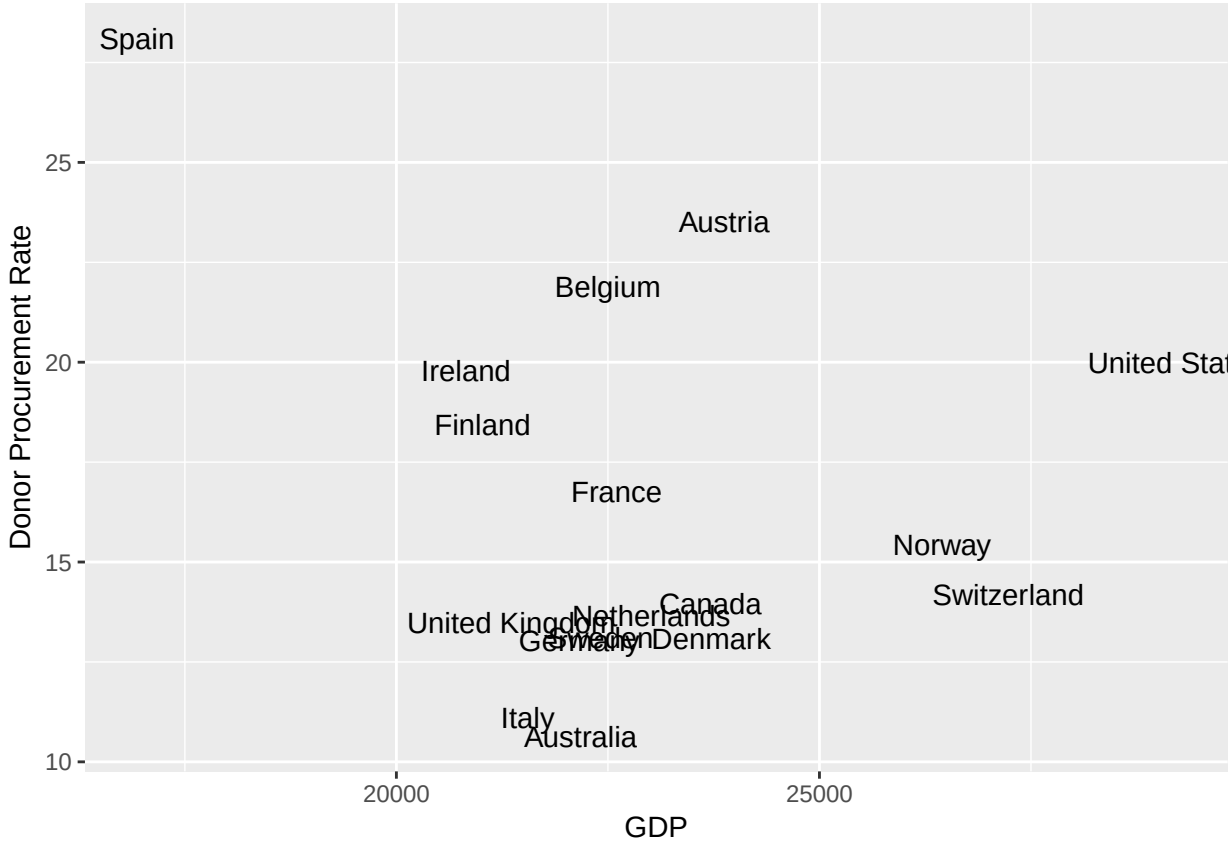
Another geoms pointrange

```
p <- ggplot(data = by_country,
            mapping = aes(x = reorder(country, don.rate),
                          y = don.rate))
p + geom_pointrange(mapping = aes(ymin= don.rate - don.sd,
                                  ymax= don.rate + don.sd)) +
  labs(x="", y="Donor Procurement Rate") +
  coord_flip()
```



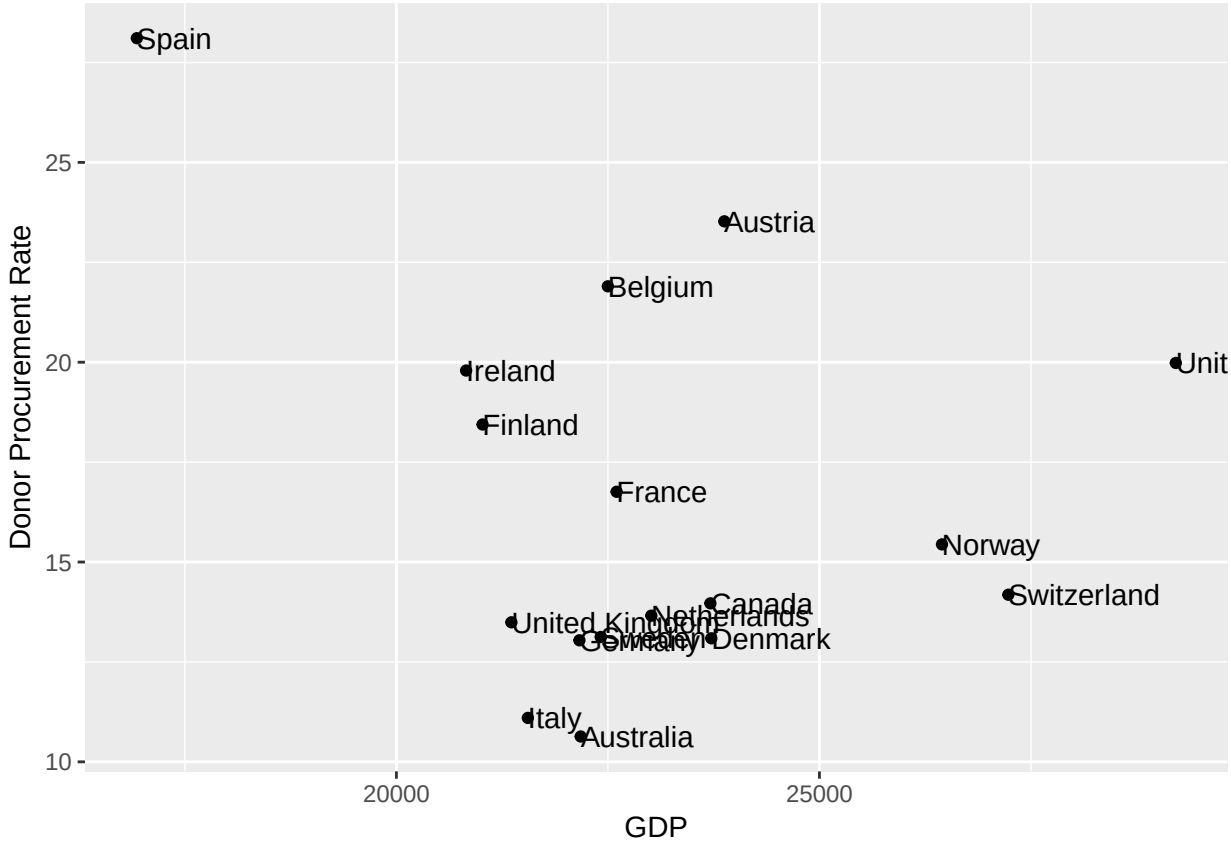
Plotting text directly

```
p <- ggplot(data = by_country,
            mapping = aes(x = don.rate, y = gdp ))
p + geom_text(mapping = aes(label = country)) +
  labs(y="GDP", x="Donor Procurement Rate") +
  coord_flip()
```



More bells with dots and hjust to get labels next to dot. Space is added to proportion of the size of the label. So not satisfactory.

```
p <- ggplot(data = by_country,
            mapping = aes(x = don.rate, y = gdp ))
p + geom_point() + geom_text(mapping = aes(label = country), hjust = 0) +
  labs(y="GDP", x="Donor Procurement Rate") +
  coord_flip()
```



We fix it with ggrepel: This library provides geom_text_repel() and geom_label_repel()

```
library("ggrepel")
```

Moar data

```
elections_historic %>% select(2:7)
```

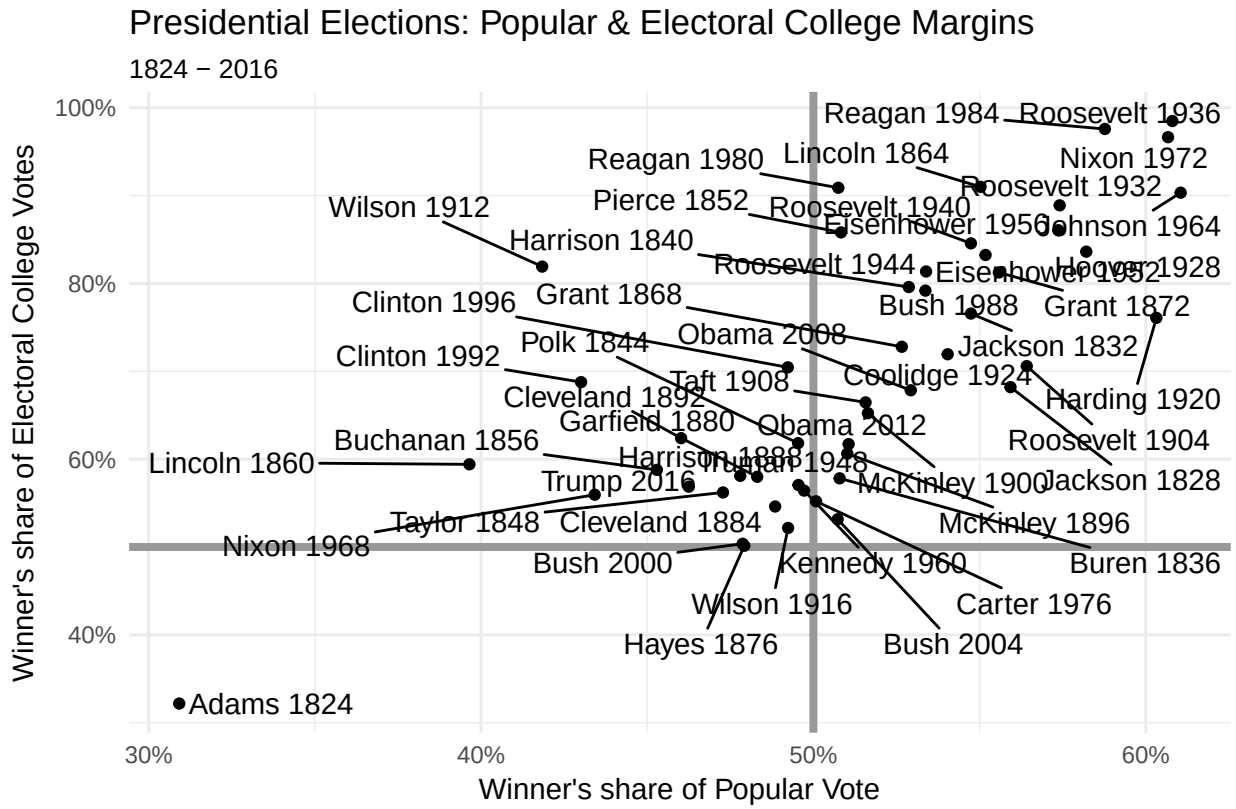
```
## # A tibble: 49 x 6
##     year                winner win_party ec_pct popular_pct
##    <int>                 <chr>     <chr>  <dbl>       <dbl>
##  1  1824      John Quincy Adams     D.-R. 0.3218      0.3092
##  2  1828         Andrew Jackson      Dem. 0.6820      0.5593
##  3  1832         Andrew Jackson      Dem. 0.7657      0.5474
##  4  1836       Martin Van Buren      Dem. 0.5782      0.5079
##  5  1840 William Henry Harrison      Whig 0.7959      0.5287
##  6  1844             James Polk      Dem. 0.6182      0.4954
##  7  1848         Zachary Taylor      Whig 0.5621      0.4728
##  8  1852        Franklin Pierce      Dem. 0.8581      0.5083
##  9  1856         James Buchanan      Dem. 0.5878      0.4529
## 10  1860        Abraham Lincoln      Rep. 0.5941      0.3965
## # ... with 39 more rows, and 1 more variables: popular_margin <dbl>
```

We'll make a graph of presidents in the US: Popular and electoral college margins.

A bit more complex. Let's make some variables with the titles and texts first, to keep graph code clean.

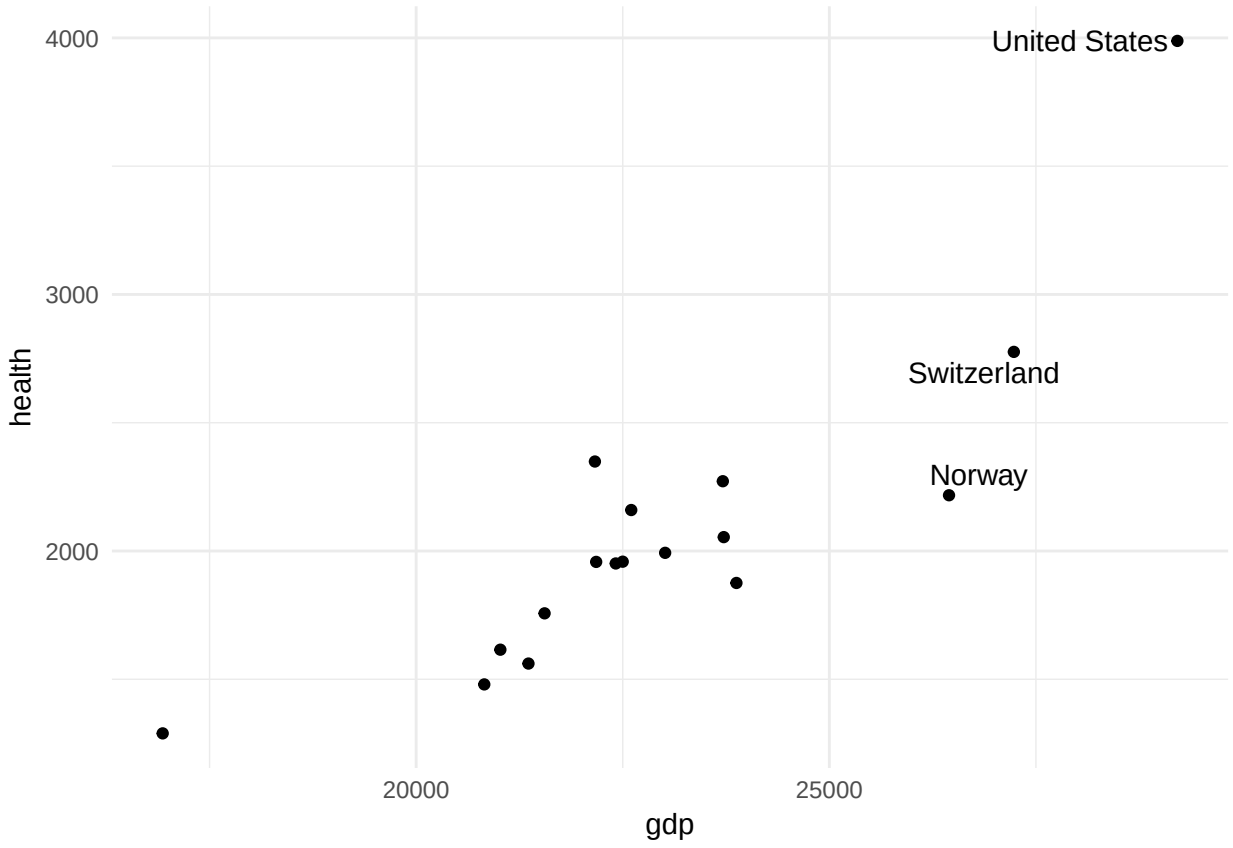
```
p_title <- "Presidential Elections: Popular & Electoral College Margins"
p_subtitle <- "1824 - 2016"
p_caption <- "Data for 2016 are provisional."
```

```
x_label <- "Winner's share of Popular Vote"
y_label <- "Winner's share of Electoral College Votes"

theme_set(theme_minimal())

p <- ggplot(elections_historic, aes(x = popular_pct,
                                    y = ec_pct,
                                    label = winner_label))
p1 <- p + geom_hline(yintercept = 0.5, size = 1.4, color = "gray60") +  # Custom plot lines first, sinc
    geom_vline(xintercept = 0.5, size = 1.4, color = "gray60") +
    geom_point() + # LAyerings: Base layer, Grid Lines, Points
    geom_text_repel() +
    scale_x_continuous(labels = scales::percent) +
    scale_y_continuous(labels = scales::percent) +
    labs(x = x_label,
         y = y_label,
         subtitle = p_subtitle,
         caption = p_caption,
         title = p_title)
print(p1)
```



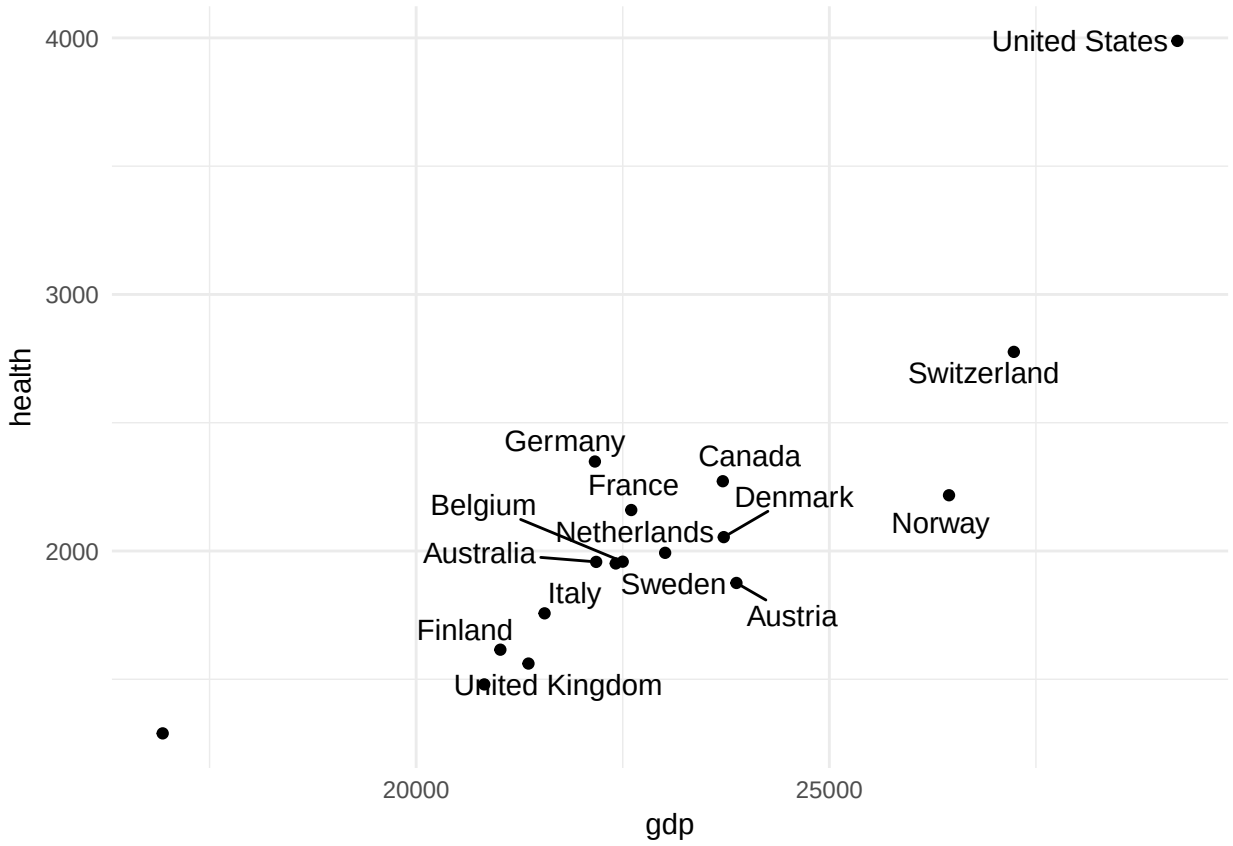
Data for 2016 are provisional.

```
p <- ggplot(data = by_country,
            mapping = aes(x = gdp, y = health))
p + geom_point() +
   geom_text_repel(data = subset(by_country, gdp > 25000),
                   mapping = aes(label = country))
```



Or

```
p <- ggplot(data = by_country,
            aes(x = gdp, y = health))
p + geom_point() +
  geom_text_repel(data = subset(by_country,
                                gdp > 25000 | health > 1500 | country %in% "Belgium"),
                  mapping = aes(label = country))
```



Adding data on the fly

```
organdata$ind <- organdata$ccode %in% c("Ita", "Spa") &
                                 organdata$year > 1988

p <- ggplot(data = organdata,
            mapping = aes(x = roads,
                          y = donors, color = ind))
p + geom_point() +
    geom_text_repel(data = subset(organdata, ind),
                    mapping = aes(label = ccode)) +
  guides(label = FALSE, color = FALSE)
```

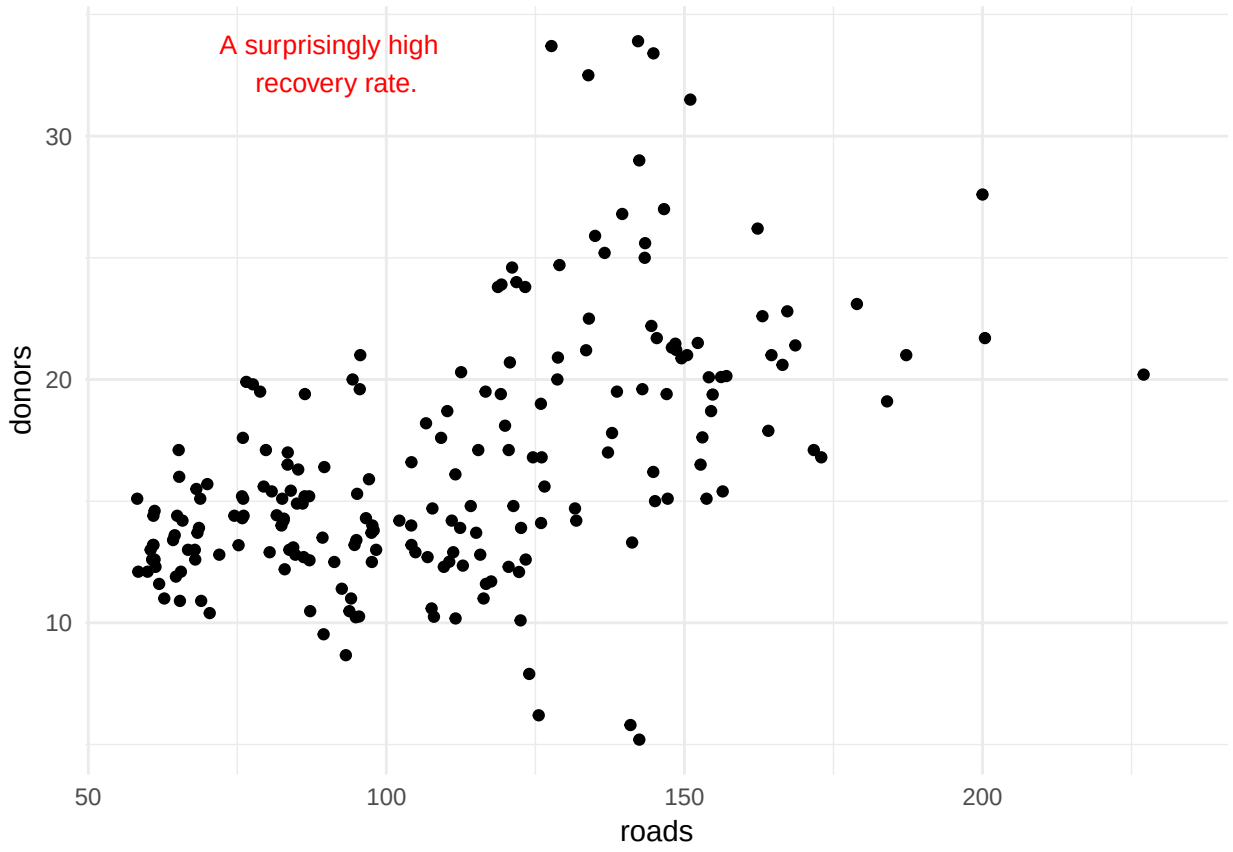
```
## Warning: Removed 34 rows containing missing values (geom_point).
```

Write and draw in the Plot Area

Sometimes you want to draw or write arbitrarily in the plot. For example explain for laypeople.

```
p <- ggplot(data = organdata,
            mapping = aes(x = roads,
                          y = donors))
p + geom_point() +
    annotate(geom = "text", x = 91, y = 33,
             label = "A surprisingly high \n recovery rate.",
             size = 3.5,
             color = "red")
```

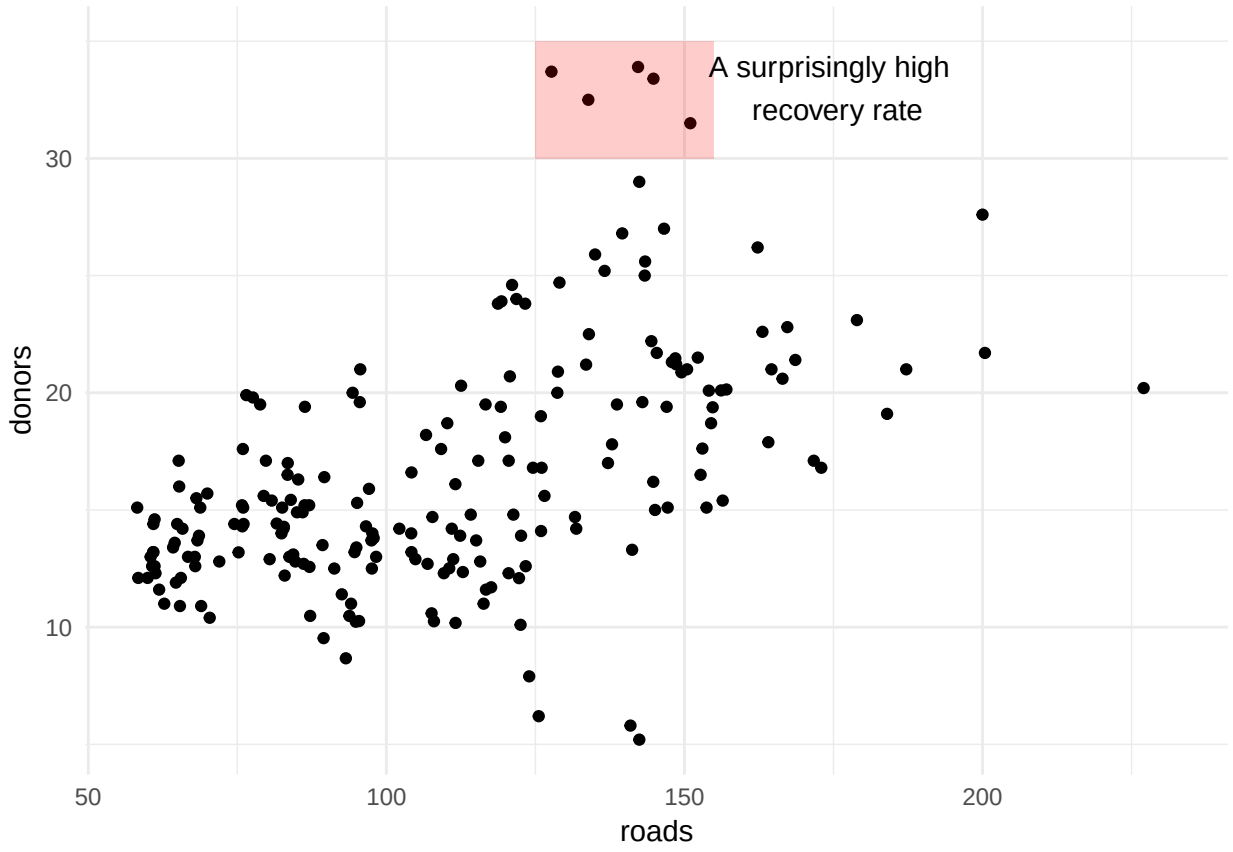
```
## Warning: Removed 34 rows containing missing values (geom_point).
```



Moar annotations

```
p <- ggplot(data = organdata,
            mapping = aes(x = roads, y = donors))
p + geom_point() +
    annotate(geom = "rect", xmin = 125, xmax = 155, ymin = 30, ymax = 35, fill = "red", alpha = 0.2) +
    annotate(geom = "text", label = "A surprisingly high \n recovery rate", x = 175, y = 33)
```

```
## Warning: Removed 34 rows containing missing values (geom_point).
```



Scales, guides and themes

Every aesthetic mapping has a scale. If you want to adjust how that scale is marked or graduated, then you use a scale_ function.

Many scales come with a legend or key to help the reader interpret the graph. These are called guides. You can make adjusments to them with the guides() function.

Features not strictly connected to the logical structure of the data being deisplayed are adjusted with the theme() function.

Scale functions control scale mappings in geoms. remember: JNot just X, and Y.

This means you can control things like color schemes *for data mappings* through scale functions.

scale___()

Scale functions are consistenly named, by mapping and kind.

scale_x_continuous()

scale_y_continuous()

scale_x_discrete()

scale_y_discrete()

scale_x_log10()

scale_x_sqrt()

scale_color_gradient() for continuous

scale_color_gradient2() its reverse

scale_color_hue() for categorical

scale_fill_gradient()

scale_fill_gradient2()

E.g. Labels, breaks, and limits.

```
p <- ggplot(data = organdata,
            mapping = aes( x = roads,
                           y = donors,
                           color = world))
p + geom_point() +
    scale_x_log10() +
    scale_y_continuous(breaks = c(5, 15, 25),
                       labels = c("Five", "Fiftum", "Twenty Fivezz")) +
    scale_color_discrete(label = c("Corporatist", "Liberal", "Social Democratic", "other")) +

    labs( x = "Road Deaths",
          y = "Donor Procurement",
          color = "Welfare State")
```

```
## Warning: Removed 34 rows containing missing values (geom_point).
```

scale___()

Working with Models

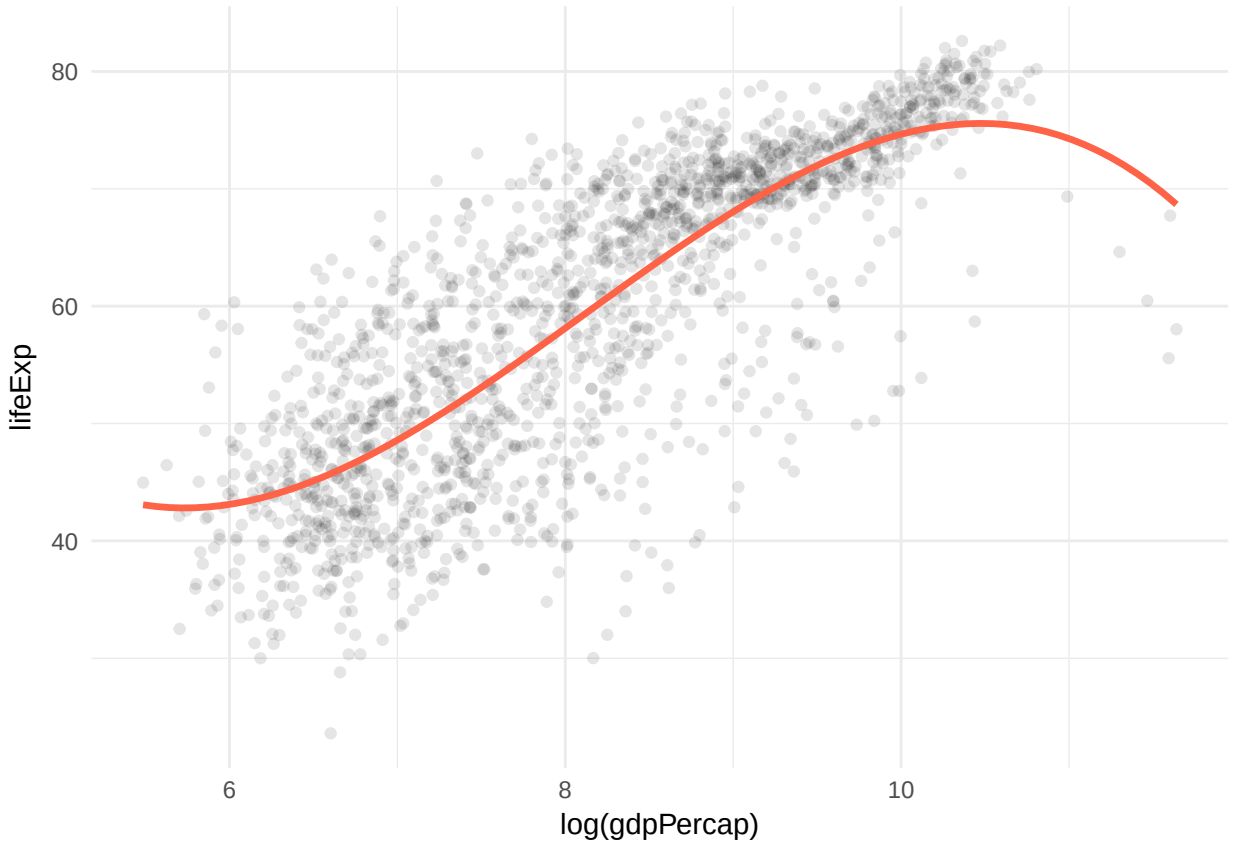
How to get data out of models. General principles. Into ggplot. broom, framework to get tidy data out of stat models.

Simple linear model by two different methods: robust regression and ordinary least squares.

```
p <- ggplot(data = gapminder,
            mapping = aes(x = log(gdpPercap), y = lifeExp))
p + geom_point(alpha=0.1) +
    geom_smooth(color = "tomato",
                fill = "tomato",
                method = MASS::rlm) +
    geom_smooth(color = "steelblue",
                fill = "steelblue",
                method = "lm")
```

Or splines

```
p + geom_point(alpha=0.1) +
    geom_smooth(color = "tomato",
                method = "lm", size = 1.2,
                formula = y ~splines::bs(x, 3),
                se = FALSE)
```



Look inside model objects

Objects have an internal structure in R. Think of them as various pieces. Sometimes strings, matrices, numbers, etc. Each one is named, and can have subpieces of their own. Think of it as a cabinet with mini cabinets inside its drawers.

```
str(gapminder)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    1704 obs. of  6 variables:
##  $ country  : Factor w/ 142 levels "Afghanistan",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ continent: Factor w/ 5 levels "Africa","Americas",..: 3 3 3 3 3 3 3 3 3 3 ...
##  $ year     : int  1952 1957 1962 1967 1972 1977 1982 1987 1992 1997 ...
##  $ lifeExp  : num  28.8 30.3 32 34 36.1 ...
##  $ pop      : int  8425333 9240934 10267083 11537966 13079460 14880372 12881816 13867957 16317921 22
##  $ gdpPercap: num  779 821 853 836 740 ...
```

First class. above: tabel dataframe. 1704 observations and 6 variables. levels are the count of different types of values.

Linear regression and summary of it:

```
out  <- lm(formula = lifeExp ~ gdpPercap + pop + continent,
           data = gapminder)
summary(out)
```

```
##
## Call:
## lm(formula = lifeExp ~ gdpPercap + pop + continent, data = gapminder)
```

```
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -49.161  -4.486   0.297   5.110  25.175
##
## Coefficients:
##                    Estimate Std. Error t value Pr(>|t|)
## (Intercept)       4.781e+01  3.395e-01 140.819  < 2e-16 ***
## gdpPercap         4.495e-04  2.346e-05  19.158  < 2e-16 ***
## pop               6.570e-09  1.975e-09   3.326 0.000901 ***
## continentAmericas 1.348e+01  6.000e-01  22.458  < 2e-16 ***
## continentAsia     8.193e+00  5.712e-01  14.342  < 2e-16 ***
## continentEurope   1.747e+01  6.246e-01  27.973  < 2e-16 ***
## continentOceania  1.808e+01  1.782e+00  10.146  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 8.365 on 1697 degrees of freedom
## Multiple R-squared:  0.5821, Adjusted R-squared:  0.5806
## F-statistic: 393.9 on 6 and 1697 DF,  p-value: < 2.2e-16
```

```
str(out)
```

```
## List of 13
##  $ coefficients : Named num [1:7] 4.78e+01 4.50e-04 6.57e-09 1.35e+01 8.19 ...
##   ..- attr(*, "names")= chr [1:7] "(Intercept)" "gdpPercap" "pop" "continentAmericas" ...
##  $ residuals    : Named num [1:1704] -27.6 -26.1 -24.5 -22.4 -20.3 ...
##   ..- attr(*, "names")= chr [1:1704] "1" "2" "3" "4" ...
##  $ effects      : Named num [1:1704] -2455.1 311.1 42.6 101.1 -17.2 ...
##   ..- attr(*, "names")= chr [1:1704] "(Intercept)" "gdpPercap" "pop" "continentAmericas" ...
##  $ rank         : int 7
##  $ fitted.values: Named num [1:1704] 56.4 56.4 56.5 56.5 56.4 ...
##   ..- attr(*, "names")= chr [1:1704] "1" "2" "3" "4" ...
##  $ assign       : int [1:7] 0 1 2 3 3 3 3
##  $ qr           :List of 5
##   ..$ qr   : num [1:1704, 1:7] -41.2795 0.0242 0.0242 0.0242 0.0242 ...
##   .. ..- attr(*, "dimnames")=List of 2
##   .. .. ..$ : chr [1:1704] "1" "2" "3" "4" ...
##   .. .. ..$ : chr [1:7] "(Intercept)" "gdpPercap" "pop" "continentAmericas" ...
##   .. ..- attr(*, "assign")= int [1:7] 0 1 2 3 3 3 3
##   .. ..- attr(*, "contrasts")=List of 1
##   .. .. ..$ continent: chr "contr.treatment"
##   ..$ qraux: num [1:7] 1.02 1.02 1 1.01 1.04 ...
##   ..$ pivot: int [1:7] 1 2 3 4 5 6 7
##   ..$ tol  : num 1e-07
##   ..$ rank : int 7
##   ..- attr(*, "class")= chr "qr"
##  $ df.residual  : int 1697
##  $ contrasts    :List of 1
##   ..$ continent: chr "contr.treatment"
##  $ xlevels      :List of 1
##   ..$ continent: chr [1:5] "Africa" "Americas" "Asia" "Europe" ...
##  $ call         : language lm(formula = lifeExp ~ gdpPercap + pop + continent, data = gapminder)
##  $ terms        :Classes 'terms', 'formula'  language lifeExp ~ gdpPercap + pop + continent
##   .. ..- attr(*, "variables")= language list(lifeExp, gdpPercap, pop, continent)
```

```
##   .. ..- attr(*, "factors")= int [1:4, 1:3] 0 1 0 0 0 0 1 0 0 0 ...
##   .. .. ..- attr(*, "dimnames")=List of 2
##   .. .. .. ..$ : chr [1:4] "lifeExp" "gdpPercap" "pop" "continent"
##   .. .. .. ..$ : chr [1:3] "gdpPercap" "pop" "continent"
##   .. ..- attr(*, "term.labels")= chr [1:3] "gdpPercap" "pop" "continent"
##   .. ..- attr(*, "order")= int [1:3] 1 1 1
##   .. ..- attr(*, "intercept")= int 1
##   .. ..- attr(*, "response")= int 1
##   .. ..- attr(*, ".Environment")=<environment: R_GlobalEnv>
##   .. ..- attr(*, "predvars")= language list(lifeExp, gdpPercap, pop, continent)
##   .. ..- attr(*, "dataClasses")= Named chr [1:4] "numeric" "numeric" "numeric" "factor"
##   .. .. ..- attr(*, "names")= chr [1:4] "lifeExp" "gdpPercap" "pop" "continent"
##  $ model        :'data.frame':   1704 obs. of  4 variables:
##   ..$ lifeExp  : num [1:1704] 28.8 30.3 32 34 36.1 ...
##   ..$ gdpPercap: num [1:1704] 779 821 853 836 740 ...
##   ..$ pop      : int [1:1704] 8425333 9240934 10267083 11537966 13079460 14880372 12881816 13867957 1
##   ..$ continent: Factor w/ 5 levels "Africa","Americas",..: 3 3 3 3 3 3 3 3 3 3 ...
##   ..- attr(*, "terms")=Classes 'terms', 'formula'  language lifeExp ~ gdpPercap + pop + continent
##   .. .. ..- attr(*, "variables")= language list(lifeExp, gdpPercap, pop, continent)
##   .. .. ..- attr(*, "factors")= int [1:4, 1:3] 0 1 0 0 0 0 1 0 0 0 ...
##   .. .. .. ..- attr(*, "dimnames")=List of 2
##   .. .. .. .. ..$ : chr [1:4] "lifeExp" "gdpPercap" "pop" "continent"
##   .. .. .. .. ..$ : chr [1:3] "gdpPercap" "pop" "continent"
##   .. .. ..- attr(*, "term.labels")= chr [1:3] "gdpPercap" "pop" "continent"
##   .. .. ..- attr(*, "order")= int [1:3] 1 1 1
##   .. .. ..- attr(*, "intercept")= int 1
##   .. .. ..- attr(*, "response")= int 1
##   .. .. ..- attr(*, ".Environment")=<environment: R_GlobalEnv>
##   .. .. ..- attr(*, "predvars")= language list(lifeExp, gdpPercap, pop, continent)
##   .. .. ..- attr(*, "dataClasses")= Named chr [1:4] "numeric" "numeric" "numeric" "factor"
##   .. .. .. ..- attr(*, "names")= chr [1:4] "lifeExp" "gdpPercap" "pop" "continent"
##  - attr(*, "class")= chr "lm"
```

We want to get the stuff out in a tidy format.

Useful plots present findings in substantive meaning. For example converting to predictive possibilites in stead of coefficients. Sensible values that are substantial and theoretically interesting.

Show your degree of confidence or uncertainty.

Show the data when you can. Like the relationship of predicted values and confirmed ones. easy due to layered nature of ggplot.

Predictions from a model

Everything prediction related works this way. R provides a family of interfaces to range of different models. The predict function is the workhorse for generating new data. We take our model, generate new predictive values, and plot them, showing also the existing data.

We generate a data frame that is a sort of clone of the existing one. We're interested in looking at the relationship between gdpPercapita and population

```
min_gdp <- min(gapminder$gdpPercap)
max_gdp <- max(gapminder$gdpPercap)

med_pop <- median(gapminder$pop)
```

```
pred_df <- expand.grid(gdpPercap = (seq(from = min_gdp,
                                        to = max_gdp,
                                        length.out = 100)),
                       pop = med_pop,
                       continent = c("Africa", "Americas", "Asia", "Europe", "Oceania"))

dim(pred_df)
```

```
## [1] 500   3
```

```
head(pred_df)
```

```
##   gdpPercap     pop continent
## 1  241.1659 7023596    Africa
## 2 1385.4282 7023596    Africa
## 3 2529.6905 7023596    Africa
## 4 3673.9528 7023596    Africa
## 5 4818.2150 7023596    Africa
## 6 5962.4773 7023596    Africa
```

Notice that you are getting a table of data back -a data frame. Underneath we are still creating and operating on tables of tidy data.

now lets predict

```
pred_out <- predict(object = out,
                    newdata = pred_df,
                    interval = "predict")

head(pred_out)
```

```
##        fit      lwr      upr
## 1 47.96863 31.54775 64.38951
## 2 48.48298 32.06231 64.90365
## 3 48.99733 32.57670 65.41797
## 4 49.51169 33.09092 65.93245
## 5 50.02604 33.60497 66.44711
## 6 50.54039 34.11885 66.96193
```

For "natural" data frames you should merge() by a common key, and not simply cbind(). here we do a cbind though

```
pred_df <- cbind(pred_df, pred_out)

head(pred_df)
```

```
##   gdpPercap     pop continent      fit      lwr      upr
## 1  241.1659 7023596    Africa 47.96863 31.54775 64.38951
## 2 1385.4282 7023596    Africa 48.48298 32.06231 64.90365
## 3 2529.6905 7023596    Africa 48.99733 32.57670 65.41797
## 4 3673.9528 7023596    Africa 49.51169 33.09092 65.93245
## 5 4818.2150 7023596    Africa 50.02604 33.60497 66.44711
## 6 5962.4773 7023596    Africa 50.54039 34.11885 66.96193
```

Back to square one, a tidy dataframe to plot.

```
p <- ggplot(data = subset(pred_df, continent %in% c("Europe", "Africa")),
            mapping = aes(x = gdpPercap,
```

```
                           y = fit,
                           ymin = lwr,
                           ymax = upr,
                           color = continent,
                           fill = continent,
                           group = continent))
p + geom_point(data = subset(gapminder,
                             continent %in% c("Europe", "Africa")),
                       mapping = aes(x = gdpPercap,
                                     y = lifeExp,
                                     color = continent),
                                     alpha = 0.5,
                                     inherit.aes = FALSE) +
    geom_smooth() +
    geom_ribbon(alpha = 0.2, color = FALSE) +
    scale_x_log10(labels = scales::dollar)
```

```
## `geom_smooth()` using method = 'loess'
```

Something like this is going on behind the scenes for what we're going to do from now on. But we would prefer not to do it ourselves.

Using broom to tidy up models

Load it

```
library("broom")
```

broom's functions turn the output of models into tidy data you can use in plots.

summary(out) is not useful for ggplot. broom thinks of things as three levels of useful things: component level, observation level, and model level information. Each of broom's three functions corresponds to these. So the first is:

tidy() - component level

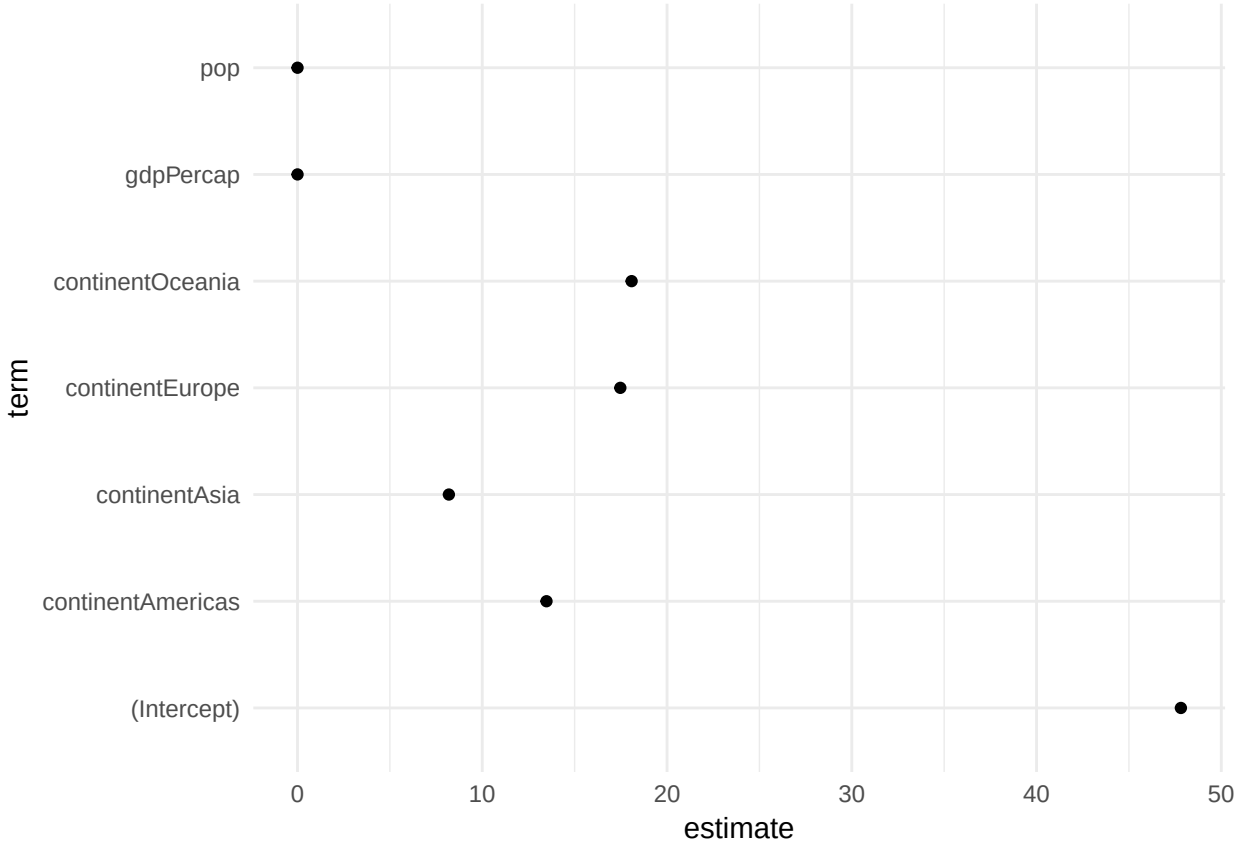
augment() - observation level

glance() - model level

```
out_comp <- tidy(out)
out_comp %>% round_df()
```

```
##                term estimate std.error statistic p.value
## 1       (Intercept)    47.81      0.34    140.82       0
## 2         gdpPercap     0.00      0.00     19.16       0
## 3               pop     0.00      0.00      3.33       0
## 4 continentAmericas    13.48      0.60     22.46       0
## 5     continentAsia     8.19      0.57     14.34       0
## 6   continentEurope    17.47      0.62     27.97       0
## 7  continentOceania    18.08      1.78     10.15       0
```

```
p <- ggplot(out_comp, aes(x = term, y = estimate))
p + geom_point() + coord_flip()
```



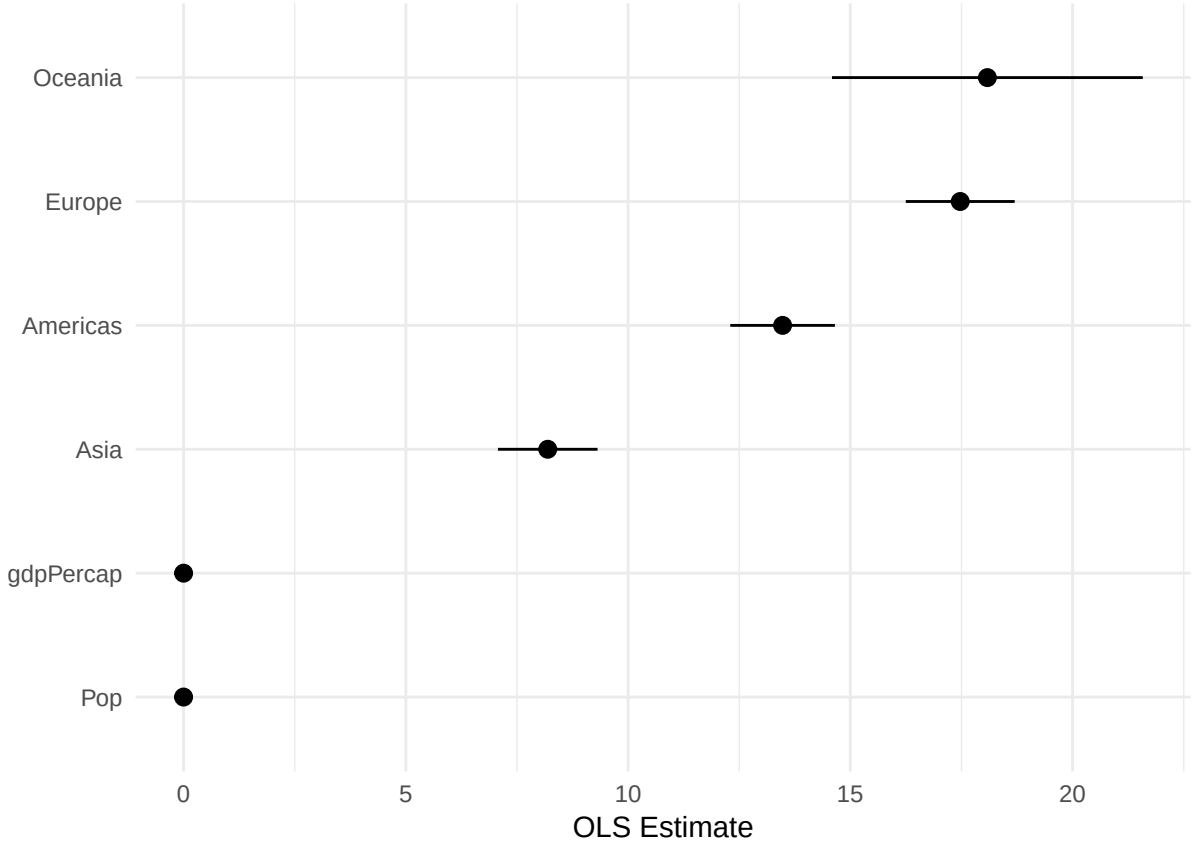
Tidy takes some arguments, like confidence intervals.

```
out_conf <- tidy(out, conf.int = TRUE)
out_conf %>% round_df()
```

```
##                term estimate std.error statistic p.value conf.low
## 1       (Intercept)    47.81      0.34    140.82       0    47.15
## 2         gdpPercap     0.00      0.00     19.16       0     0.00
## 3               pop     0.00      0.00      3.33       0     0.00
## 4 continentAmericas    13.48      0.60     22.46       0    12.30
## 5     continentAsia     8.19      0.57     14.34       0     7.07
## 6   continentEurope    17.47      0.62     27.97       0    16.25
## 7  continentOceania    18.08      1.78     10.15       0    14.59
##   conf.high
## 1     48.48
## 2      0.00
## 3      0.00
## 4     14.65
## 5      9.31
## 6     18.70
## 7     21.58
```

With those in place, we can plot it with geom_pointrange. But at this point we would want to clean up our labels etc. And drop the intercept, since its so much larger.

```
out_conf <- subset(out_conf, term %nin% "(Intercept)") # %nin% = not in. Opposite of %in%. Drop everyth
out_conf$nicelabs <- prefix_strip(out_conf$term, "continent")

p <- ggplot(out_conf, mapping = aes(x = reorder(nicelabs, estimate),
                                    y = estimate,
                                    ymin = conf.low,
                                    ymax = conf.high))
p + geom_pointrange() + coord_flip() +
  labs(x="", y="OLS Estimate")
```



Down to observation level

This is handled in broom by augment(). Ads

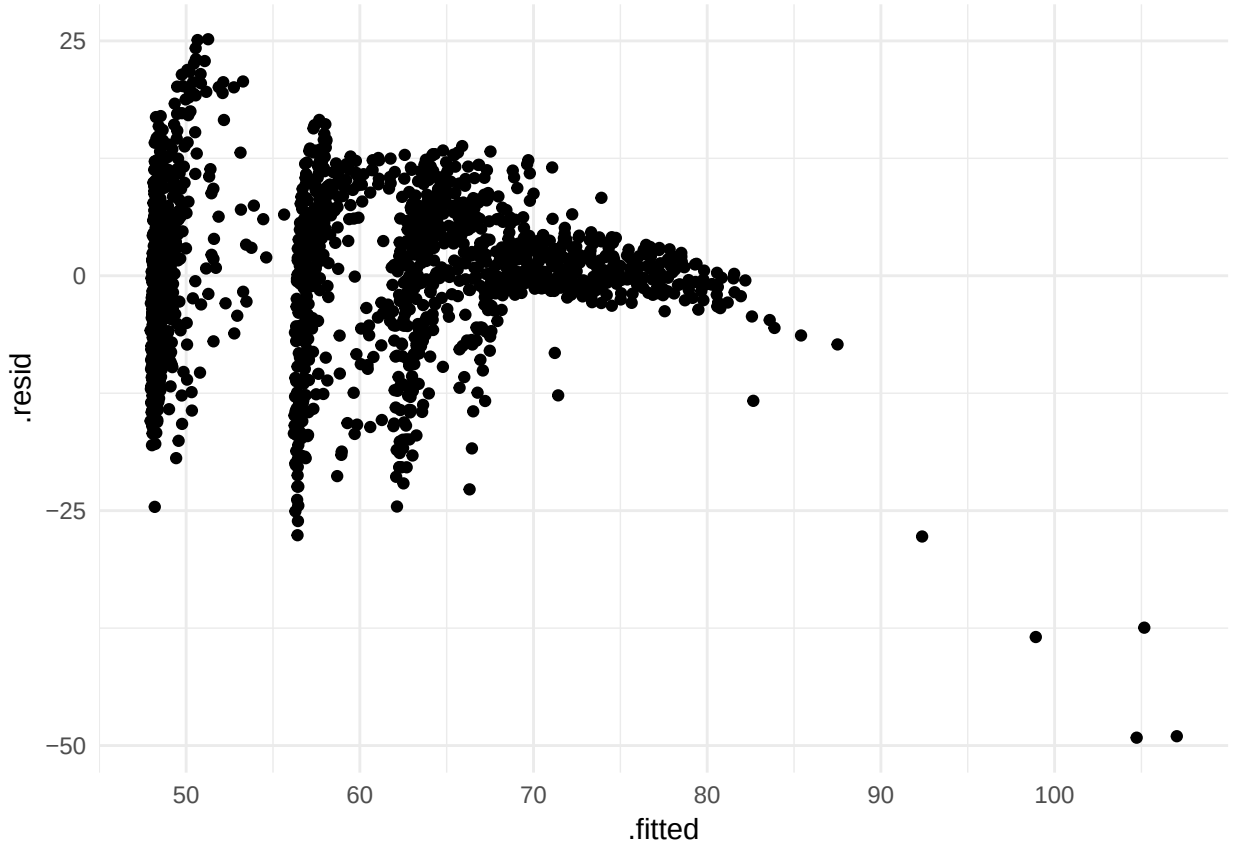
- .fitted . fitted values of the model
- .se.fit - The standard errors
- .resid,
- .hat
- .sigma
- .cooksd
- .std.resid

```
out_aug <- augment(out)
head(out_aug) %>% round_df()
```

```
##   lifeExp gdpPercap      pop continent .fitted .se.fit .resid .hat .sigma
## 1   28.80    779.45  8425333      Asia   56.41    0.47 -27.61    0   8.34
## 2   30.33    820.85  9240934      Asia   56.44    0.47 -26.10    0   8.34
## 3   32.00    853.10 10267083      Asia   56.46    0.47 -24.46    0   8.35
## 4   34.02    836.20 11537966      Asia   56.46    0.47 -22.44    0   8.35
## 5   36.09    739.98 13079460      Asia   56.43    0.47 -20.34    0   8.35
## 6   38.44    786.11 14880372      Asia   56.46    0.47 -18.02    0   8.36
##   .cooksd .std.resid
## 1    0.01      -3.31
## 2    0.00      -3.13
## 3    0.00      -2.93
## 4    0.00      -2.69
```

```
## 5    0.00      -2.44
## 6    0.00      -2.16
```

```
p <- ggplot(data = out_aug,
            mapping = aes(x = .fitted, y = .resid))
p + geom_point()
```



Glance is the function to see the model level:

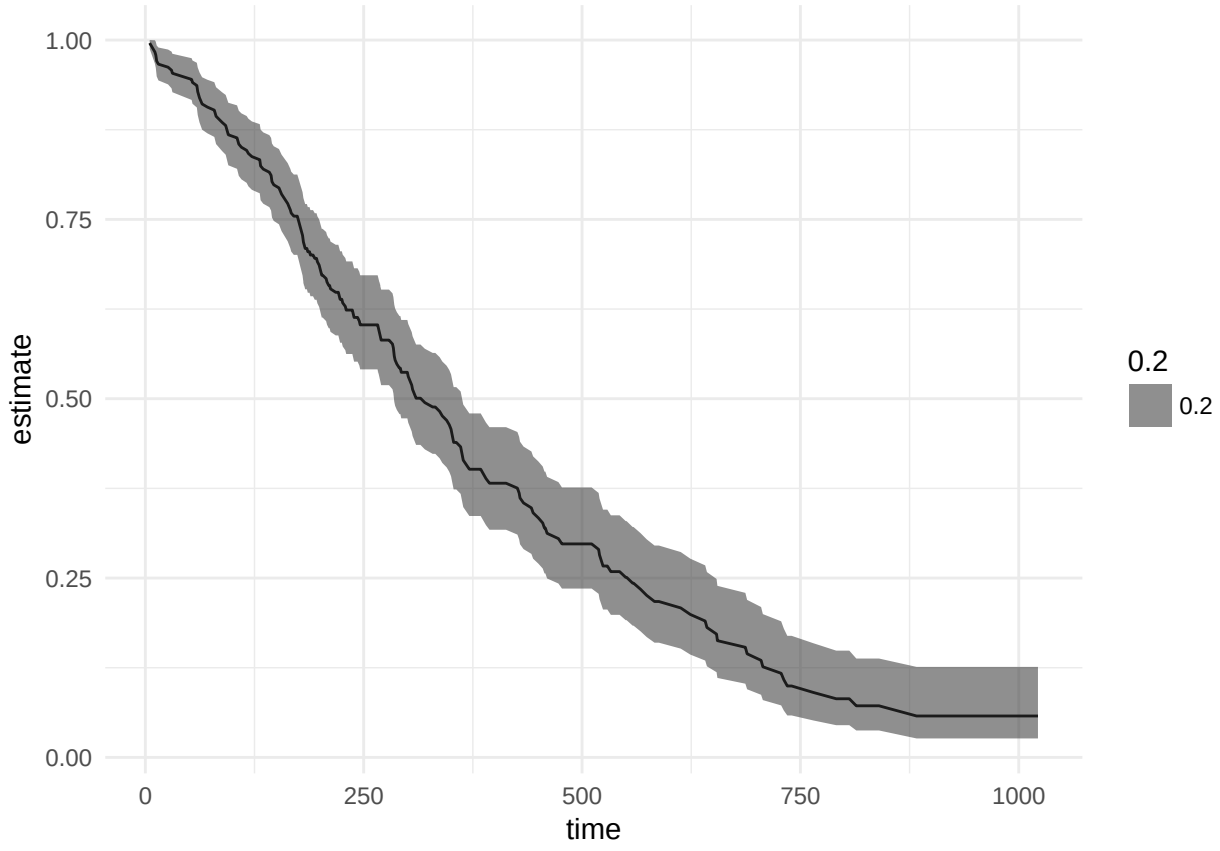
```
glance(out) %>% round_df()
```

```
##   r.squared adj.r.squared sigma statistic p.value df   logLik      AIC
## 1      0.58          0.58  8.37    393.91       0  7 -6033.83 12083.65
##        BIC deviance df.residual
## 1 12127.18 118754.5        1697
```

Broom's strength is twofold. These functions are available for an increasing number of libraries.

```
library(survival)

out_cph  <- coxph(Surv(time, status) ~ age + sex, data = lung)
out_surv <- survfit(out_cph)

out_tidy <- tidy(out_surv)

p <- ggplot(data = out_tidy, mapping = aes(time, estimate))

p + geom_line() +
    geom_ribbon(mapping = aes(ymin = conf.low,
```

```
                            ymax = conf.high,
                            alpha = .2))
```



lets make another subset by filter

```
eu77 <- gapminder %>% filter(continent == "Europe", year == 1977)
eu77
```

```
## # A tibble: 30 x 6
##                    country continent  year lifeExp      pop gdpPercap
##                     <fctr>    <fctr> <int>   <dbl>    <int>     <dbl>
##  1                Albania    Europe  1977   68.93  2509048  3533.004
##  2                Austria    Europe  1977   72.17  7568430 19749.422
##  3                Belgium    Europe  1977   72.80  9821800 19117.974
##  4 Bosnia and Herzegovina    Europe  1977   69.86  4086000  3528.481
##  5               Bulgaria    Europe  1977   70.81  8797022  7612.240
##  6                Croatia    Europe  1977   70.64  4318673 11305.385
##  7         Czech Republic    Europe  1977   70.71 10161915 14800.161
##  8                Denmark    Europe  1977   74.69  5088419 20422.901
##  9                Finland    Europe  1977   72.52  4738902 15605.423
## 10                 France    Europe  1977   73.83 53165019 18292.635
## # ... with 20 more rows
```

Lets fit a model to this subset

```
fit <- lm(lifeExp ~ log(gdpPercap), data = eu77)
summary(fit)
```

```
##
```

```
## Call:
## lm(formula = lifeExp ~ log(gdpPercap), data = eu77)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -7.4956 -1.0306  0.0935  1.1755  3.7125
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)      29.489      7.161   4.118 0.000306 ***
## log(gdpPercap)    4.488      0.756   5.936 2.17e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.114 on 28 degrees of freedom
## Multiple R-squared:  0.5572, Adjusted R-squared:  0.5414
## F-statistic: 35.24 on 1 and 28 DF,  p-value: 2.173e-06
```

How do we do this for all years and continents? Broom is efficient. Lets pipe it and use a do loop.

```
out_le <- gapminder %>% group_by(continent, year) %>%
  do(fit = lm(lifeExp ~gdpPercap, data = .)) # The dot gets the gapminder in lm, because lm is stupid (

out_le # the fit is a list object, full of complex lm objects.
```

```
## Source: local data frame [60 x 3]
## Groups: <by row>
##
## # A tibble: 60 x 3
##    continent  year      fit
##  *    <fctr> <int>   <list>
##  1    Africa  1952 <S3: lm>
##  2    Africa  1957 <S3: lm>
##  3    Africa  1962 <S3: lm>
##  4    Africa  1967 <S3: lm>
##  5    Africa  1972 <S3: lm>
##  6    Africa  1977 <S3: lm>
##  7    Africa  1982 <S3: lm>
##  8    Africa  1987 <S3: lm>
##  9    Africa  1992 <S3: lm>
## 10    Africa  1997 <S3: lm>
## # ... with 50 more rows
```

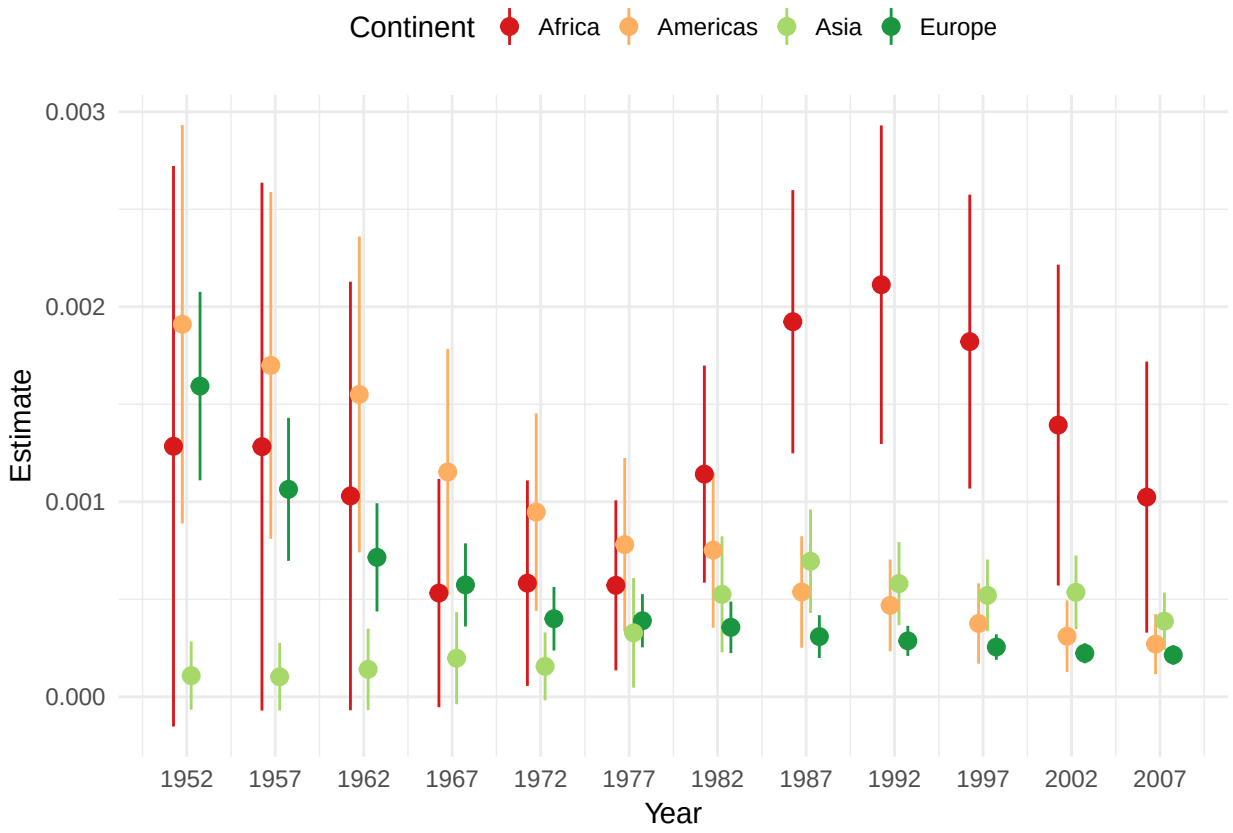
We can also filter out europe

```
out_le %>% filter(continent == "Europe", year == 1977)
```

```
## Source: local data frame [1 x 3]
## Groups: <by row>
##
## # A tibble: 1 x 3
##   continent  year      fit
##      <fctr> <int>   <list>
## 1    Europe  1977 <S3: lm>
```

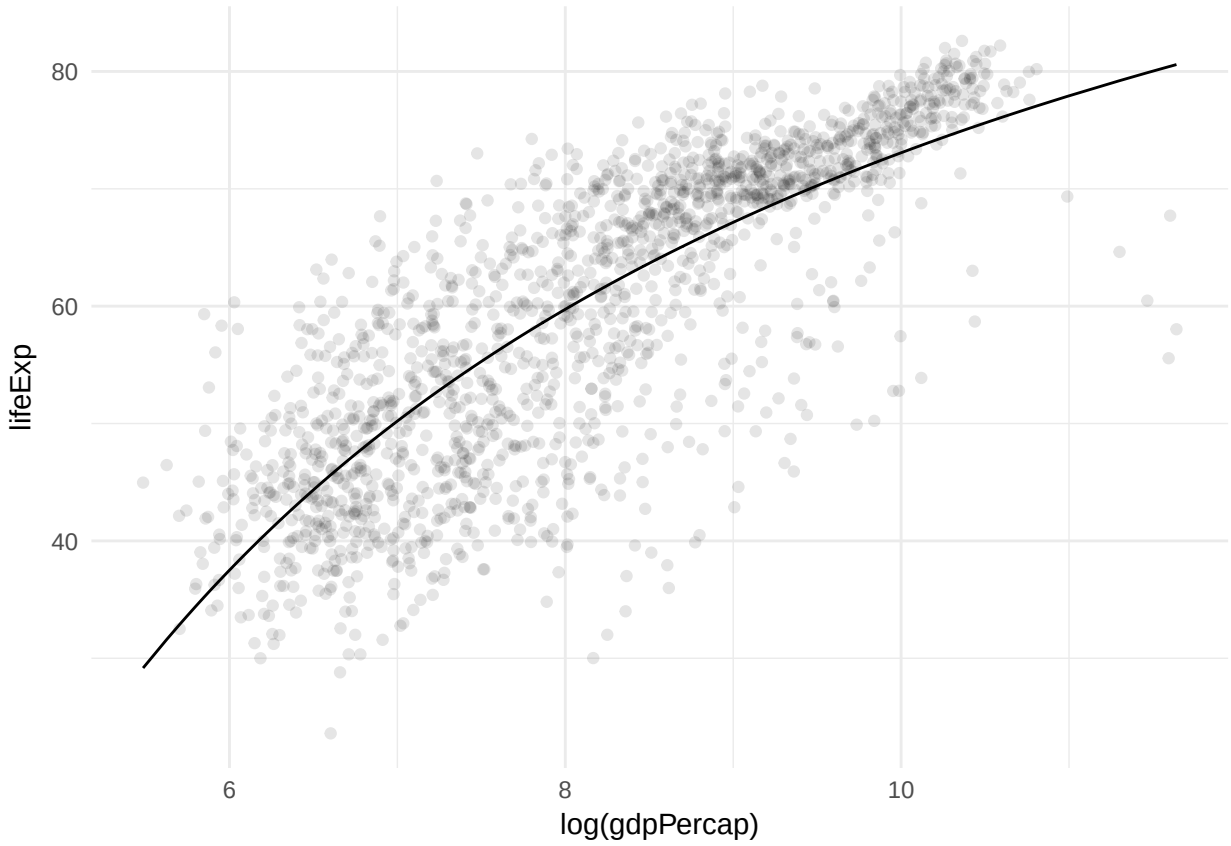
now lets plot it.

```
out_tidy <- out_le %>% tidy(fit) %>%
  filter(term %nin% "(Intercept)" &
           continent %nin% "Oceania")

p <- ggplot(data = out_tidy,
            mapping = aes( x = year,
                           y = estimate,
                           ymin = estimate - 2*std.error,
                           ymax = estimate + 2*std.error,
                           group = continent,
                           color = continent))
p + geom_pointrange(position = position_dodge(width = 2)) + # very useful since you can plot multiple e
  scale_x_continuous(breaks = unique(gapminder$year)) +
  scale_colour_brewer(type = "div", palette = 8) +
  theme(legend.position = "top") +
  labs(x = "Year", y = "Estimate", color = "Continent")
```



Lets plot a nonlinear (weighted) least-squares estimates of the parameters of a nonlinear model.

```
out_nls <- nls(lifeExp ~ k / log(gdpPercap) + b, # nls requires k and b specified
               data = gapminder,
               start = list(k = 1,
                            b = 0))
p <- ggplot(data = gapminder,
            aes(x = log(gdpPercap), y = lifeExp))
p + geom_point(alpha=0.1) +
    geom_line(aes(y = predict(out_nls)))
```



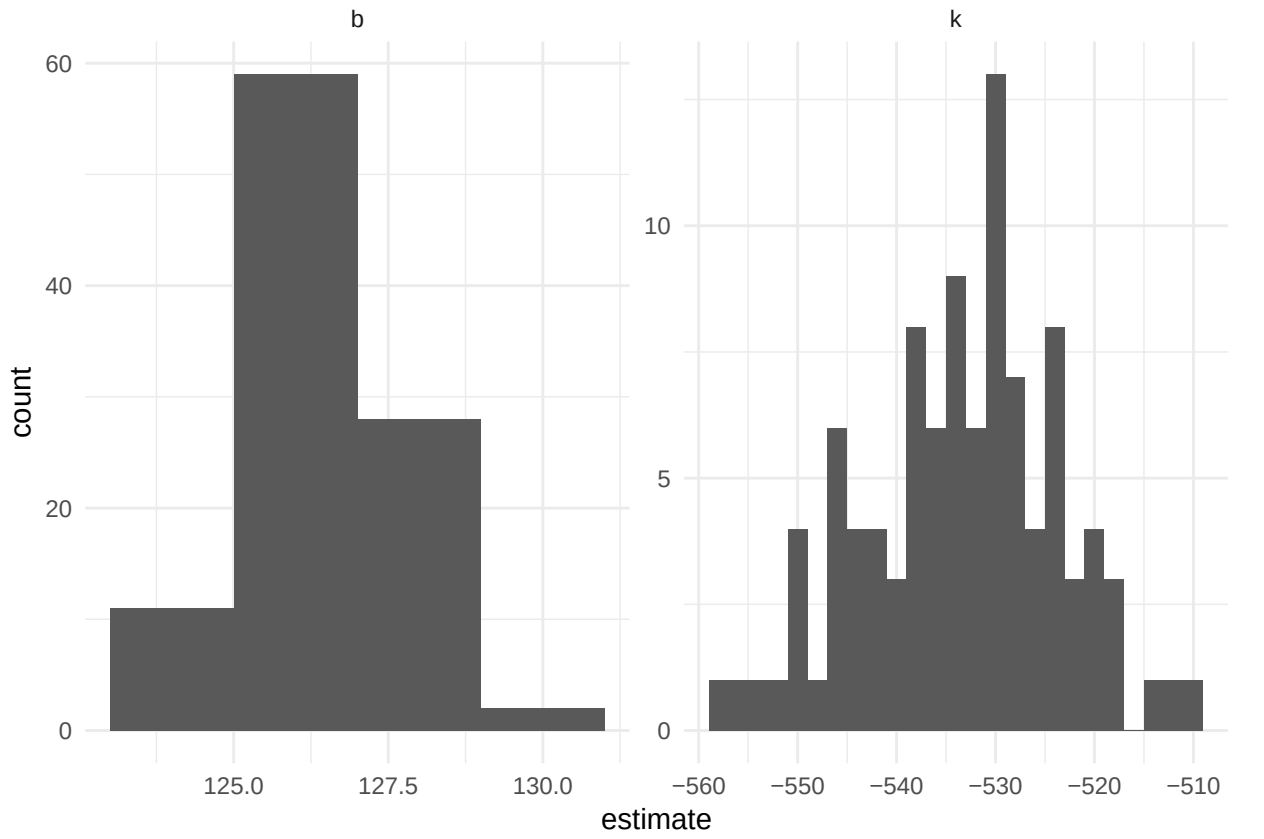
tidy() output from one hundred bootstrap replicates of this model

```
set.seed(112016)
out_bootnls <- gapminder %>% bootstrap(100) %>%
  do(tidy(nls(lifeExp ~ k / log(gdpPercap) + b, ., # exactly the same as the nls above.
              start=list(k=1, b=0))))

out_bootnls
```

```
## Source: local data frame [200 x 6]
## Groups: replicate [100]
##
## # A tibble: 200 x 6
##    replicate  term  estimate std.error statistic p.value
##        <int> <chr>     <dbl>     <dbl>     <dbl>   <dbl>
##  1         1     k -529.1990  9.986083 -52.99365       0
##  2         1     b  125.8627  1.260548  99.84762       0
##  3         2     k -526.5463  9.834573 -53.54034       0
##  4         2     b  125.3352  1.248739 100.36935       0
##  5         3     k -521.8463  9.711754 -53.73348       0
##  6         3     b  125.0659  1.236656 101.13232       0
##  7         4     k -528.8287  9.839724 -53.74427       0
##  8         4     b  125.9152  1.237296 101.76648       0
##  9         5     k -517.4593  9.683355 -53.43802       0
## 10         5     b  124.5752  1.231880 101.12606       0
## # ... with 190 more rows
```

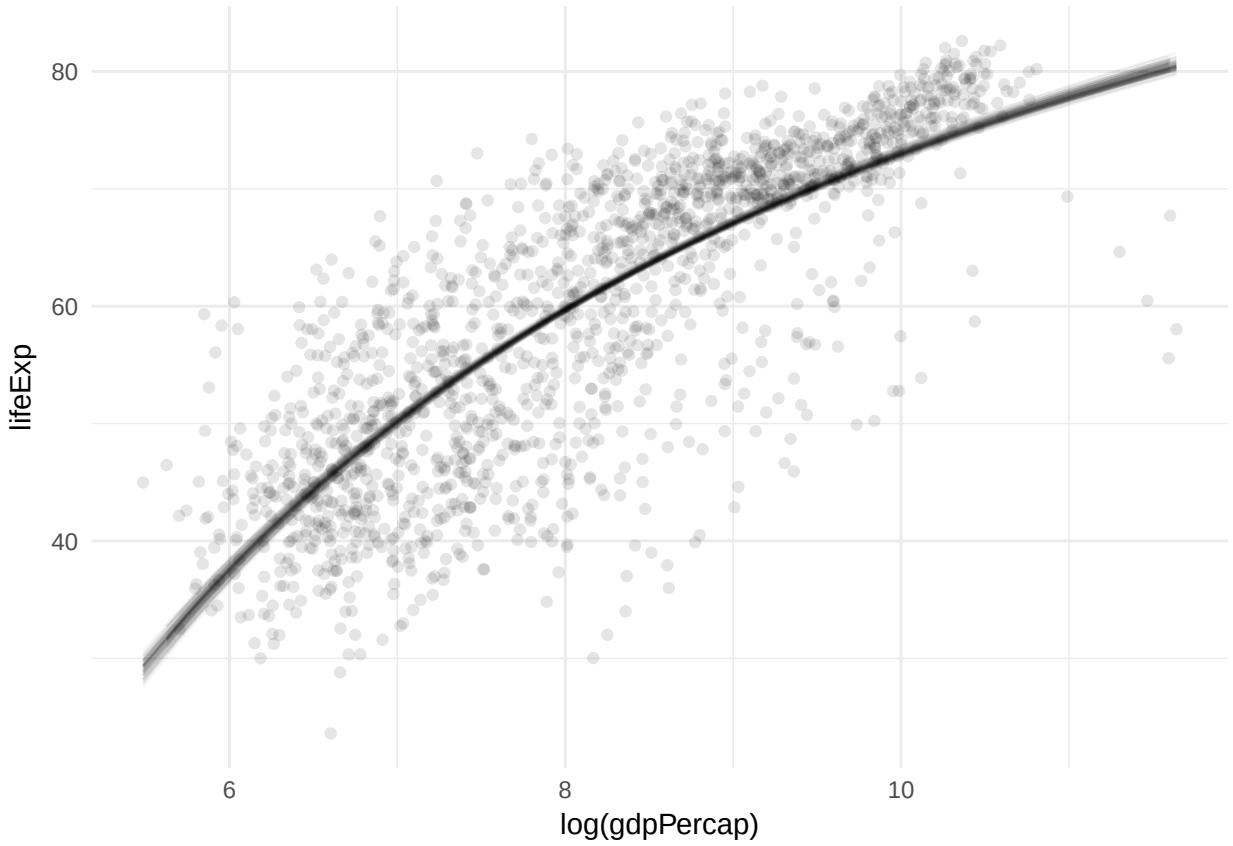
graph it

```
p <- ggplot(out_bootnls, aes(estimate))
p + geom_histogram(binwidth = 2) +
  facet_wrap( ~ term, scales="free")
```



So we do this for the observation level as well:

```
set.seed(112016)
out_bootnls <- gapminder %>% bootstrap(100) %>%
  do(augment(nls(lifeExp ~ k / log(gdpPercap) + b, ., # exactly the same as the nls above.
             start=list(k=1, b=0))))
p <- ggplot(data = gapminder,
            mapping = aes(x = log(gdpPercap), y = lifeExp))
p + geom_point(alpha = 0.1) +
  geom_line(data = out_bootnls,
            mapping = aes(y=.fitted,
                          group=replicate), alpha=0.04)
```



Lets group by continent first

Marginal effect and how to plot them

margins package. Newly developed for R, pride and joy of STATA. Gold standard in recent polsci, on the background of misinterpretations of logistic regression models. Standard way of estimating average causal effects in models.

In development. Read documentation if you use it.

```
library(margins)

gss_sm$polviews_m <- relevel(gss_sm$polviews,
                             ref = "Moderate")
out_bo <- glm(obama ~ polviews_m + sex*race,
              family = "binomial", data = gss_sm)
summary(out_bo)
```

```
##
## Call:
## glm(formula = obama ~ polviews_m + sex * race, family = "binomial",
##     data = gss_sm)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.9045  -0.5541   0.1772   0.5418   2.2437
##
```

```
## Coefficients:
##                                  Estimate Std. Error z value Pr(>|z|)
## (Intercept)                      0.296493   0.134091   2.211  0.02703 *
## polviews_mExtremely Liberal      2.372950   0.525045   4.520 6.20e-06 ***
## polviews_mLiberal                2.600031   0.356666   7.290 3.10e-13 ***
## polviews_mSlightly Liberal       1.293172   0.248435   5.205 1.94e-07 ***
## polviews_mSlightly Conservative -1.355277   0.181291  -7.476 7.68e-14 ***
## polviews_mConservative          -2.347463   0.200384 -11.715  < 2e-16 ***
## polviews_mExtremely Conservative -2.727384   0.387210  -7.044 1.87e-12 ***
## sexFemale                        0.254866   0.145370   1.753  0.07956 .
## raceBlack                        3.849526   0.501319   7.679 1.61e-14 ***
## raceOther                       -0.002143   0.435763  -0.005  0.99608
## sexFemale:raceBlack             -0.197506   0.660066  -0.299  0.76477
## sexFemale:raceOther              1.574829   0.587657   2.680  0.00737 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 2247.9  on 1697  degrees of freedom
## Residual deviance: 1345.9  on 1686  degrees of freedom
##   (1169 observations deleted due to missingness)
## AIC: 1369.9
##
## Number of Fisher Scoring iterations: 6
```

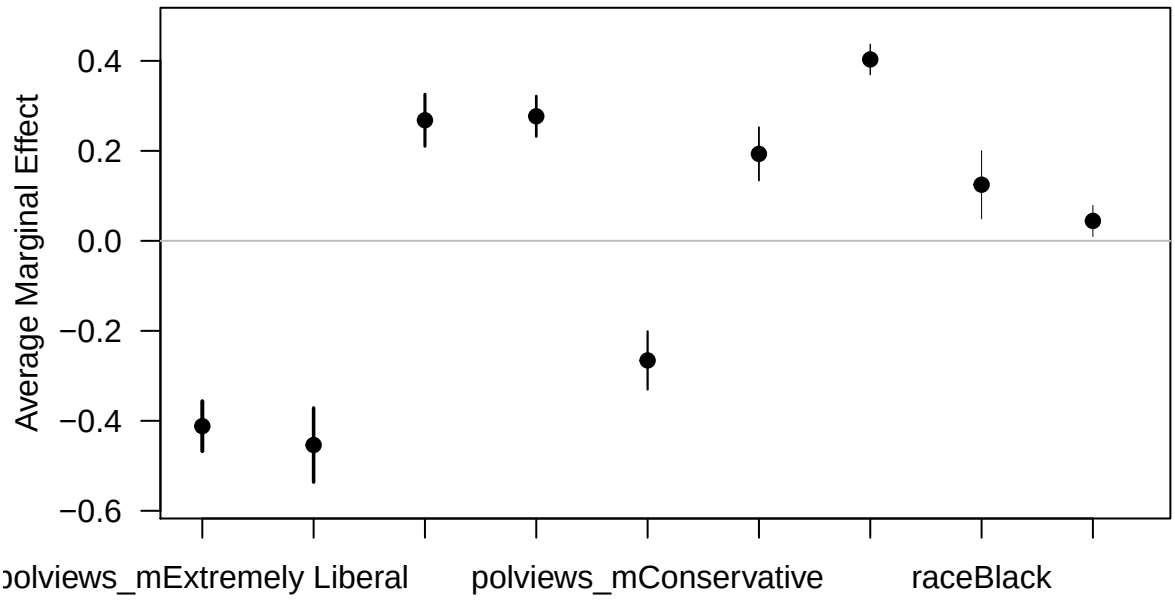
```
bo_m <- margins(out_bo)
```

```
## Warning in warn_for_weights(model): 'weights' used in model estimation are
## currently ignored!
```

```
summary(bo_m)
```

```
##                            factor     AME     SE        z      p   lower
##            polviews_mConservative -0.4119 0.0283 -14.5394 0.0000 -0.4674
##  polviews_mExtremely Conservative -0.4538 0.0420 -10.7971 0.0000 -0.5361
##       polviews_mExtremely Liberal  0.2681 0.0295   9.0996 0.0000  0.2103
##                  polviews_mLiberal  0.2768 0.0229  12.0736 0.0000  0.2319
##   polviews_mSlightly Conservative -0.2658 0.0330  -8.0596 0.0000 -0.3304
##        polviews_mSlightly Liberal  0.1933 0.0303   6.3896 0.0000  0.1340
##                          raceBlack  0.4032 0.0173  23.3568 0.0000  0.3694
##                          raceOther  0.1247 0.0386   3.2297 0.0012  0.0490
##                          sexFemale  0.0443 0.0177   2.5073 0.0122  0.0097
##    upper
##  -0.3564
##  -0.3714
##   0.3258
##   0.3218
##  -0.2011
##   0.2526
##   0.4371
##   0.2005
##   0.0789
```

```
plot(bo_m) #plots using margin library
```



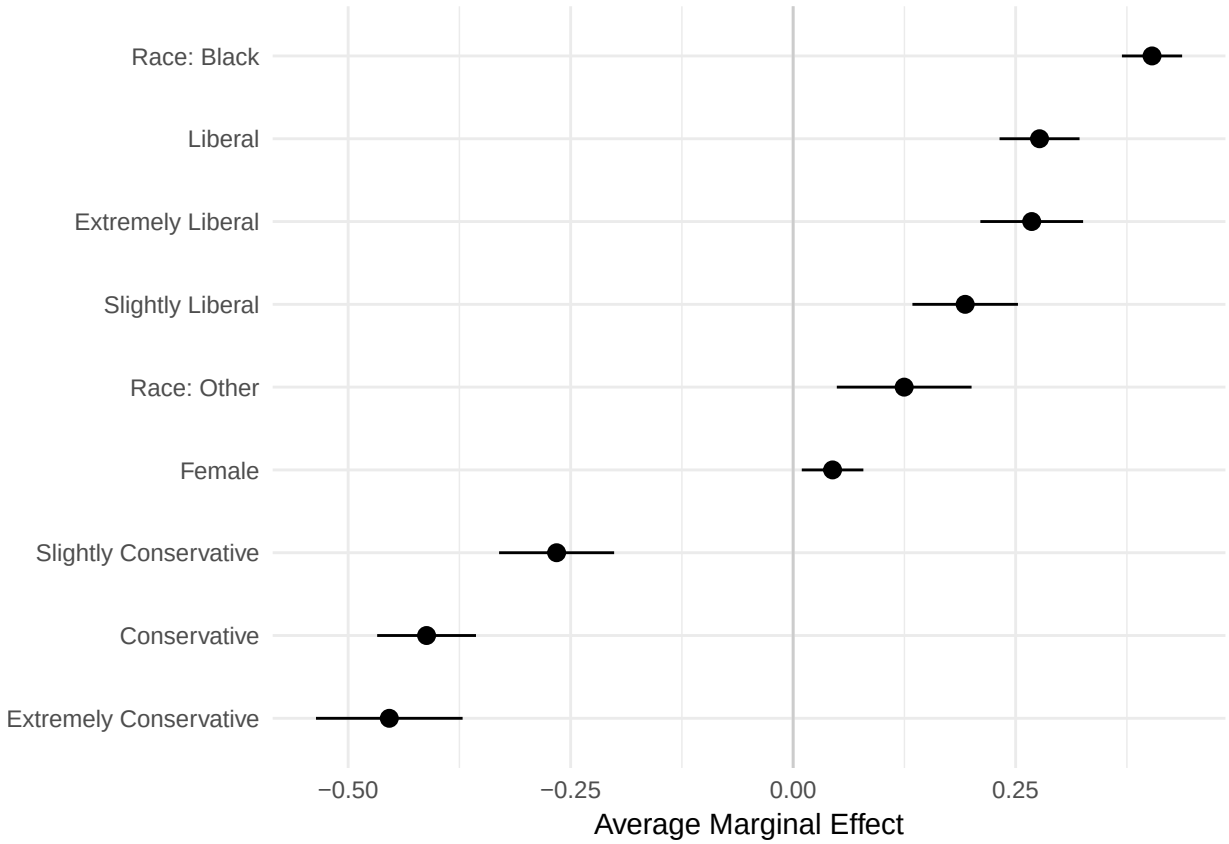
But better in ggplot:

```
bo_gg <- data.frame(summary(bo_m))
prefixes <- c("polviews_m", "sex")
bo_gg$factor <- prefix_strip(bo_gg$factor, prefixes)
bo_gg$factor <- prefix_replace(bo_gg$factor, "race", "Race: ")

bo_gg %>% select(factor, AME, lower, upper)
```

```
##                   factor         AME        lower       upper
## 1           Conservative -0.41188222 -0.467405438 -0.35635900
## 2 Extremely Conservative -0.45376935 -0.536140431 -0.37139827
## 3      Extremely Liberal  0.26805480  0.210318396  0.32579120
## 4                Liberal  0.27683785  0.231897478  0.32177822
## 5  Slightly Conservative -0.26578277 -0.330416626 -0.20114891
## 6       Slightly Liberal  0.19328735  0.133997967  0.25257674
## 7            Race: Black  0.40322793  0.369391373  0.43706449
## 8            Race: Other  0.12474996  0.049044708  0.20045521
## 9                 Female  0.04428648  0.009666921  0.07890603
```

Plot it

```
p <- ggplot(data = bo_gg, aes(x = reorder(factor, AME),
                              y = AME, ymin = lower, ymax = upper))
p + geom_hline(yintercept = 0, color = "gray80") +
    geom_pointrange() + coord_flip() +
    labs(x = NULL, y = "Average Marginal Effect")
```

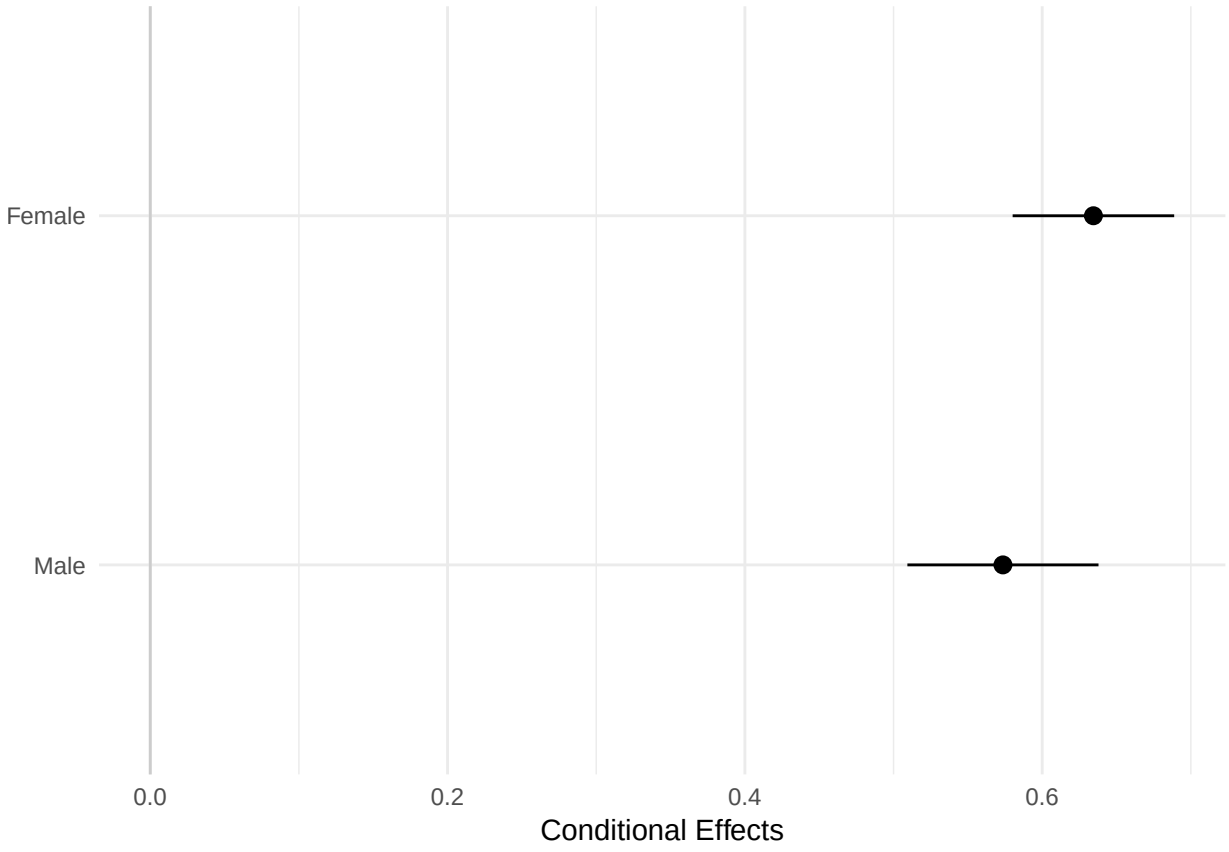


Margin can also show the conditional effects, using cplot, just tell it not to draw a plot

```
pv_cp <- cplot(out_bo, x = "sex", draw = FALSE)
pv_cp
```

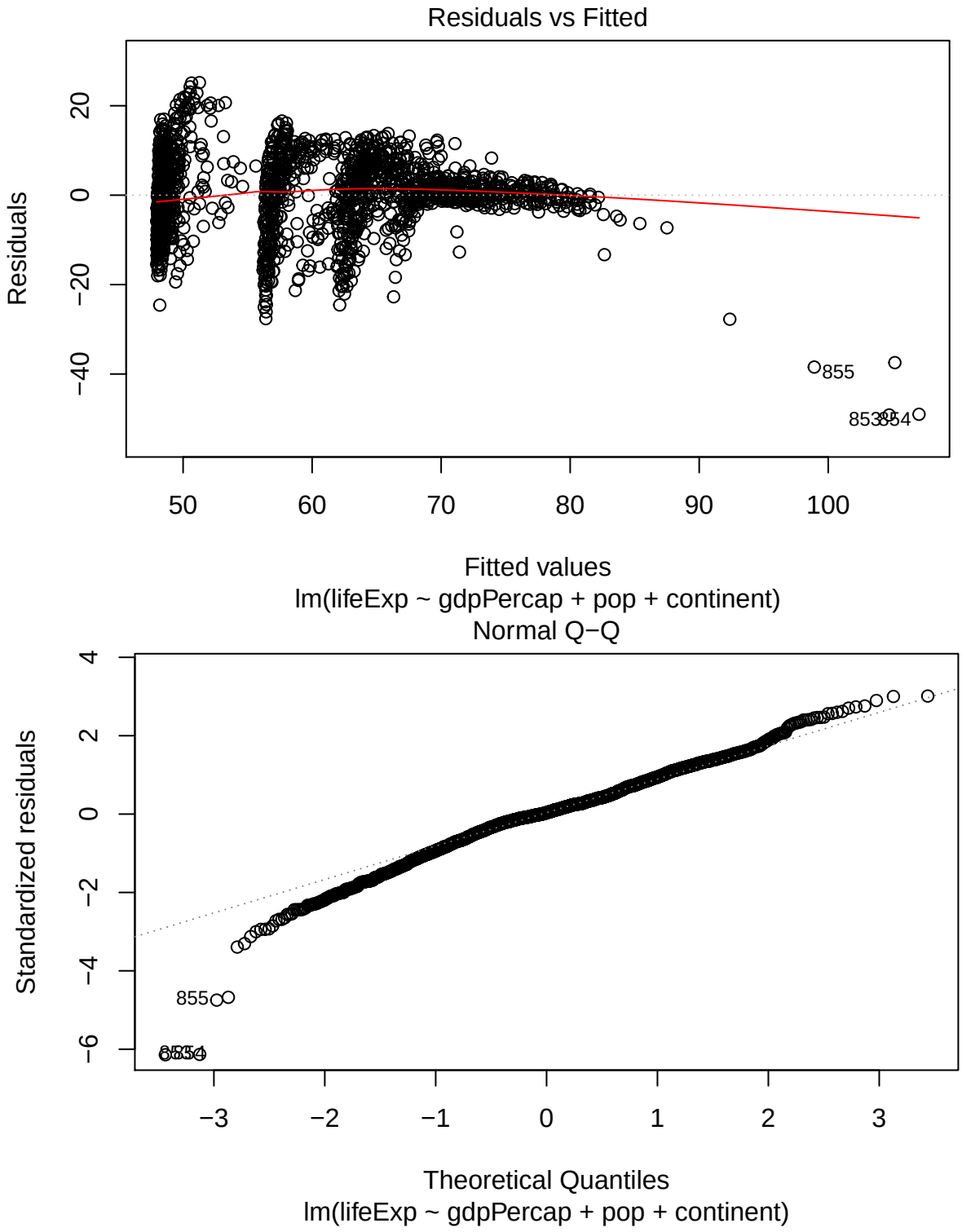
```
##    xvals     yvals     upper     lower
## 1   Male 0.5735849 0.6378653 0.5093045
## 2 Female 0.6344507 0.6887845 0.5801169
```

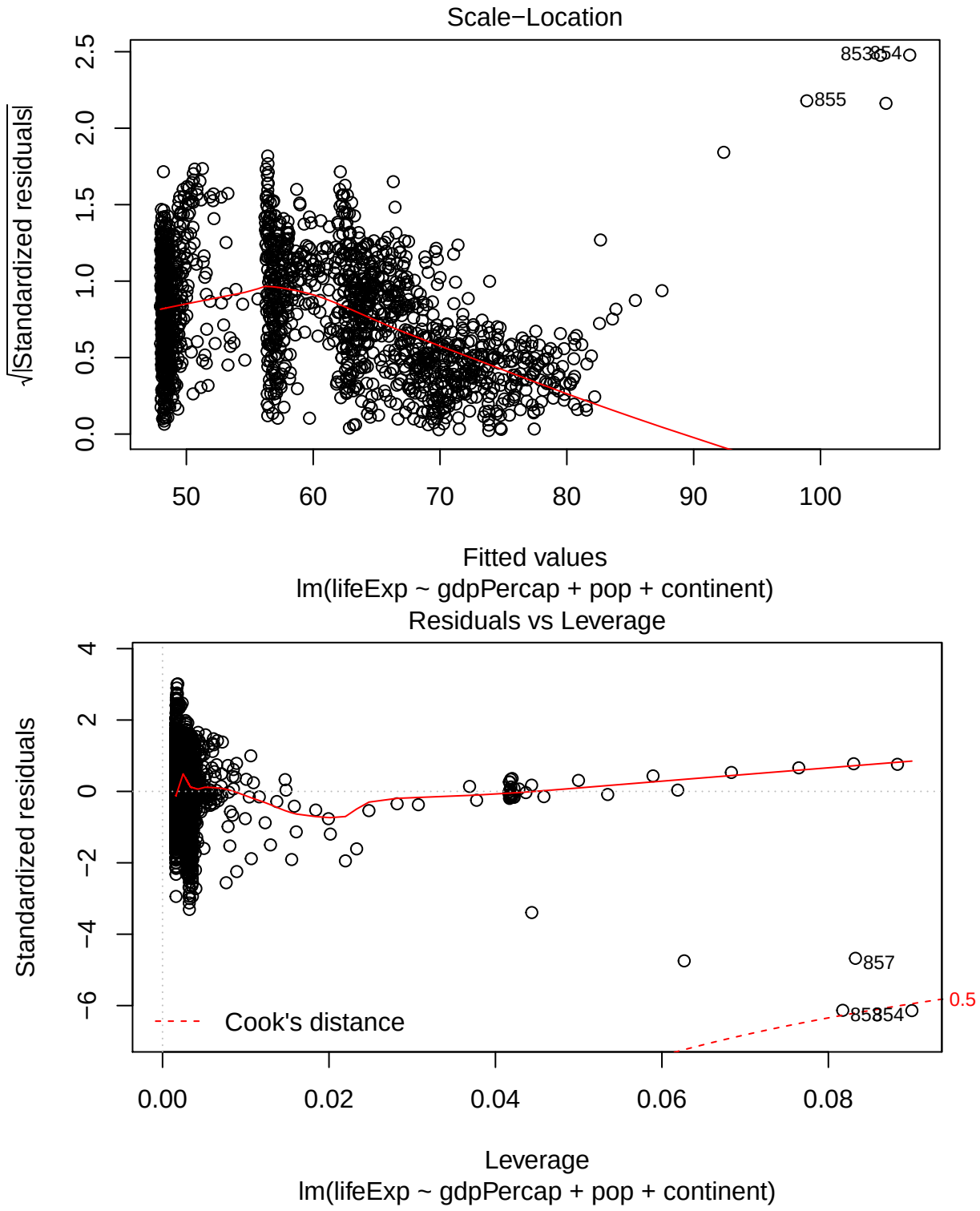
```
p <- ggplot(pv_cp, aes(x = reorder(xvals, yvals),
                       y = yvals, ymin = lower, ymax = upper))

p + geom_hline(yintercept = 0, color = "gray80") +
    geom_pointrange() + coord_flip() +
   labs(x = NULL, y = "Conditional Effects")
```



Default plot() methods should not be forgotten

```
out <- lm(formula = lifeExp ~ gdpPercap + pop + continent,
          data = gapminder)
plot(out, ask=FALSE)
```


Residuals vs Fitted
Residuals
20
0
−20
−40
855
50
60
70
80
90
100
Fitted values
lm(lifeExp ~ gdpPercap + pop + continent)
Normal Q−Q
Standardized residuals
4
2
0
−2
−4
−6
855
−3
−2
−1
0
1
2
3
Theoretical Quantiles
lm(lifeExp ~ gdpPercap + pop + continent)



The coefplot library, built on ggplot, but not necessarily actively maintained anymore

```
library(coefplot)
```

```
##
## Attaching package: 'coefplot'
```

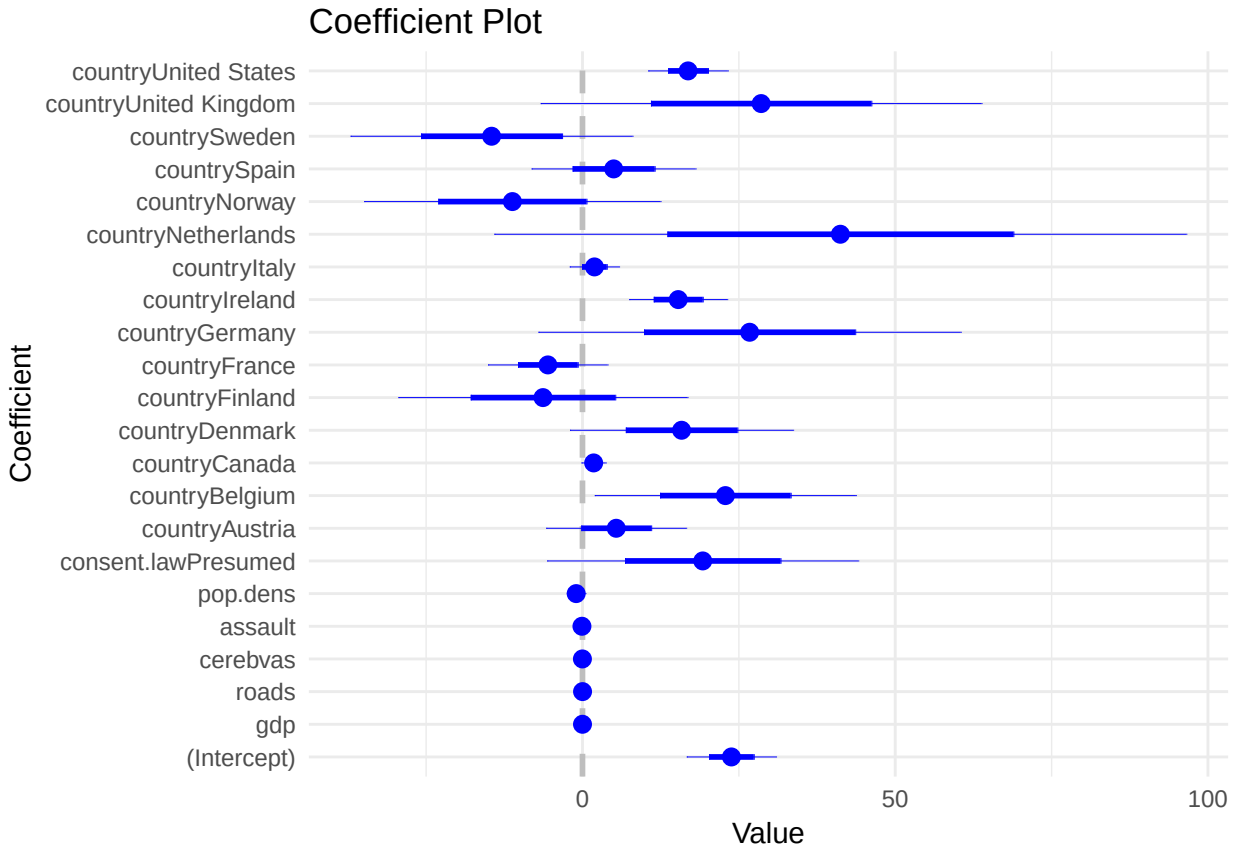
```
## The following object is masked from 'package:socviz':
##
##     multiplot
```

```
out <- lm(formula = lifeExp ~ gdpPercap + pop + continent,
          data = gapminder)
coefplot(out)
```

```
## Warning: Ignoring unknown aesthetics: xmin, xmax
```

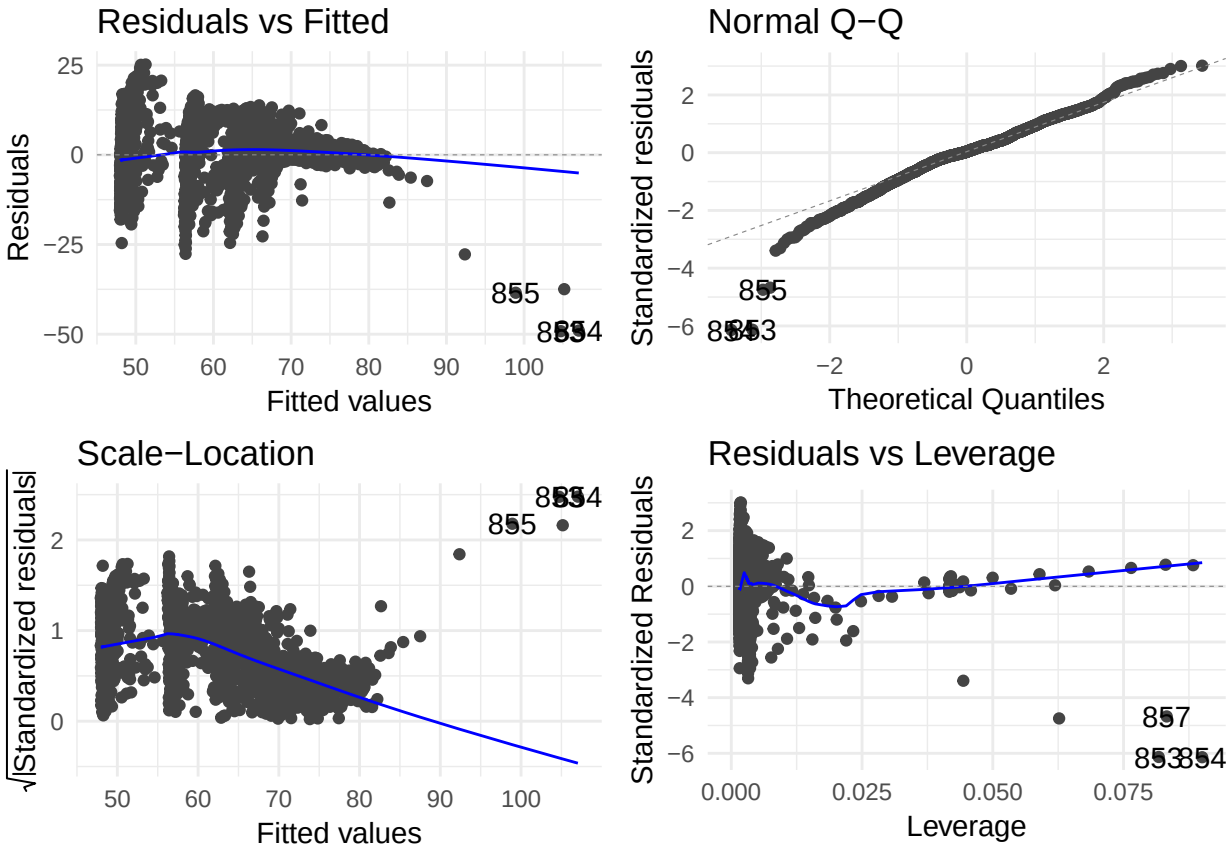

```
out2 <- lm(formula = donors ~ gdp + roads + cerebvas + assault + pop.dens + consent.law + country,
           data = organdata)
coefplot(out2)
```

```
## Warning: Ignoring unknown aesthetics: xmin, xmax
```



ggfortify means to replace all the default plots with ggplot based alternative. Tries to automatically guess the best plots for the model you're working on.

```
library(ggfortify)
autoplot(out)
```



Day 3: 08-18-2017

Making maps

Choropleths

R is not a GIS, but can make maps and read spatial data. Choropleths are doable. Lets make some maps with US election data.

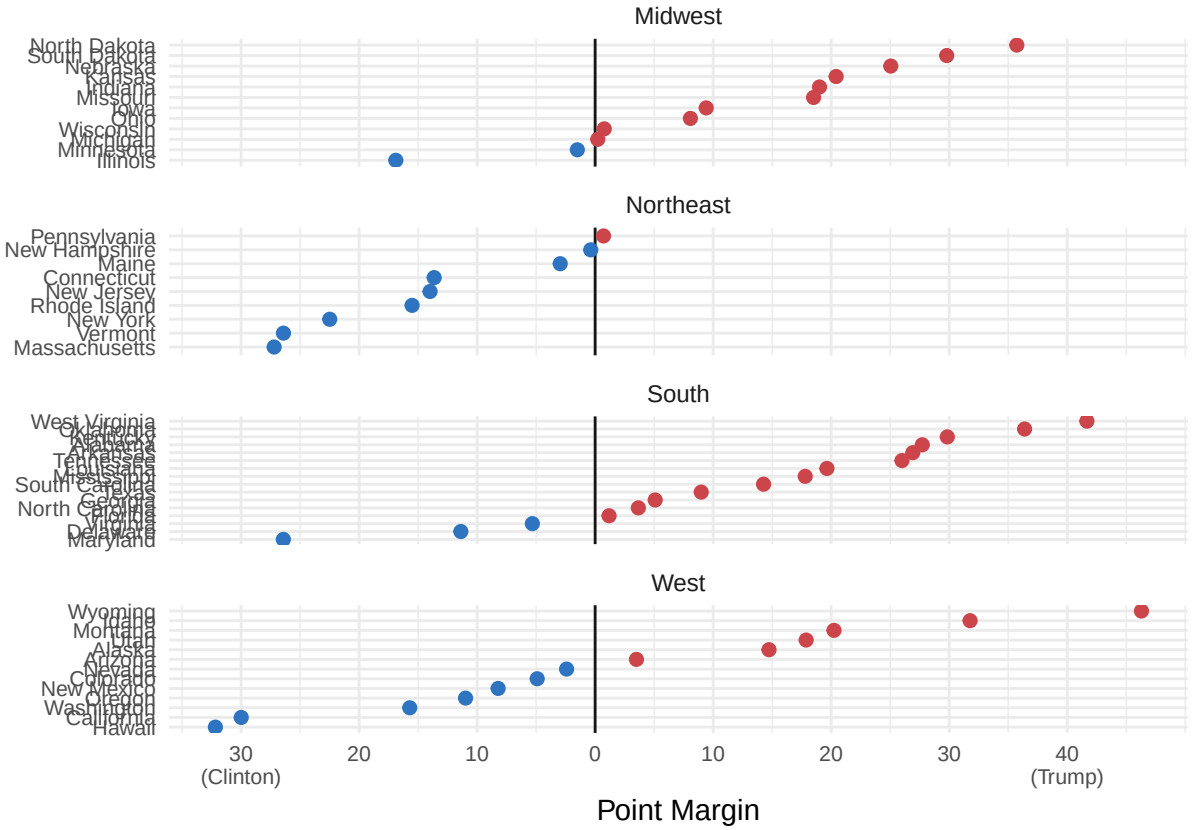
```
election %>% select(state, total_vote, r_points, pct_trump, party, census) %>% sample_n(5)
```

```
## # A tibble: 5 x 6
##           state total_vote r_points pct_trump      party    census
##           <chr>      <dbl>    <dbl>     <dbl>      <chr>     <chr>
## 1 Massachusetts    3325046   -27.20     32.81   Democrat Northeast
## 2  Pennsylvania    6166710     0.71     48.17 Republican Northeast
## 3        Alaska     318608    14.73     51.28 Republican      West
## 4      Colorado    2780247    -4.91     43.25   Democrat      West
## 5     Minnesota    2945233    -1.51     44.93   Democrat   Midwest
```

First we make a nice dotplot to remind us

```
party_colors <- c("#2E74C0", "#CB454A")
p <- ggplot(data = subset(election, st %nin% "DC"),
            mapping = aes(x = r_points,
                          y = reorder(state, r_points),
                          color = party))
```

```
p + geom_vline(xintercept = 0, alpha = 0.9) +
    geom_point(size = 2) +
    scale_color_manual(values = party_colors) +
    scale_x_continuous(breaks = c(-30, -20, -10, 0, 10, 20, 30, 40),
                       labels = c("30\n(Clinton)", "20", "10", "0", "10", "20", "30", "40\n(Trump)")) +
facet_wrap(~ census, scales = "free_y", ncol=1) +
  guides(color=FALSE) + labs(x = "Point Margin", y = "") +
    theme(axis.text=element_text(size=8))
```



OK. And our first map, load library and included dataset. Check it out:

```
library(maps)
```

```
##
## Attaching package: 'maps'
```

```
## The following object is masked from 'package:purrr':
##
##     map
```

```
us_states <- map_data("state")
head(us_states)
```

```
##        long      lat group order  region subregion
## 1 -87.46201 30.38968     1     1 alabama      <NA>
## 2 -87.48493 30.37249     1     2 alabama      <NA>
## 3 -87.52503 30.37249     1     3 alabama      <NA>
## 4 -87.53076 30.33239     1     4 alabama      <NA>
## 5 -87.57087 30.32665     1     5 alabama      <NA>
```

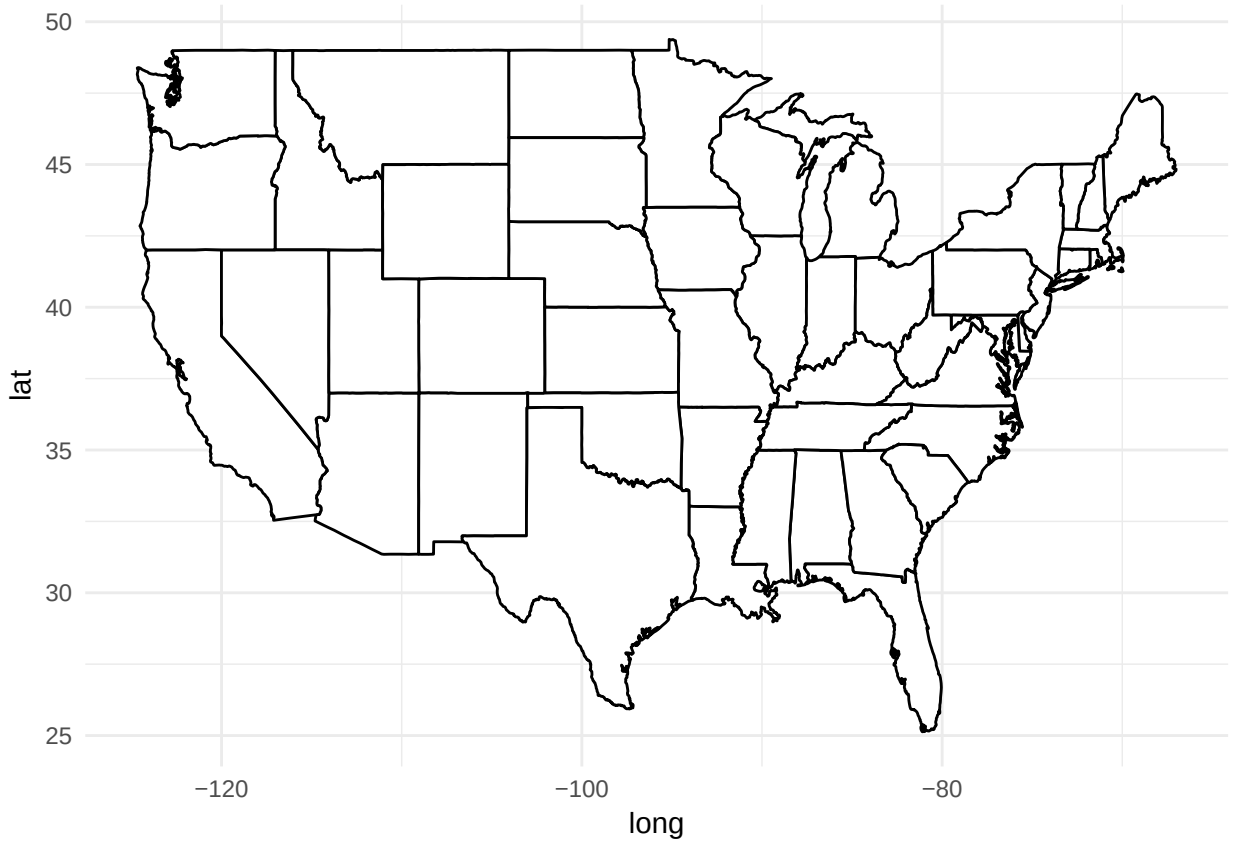
```
## 6 -87.58806 30.32665     1     6 alabama      <NA>
```

```
dim(us_states)
```

```
## [1] 15537     6
```

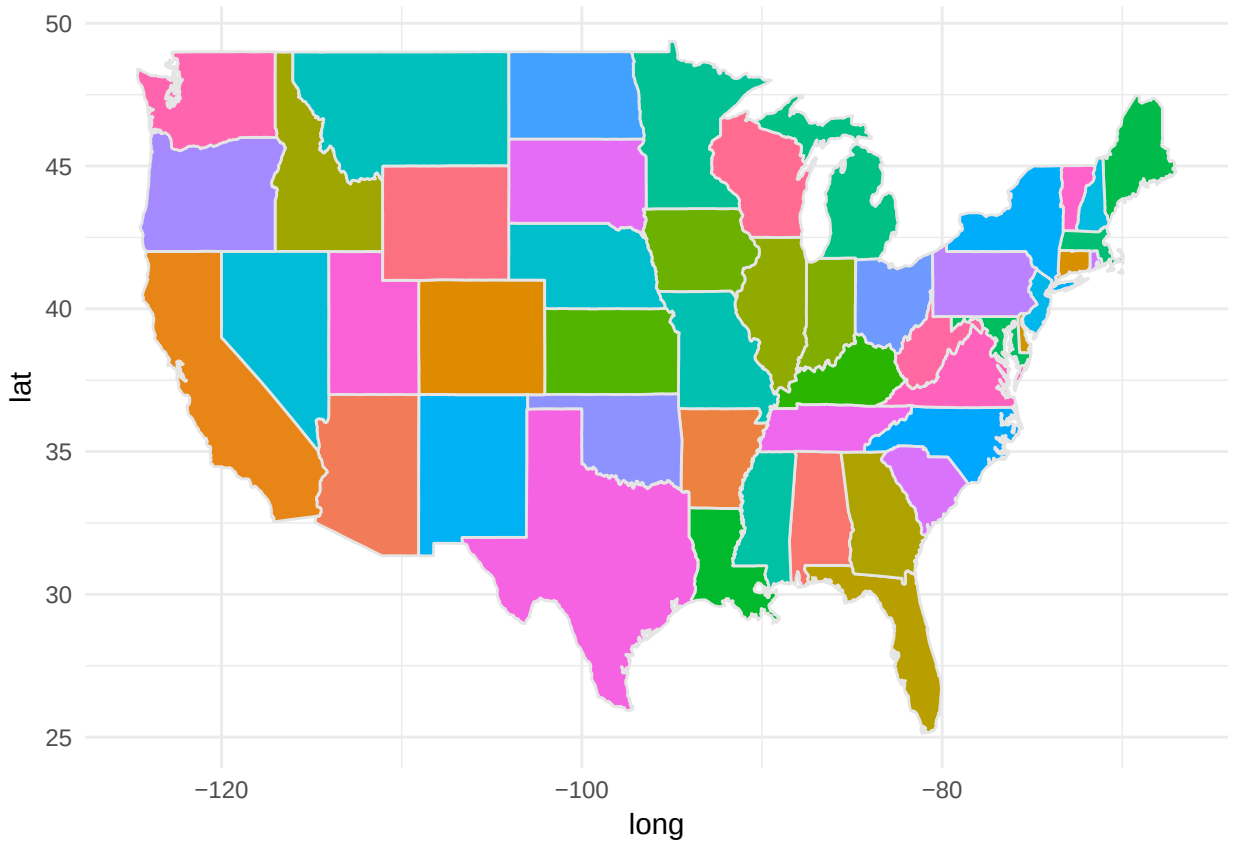
And the map:

```
p <- ggplot(data = us_states, aes(x = long,
                                  y = lat,
                                  group = group))
p + geom_polygon(fill = "white", color = "black")
```



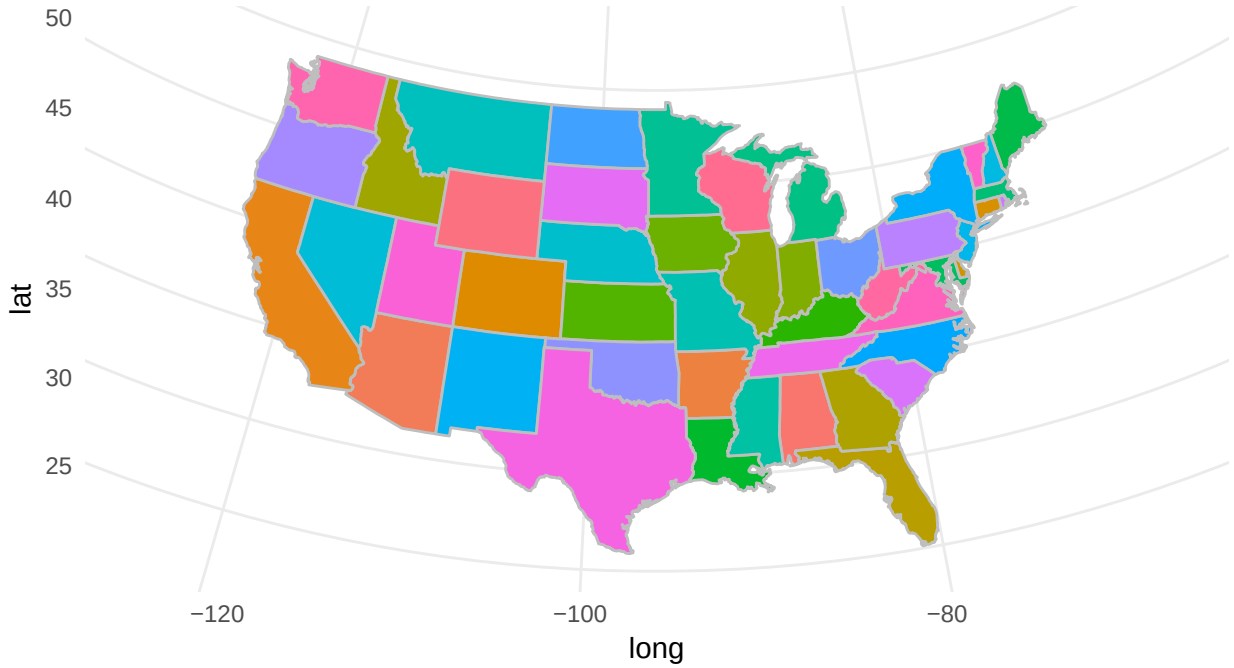
With colors:

```
p <- ggplot(data = us_states, aes(x = long,
                                  y = lat,
                                  group = group,
                                  fill = region))
p + geom_polygon(color = "grey90") + guides(fill = FALSE)
```



But maps should have projections!

```
p <- ggplot(data = us_states, aes(x = long,
                                  y = lat,
                                  group = group,
                                  fill = region))
p + geom_polygon(color = "grey") +
    coord_map(projection = "albers", lat0 = 39, lat1 = 45) +
    guides(fill = FALSE)
```



Joining data set. Don't do this part blindly. It's the most important piece! Be aware of the variables you are merging the data on and why you are doing it.

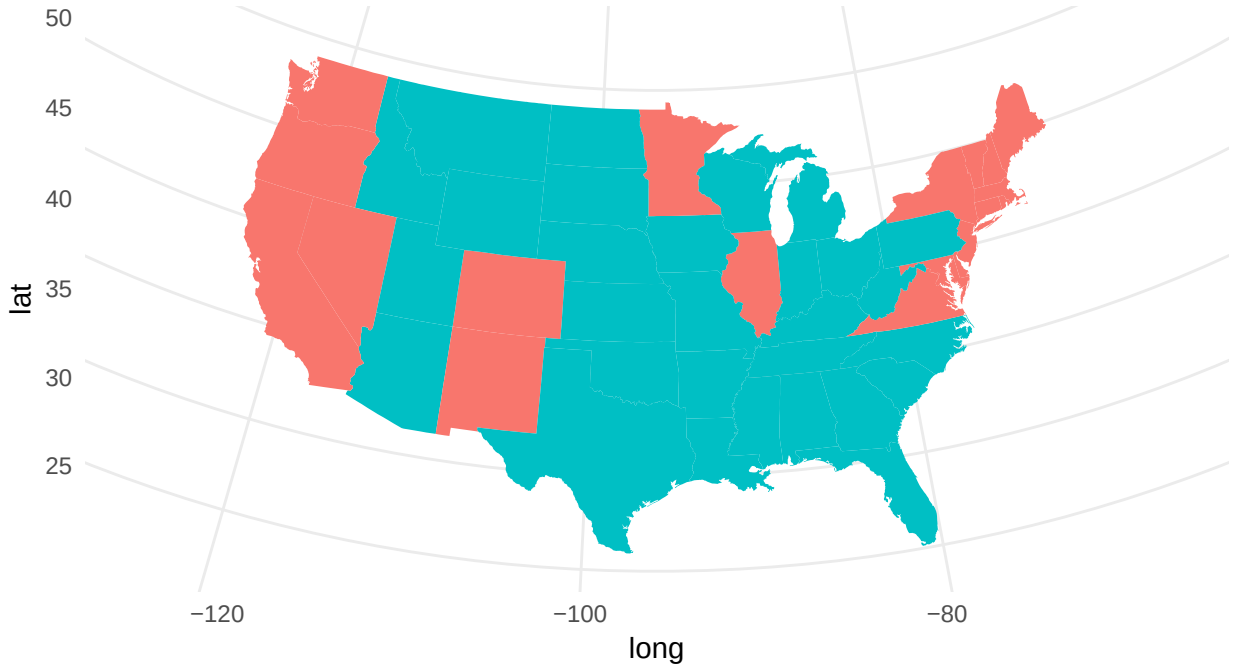
```
election$region <- tolower(election$state)
us_states_elec <- left_join(us_states, election)
```

```
## Joining, by = "region"
```

```
dim(us_states)
```

```
## [1] 15537     6
```

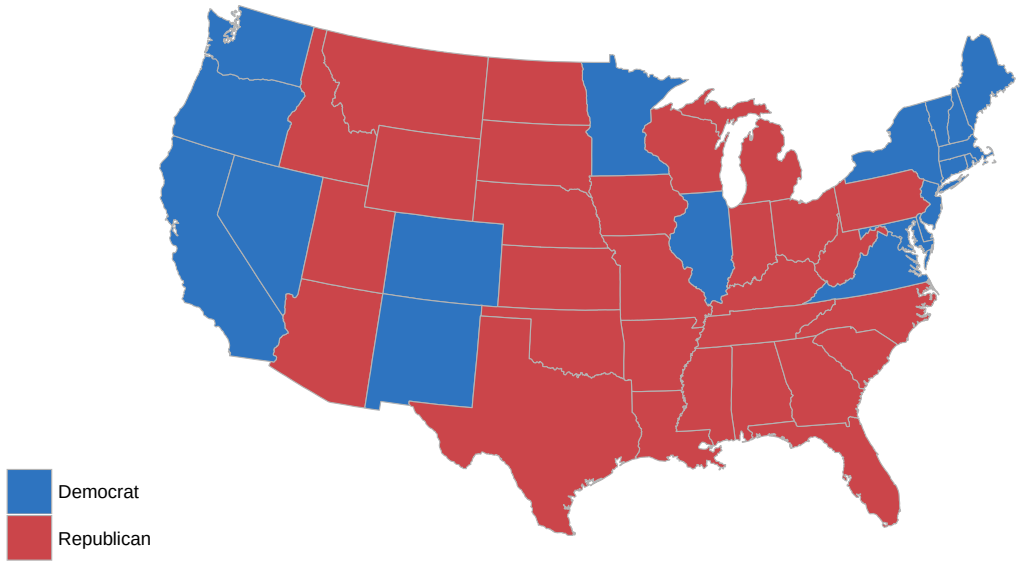
```
p <- ggplot(data = us_states_elec,
            aes(x = long, y = lat,
                group = group, fill = party))

p + geom_polygon() +
    coord_map(projection = "albers", lat0 = 39, lat1 = 45) +
    guides(fill = FALSE)
```



With some added bells and whistles:

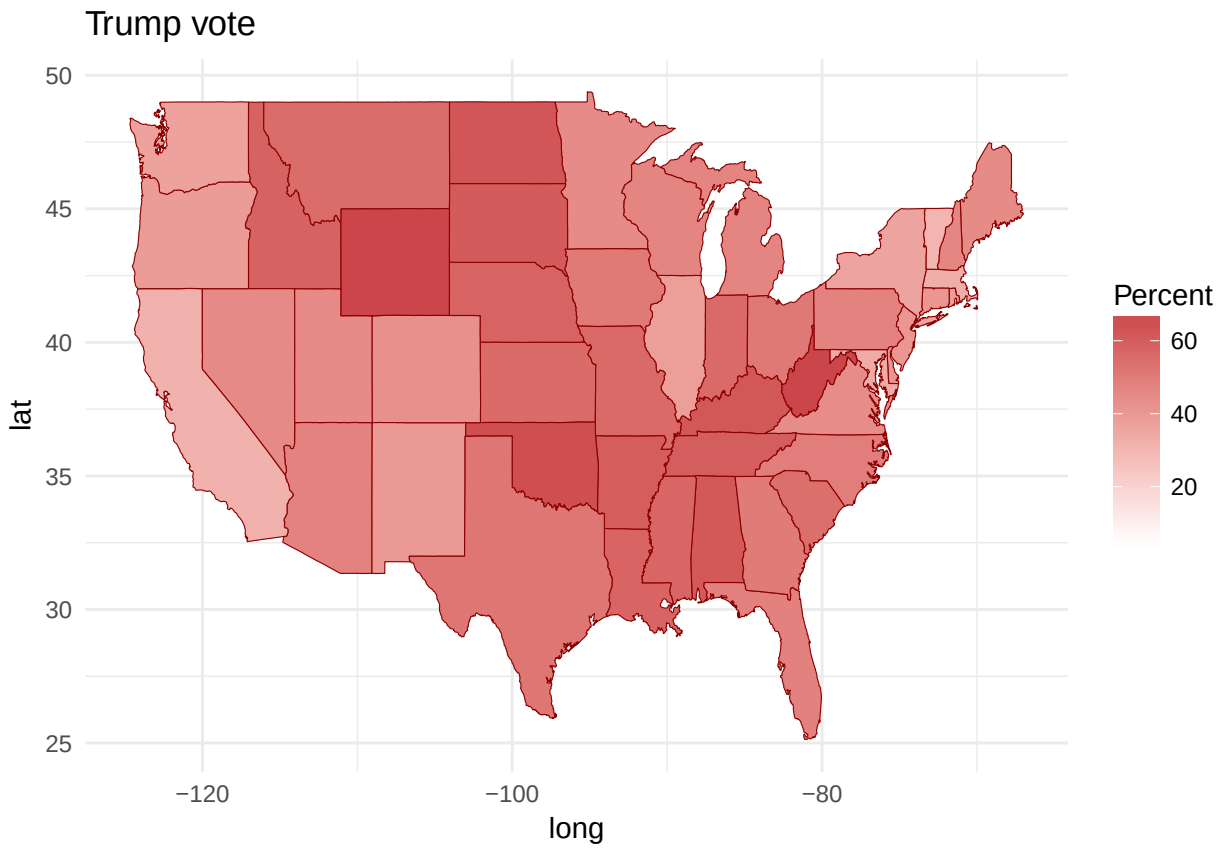
```
theme_map <- function(base_size=9, base_family="") {
    require(grid)
    theme_bw(base_size=base_size, base_family=base_family) %+replace%
        theme(axis.line=element_blank(),
              axis.text=element_blank(),
              axis.ticks=element_blank(),
              axis.title=element_blank(),
              panel.background=element_blank(),
              panel.border=element_blank(),
              panel.grid=element_blank(),
              panel.spacing=unit(0, "lines"),
              plot.background=element_blank(),
              legend.justification = c(0,0),
              legend.position = c(0,0)
              )
}

p <- ggplot(data = us_states_elec,
            aes(x = long, y = lat,
                group = group, fill = party))

p + geom_polygon(color = "gray70", size = 0.2) +
    scale_fill_manual(values = party_colors) +
    labs(title = "Election Results 2016", fill = NULL, color = "Party") +
    coord_map(projection = "albers", lat0 = 39, lat1 = 45) +
    theme_map()
```

```
## Loading required package: grid
```



Another variation:

```
p <- ggplot(data = us_states_elec,
            aes(x = long, y = lat,
                group = group, fill = pct_trump))

p + geom_polygon(color = "darkred", size = 0.2) +
    labs(title = "Trump vote", fill = "Percent") +
    scale_fill_gradient(low ="white", high = "#CB454A")
```

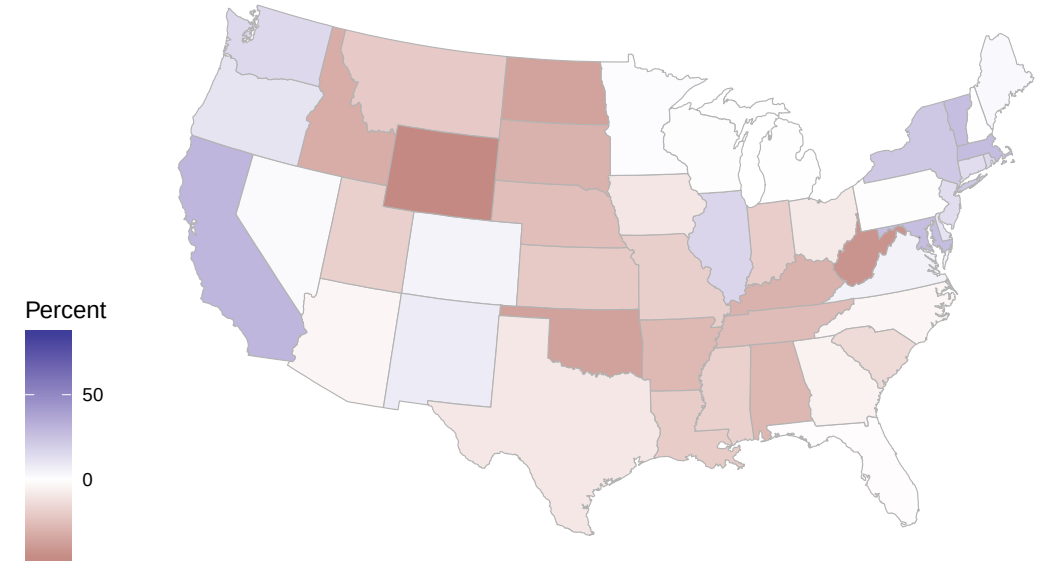



```
    coord_map(projection = "albers", lat0 = 39, lat1 = 45) +
    theme_map()
```

```
## NULL
```

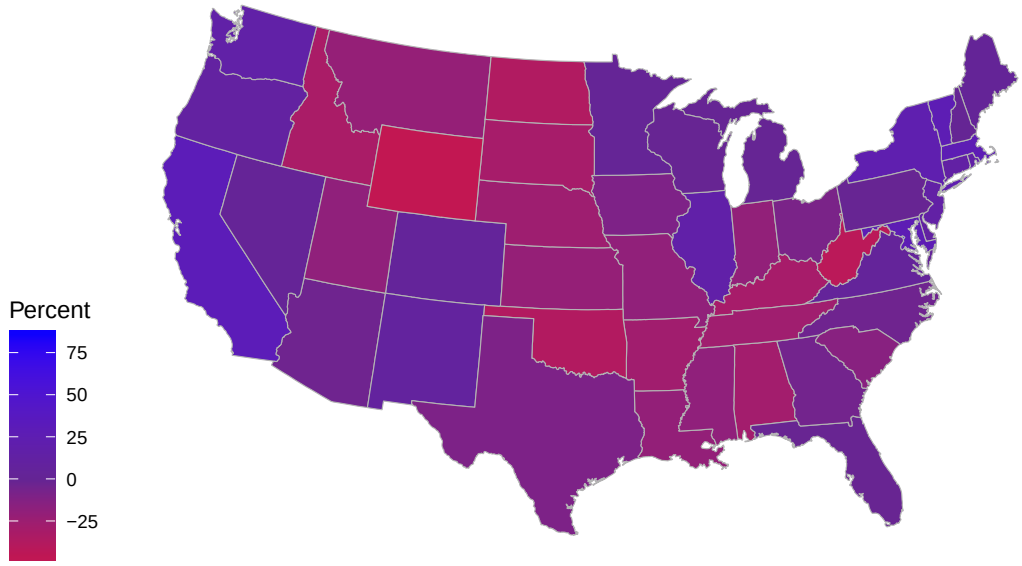
Another kind of gradation:

```
p <- ggplot(data = us_states_elec,
            aes(x = long, y = lat,
                group = group, fill = d_points))

p + geom_polygon(color = "grey70", size = 0.2) +
    labs(title = "Winning margins", fill = "Percent") +
    scale_fill_gradient2() +  # The reason the 0 is not in the middle is because of the outlier DC
    coord_map(projection = "albers", lat0 = 39, lat1 = 45) +
    theme_map()
```



Add some values to scale-gradient2

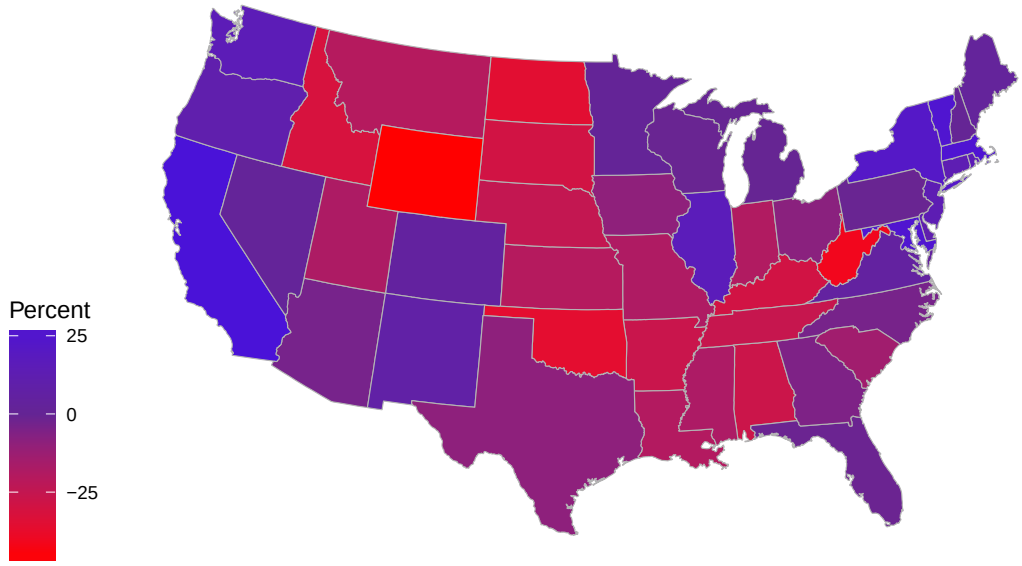
```
p <- ggplot(data = us_states_elec,
            aes(x = long, y = lat,
                group = group, fill = d_points))

p + geom_polygon(color = "grey70", size = 0.2) +
    labs(title = "Winning margins", fill = "Percent") +
    scale_fill_gradient2(low = "red",
                         mid = scales::muted("purple"),
                         high = "blue",
                         breaks = c(-25, 0, 25, 50, 75)) +  # The reason the 0 is not in the middle is
    coord_map(projection = "albers", lat0 = 39, lat1 = 45) +
    theme_map()
```



lets drop District of columbia, since it doesnt even show up on the map.

```
p <- ggplot(data = subset(us_states_elec,
                          region %nin% "district of columbia"),
            aes(x = long, y = lat,
                group = group, fill = d_points))

p + geom_polygon(color = "grey70", size = 0.2) +
    labs(title = "Winning margins", fill = "Percent") +
    scale_fill_gradient2(low = "red",
                         mid = scales::muted("purple"),
                         high = "blue",
                         breaks = c(-25, 0, 25, 50, 75)) +  # The reason the 0 is not in the middle is
    coord_map(projection = "albers", lat0 = 39, lat1 = 45) +
    theme_map()
```

Winning margins



The fallacy of misplaced concreteness:

Americas coropleth map.

```
county_map %>% sample_n(10)
```

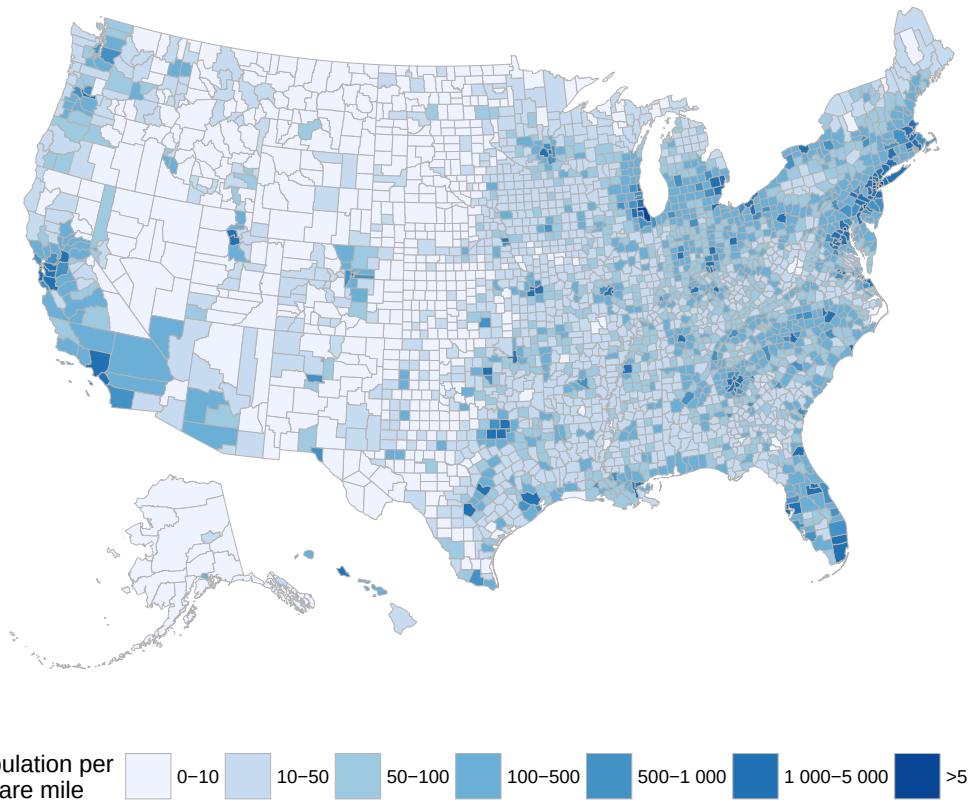
```
##              long         lat  order  hole piece            group    id
## 117606  -717773.6 -1246666.97 117606 FALSE     1 0500000US35053.1 35053
## 120573  2057191.4   -76591.56 120573 FALSE     1 0500000US36105.1 36105
## 73404   1035596.7 -1698571.44  73404 FALSE     1 0500000US22075.1 22075
## 66815   1455409.4  -712169.43  66815 FALSE     1 0500000US21131.1 21131
## 15908  -1502944.5 -2065675.32  15908 FALSE     1 0500000US02270.1 02270
## 49023  -1111962.9    78299.97  49023 FALSE     1 0500000US16037.1 16037
## 85790   1115028.0  -216602.70  85790 FALSE     1 0500000US26021.1 26021
## 53715    874298.9  -287851.55  53715 FALSE     1 0500000US17141.1 17141
## 103336   908608.6  -945263.92 103336 FALSE     1 0500000US29155.1 29155
## 93767    239138.8   250489.27  93767 FALSE     1 0500000US27107.1 27107
```

First we merge county_map with county_data

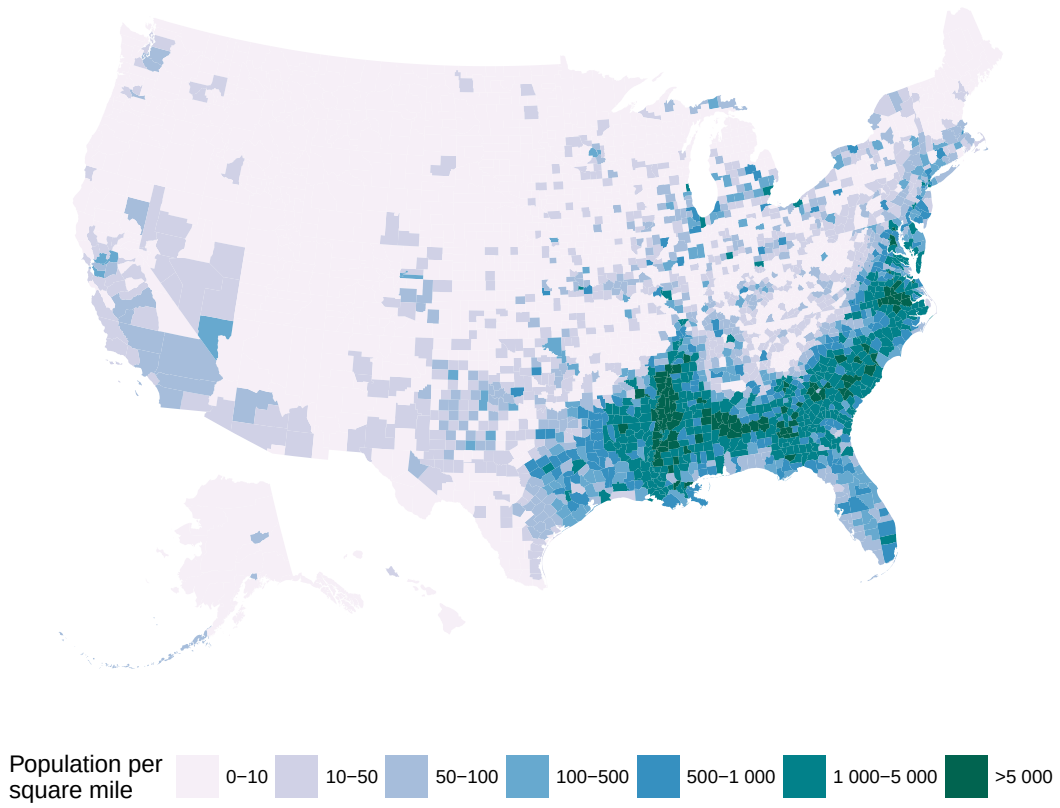
```
county_full <- left_join(county_map, county_data, by = "id")
```

Now we make the map!

```
p <- ggplot(data = county_full,
            mapping = aes(x = long, y = lat,
                          fill = pop_dens,
                          group = group))
p1 <- p + geom_polygon(color = "gray70", size = 0.1) + coord_equal()

p2 <- p1 + scale_fill_brewer(palette="Blues",
                             labels = c("0-10", "10-50", "50-100", "100-500", "500-1 000", "1 000-5 000
p2 + labs(fill = "Population per\nsquare mile") + theme_map() + guides(fill = guide_legend(nrow = 1)) +
```




```
p <- ggplot(data = county_full,
            mapping = aes(x = long, y = lat,
                          fill = pct_black,
                          group = group))
p1 <- p + geom_polygon() + coord_equal()

p2 <- p1 + scale_fill_brewer(palette="PuBuGn",
                             labels = c("0-10", "10-50", "50-100", "100-500", "500-1 000", "1 000-5 000
p2 + labs(fill = "Population per\nsquare mile") + theme_map() + guides(fill = guide_legend(nrow = 1)) +
```



Lets create a custom pallette:

```
orange_pal <- RColorBrewer::brewer.pal(n = 6, name = "Oranges")
orange_pal
```

```
## [1] "#FEEDDE" "#FDD0A2" "#FDAE6B" "#FD8D3C" "#E6550D" "#A63603"
```

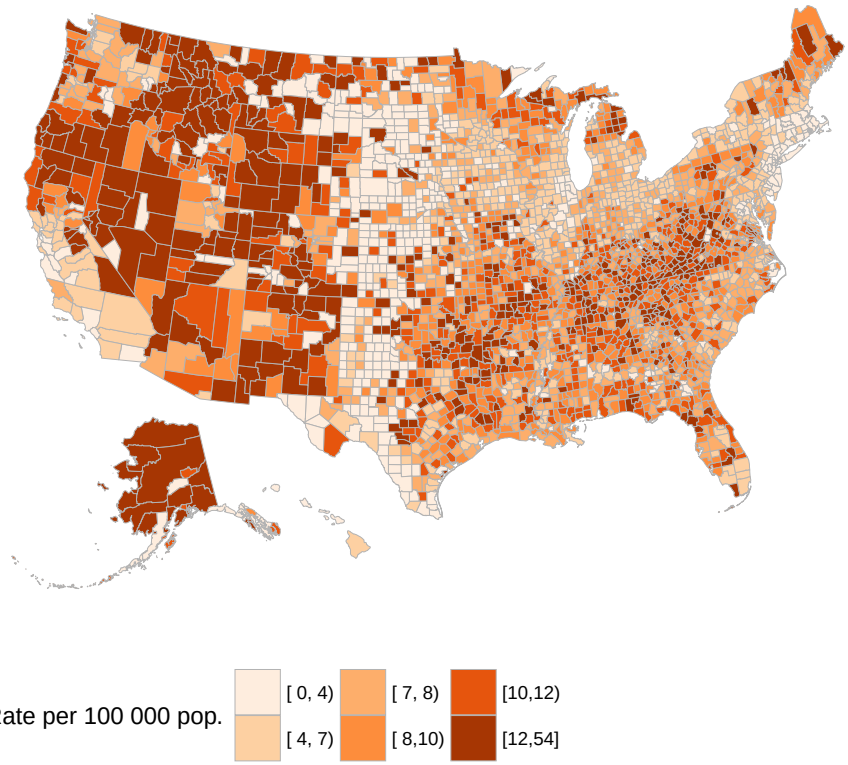
```
orange_rev <- rev(orange_pal)
orange_rev
```

```
## [1] "#A63603" "#E6550D" "#FD8D3C" "#FDAE6B" "#FDD0A2" "#FEEDDE"
```

and a gun map:

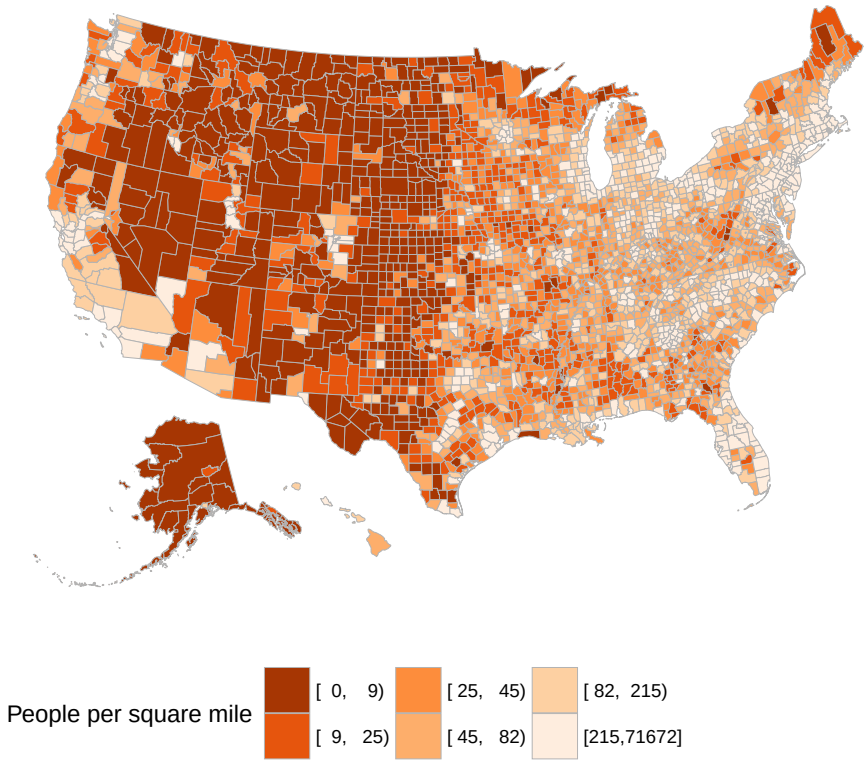
```
gun_p <- ggplot(data = county_full,
                mapping = aes(x = long, y = lat,
                              fill = su_gun6,
                              group = group))
gun_p1 <- gun_p + geom_polygon(color = "gray70", size = 0.1) + coord_equal()

gun_p2 <- gun_p1 + scale_fill_manual(values = orange_pal)

gun_p2 + labs(title="Gun-Related Suicides, 1999 - 2015",
              fill = "Rate per 100 000 pop.") +
  theme_map() +
  theme(legend.position = "bottom")
```

Gun−Related Suicides, 1999 − 2015




```
pop_p <- ggplot(data = county_full,
                mapping = aes(x = long, y = lat,
                              fill = pop_dens6,
                              group = group))
pop_p1 <- pop_p + geom_polygon(color = "gray70", size = 0.1) + coord_equal()

pop_p2 <- pop_p1 + scale_fill_manual(values = orange_rev)

pop_p2 + labs(title="Reverse-coded Population Density",
              fill = "People per square mile") +
  theme_map() +
  theme(legend.position = "bottom")
```

Reverse−coded Population Density




```
head(opiates)
```

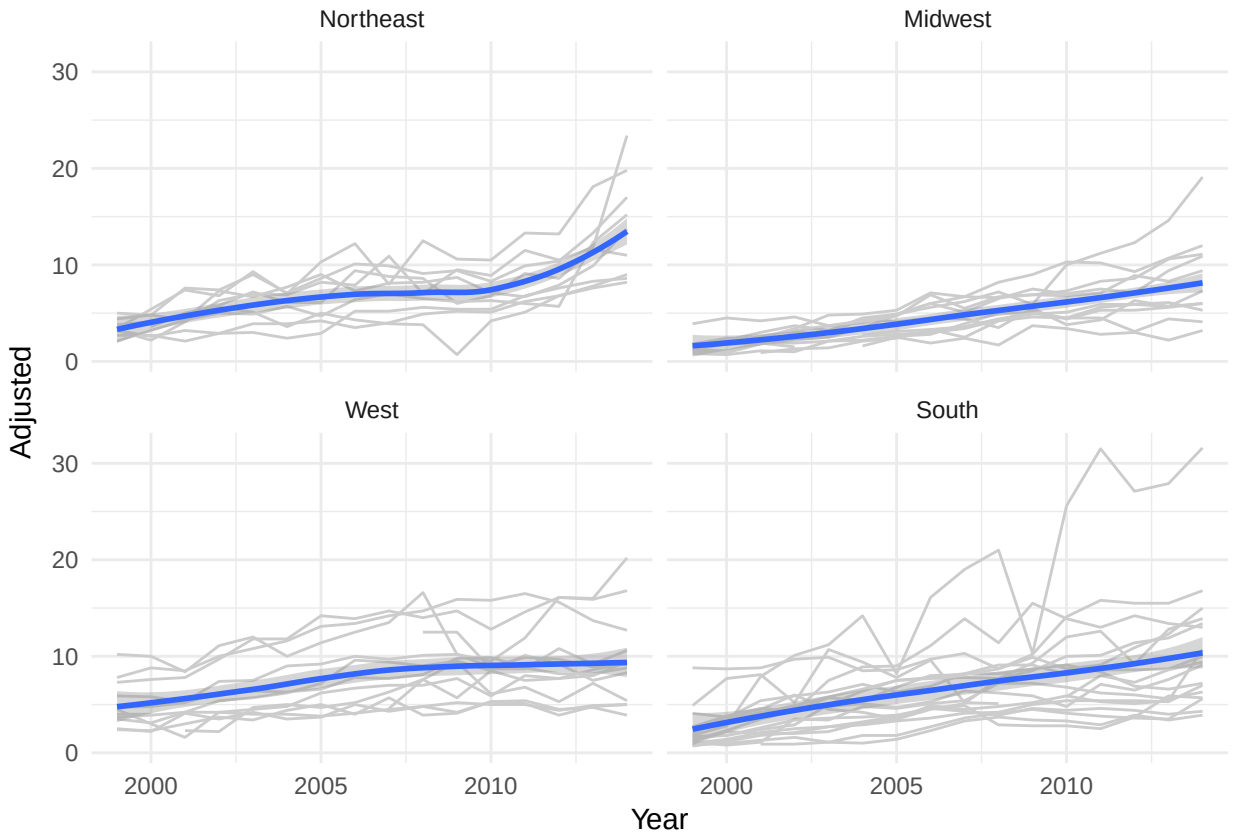
```
## # A tibble: 6 x 10
##    Year      State  FIPS Deaths Population Crude Adjusted Adjusted.se
##   <int>      <chr> <int>  <int>      <int> <dbl>    <dbl>       <dbl>
## 1  1999    Alabama     1     37    4430141   0.8      0.8         0.1
## 2  1999     Alaska     2     27     624779   4.3      4.0         0.8
## 3  1999    Arizona     4    229    5023823   4.6      4.7         0.3
## 4  1999   Arkansas     5     28    2651860   1.1      1.1         0.2
## 5  1999 California     6   1474   33499204   4.4      4.5         0.1
## 6  1999   Colorado     8    164    4226018   3.9      3.7         0.3
## # ... with 2 more variables: Region <ord>, Abbr <chr>
```

```
p <- ggplot(data = opiates,
            aes(x = Year, y = Adjusted, group = State))

p + geom_line(color = "gray80") + geom_smooth(aes(group = Region)) +
    facet_wrap(~ Region)
```

```
## `geom_smooth()` using method = 'loess'
```

```
## Warning: Removed 28 rows containing non-finite values (stat_smooth).
```

```
## Warning: Removed 17 rows containing missing values (geom_path).
```

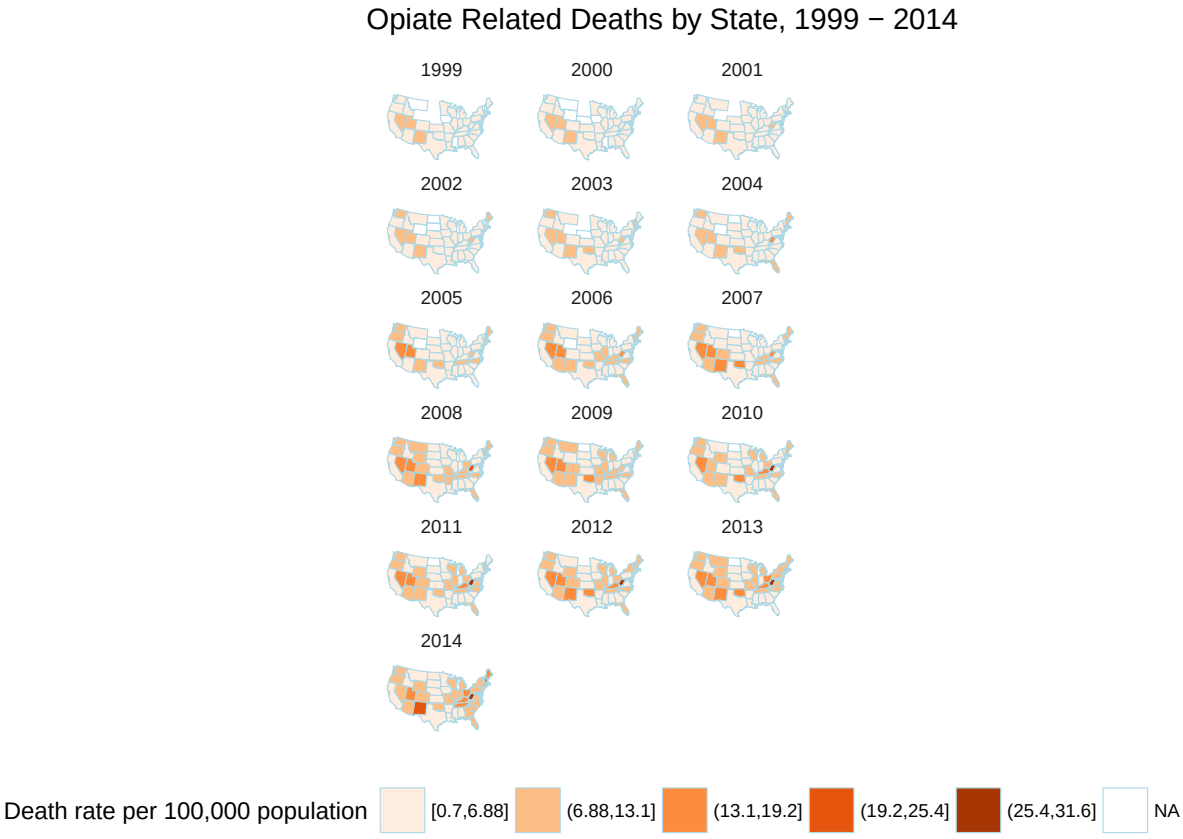



```
opiates$region <- tolower(opiates$State)
opiates_map <- left_join(us_states, opiates)
```

```
## Joining, by = "region"
```

mapping with facets

```
p0 <- ggplot(data = opiates_map,
             aes(x = long, y = lat,
                 group = group,
                 fill = cut_interval(Adjusted, n = 5)))
p1 <- p0 + geom_polygon(color = "lightblue", size = 0.2) +
            coord_map(projection = "albers", lat0 = 39, lat1 = 45)

p2 <- p1 + scale_fill_brewer(type = "seq", palette = "Oranges")

p2 + theme_map() + facet_wrap(~ Year, ncol = 3) +
  guides(fill = guide_legend(nrow = 1)) +
  theme(legend.position = "bottom",
        strip.background = element_blank()) +
  labs(fill = "Death rate per 100,000 population",
       title =  "Opiate Related Deaths by State, 1999 - 2014")
```


Opiate Related Deaths by State, 1999 − 2014
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
Death rate per 100,000 population
[0.7,6.88]
(6.88,13.1]
(13.1,19.2]
(19.2,25.4]
(25.4,31.6]
NA