Dataviz course

Lasse Gullvåg Sætre

08-16-2017

Kieran explains Markdown

I'm good, thank you.

Figure 1: R Markdown, familiar

So let's try to load the tidyverse

Tidyverse

```r
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.

```r
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.

```r
getwd()
```

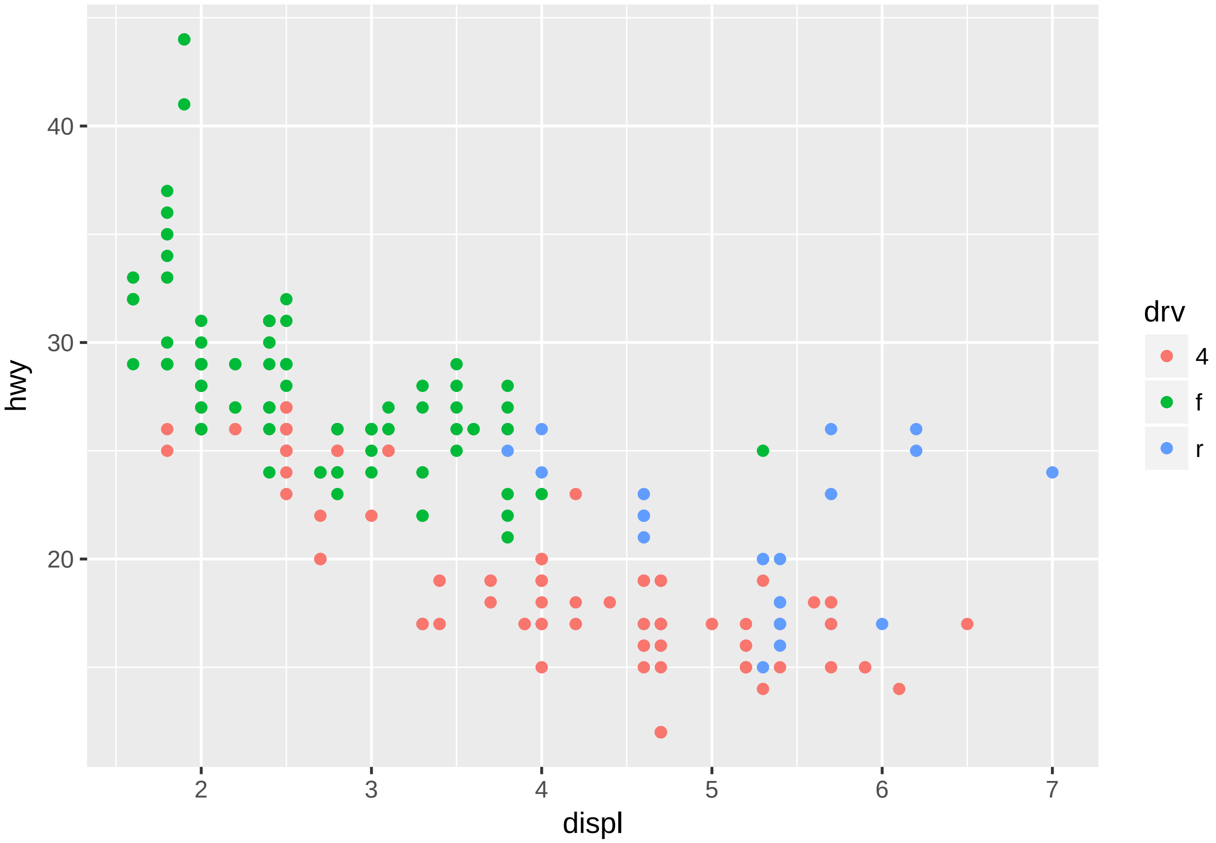
```
## [1] "/home/lassegs/Documents/sos9028/notes/rmd"
```

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

```r
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)
```

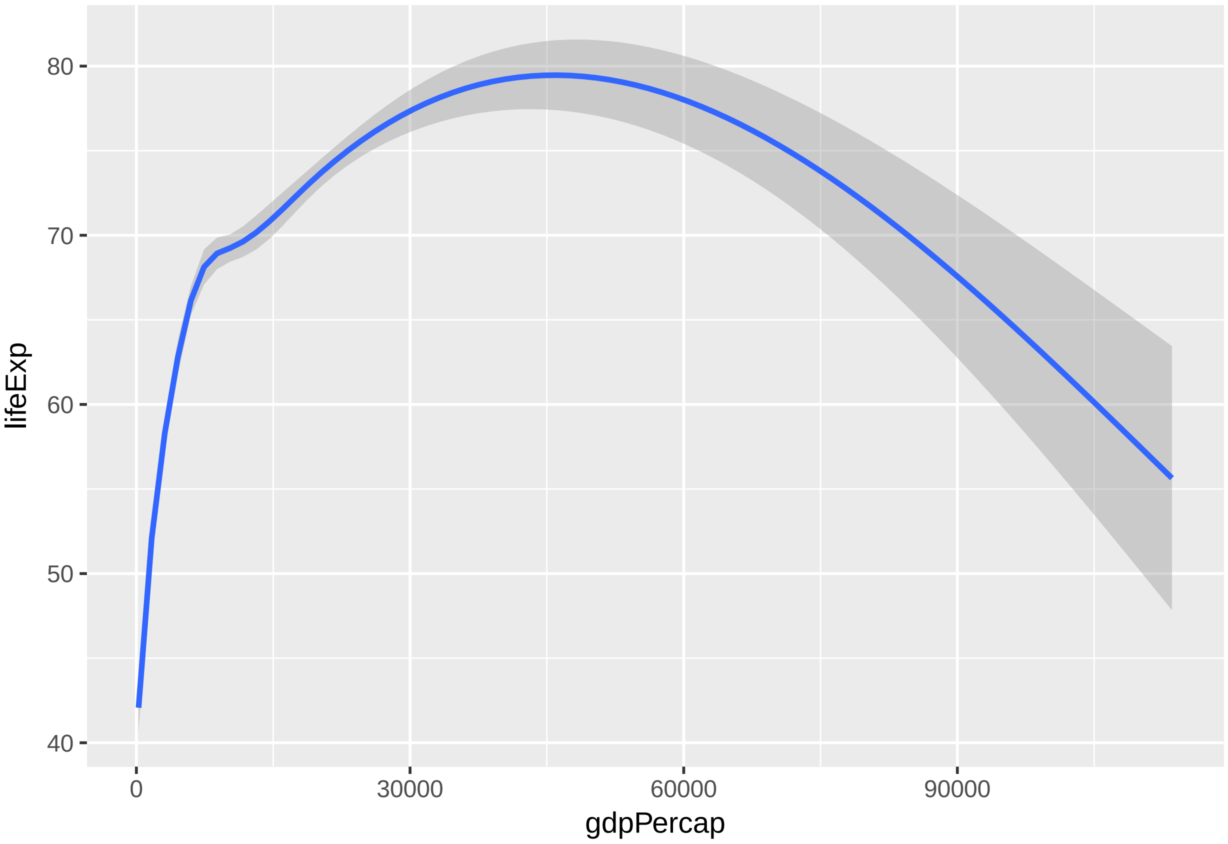
```
## # A tibble: 6 x 6
##       country continent  year lifeExp      pop gdpPercap
##        <fctr>    <fctr> <int>   <dbl>    <int>     <dbl>
## 1 Afghanistan      Asia  1952  28.801  8425333  779.4453
## 2 Afghanistan      Asia  1957  30.332  9240934  820.8530
## 3 Afghanistan      Asia  1962  31.997 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

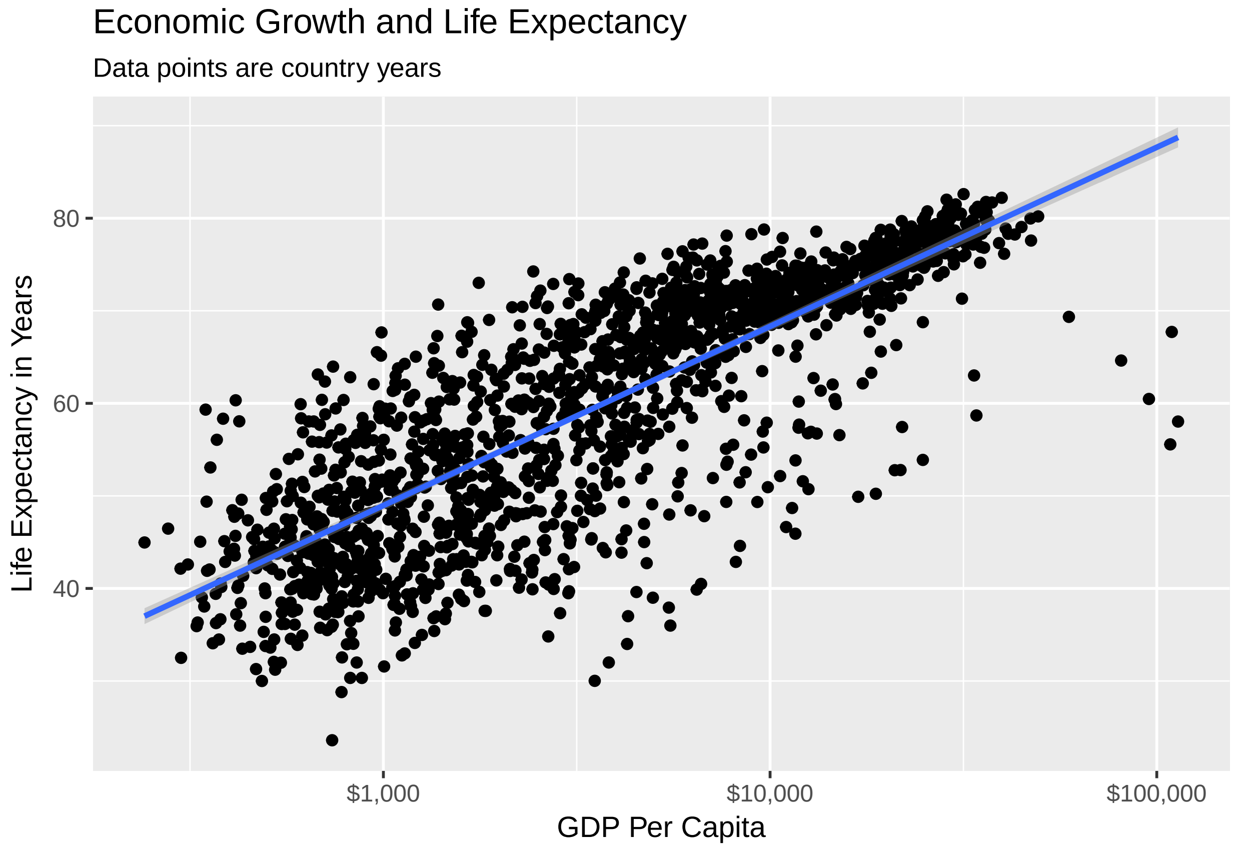
Data in long format vs data in wide format. GGPlot wants your data in long format. If the data is in the right shape, everything goes much more smoothly. wants two-dimensional data, with variables in the columns, observations in the rows.

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")
```

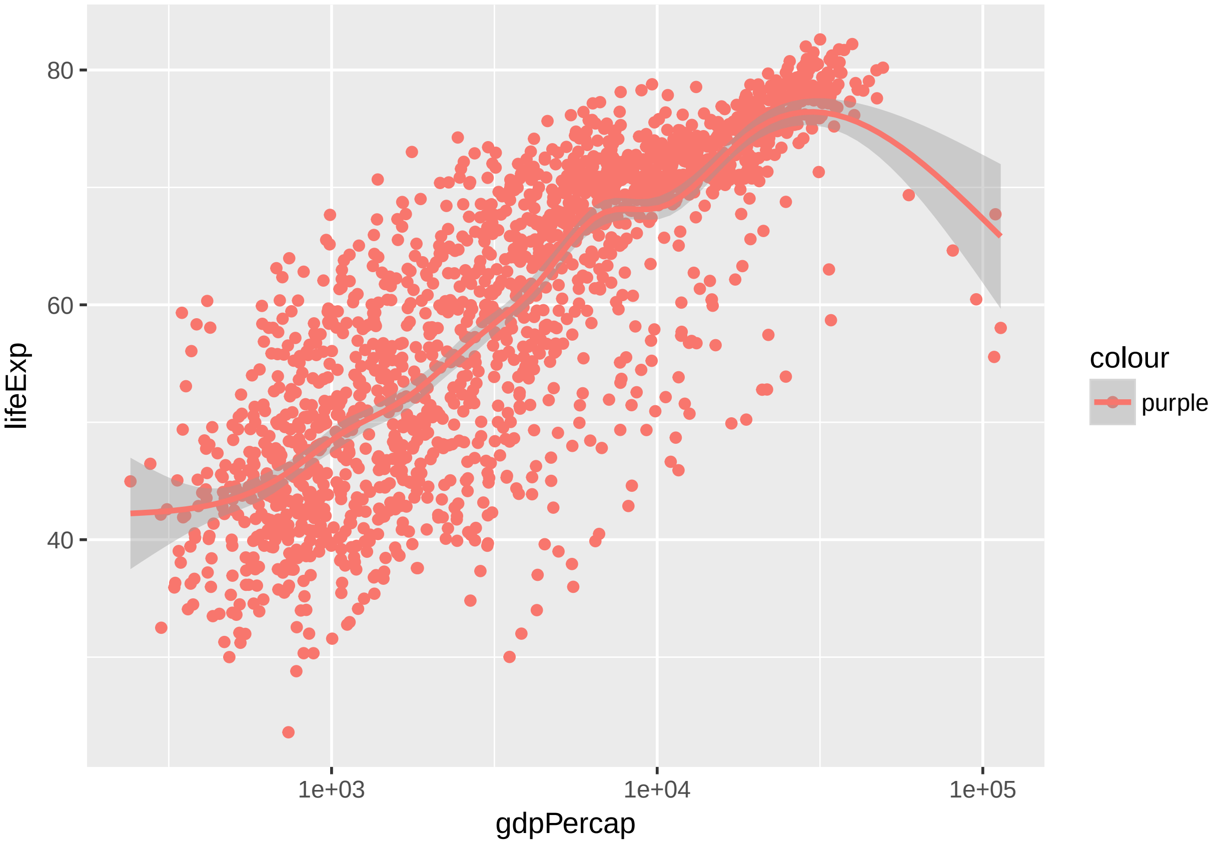


Mapping vs setting aesthetics

Crucial distinction. Makes everything clearer.

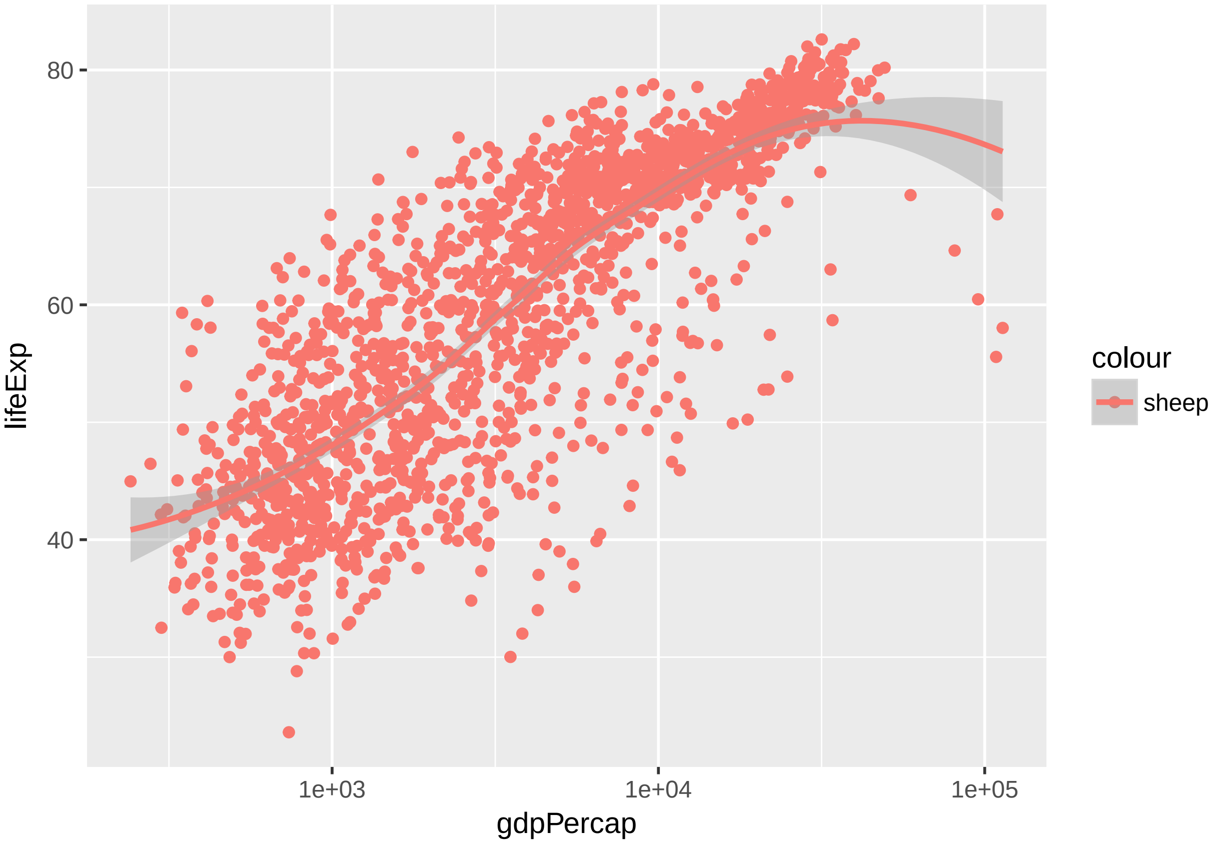
```
p <- ggplot(data = gapminder,
            mapping = aes( x = gdpPercap,
                           y = lifeExp,
                           color = "purple"))

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.

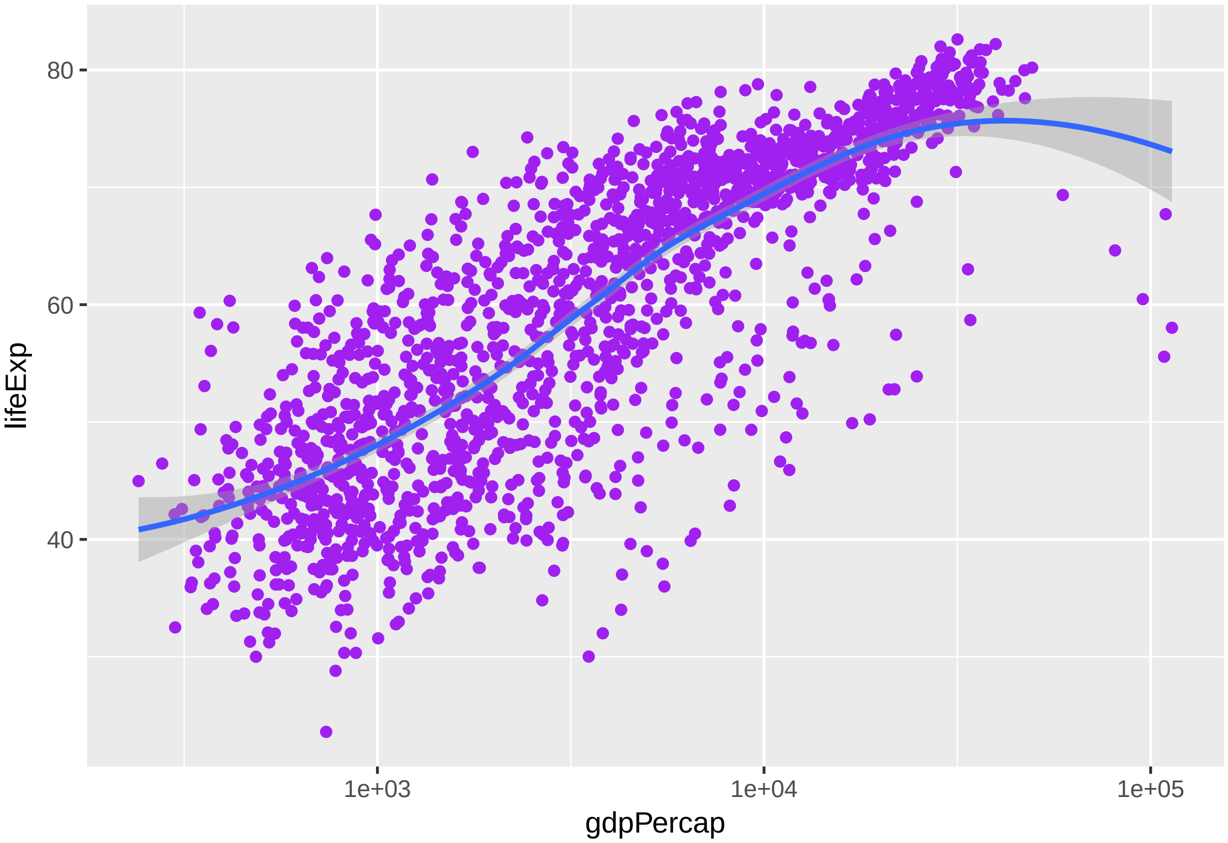
```r
p <- ggplot(data = gapminder,
            mapping = aes(x = gdpPercap,
                          y = lifeExp,
                          color = "sheep"))
p + geom_point() +
  geom_smooth(method='loess') +
  scale_x_log10()
```



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:

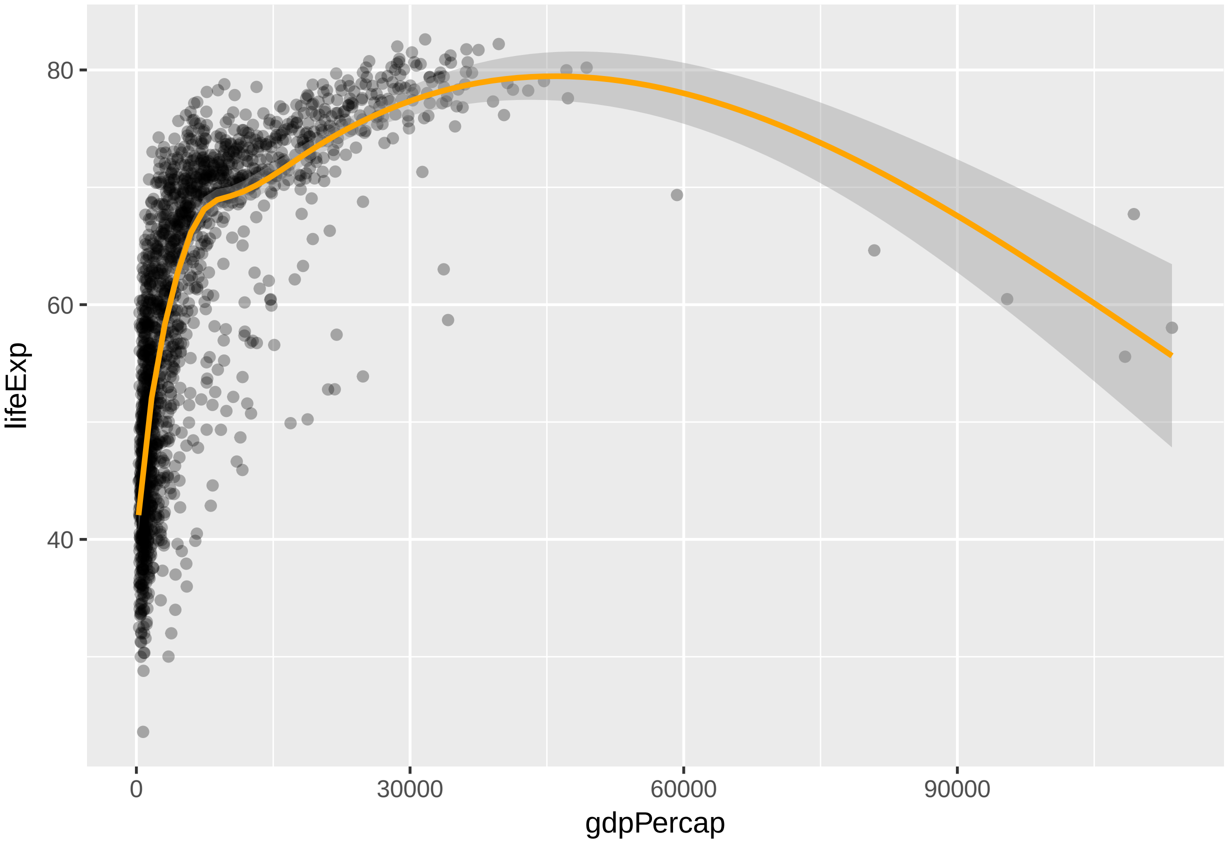
```
p <- ggplot(data = gapminder,
            mapping = aes(x = gdpPercap,
                          y = lifeExp))
p + geom_point(color = "purple") +
    geom_smooth(method='loess') +
    scale_x_log10()
```



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'
```

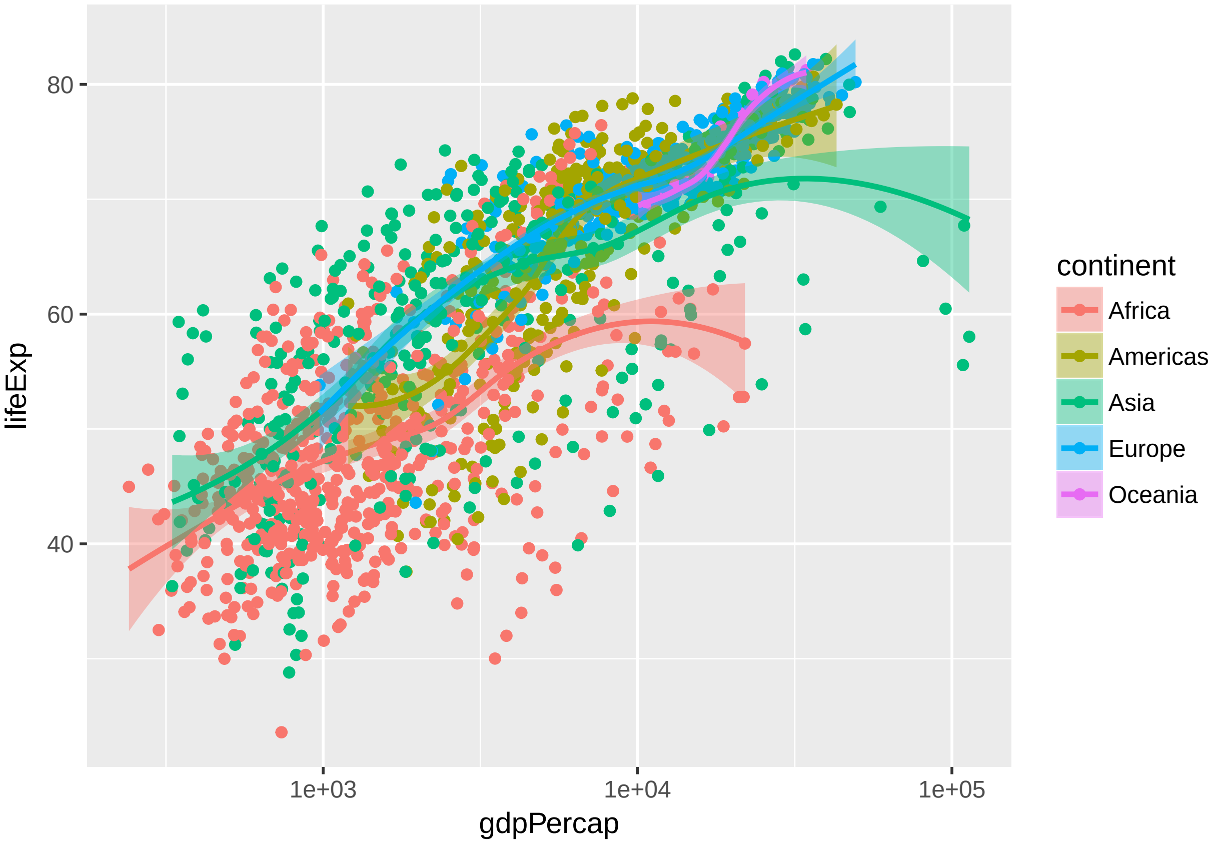


```
    scale_x_log10()
```

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

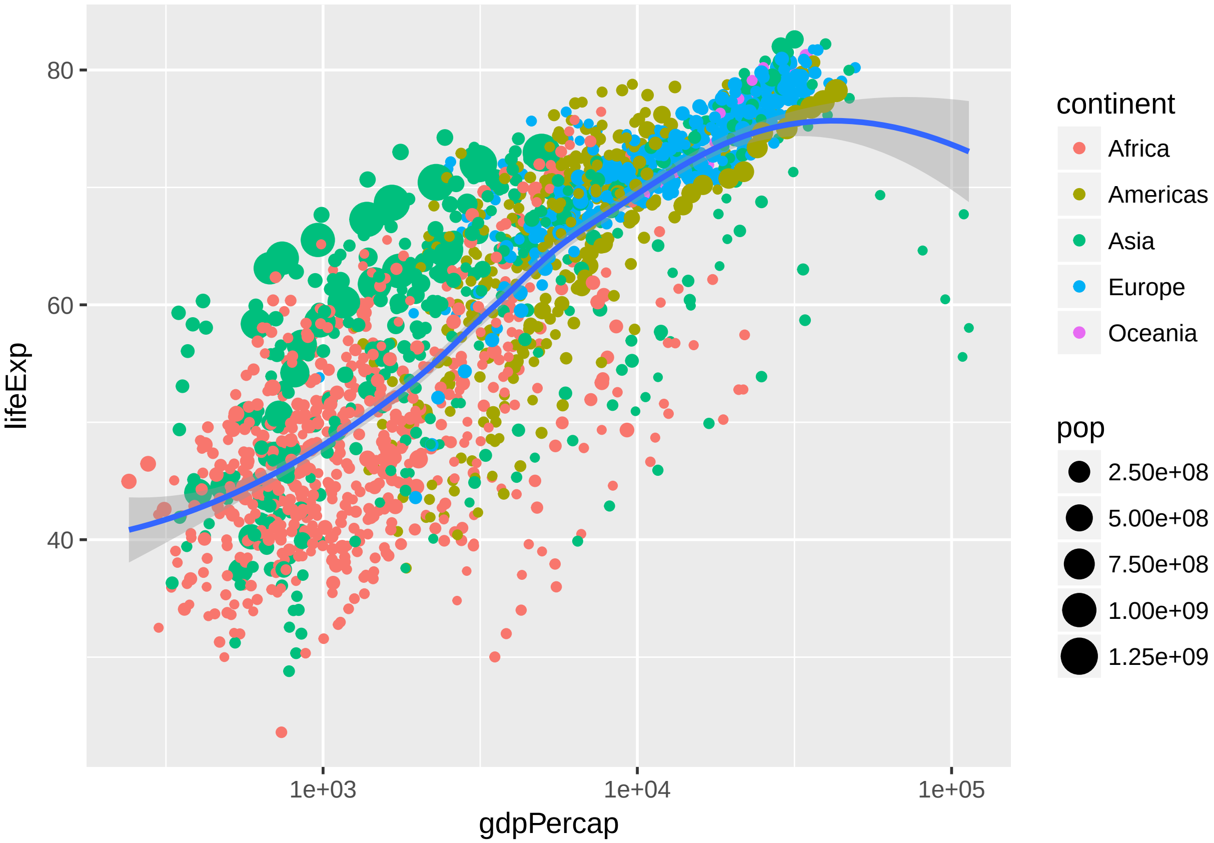
Now lets try to map it reasonably:

```
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()
```



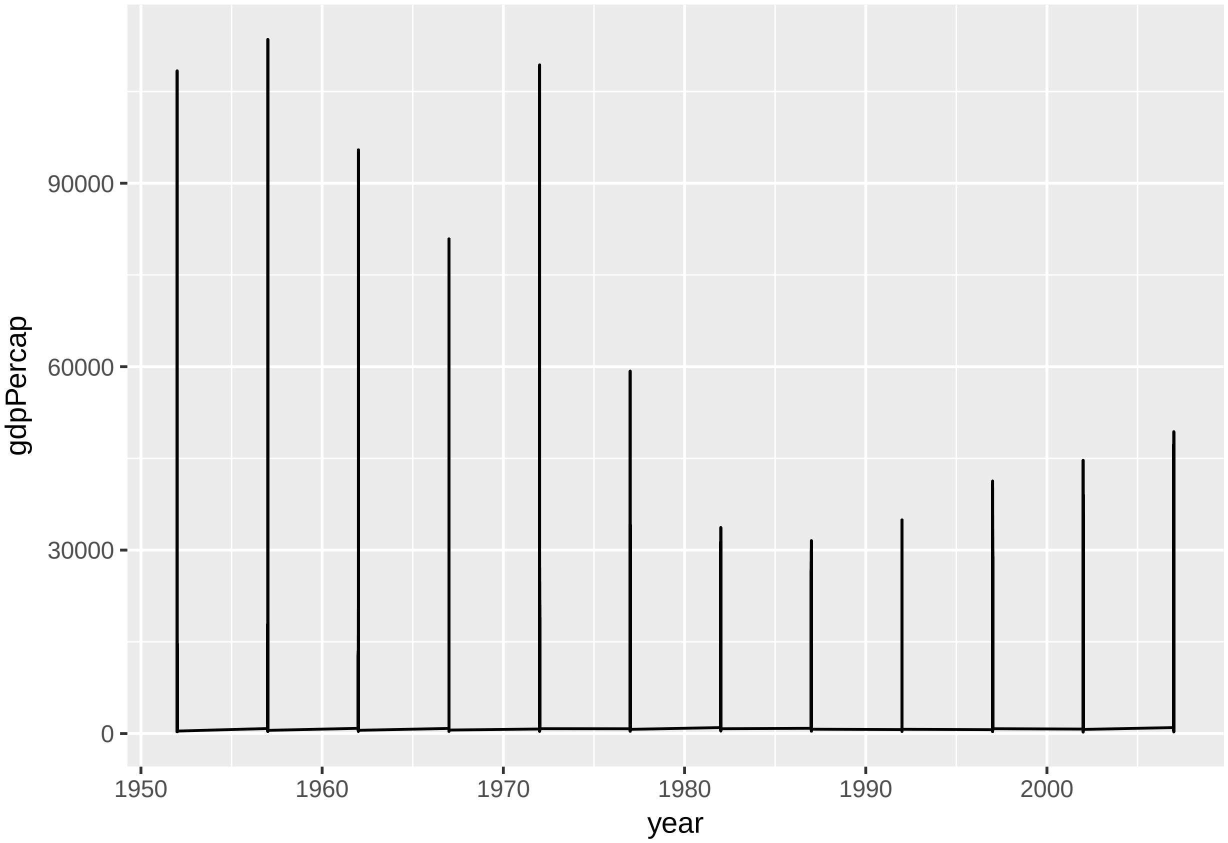
So both mappings can be set on both a base and layer basis.

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

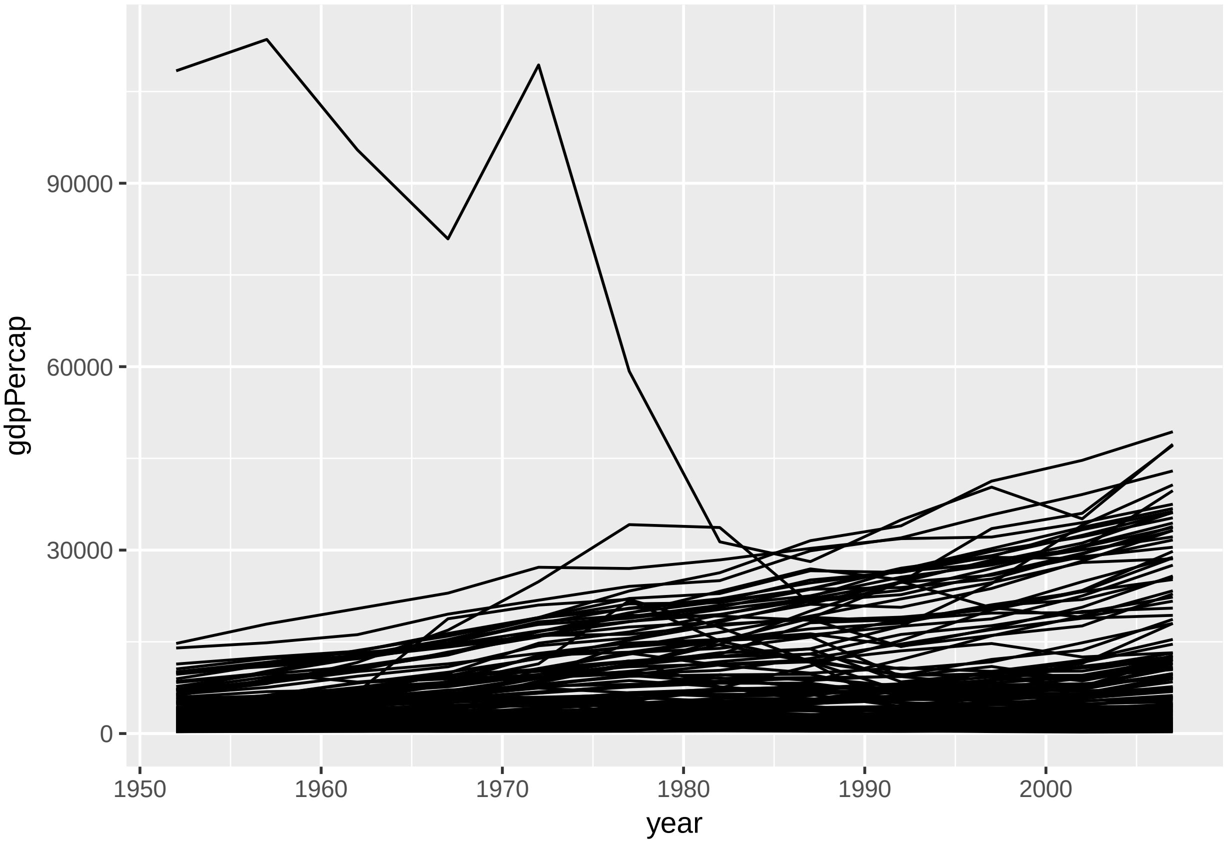
```
head(gapminder)
```

```
## # 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.

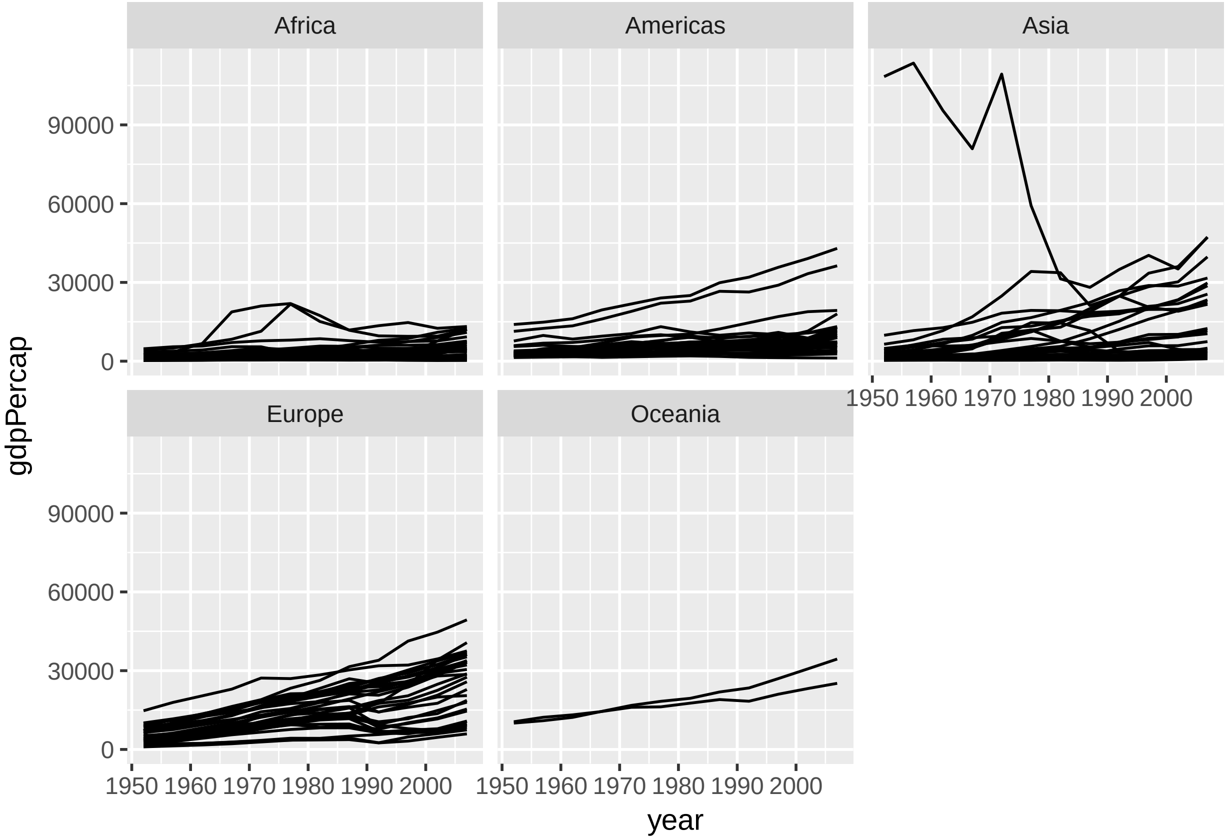
Can be told though:

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



Faceting your geoms

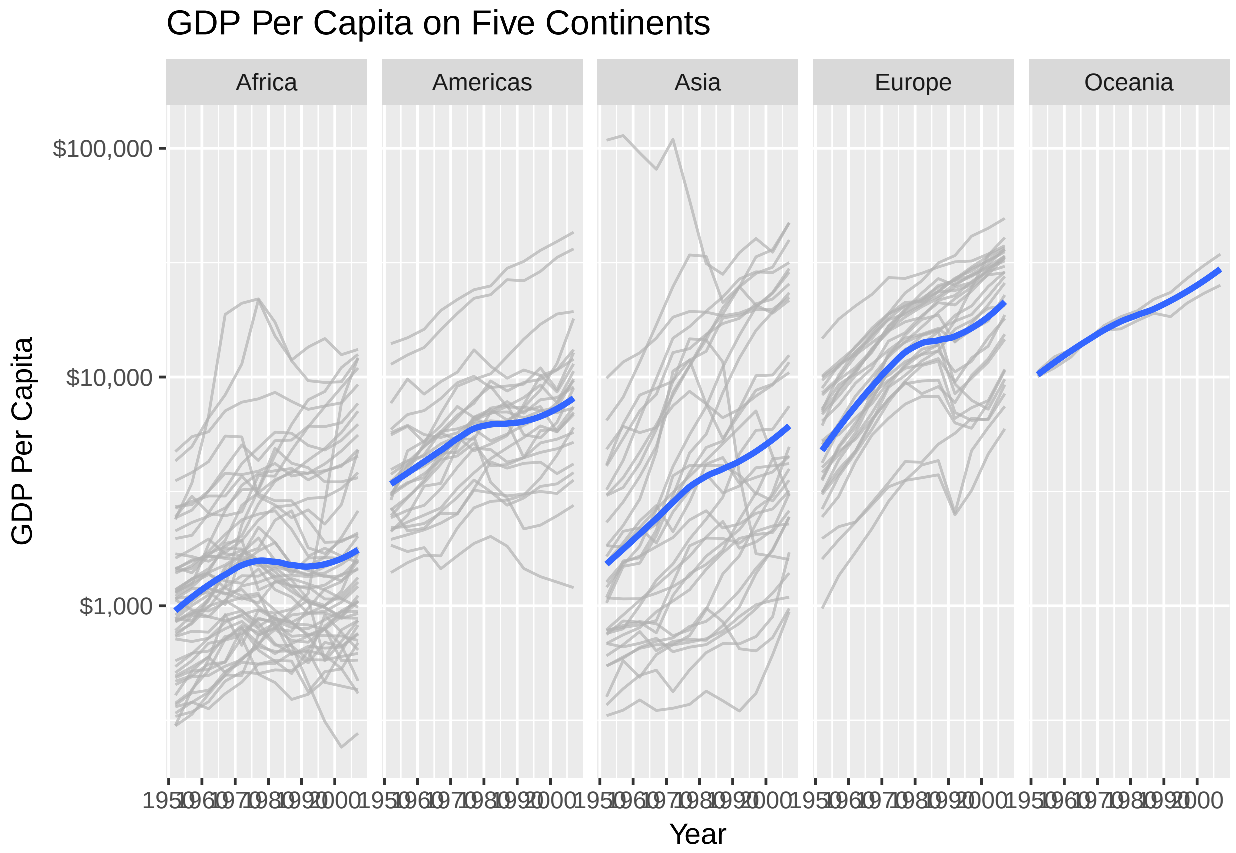
```
p <- ggplot(data = gapminder,
            mapping = aes(x = year,
                          y = gdpPercap))
p + geom_line(mapping = aes(group = country)) +
  facet_wrap(~ continent)
```



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")
```

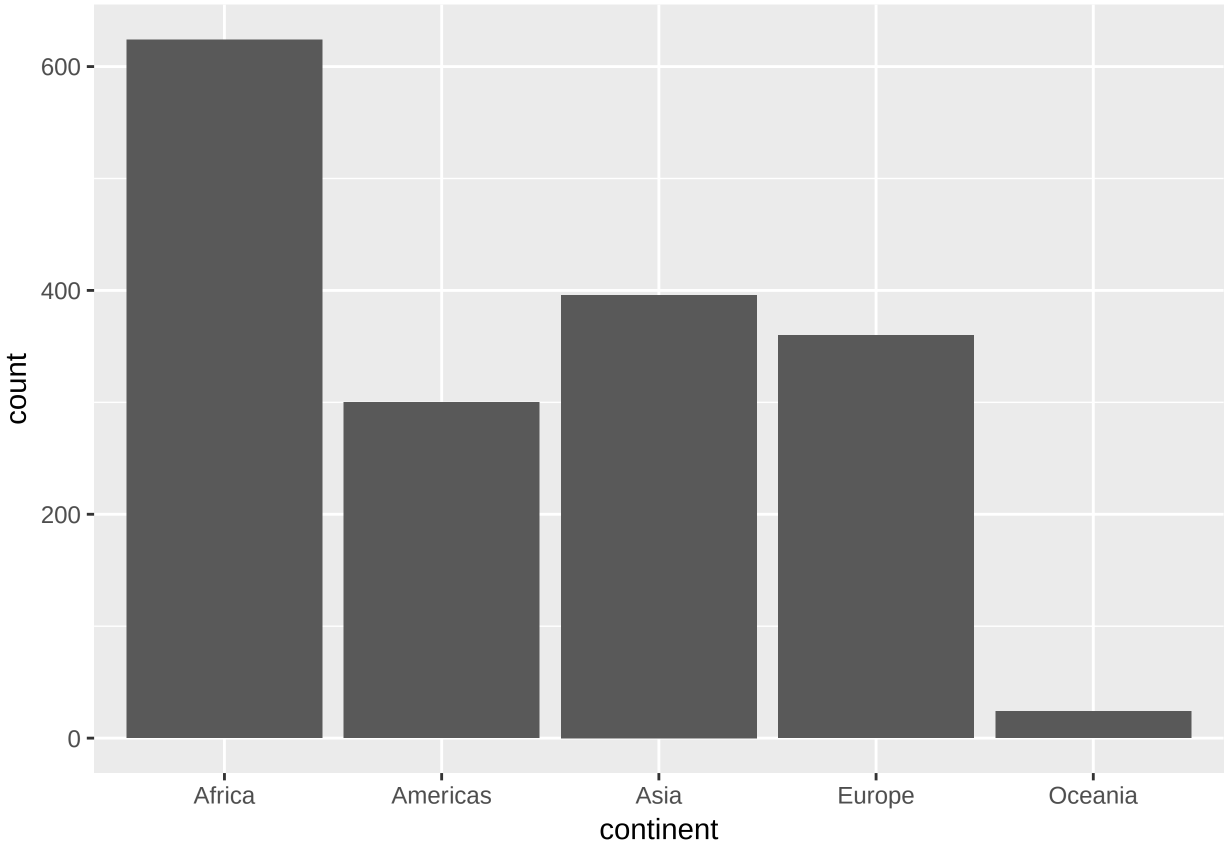


It did a lot of things behind the scenes: Nice logarithmic labels for dollars at 1000 10000 and 100000.

More about stats

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

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

p + geom_bar()
```

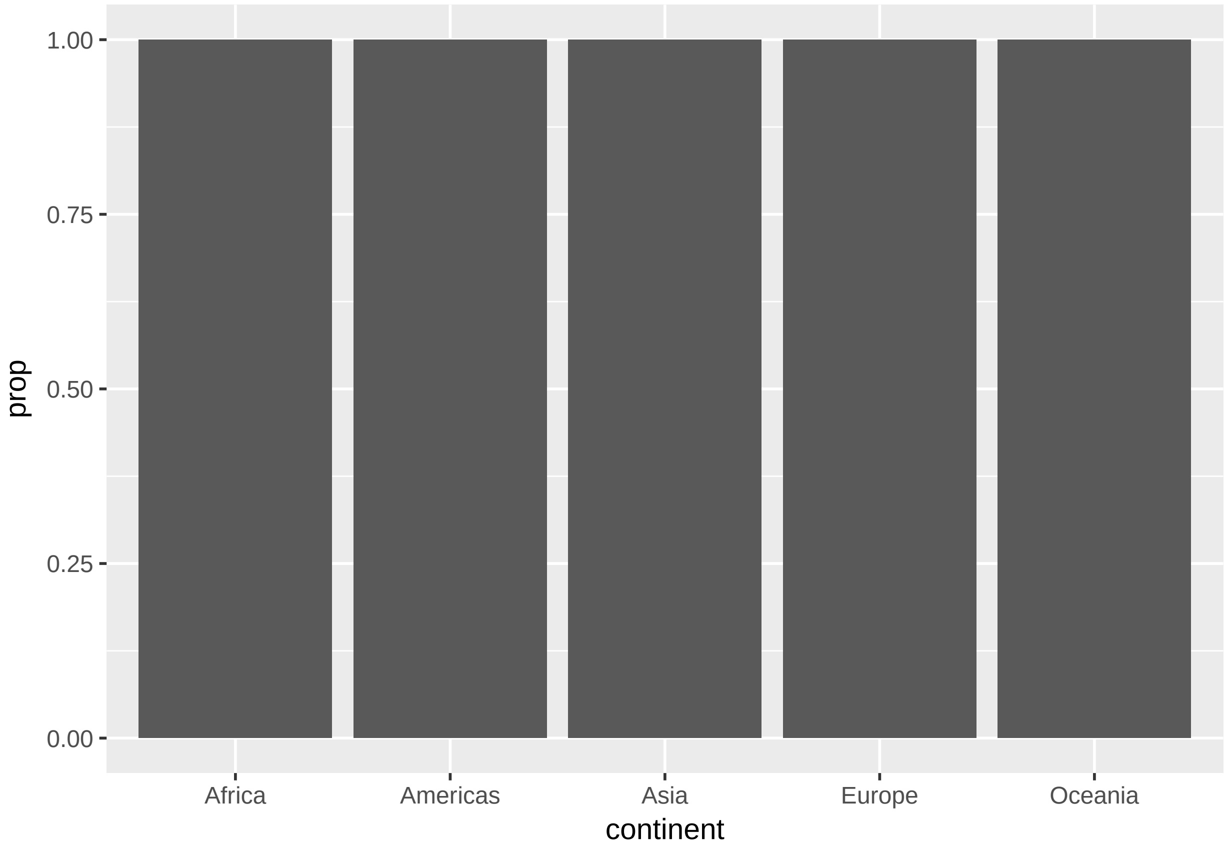


We only specified X, but also got Y. geom_bar calculated the count of observations, to display itself. So this graph lists the total number of country years in the dataset.

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..))
```

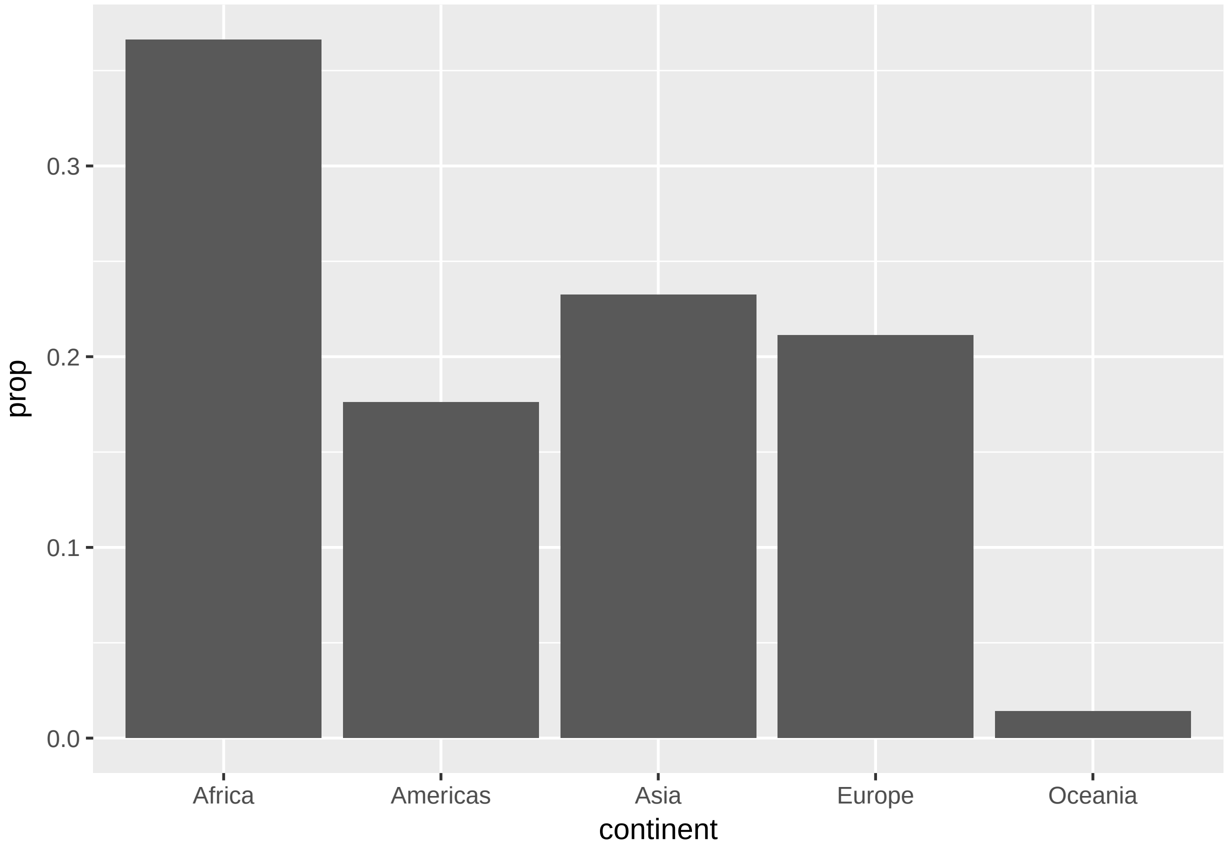


It doesn't do what we want it to, though. Not informative. We infer that this is a grouping issue.

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

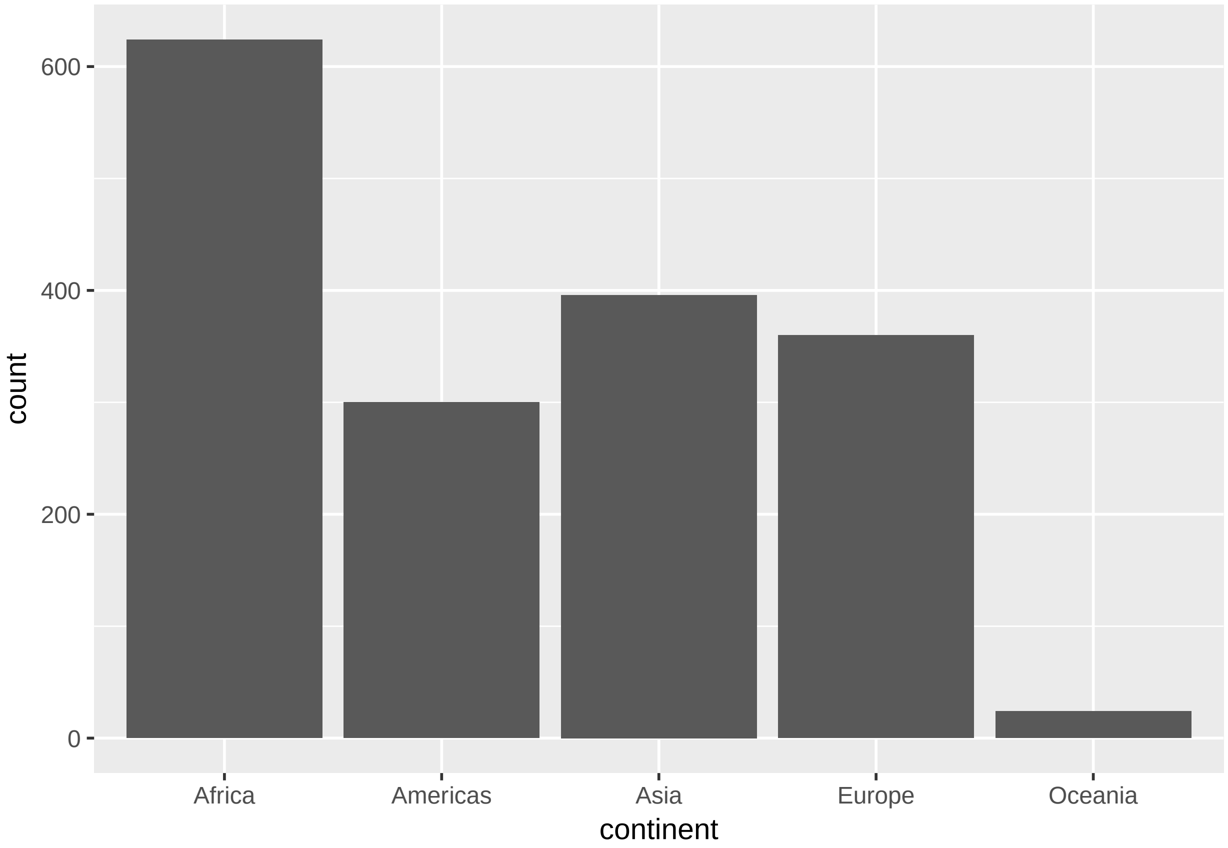
So let's make it sensible, by telling geom_bar to treat the whole dataset as one.

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

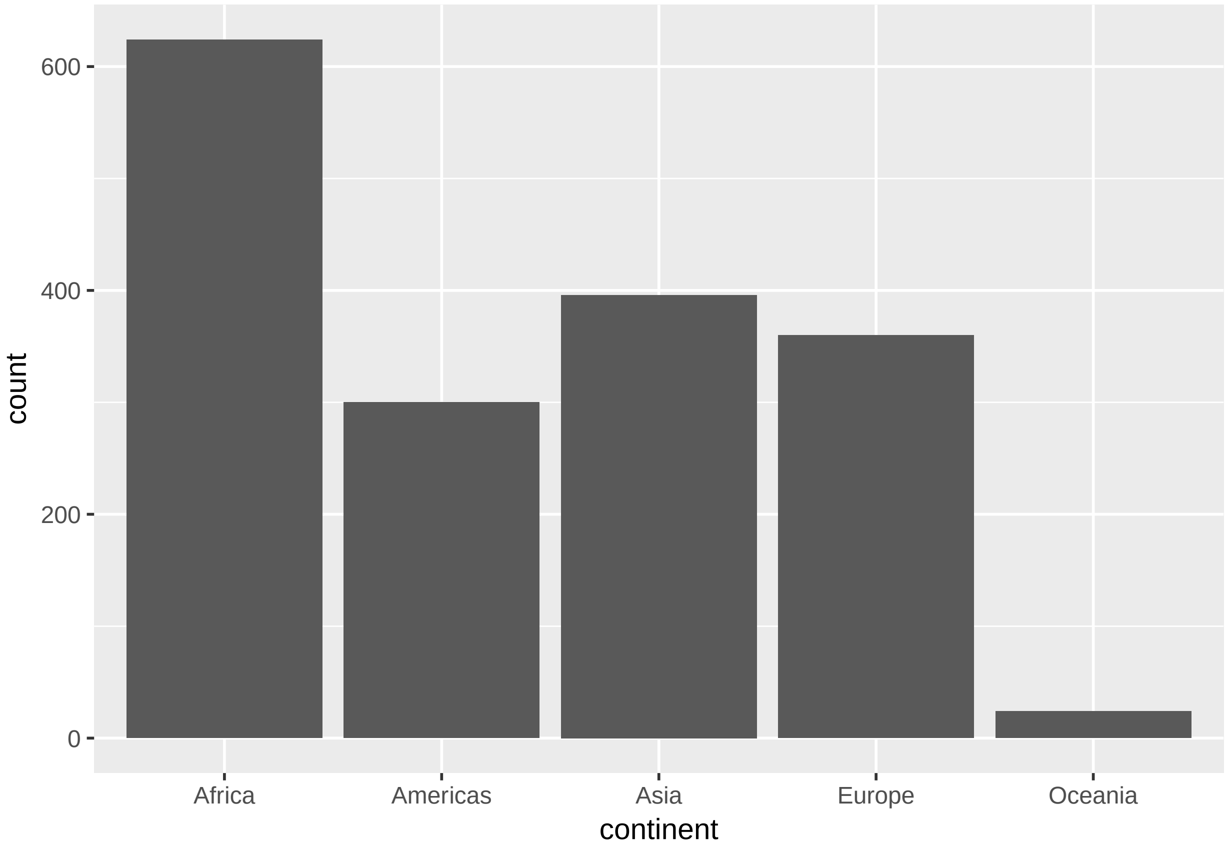


Geom functions call their default stat functions, and vice versa

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

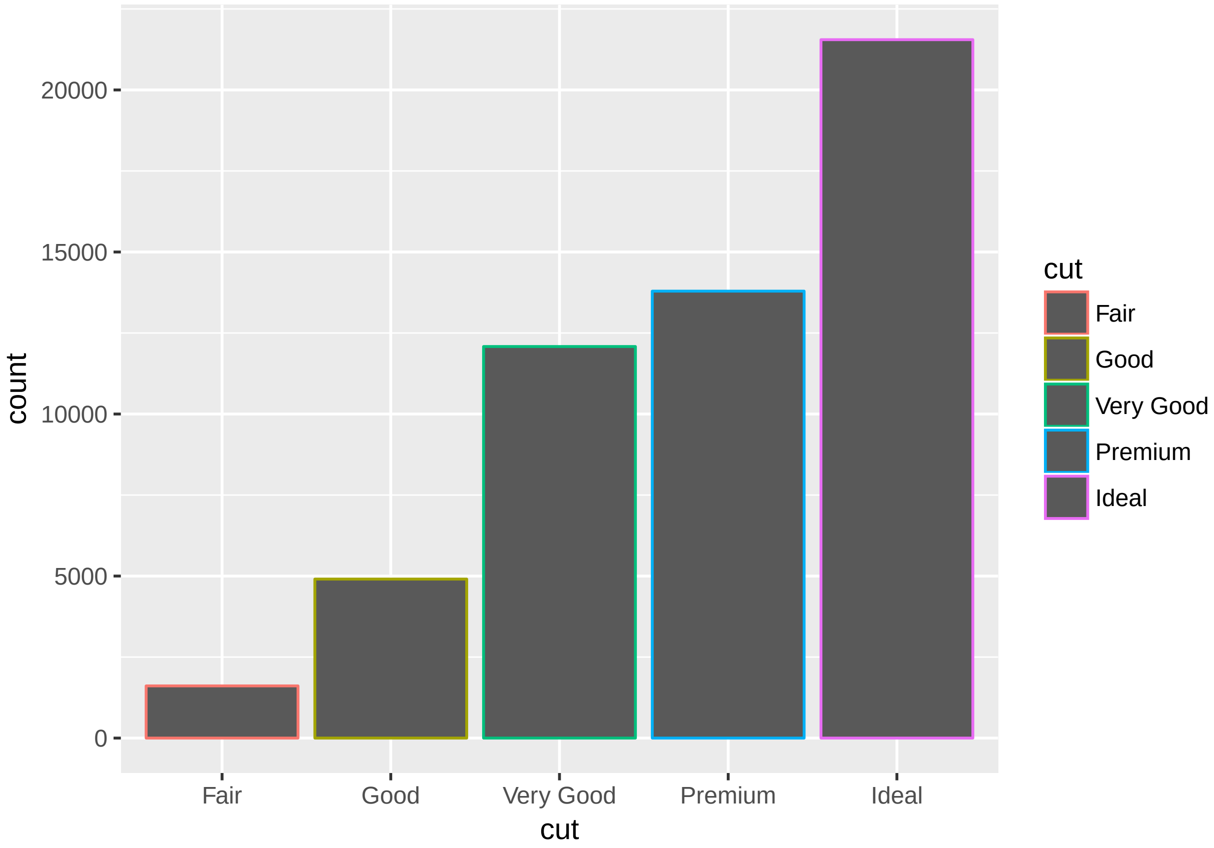


```
# is equal to
p + stat_count()
```



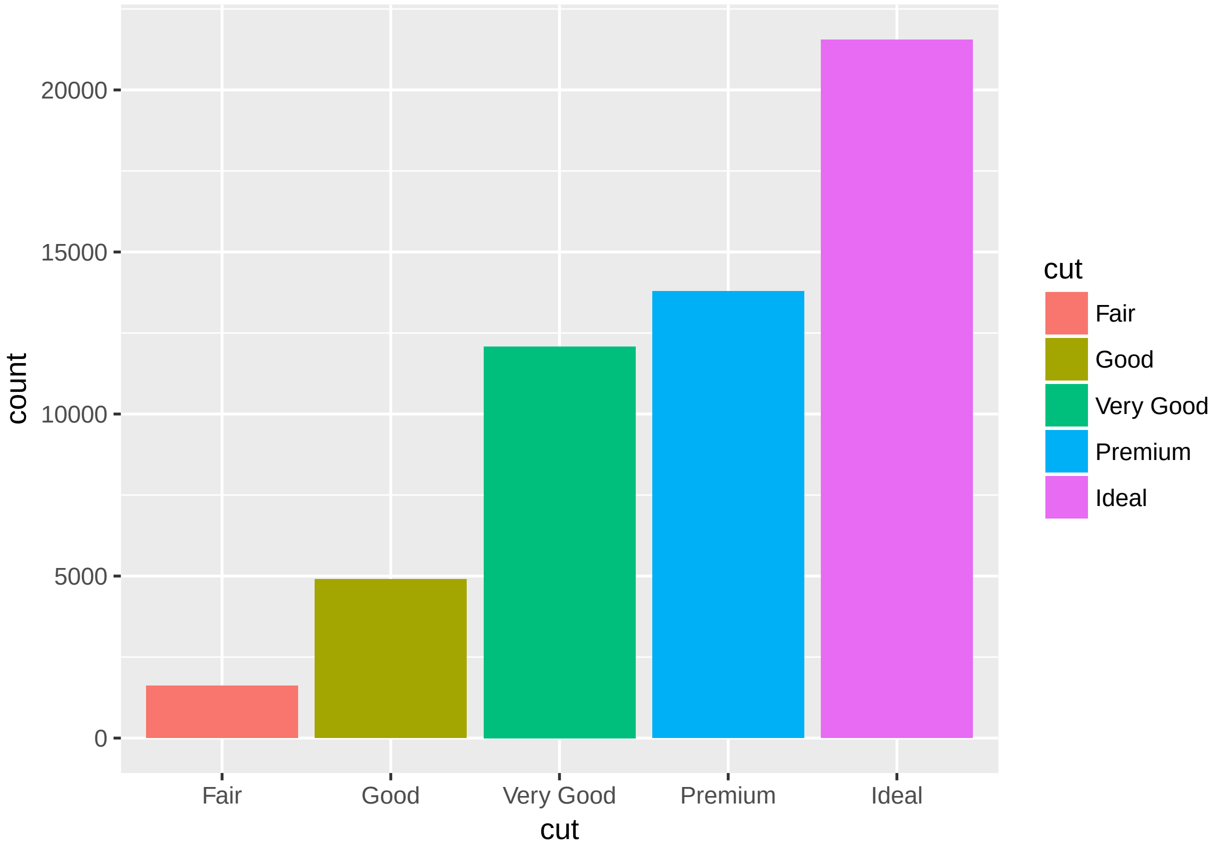
Position adjustments

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



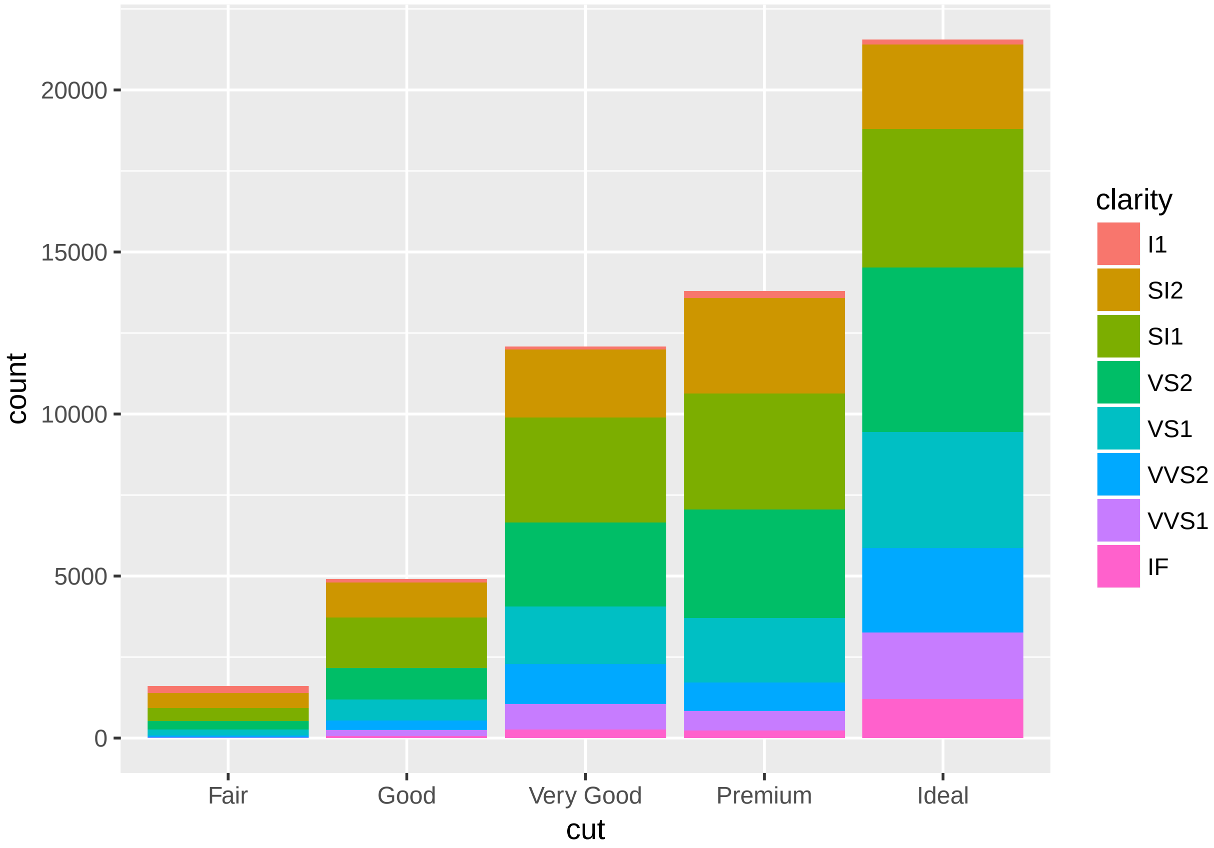
Color gets the outside bar. Fill takes the inside

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



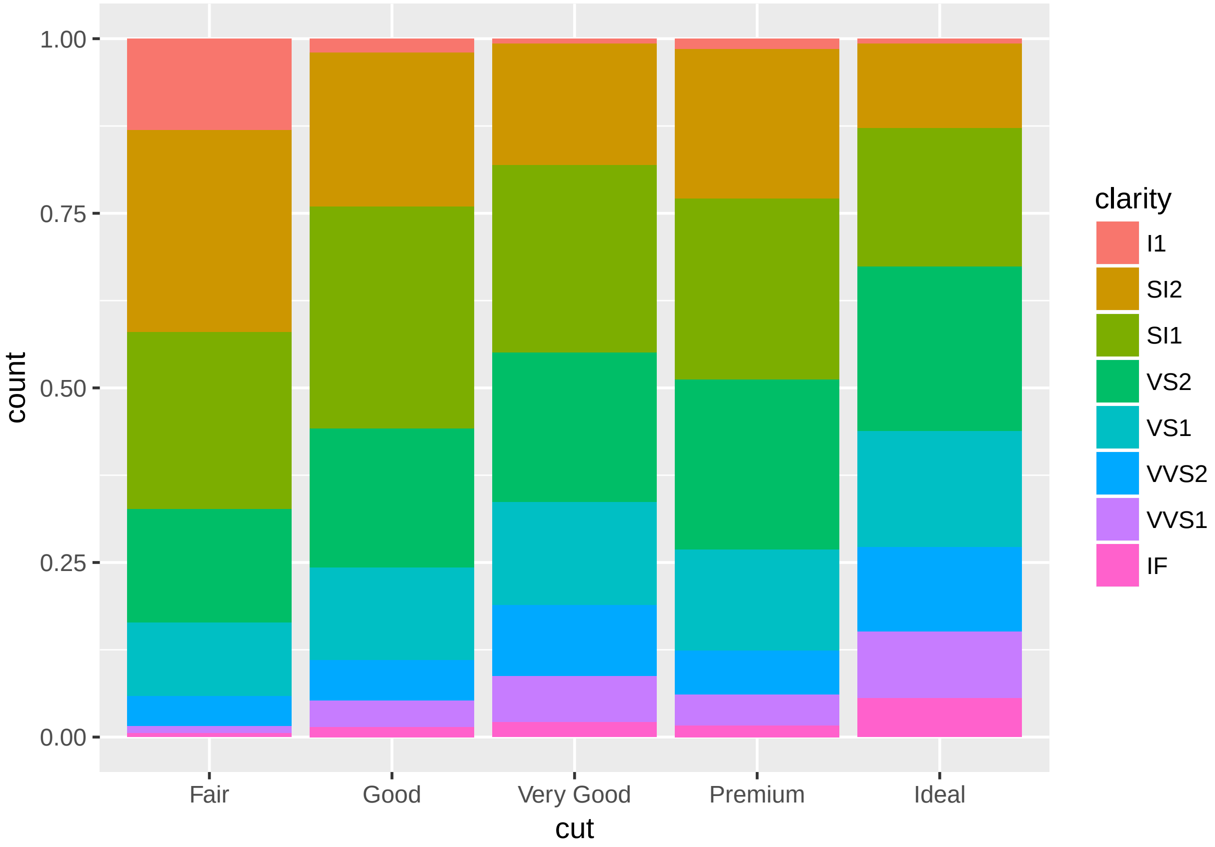
This legend is redundant. Let's cross-classify cut/clarity:

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



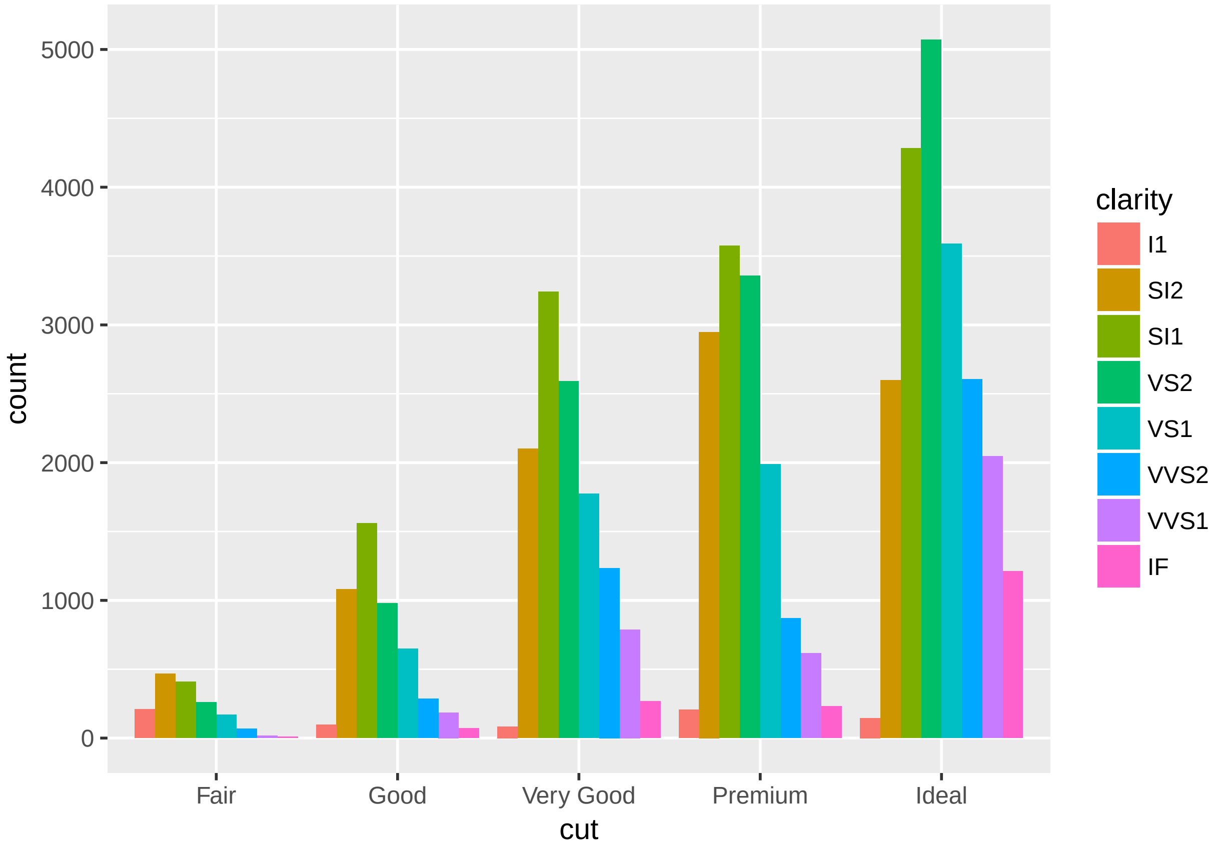
or to make more useful with fill

```
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')
```



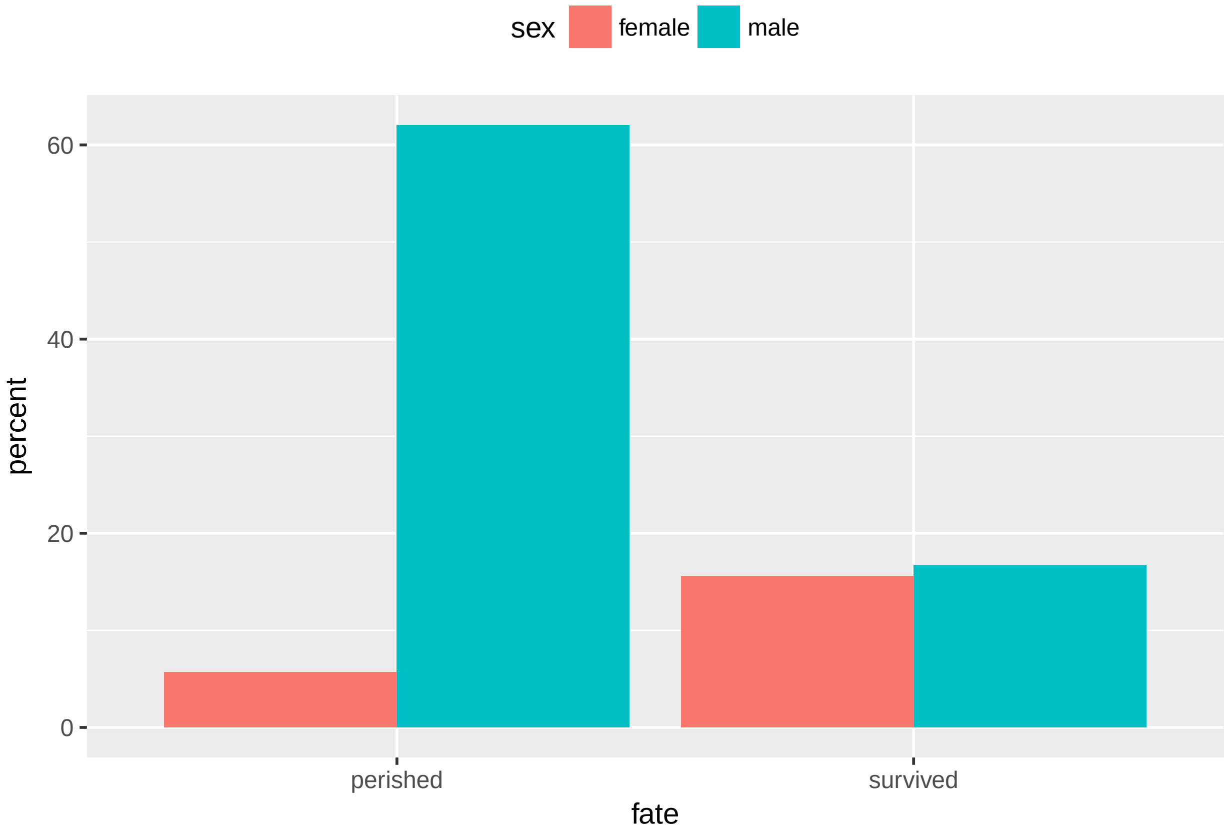
Plotting a finished table

```
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")
```



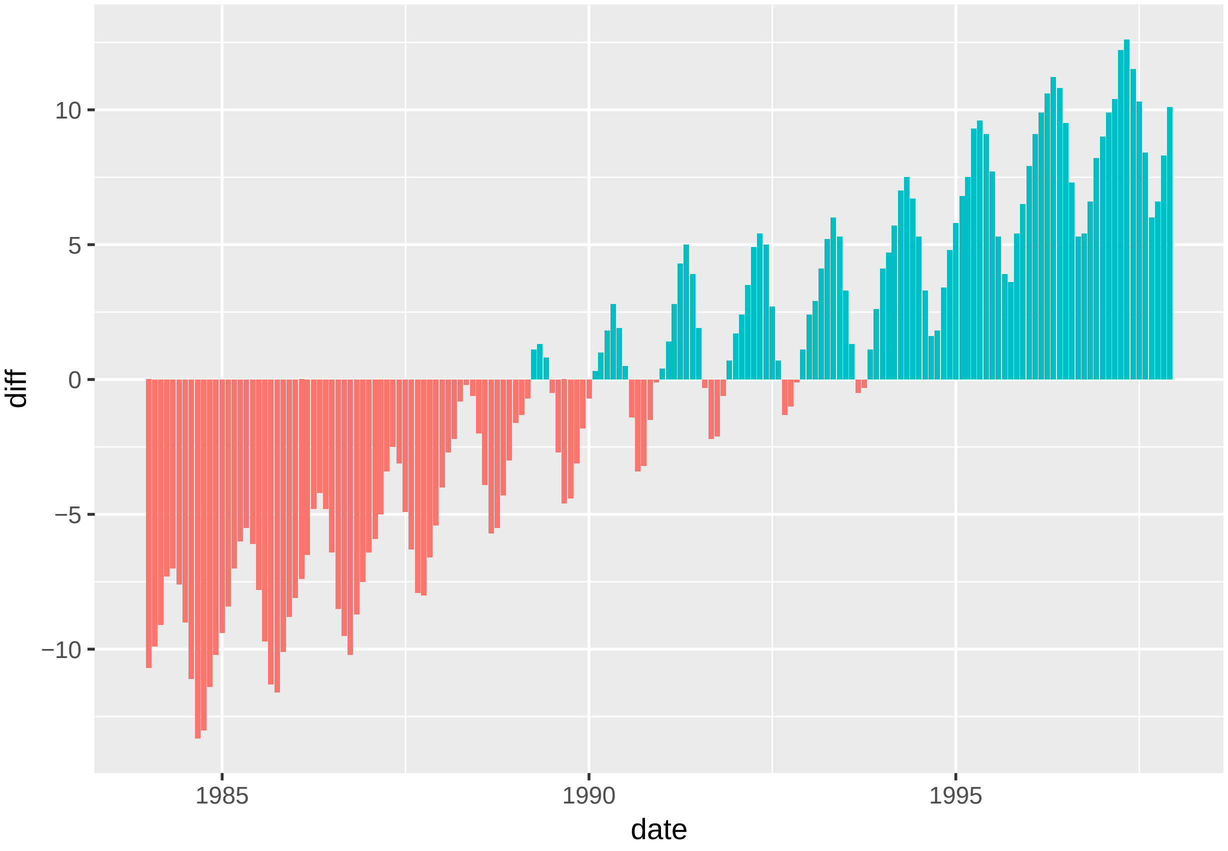
The theme() function controls parts of the plot that don't belong to its "grammatical" structure.

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)
```

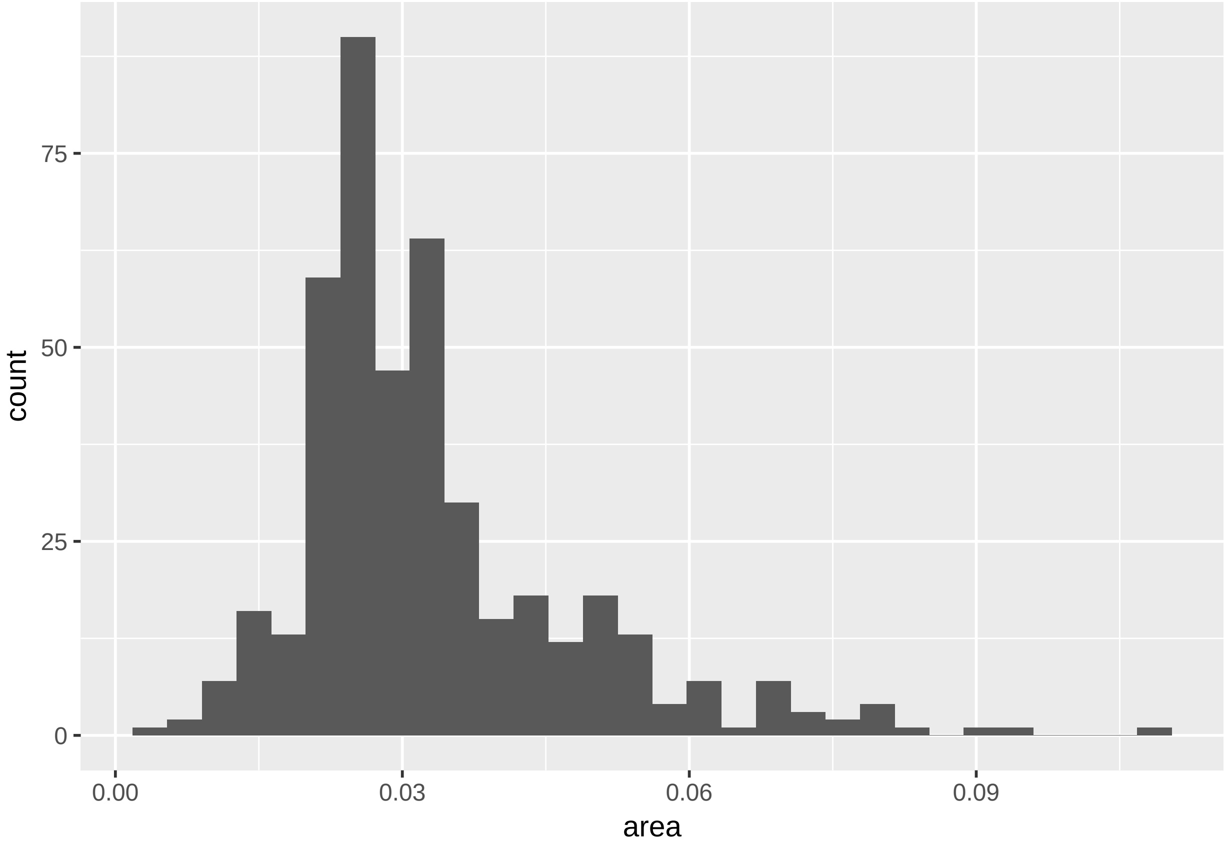
```
## # A tibble: 6 x 28
##     PID    county state  area poptotal popdensity popwhite popblack
##   <int>     <chr> <chr> <dbl>    <int>      <dbl>    <int>    <int>
## 1   561     ADAMS    IL 0.052    66090  1270.9615    63917     1702
## 2   562 ALEXANDER    IL 0.014    10626   759.0000     7054     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))
```

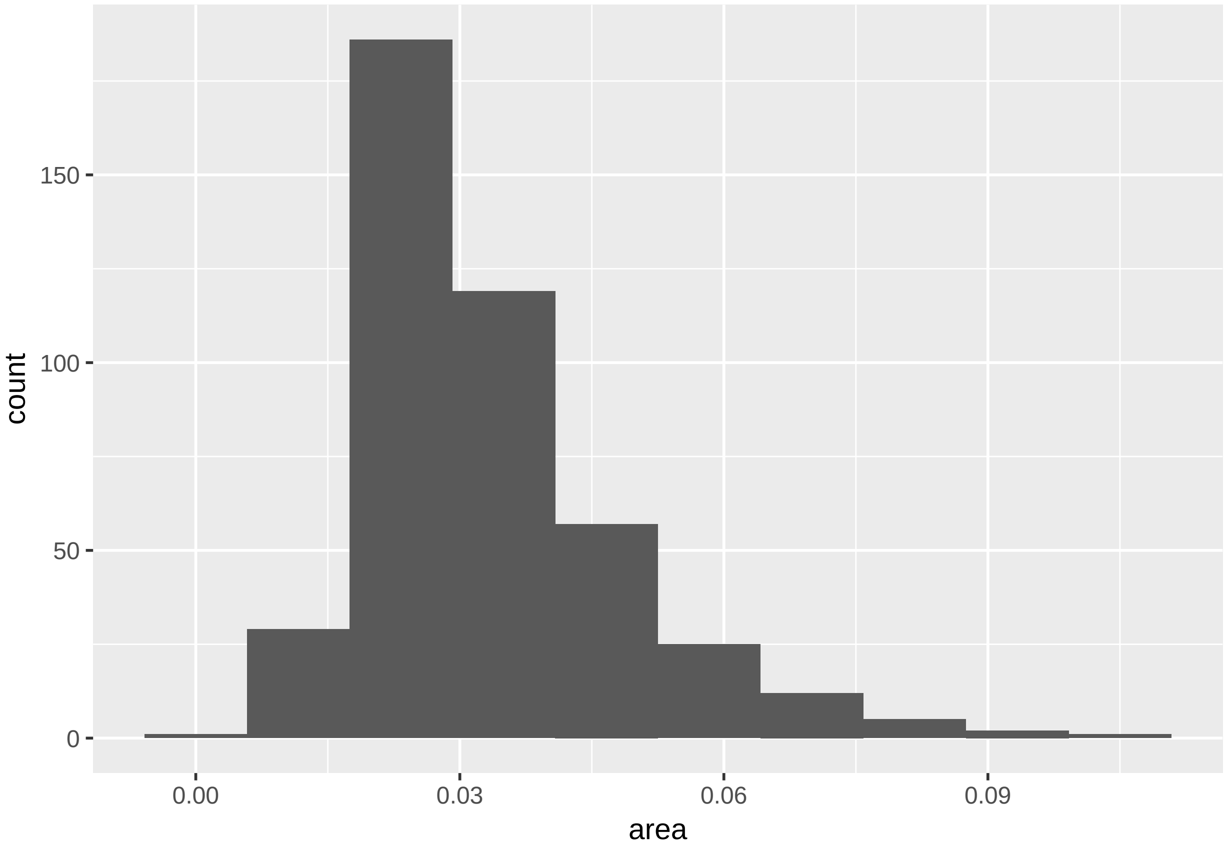
```
p + geom_histogram()
```

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



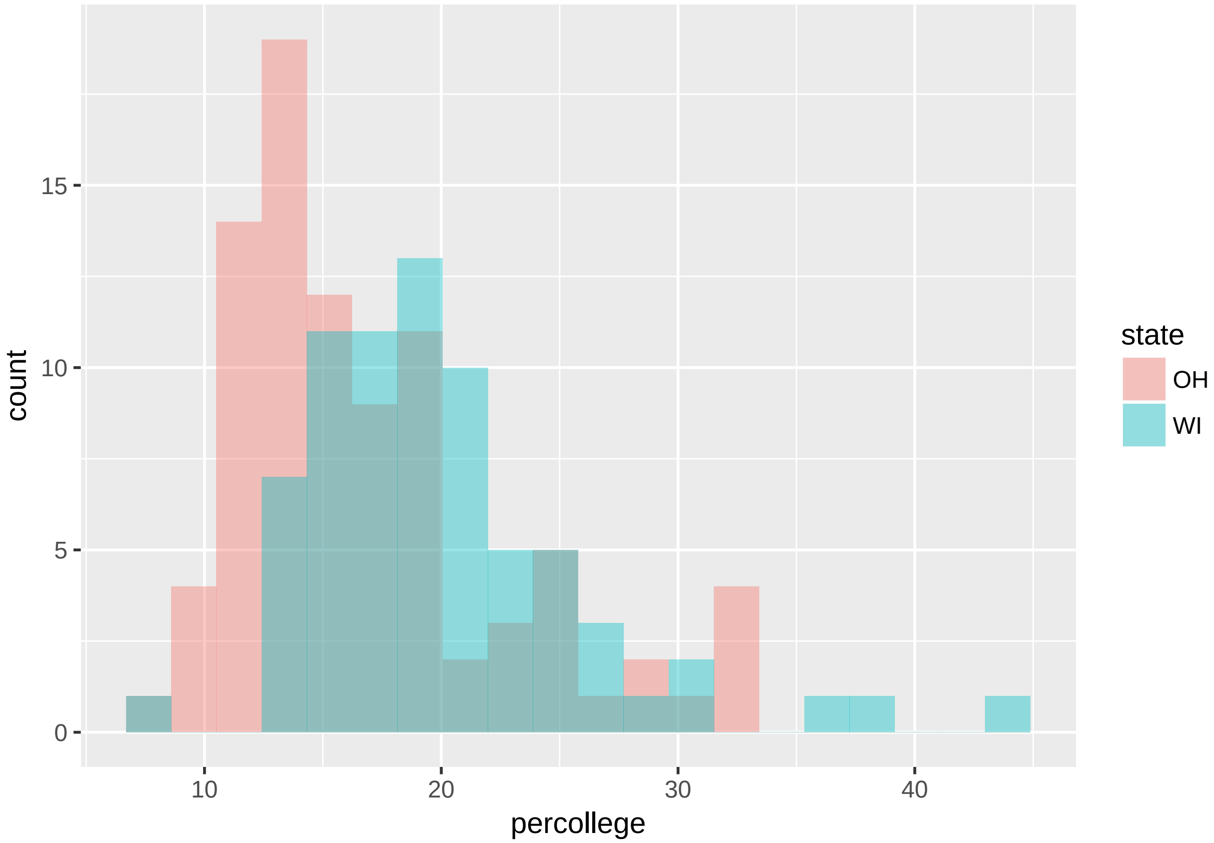
To follow its recommendation, lets set bins to 10 to make it coarser:

```
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)
```

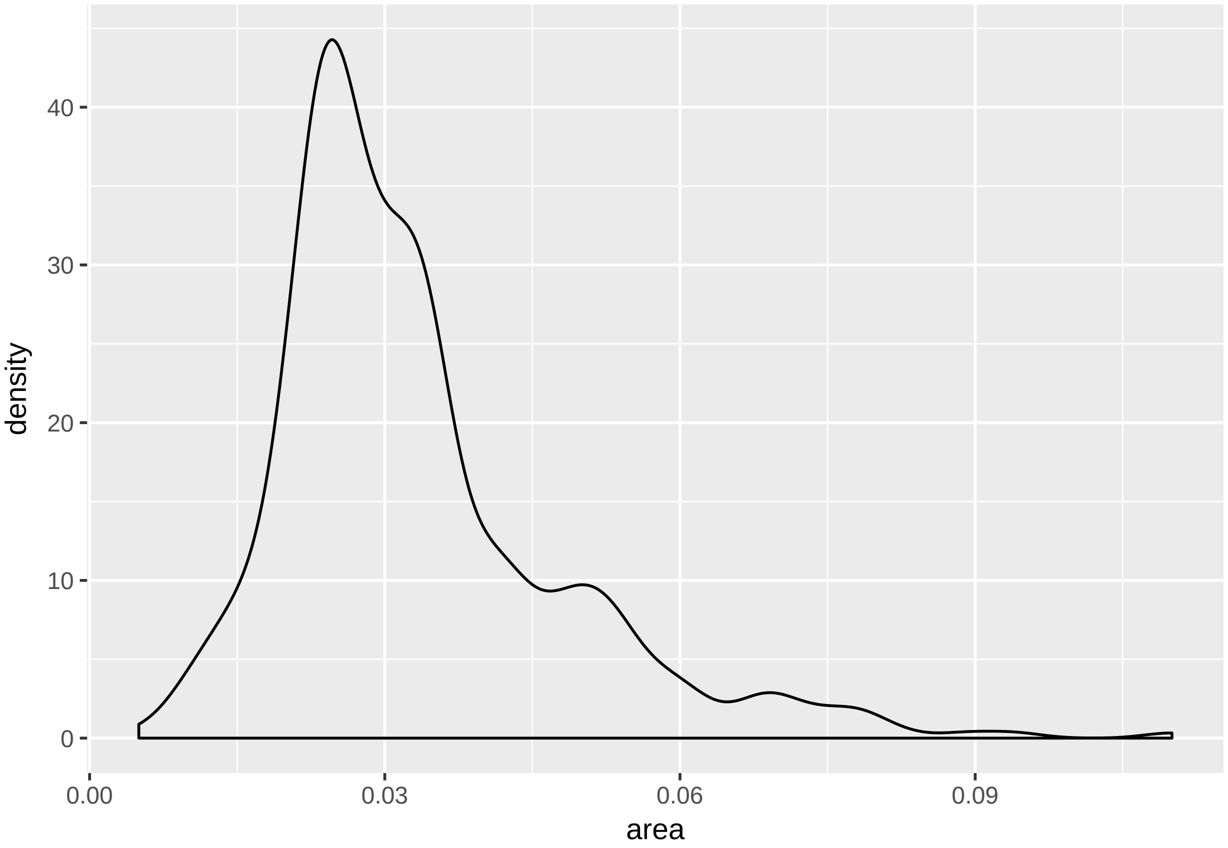


New operator as well %in%. A convinient built in operator that returns a true false value. Asks for a subset in the data WHERE the state is in OH.WI

geom_hist()'s continious counterpart, geom_density()

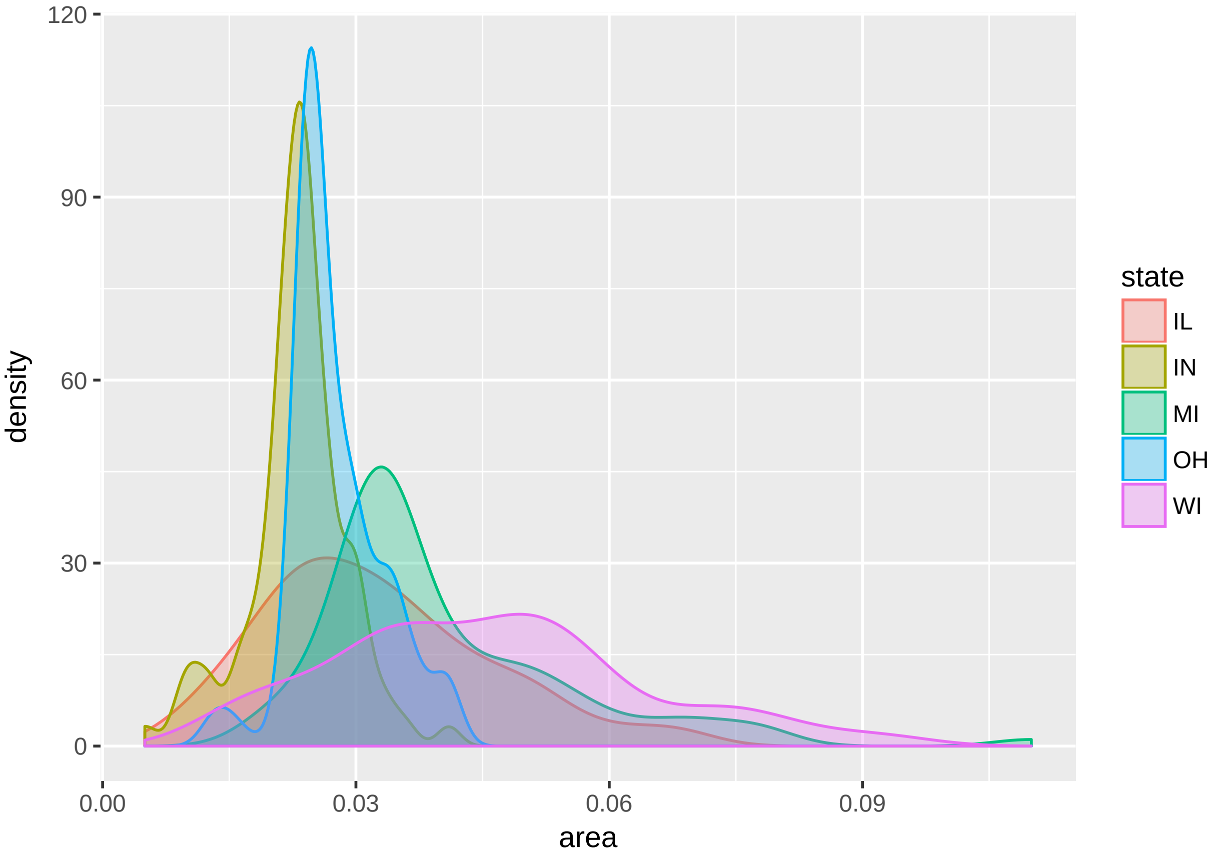
Newer, depend on the availability of quick computers. Estimates kernel density. Think of it as a one dimensional smoother. What sort of continious distribution does this look like?

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



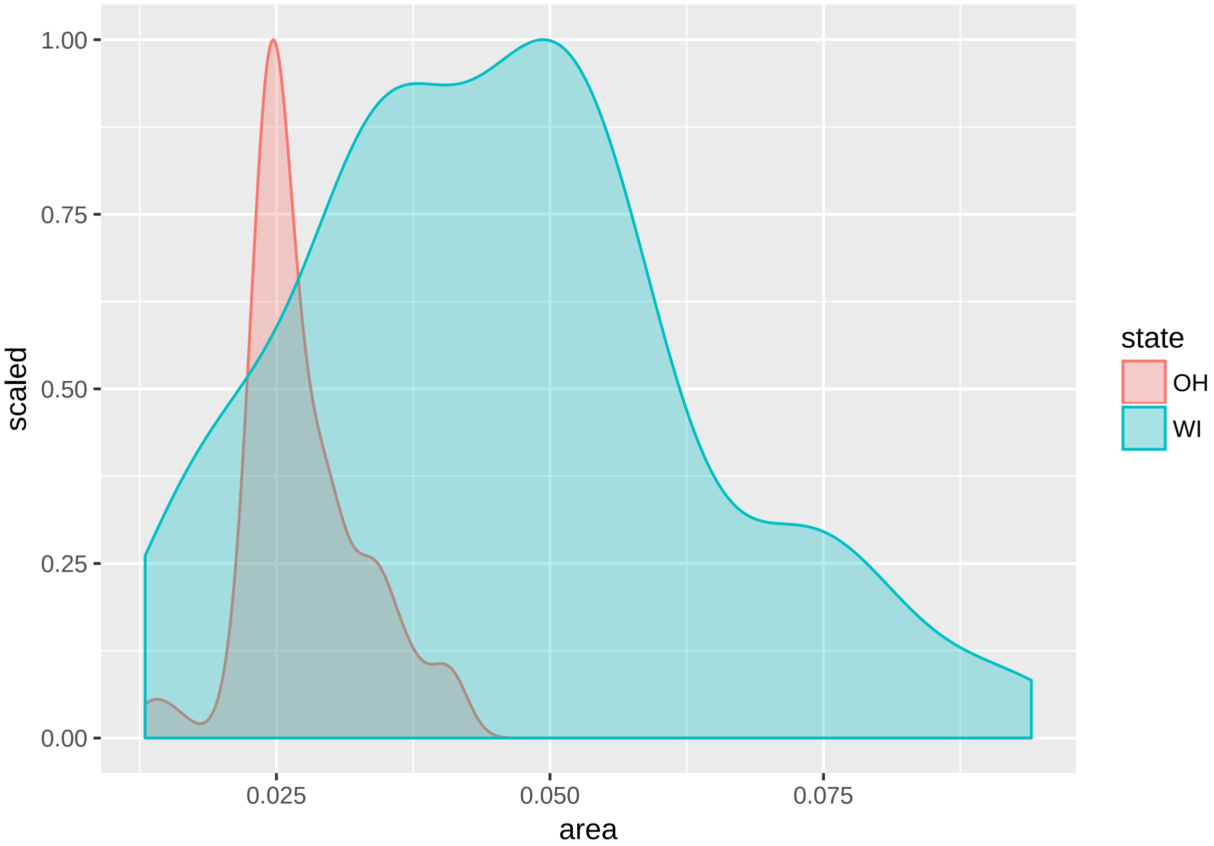
Make it nice:

```
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)
```

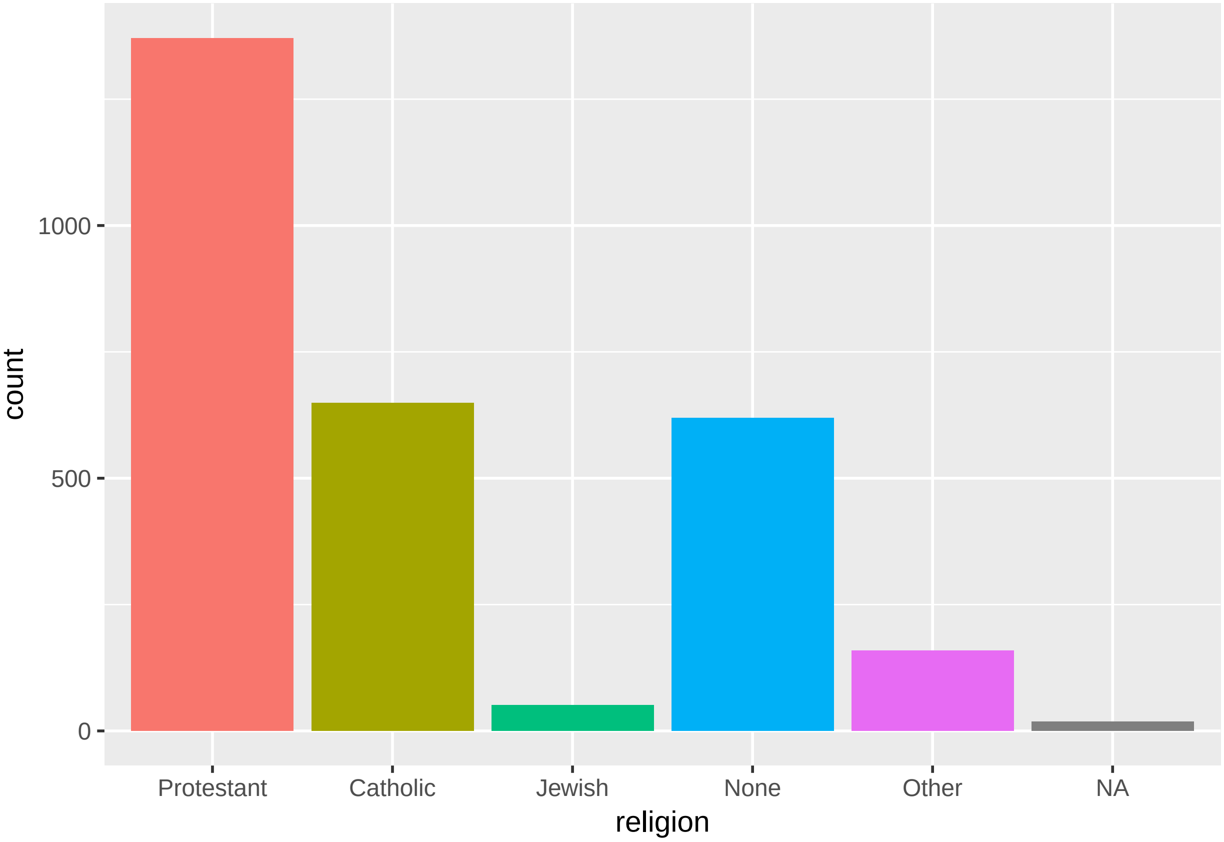
```
## # A tibble: 6 x 32
##    year    id    ballot   age childs      sibs         degree   race
##   <dbl> <dbl> <dbl+lbl> <dbl>  <dbl> <dbl+lbl>         <fctr> <fctr>
## 1  2016     1          1    47      3          2       Bachelor  White
## 2  2016     2          2    61      0          3    High 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
```

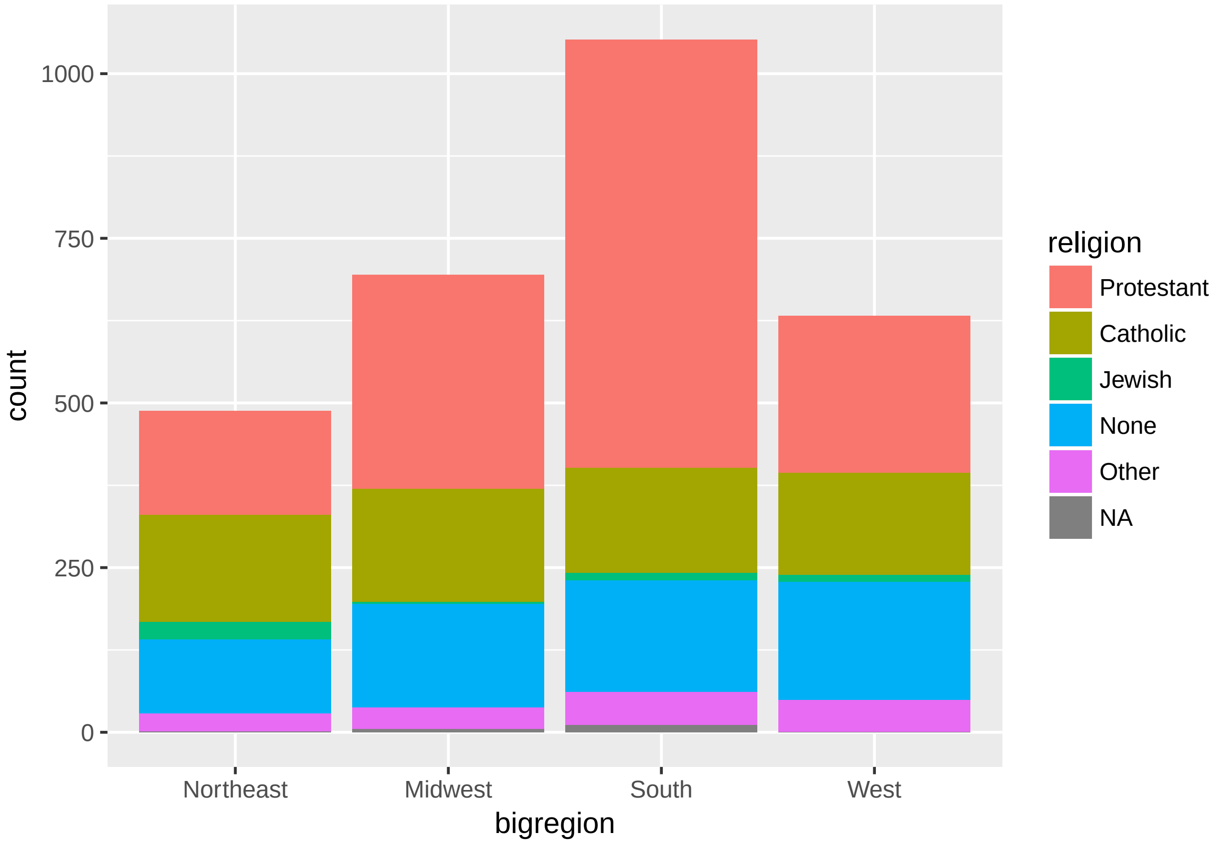
```
p <- ggplot(data = gss_sm,
            aes(religion, fill = religion))
p + geom_bar() + guides(fill = FALSE)
```



A

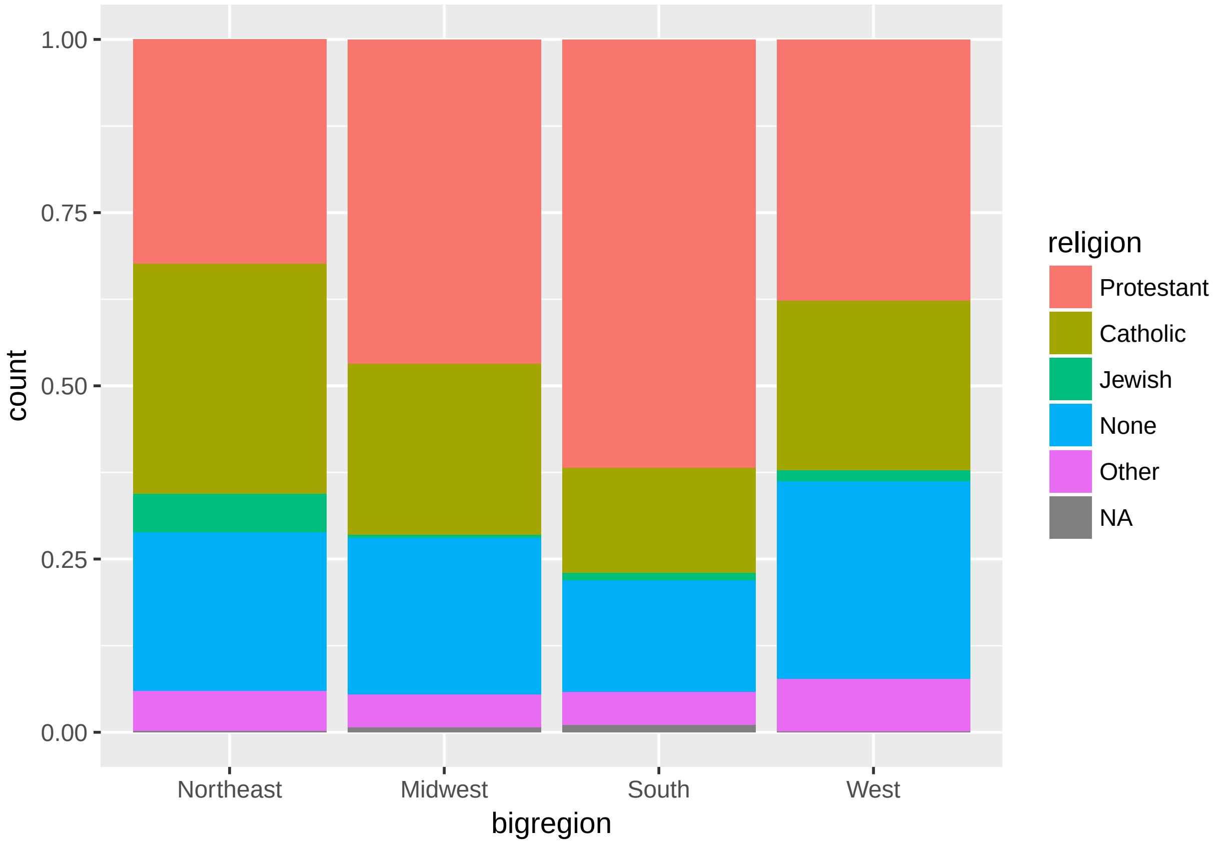
more interesting way to use fill is this:

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



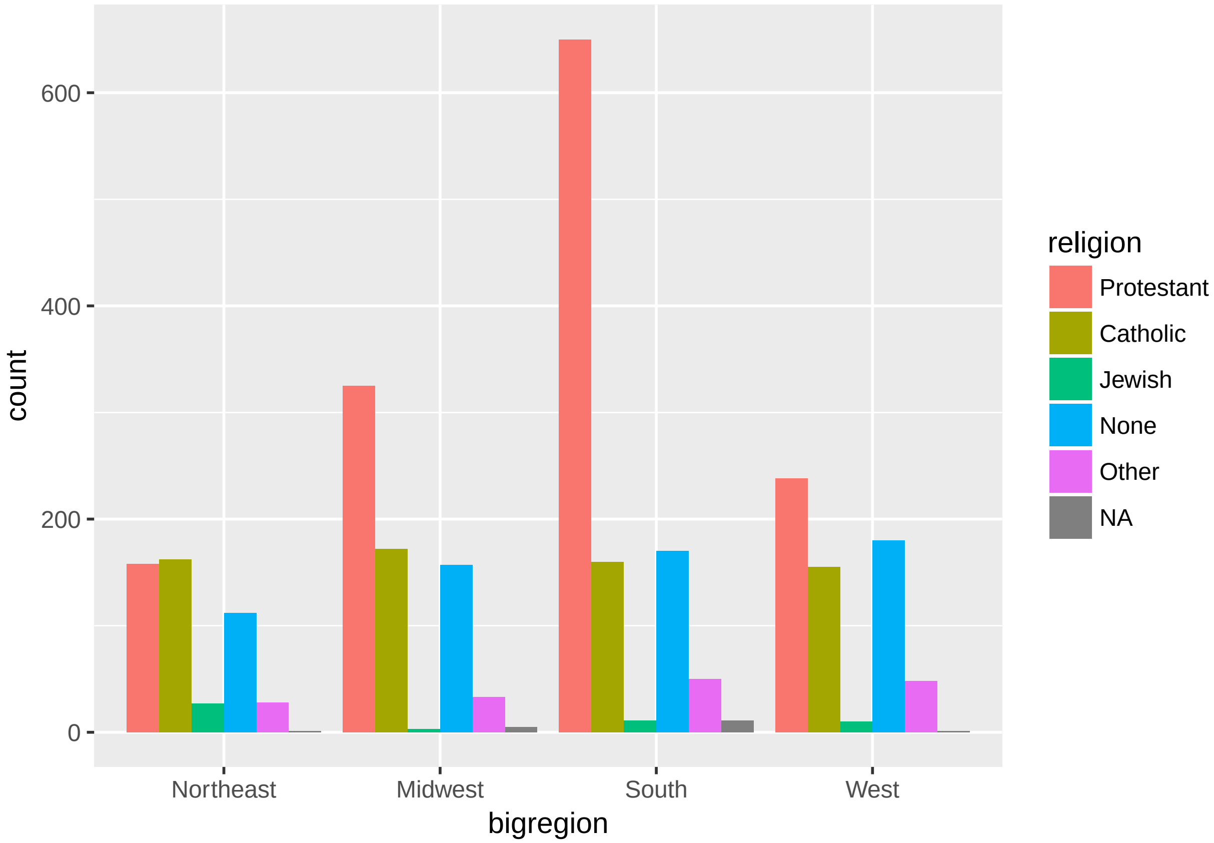
Better to show it like this maybe:

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



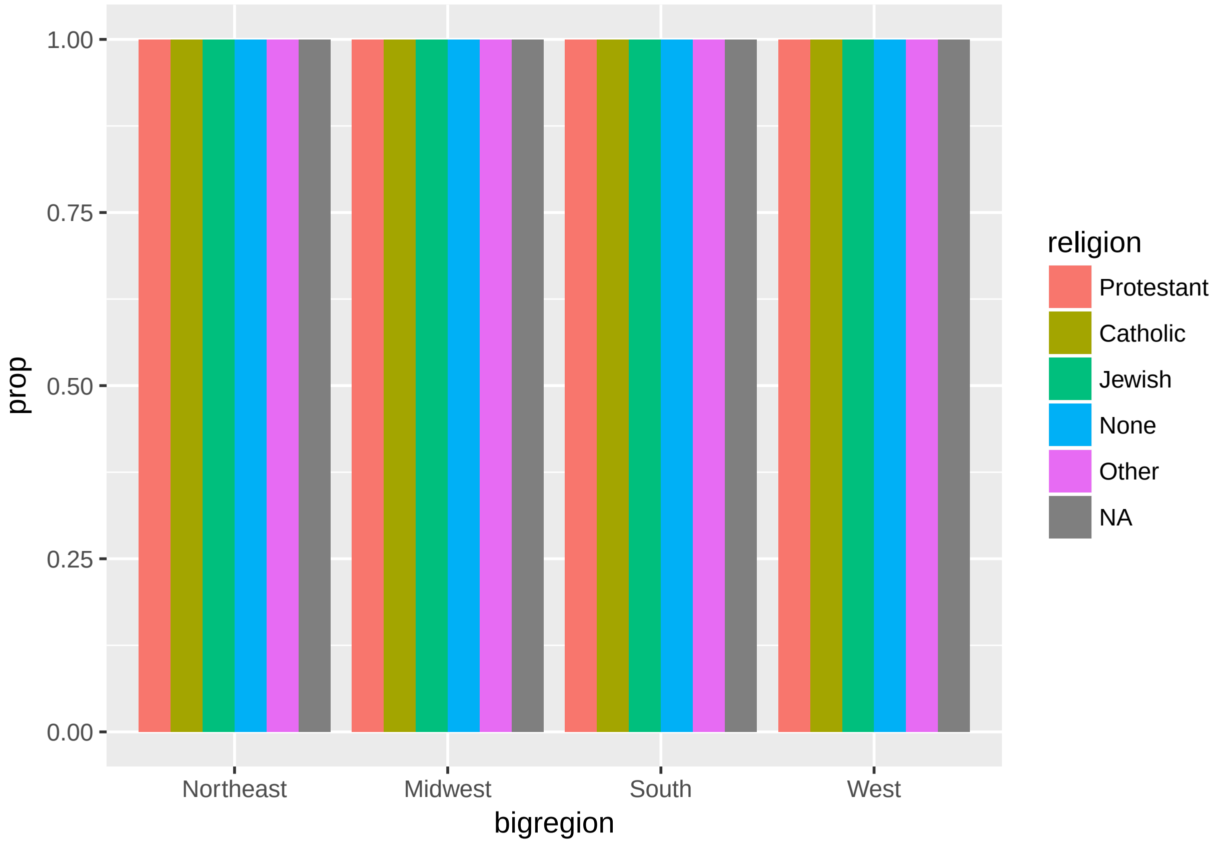
Or maybe with dodge

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



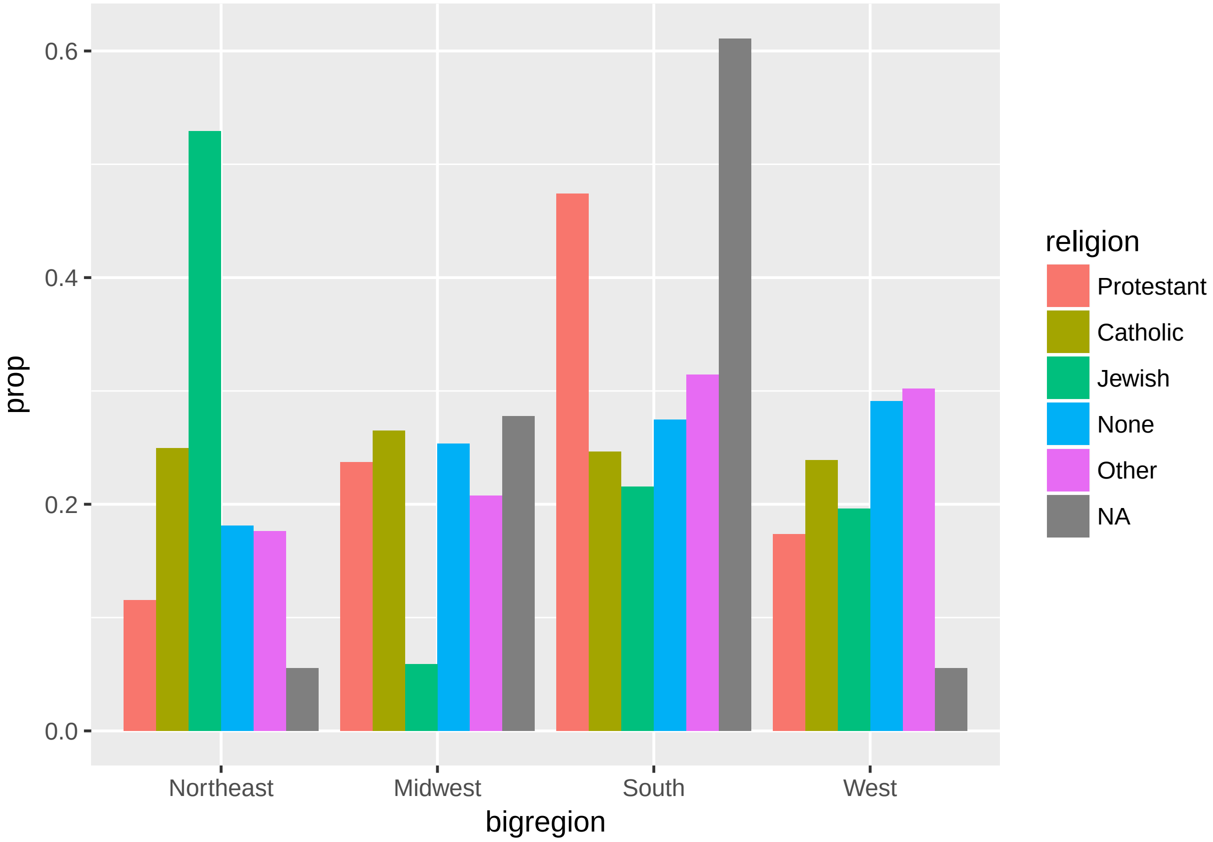
But if we want to properly show proportions within regions. How to?

```
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.

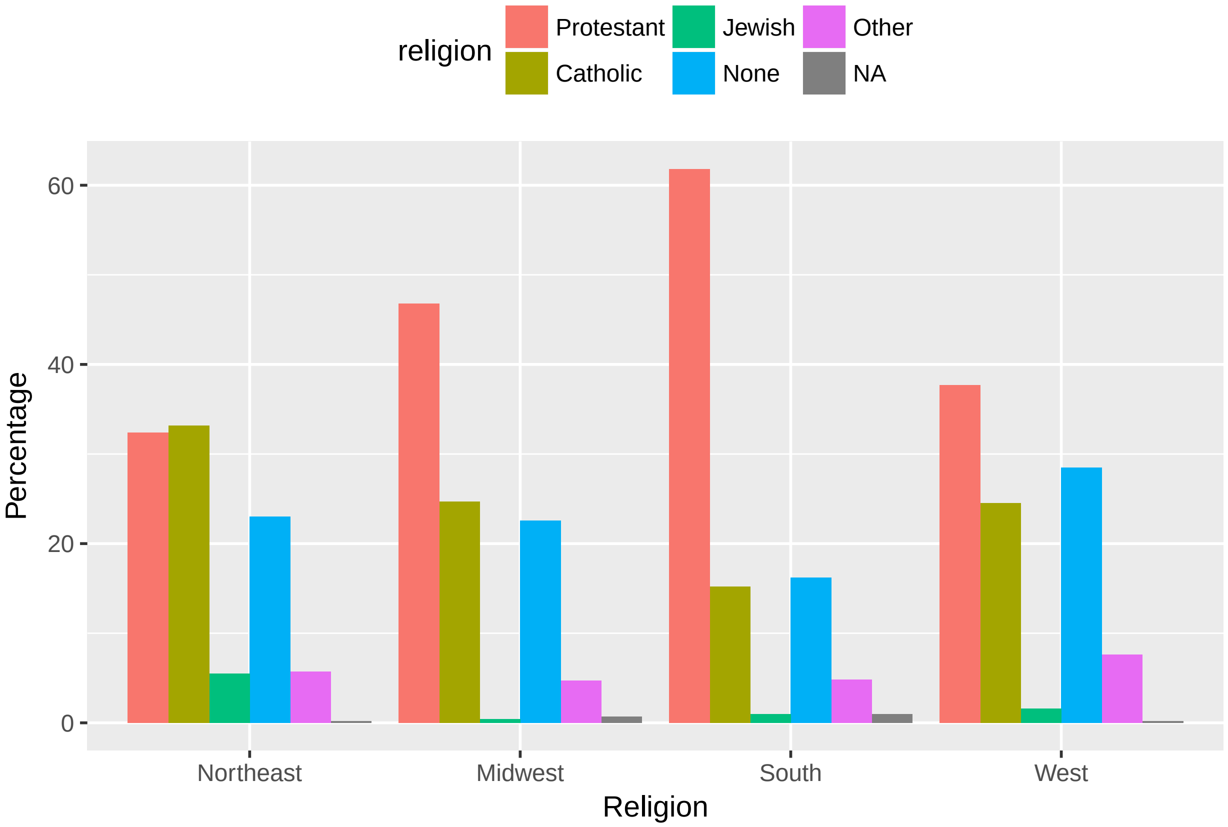
Lets do a sanity check:

```
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
```

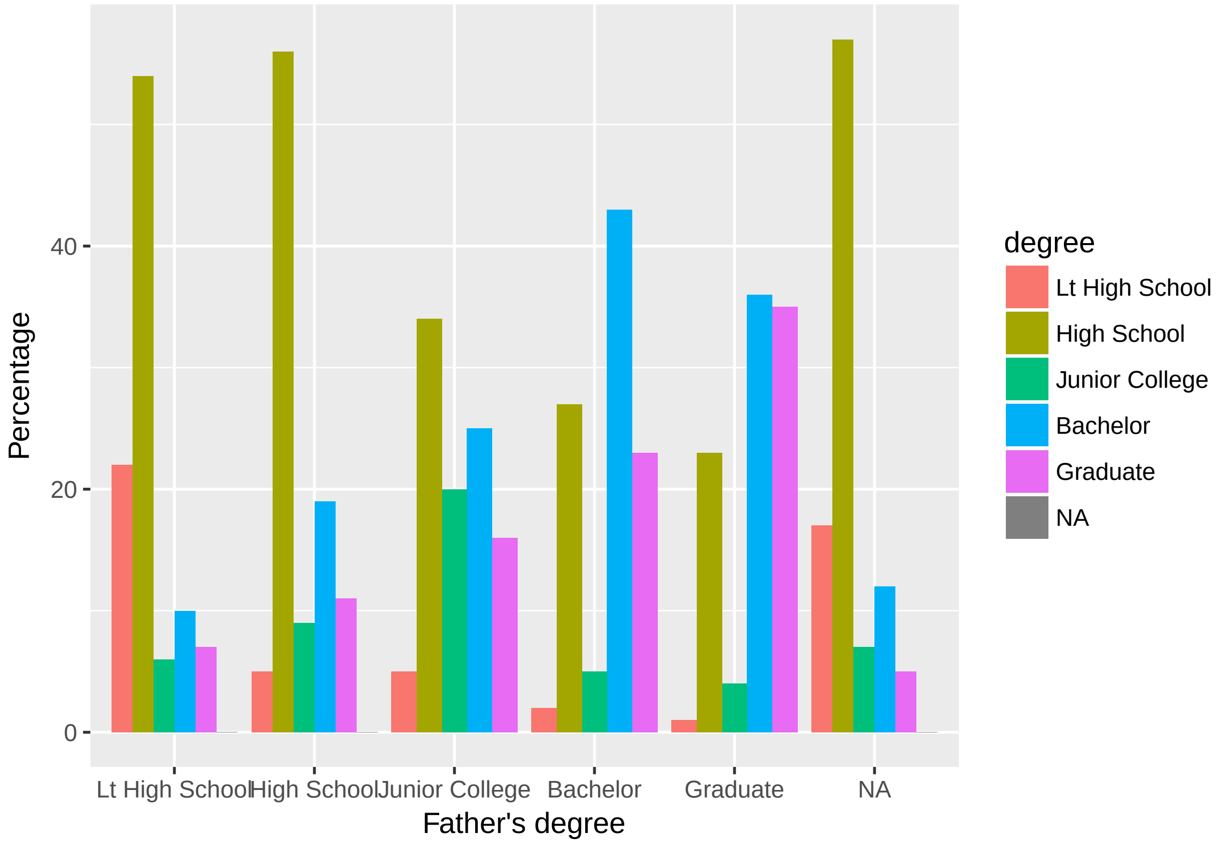
Perfect (or kinda: errors from summing up. ACCEPTABLE)

```
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")
```



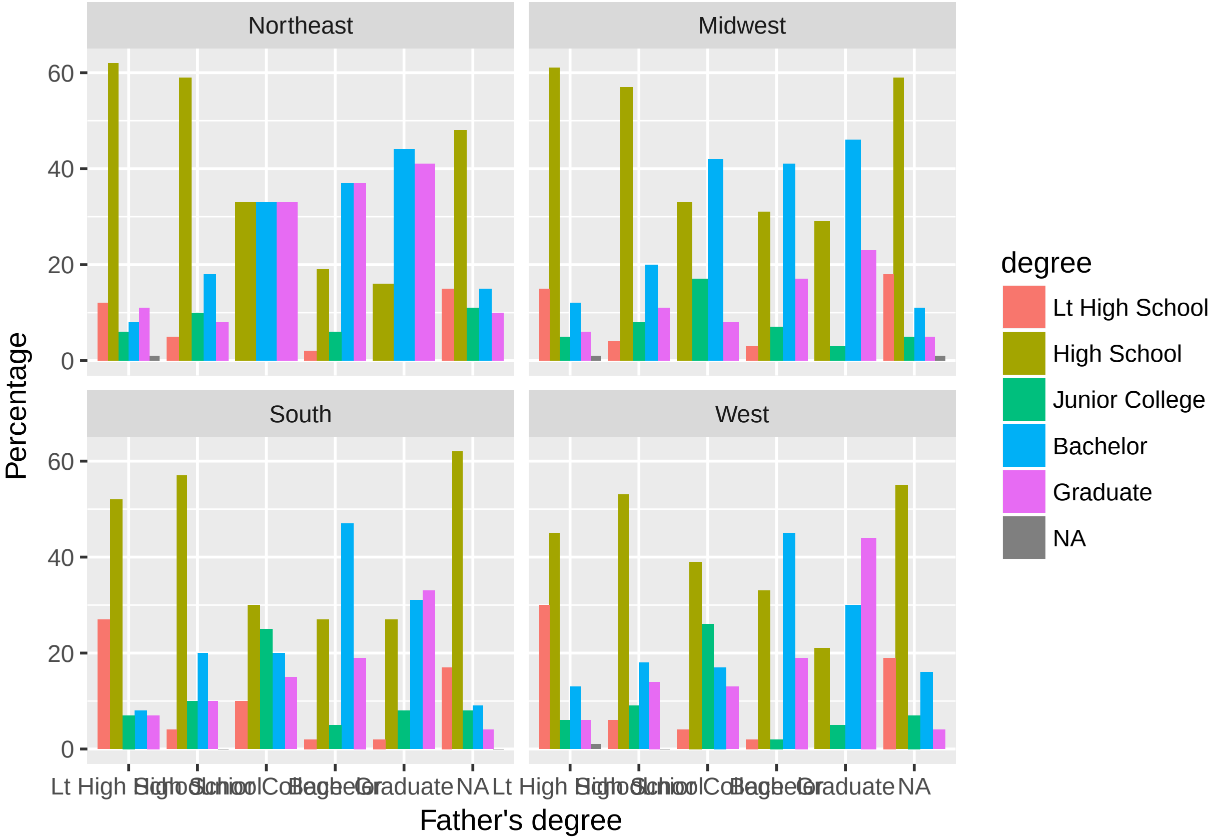
So lets look at education mobility:

```
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

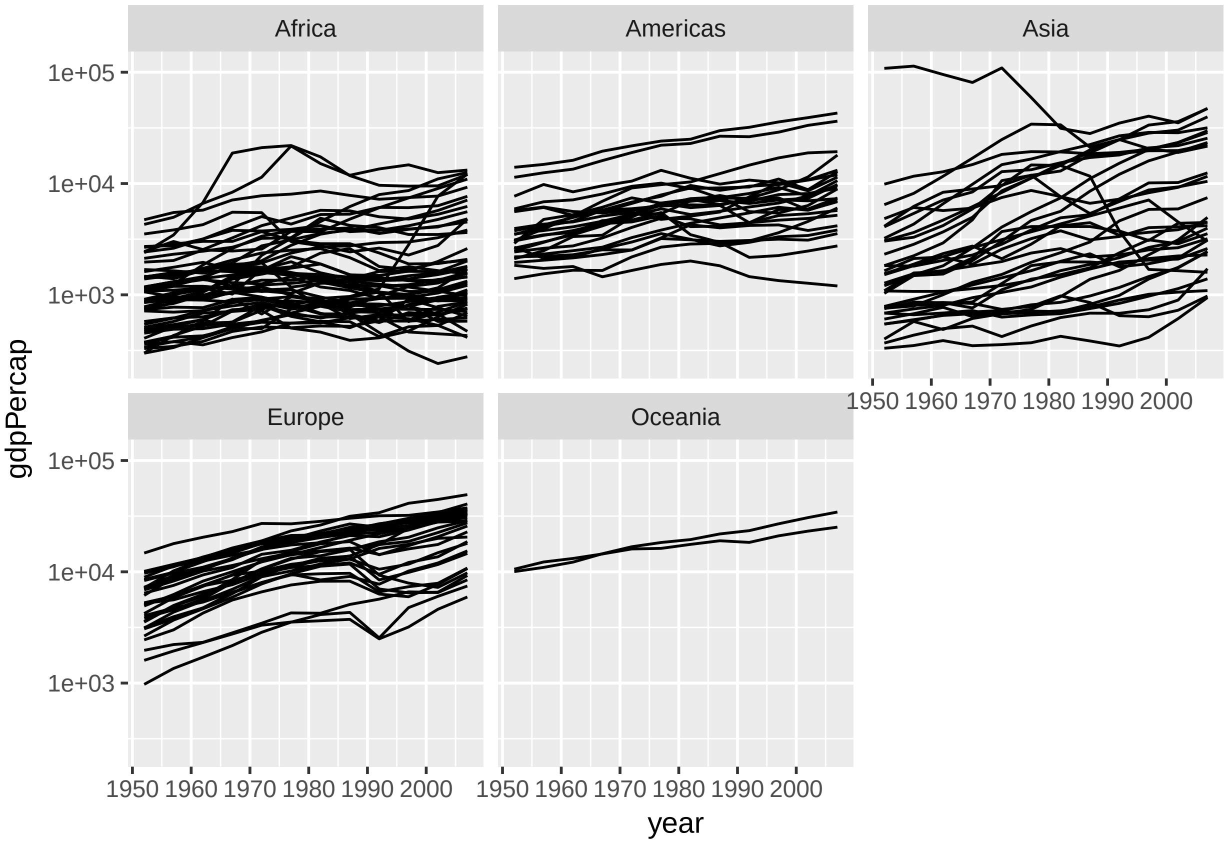
We're expanding our vocabulary. We'll also work with multiple layered plots.

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:

```r
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

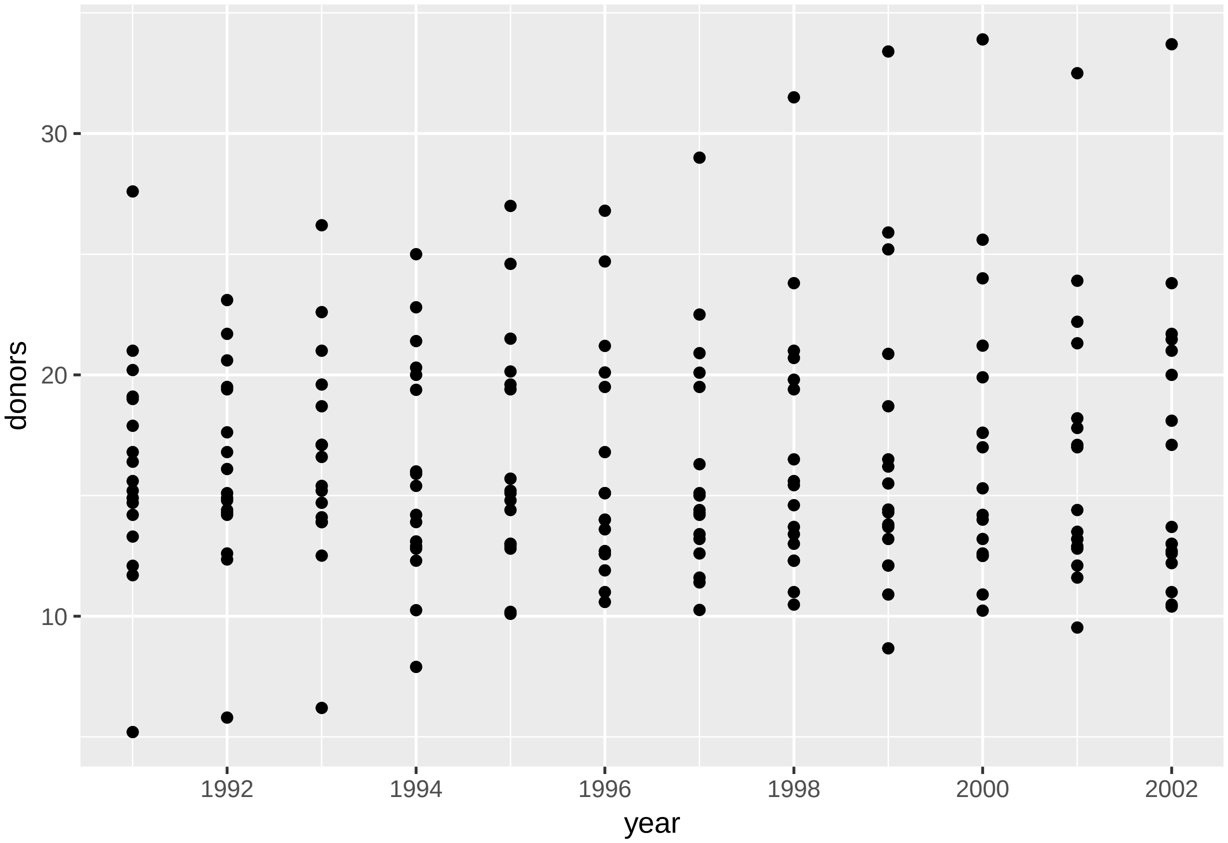
Collected by Kieran. Looking at it with pipelines:

```
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 United Kingdom 1995-01-01  14.40 58005 23.879215 19998
##  2        Germany 1995-01-01  12.80 81678 22.877069 21411
##  3        Ireland 1999-01-01  18.70  3756  5.345097 25936
##  4         Norway 1996-01-01  15.10  4381  1.352661 26218
##  5        Finland 1998-01-01  19.80  5154  1.524176 23267
##  6    Switzerland 1998-01-01  15.43  7110 17.219666 28733
##  7 United Kingdom 1998-01-01  12.30 58440 24.058293 23343
##  8        Austria 1992-01-01  23.10  7841  9.350107 20601
##  9          Italy 1994-01-01   7.90 57204 18.983208 19903
## 10          Spain 2002-01-01  33.70 41874  8.275658 21592
```

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

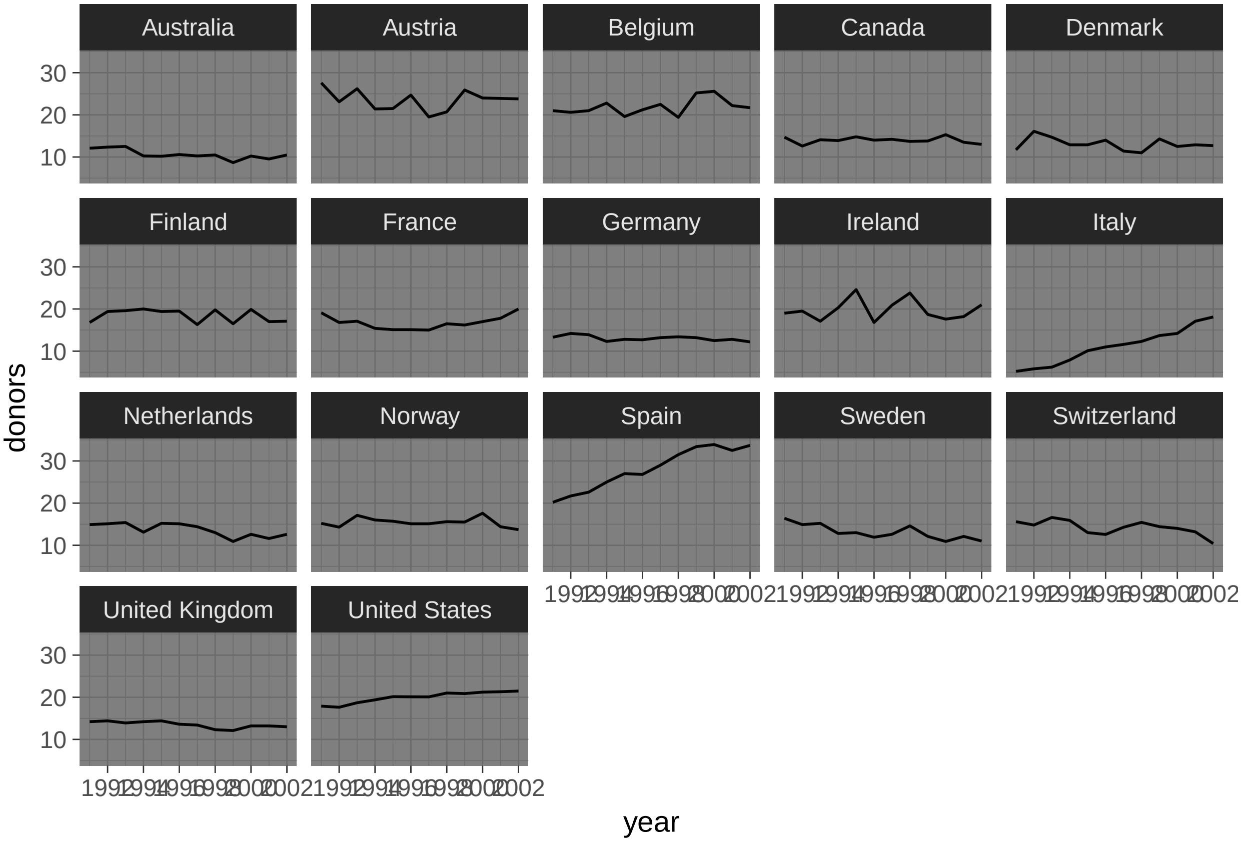
```
## Warning: Removed 34 rows containing missing values (geom_point).
```



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()
```

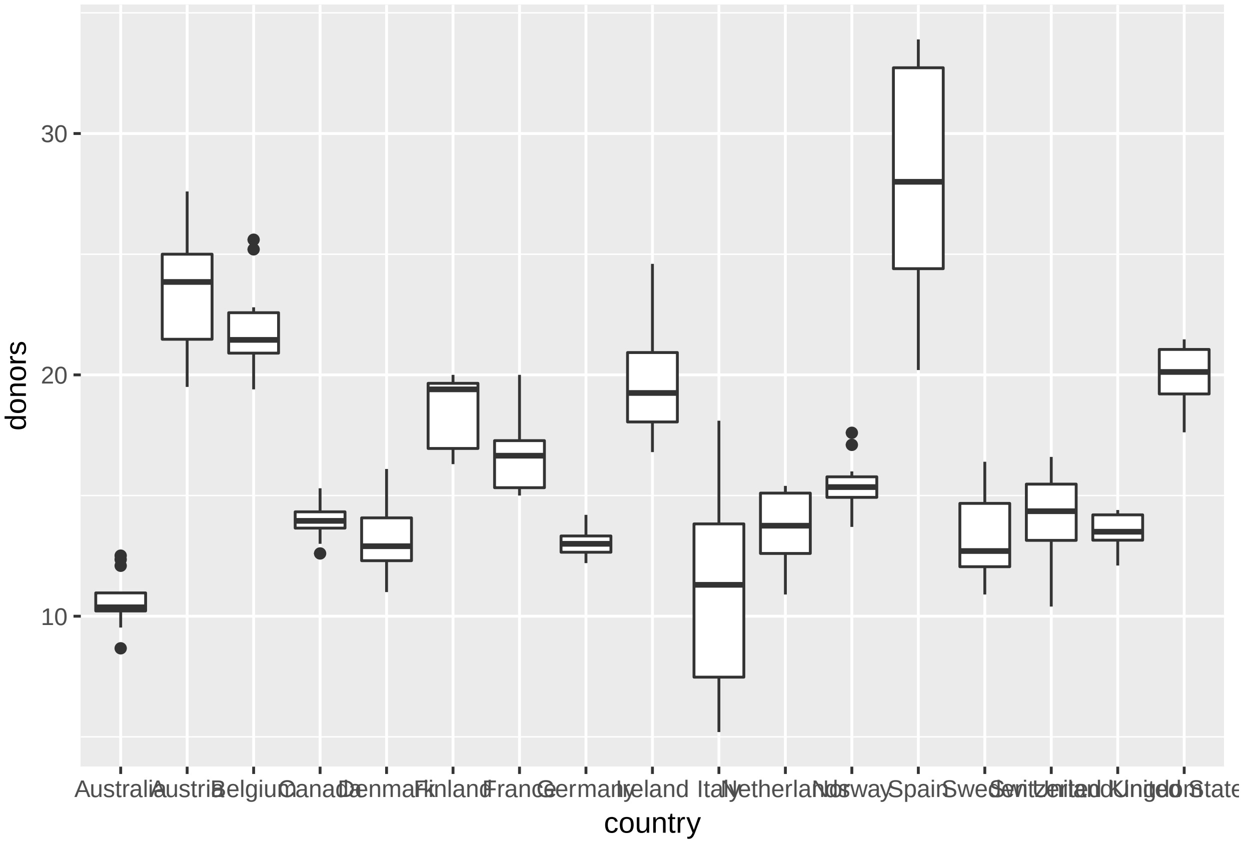
```
## Warning: Removed 2 rows containing missing values (geom_path).
```



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).
```

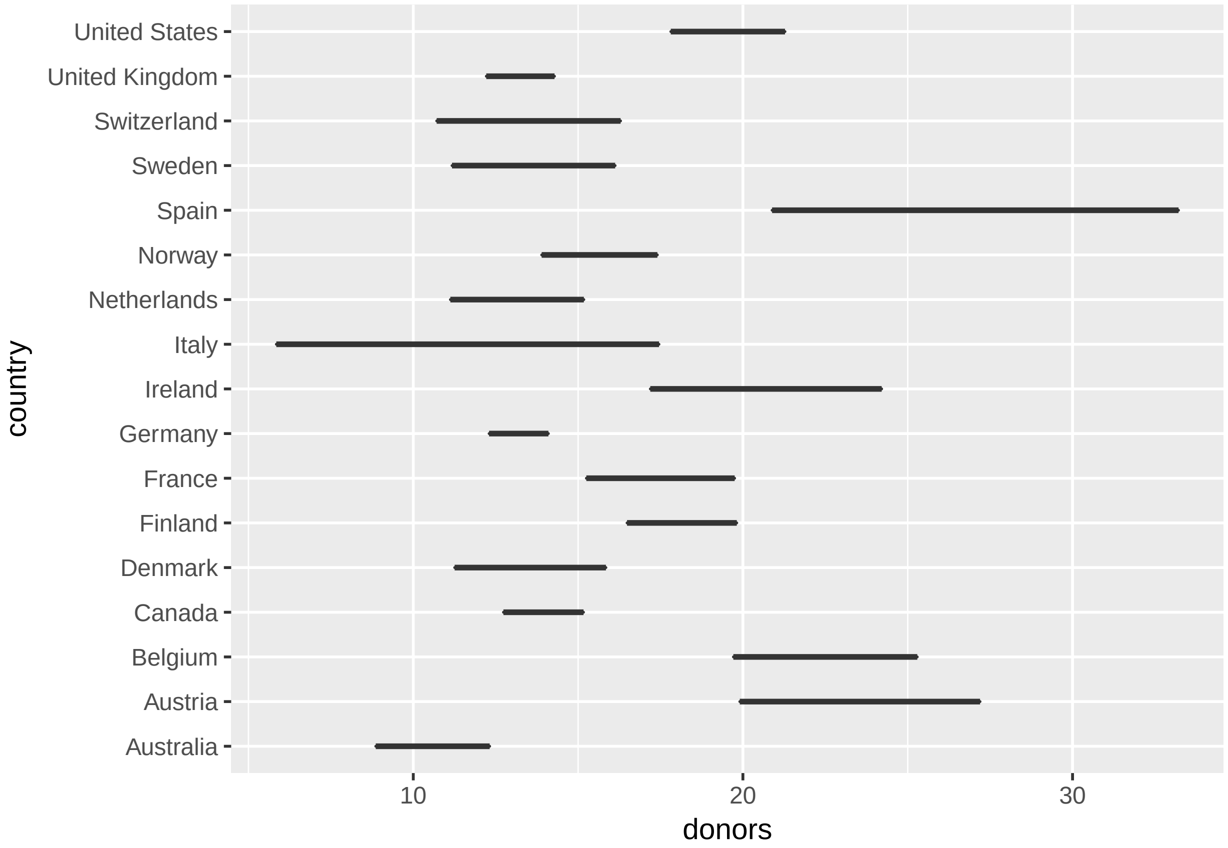


Boxplots are similar to scatterplots, they should probably have X and Y. But it's not showing us the crossregional distribution.

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

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

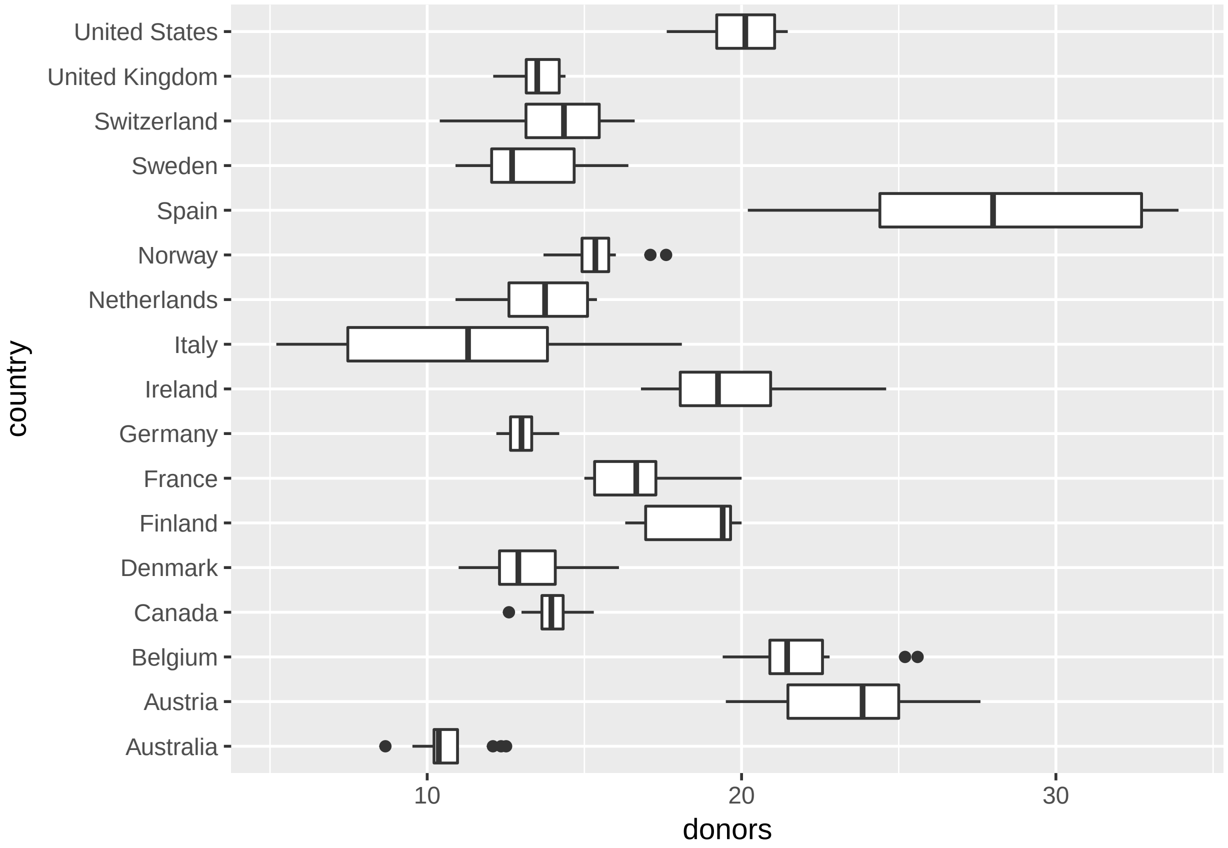
```
## Warning: position_dodge requires non-overlapping x intervals
```



Boxplot does not like it that way.

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

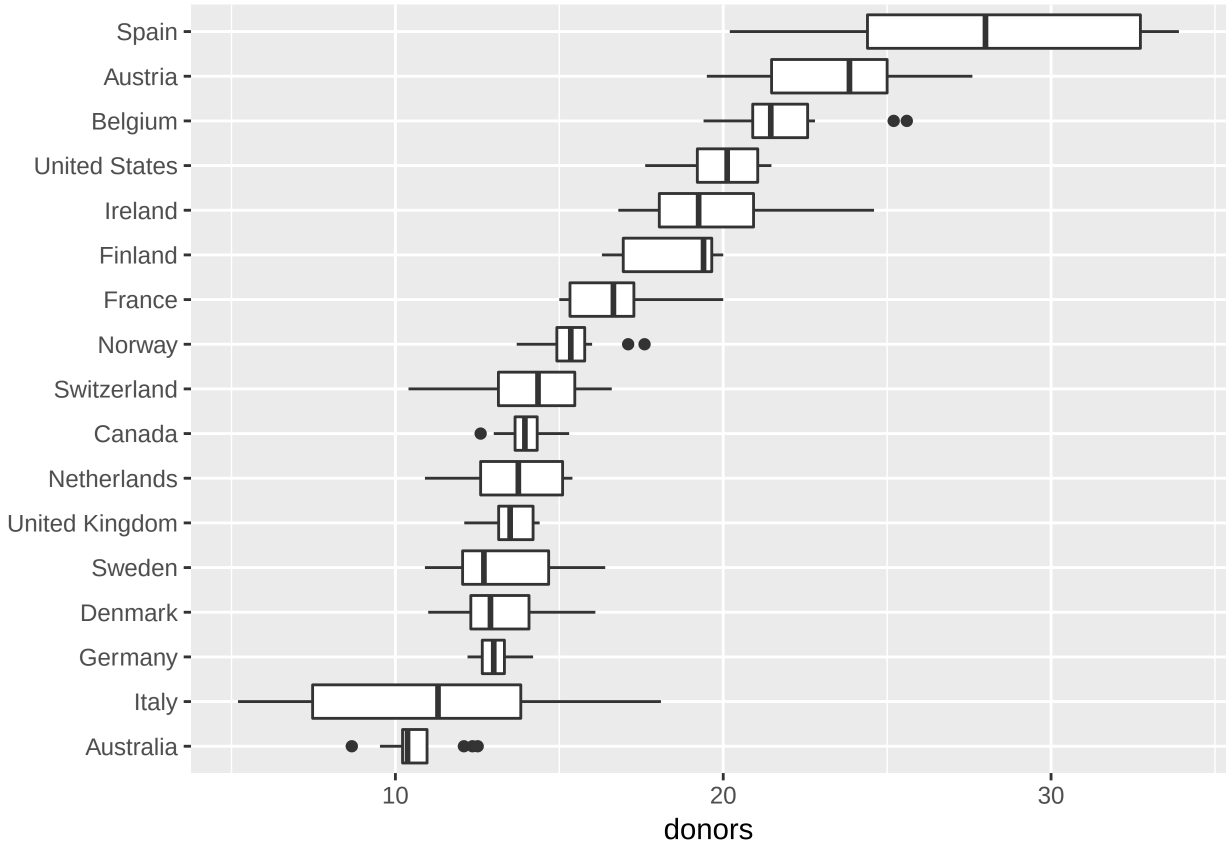
```
## Warning: Removed 34 rows containing non-finite values (stat_boxplot).
```



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).
```

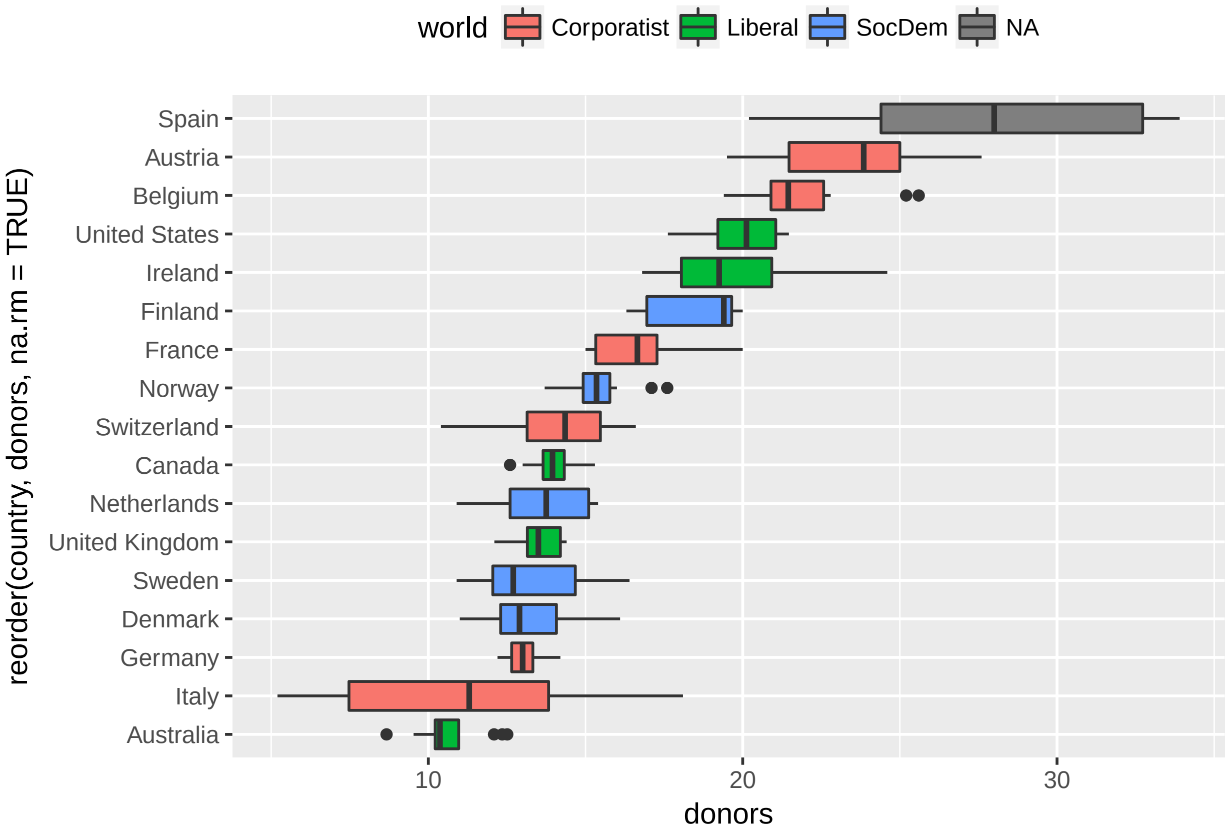


The reorder function takes variable, by-variable, the default function is mean() (but can also take median() std()). And we asked it to not calculate NA into the mean.

Another data set

```r
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")
```

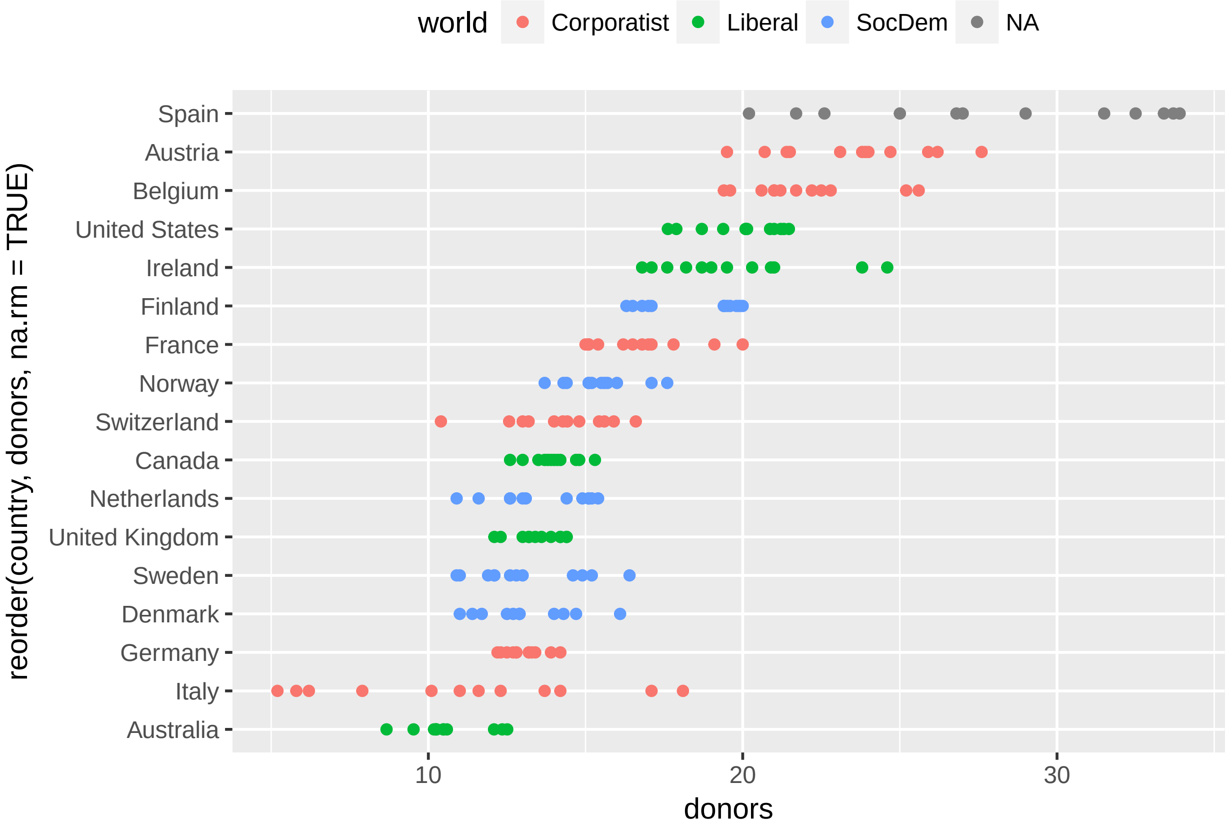
```
## Warning: Removed 34 rows containing non-finite values (stat_boxplot).
```



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")
```

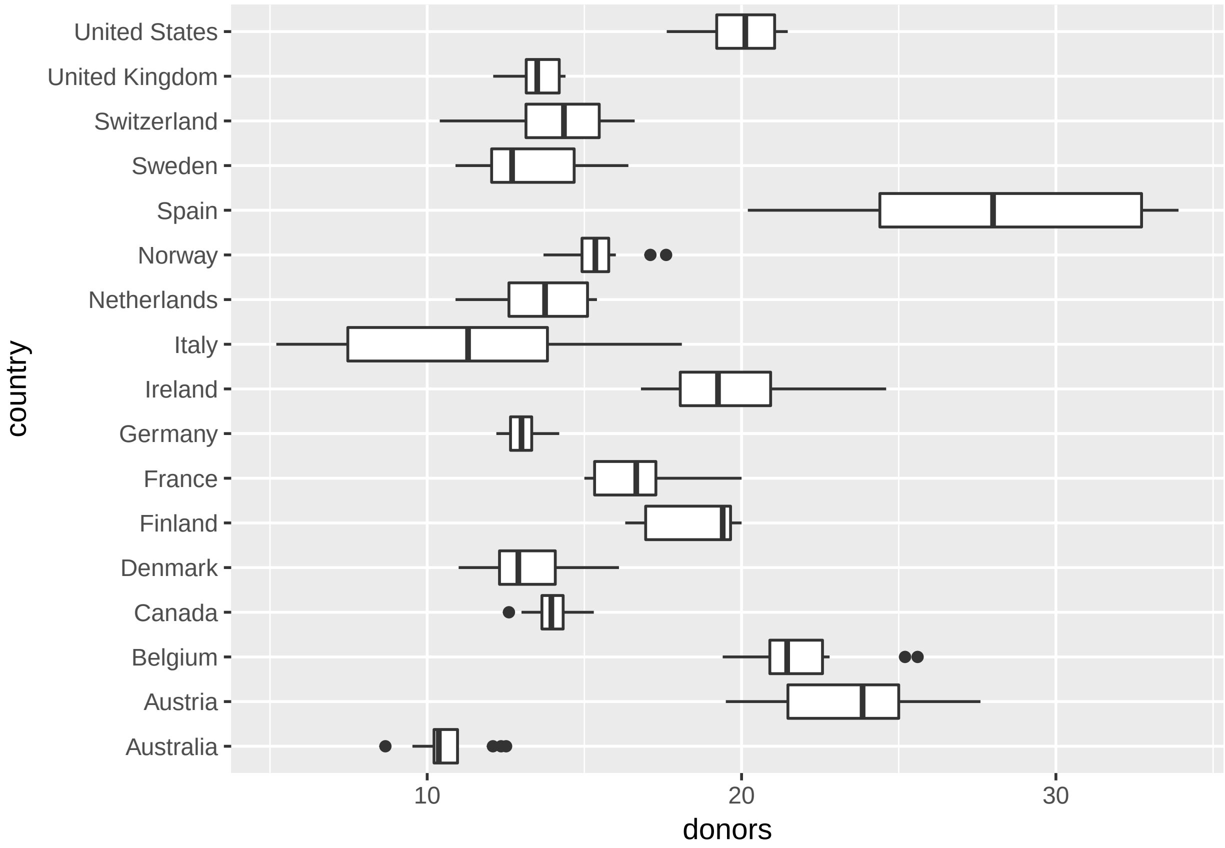
```
## Warning: Removed 34 rows containing missing values (geom_point).
```



geom_jitter can help with overplotting

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

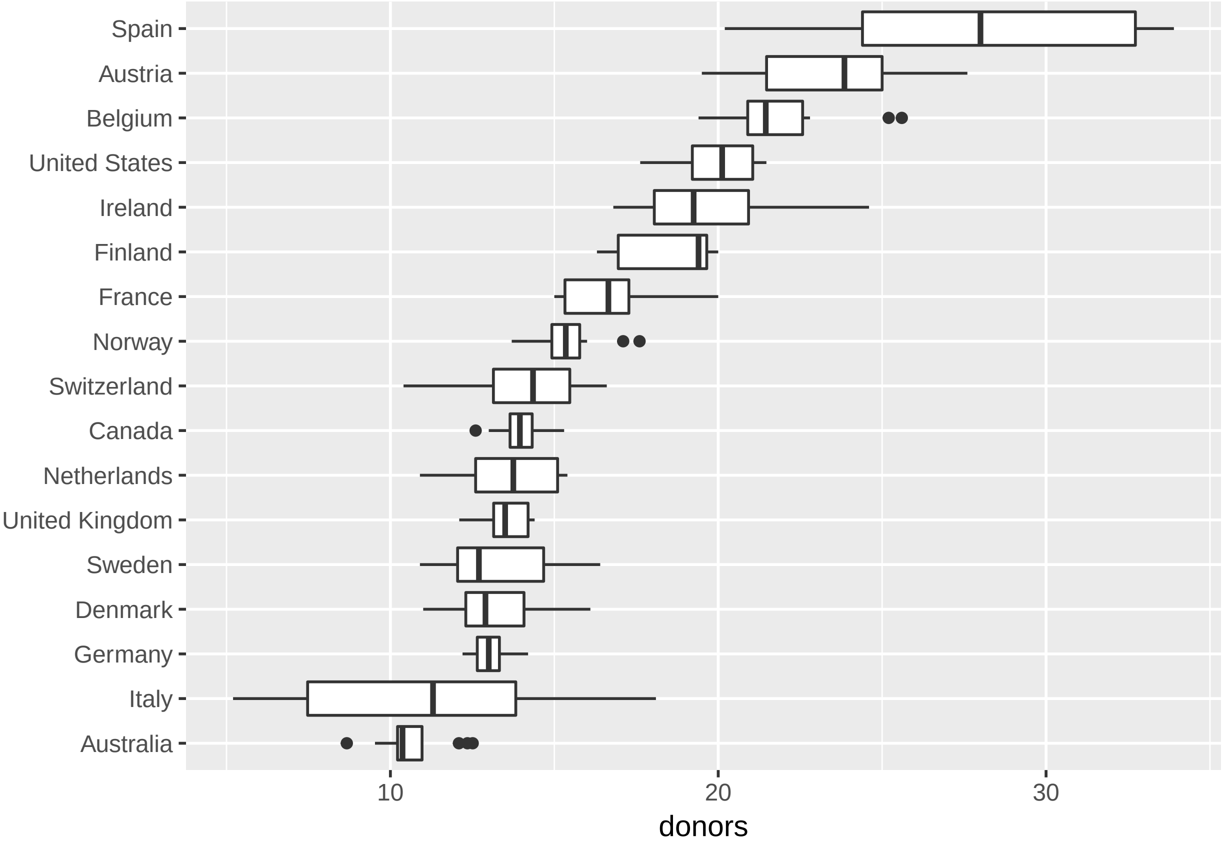
```
## Warning: Removed 34 rows containing non-finite values (stat_boxplot).
```



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).
```

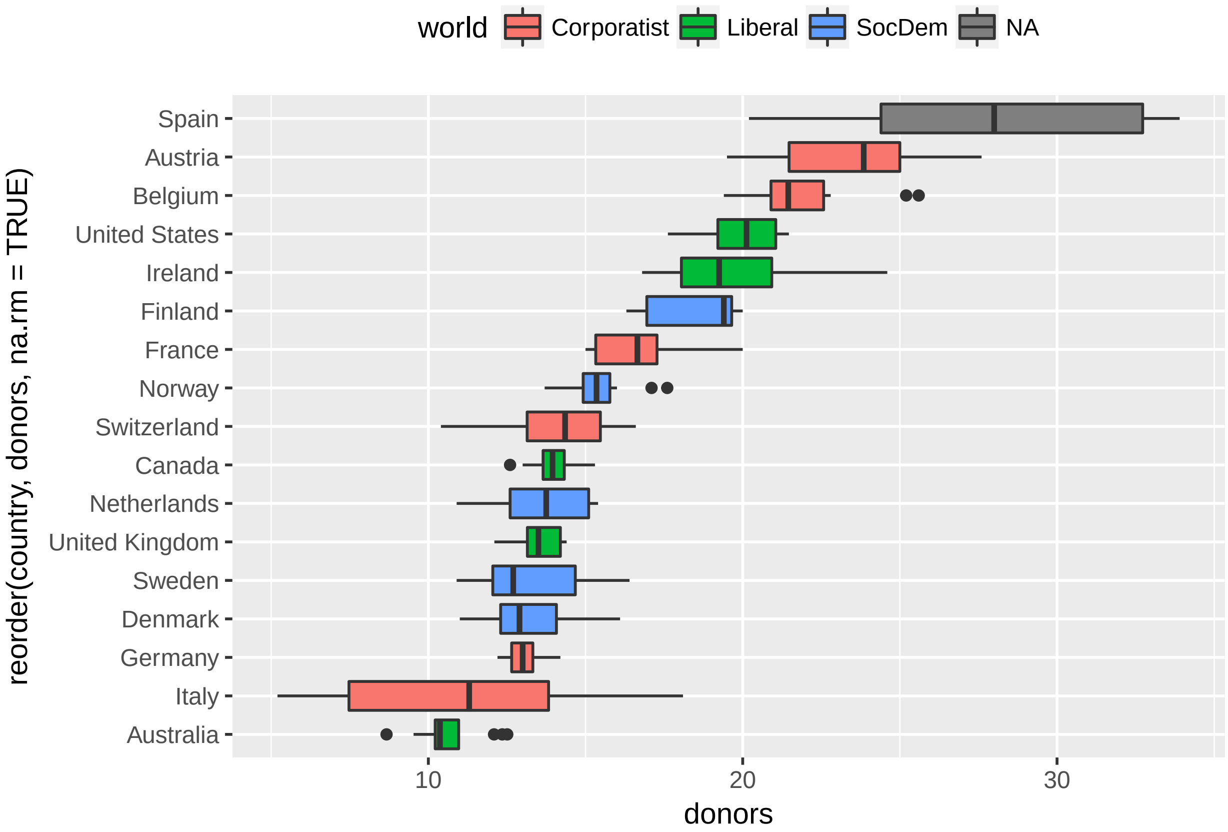


The reorder function takes variable, by-variable, the default function is mean() (but can also take median() std()). And we asked it to not calculate NA into the mean.

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")
```

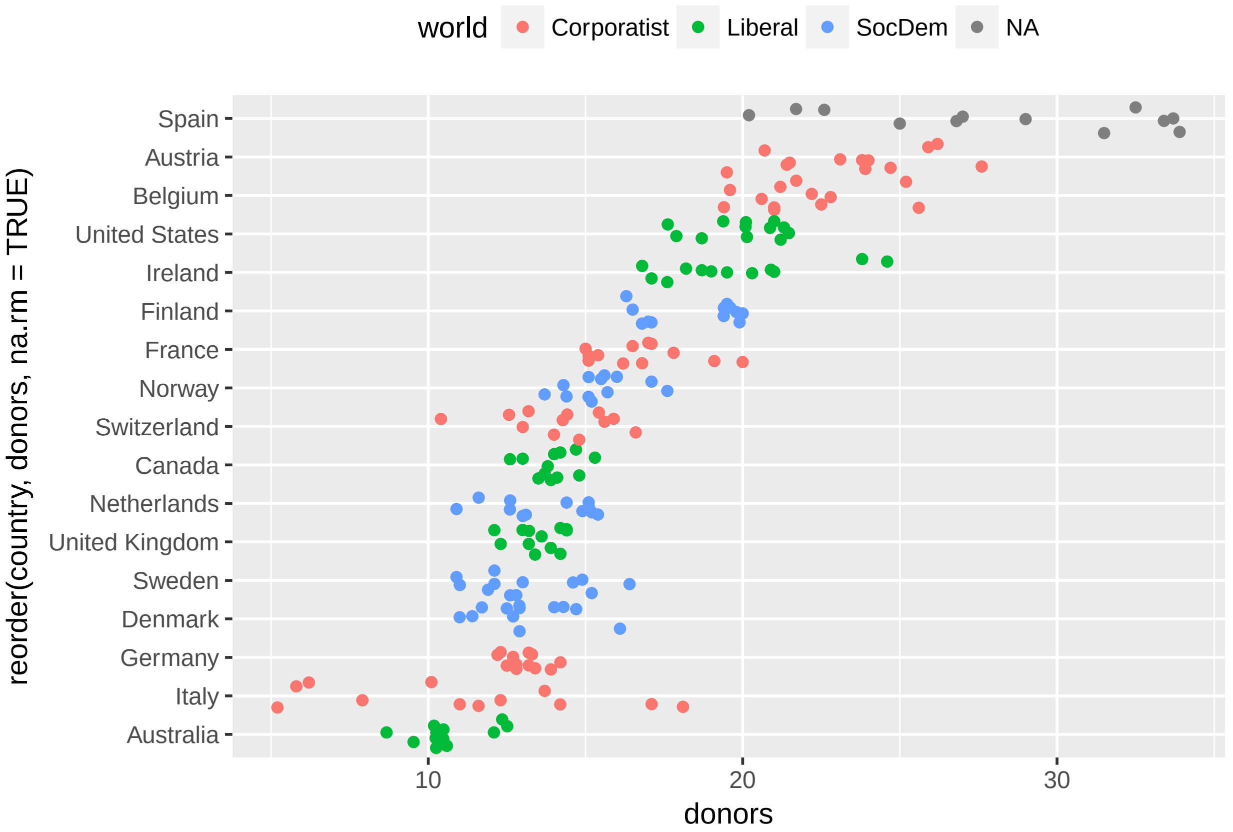
```
## Warning: Removed 34 rows containing non-finite values (stat_boxplot).
```



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")
```

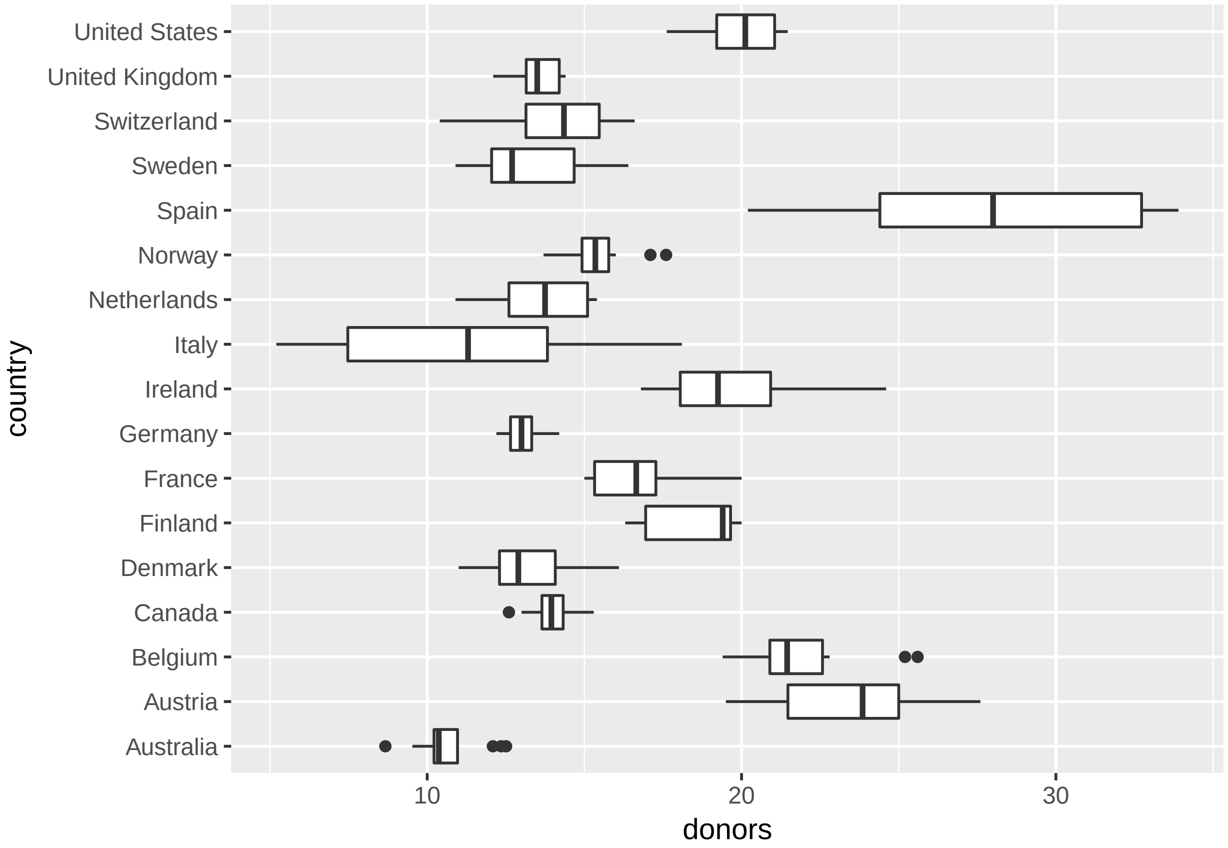
```
## Warning: Removed 34 rows containing missing values (geom_point).
```



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
```

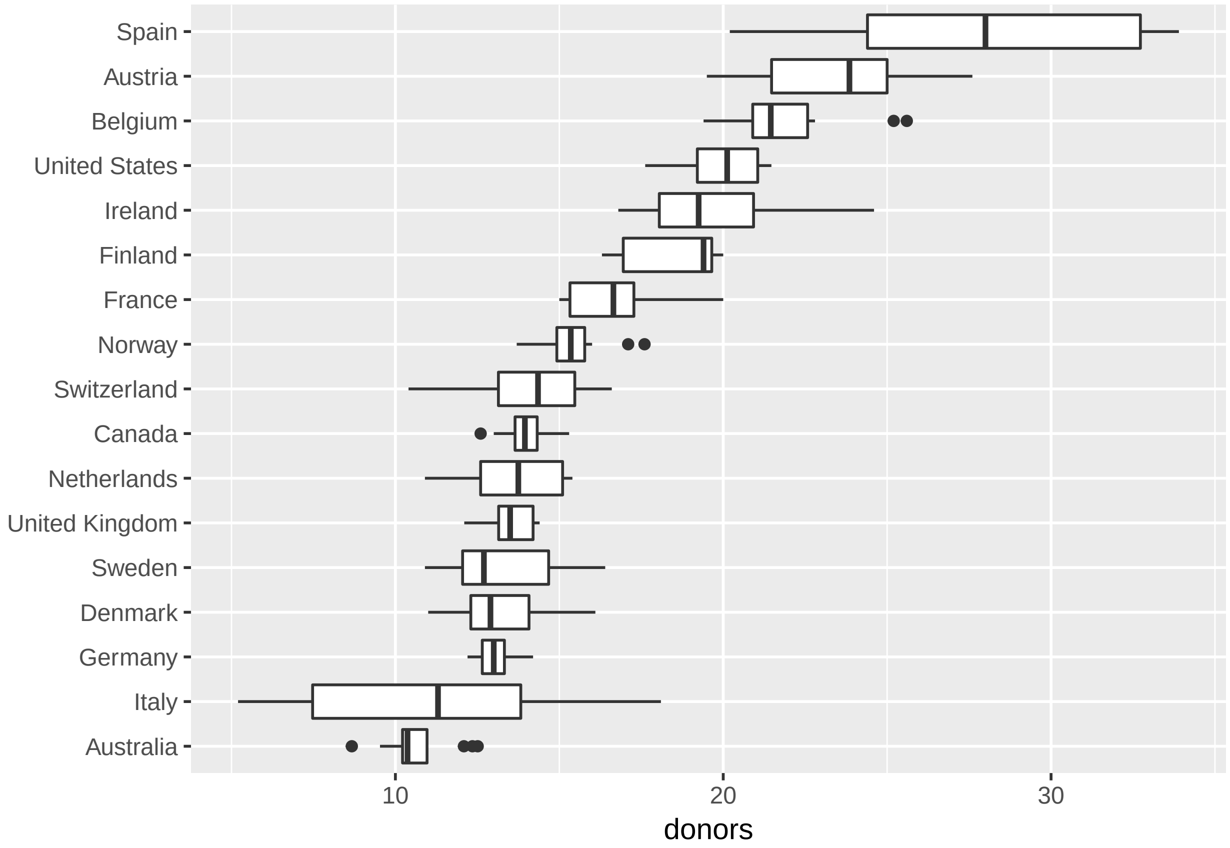
```
## Warning: Removed 34 rows containing non-finite values (stat_boxplot).
```



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).
```

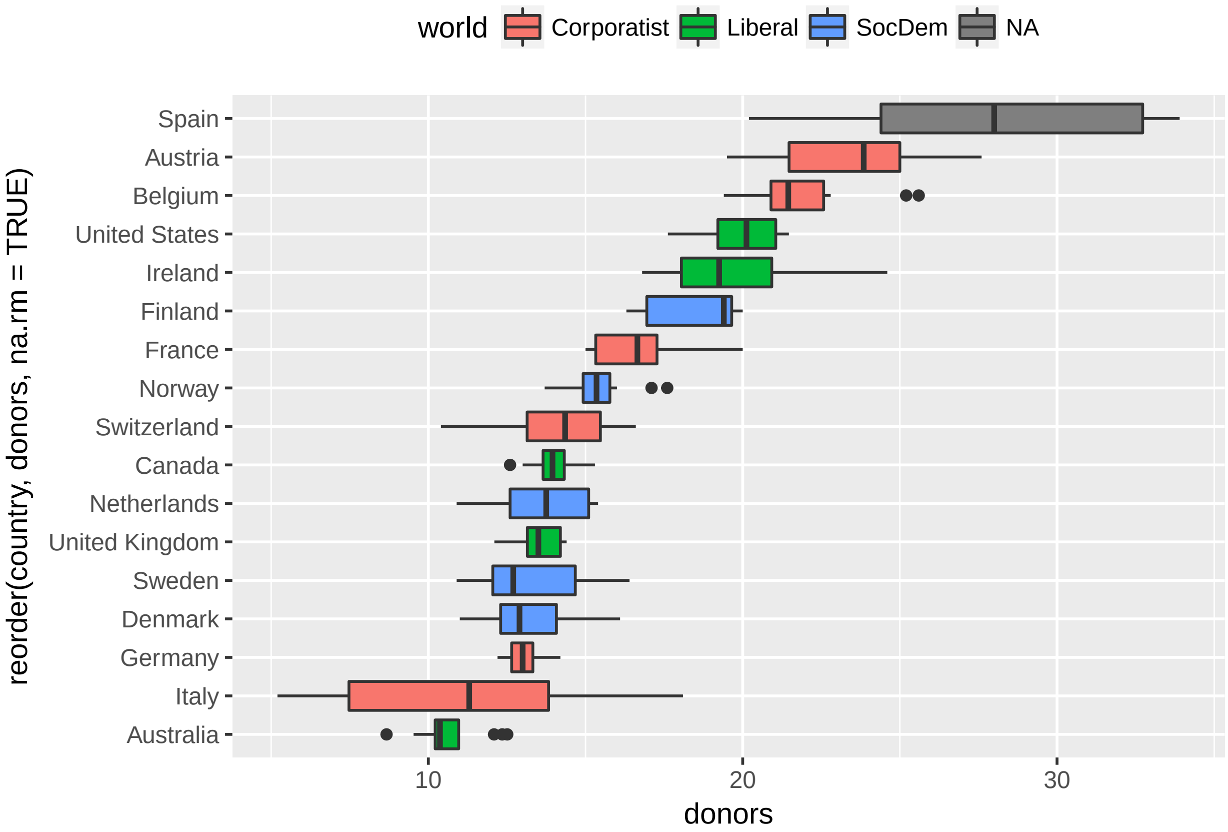


The reorder function takes variable, by-variable, the default function is mean() (but can also take median() std()). And we asked it to not calculate NA into the mean.

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")
```

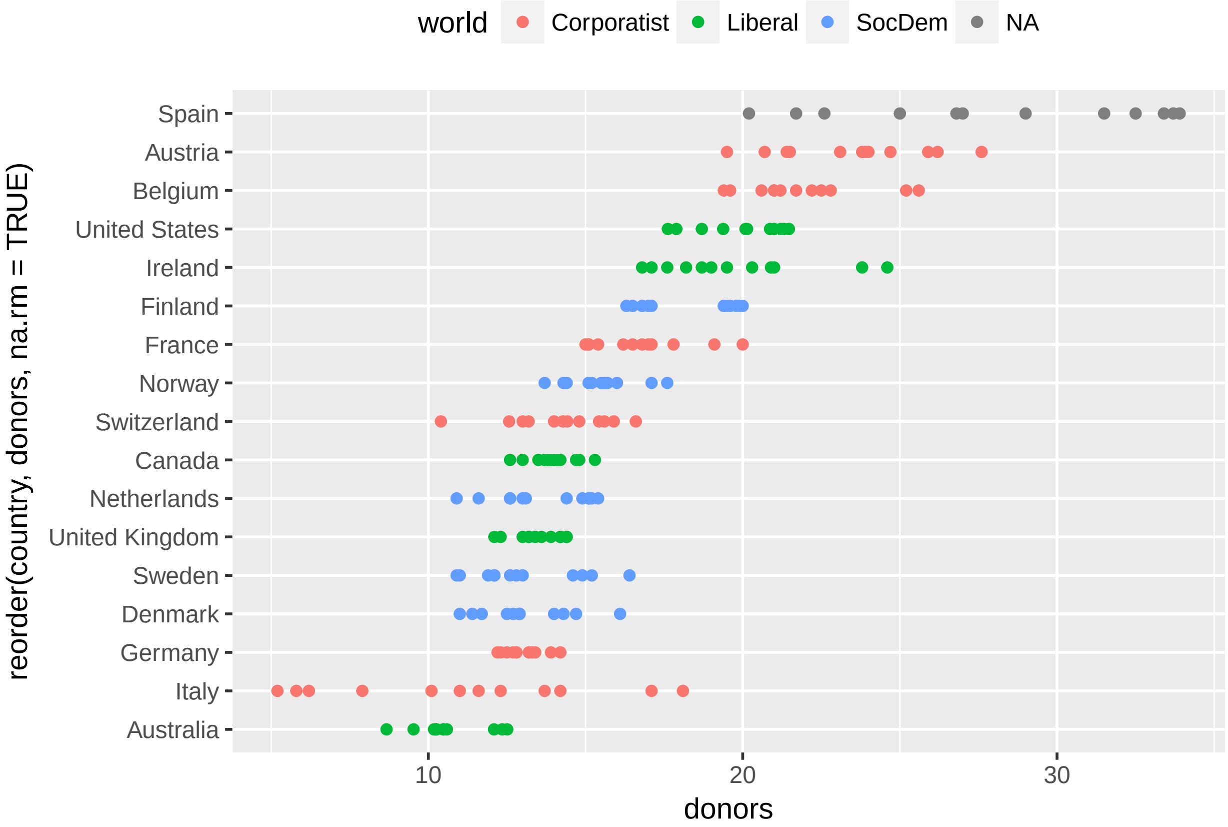
```
## Warning: Removed 34 rows containing non-finite values (stat_boxplot).
```



Lets try to make it a dotplot

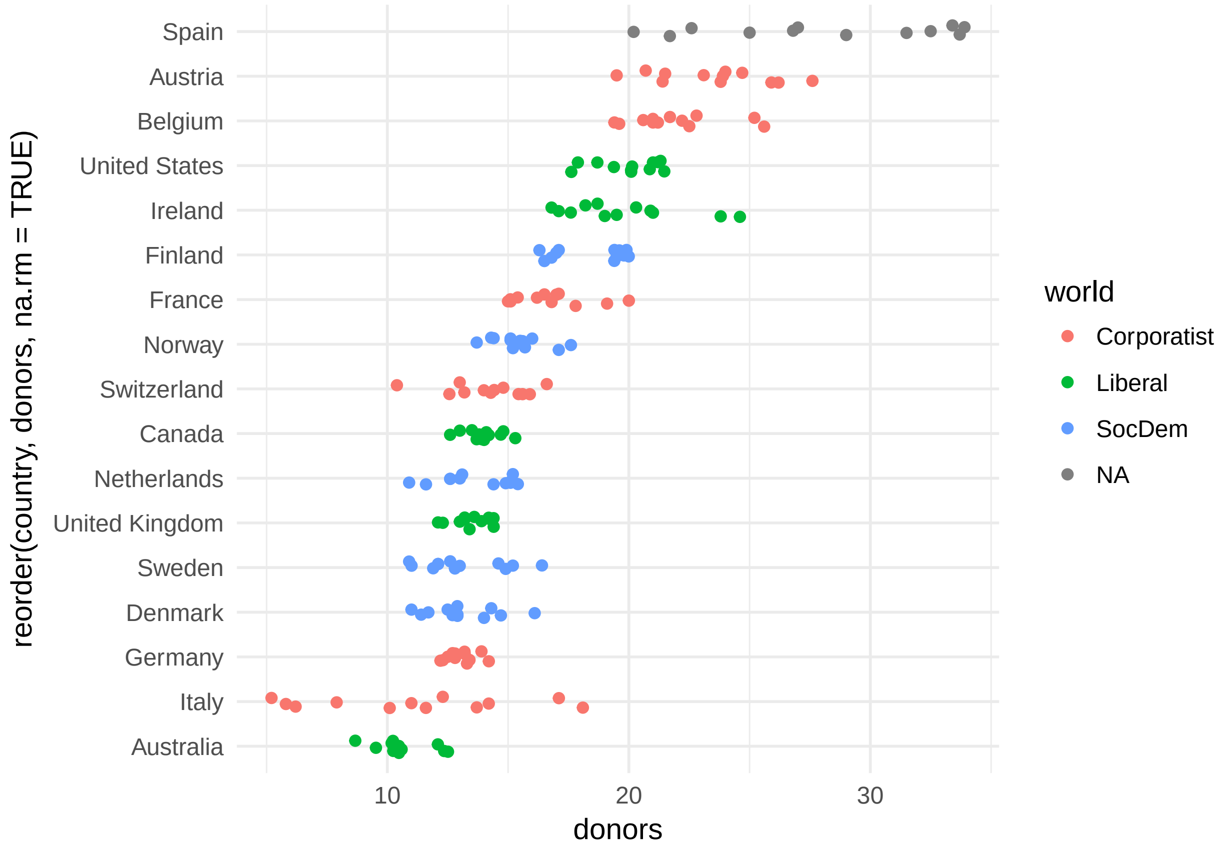
```
p <- ggplot(data = organdata,
            aes(x = reorder(country, donors, na.rm=TRUE),
                y = donors,
                color = world))
p + geom_point() +
  coord_flip() +
  theme(legend.position = "top")
```

```
## 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).
```



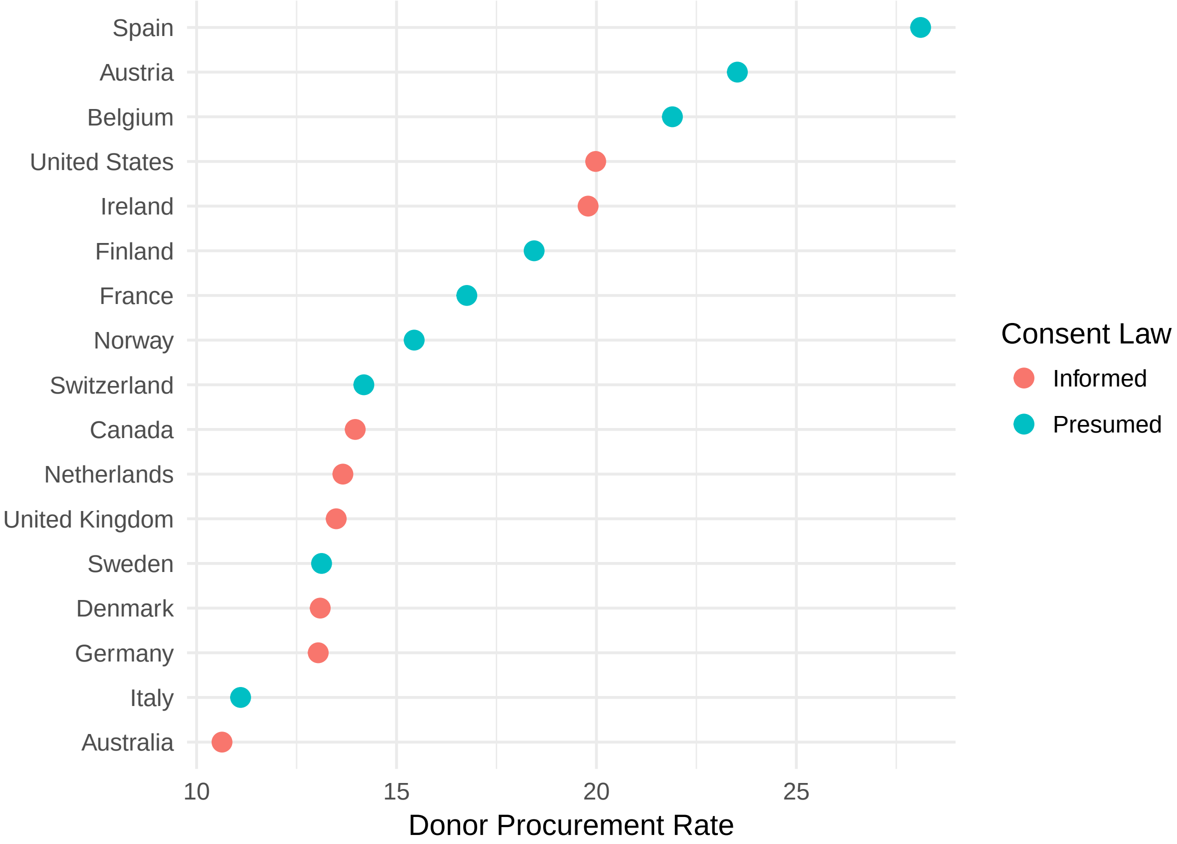
Summarize with dplyr

```r
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>
```

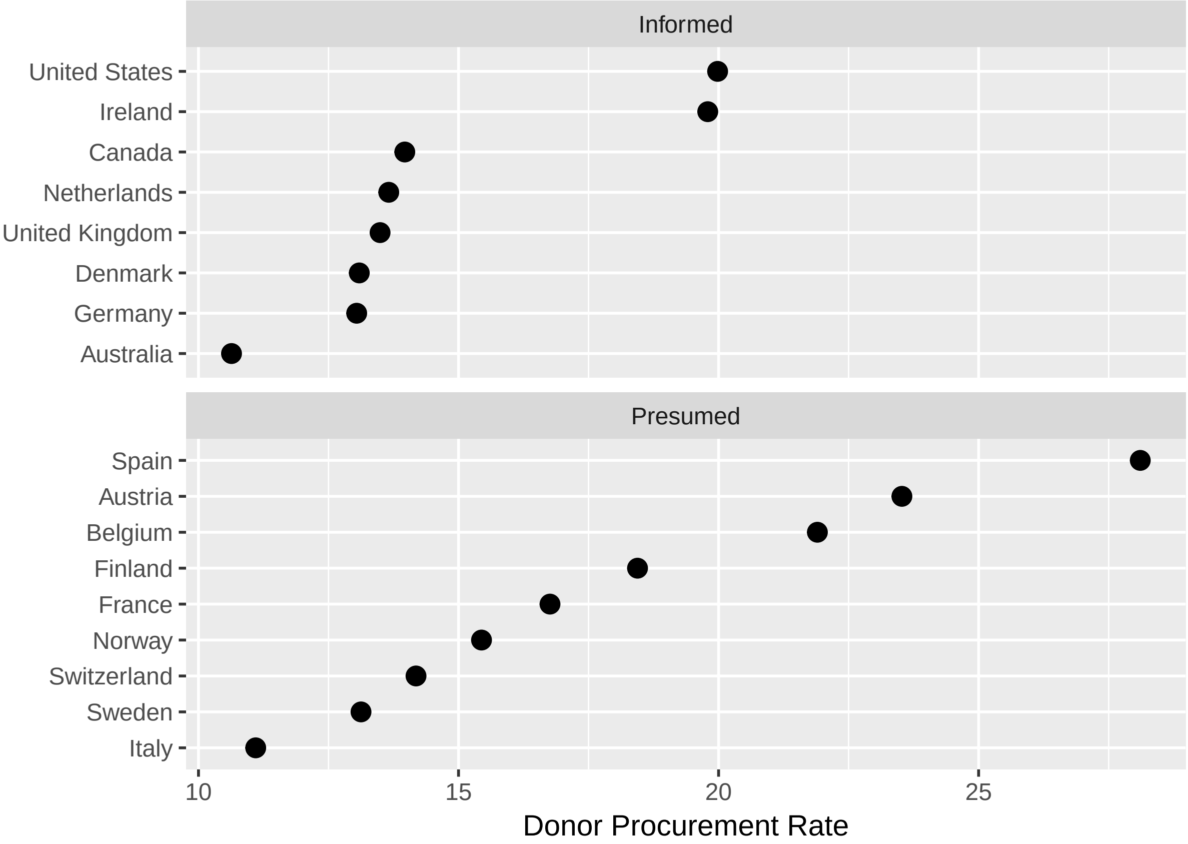
Now that we've made a nice object, lets ggplot it

```
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()
```



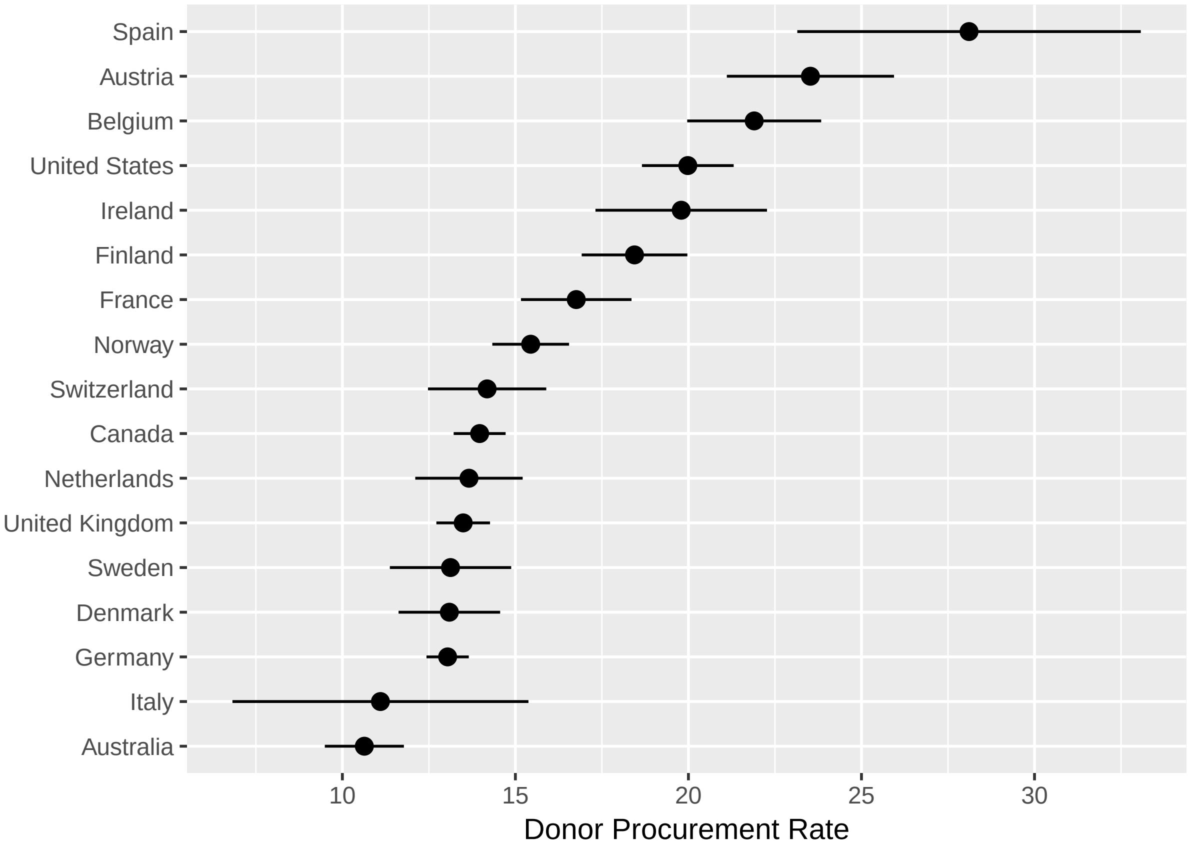
Or we can facet it. See that we use free_y to get countries at each facet, and ncol=1 to only one column with 2 rows.

```
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= "")
```



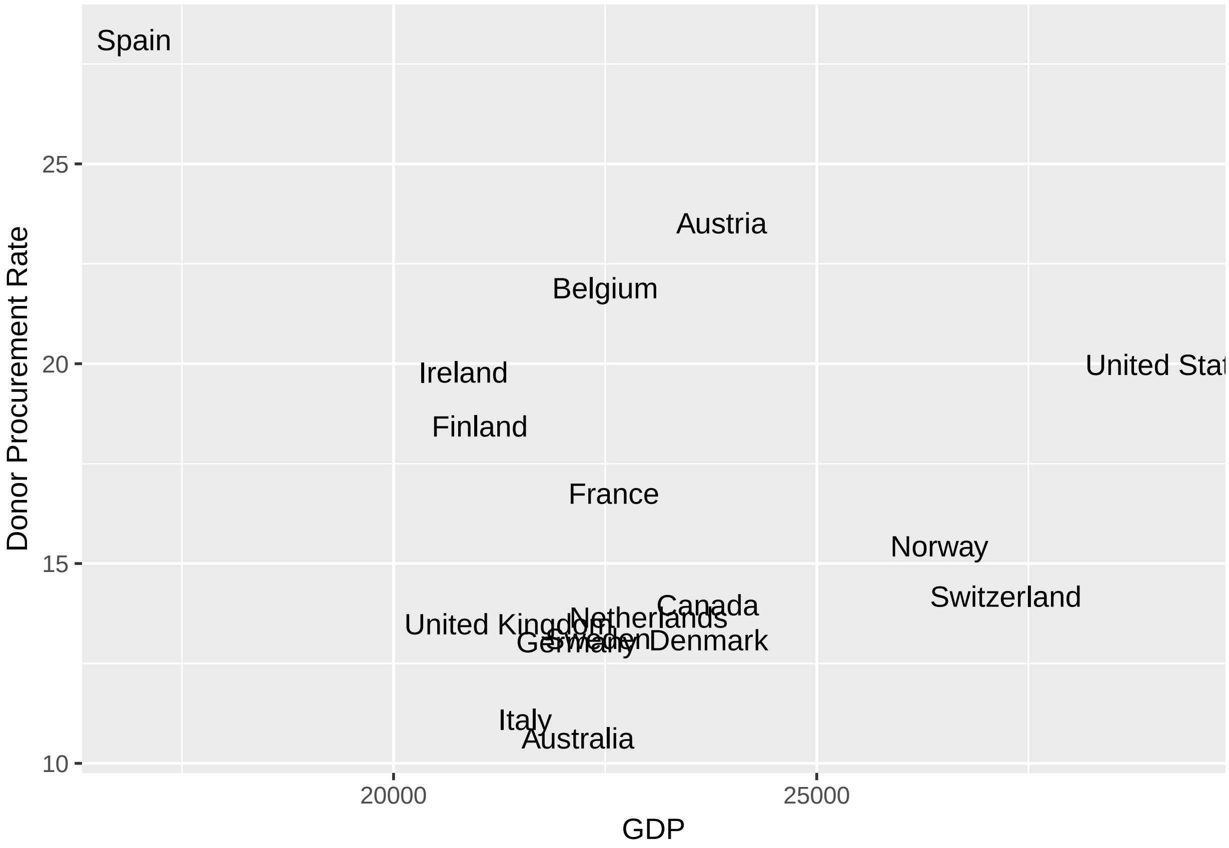
Another geoms pointrange

```
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()
```



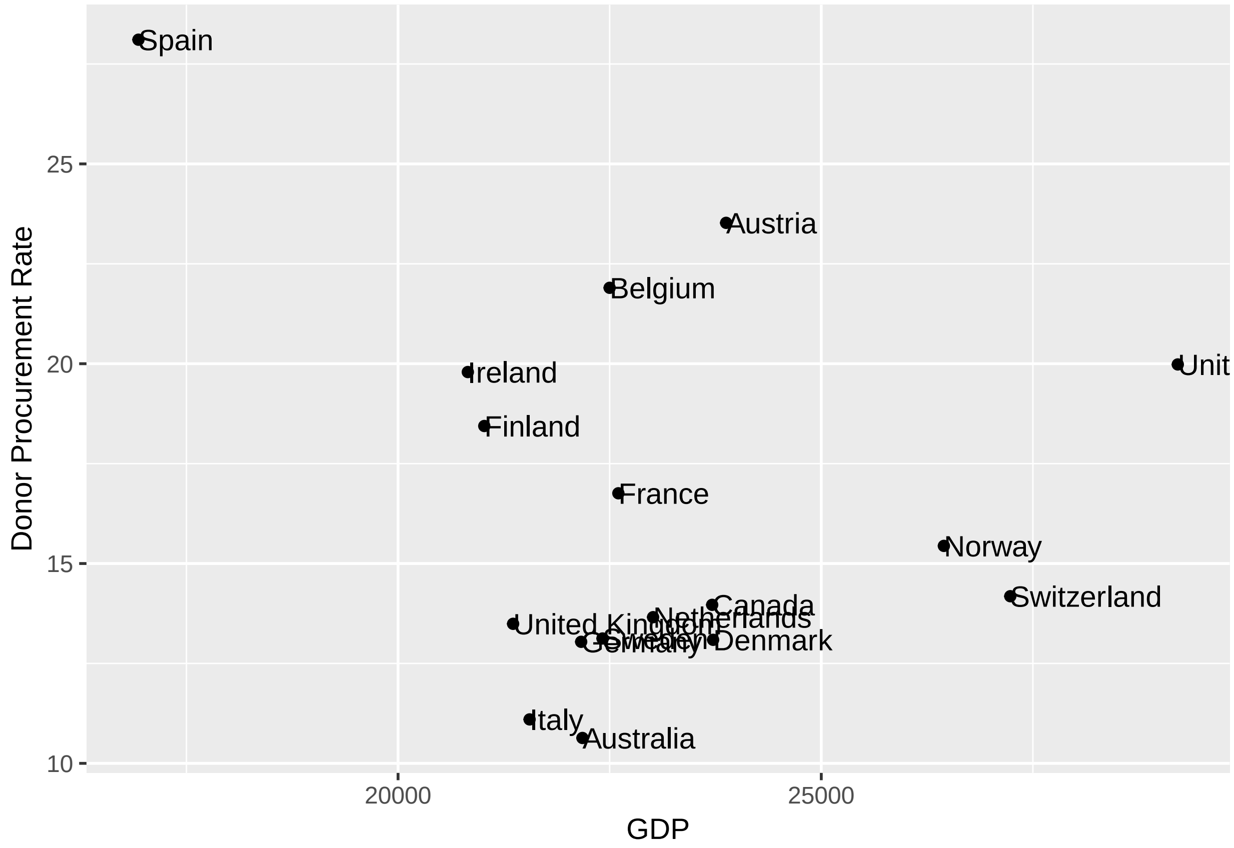
Plotting text directly

```
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)
```

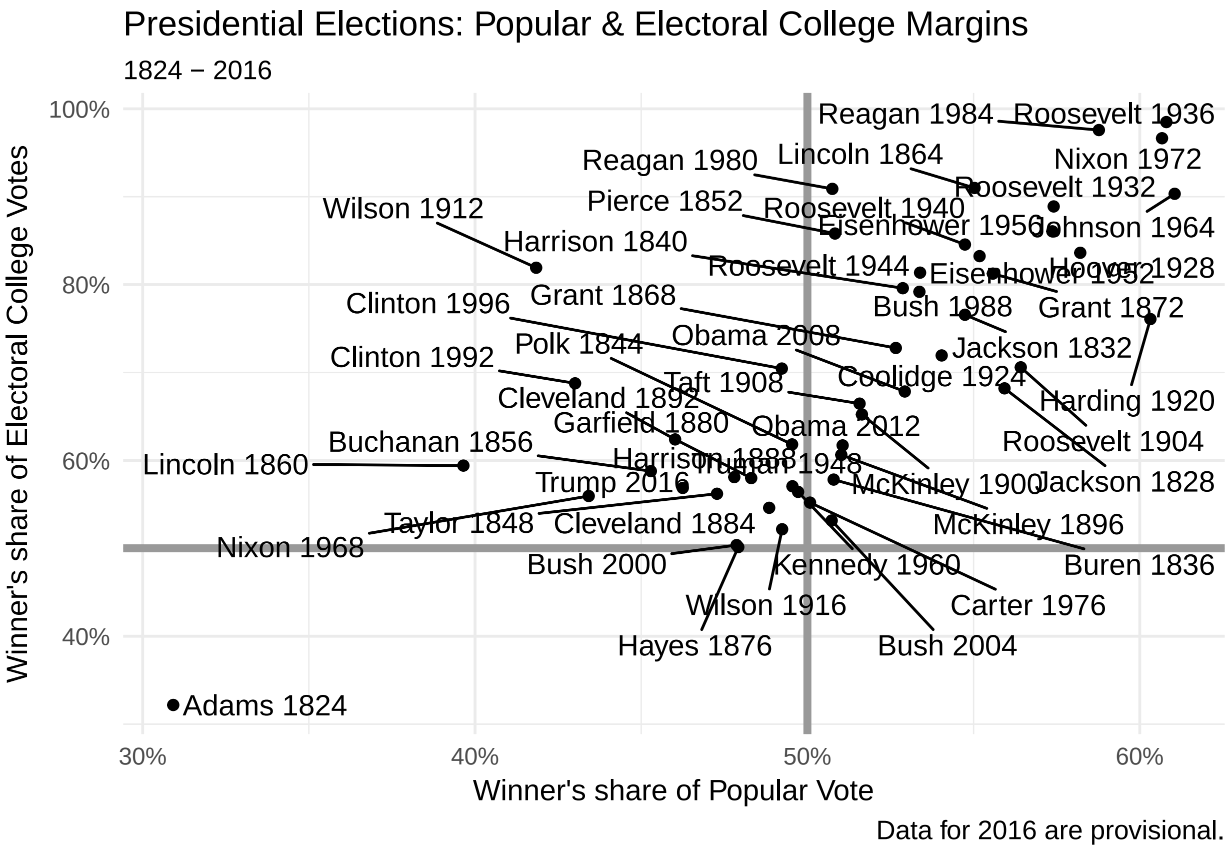
```
## # A tibble: 49 x 6
##     year                 winner win_party ec_pct popular_pct
##    <int>                  <chr>     <chr>  <dbl>       <dbl>
##  1  1824      John Quincy Adams     D.-R. 0.3218      0.3092
##  2  1828         Andrew Jackson      Dem. 0.6820      0.5593
##  3  1832         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.

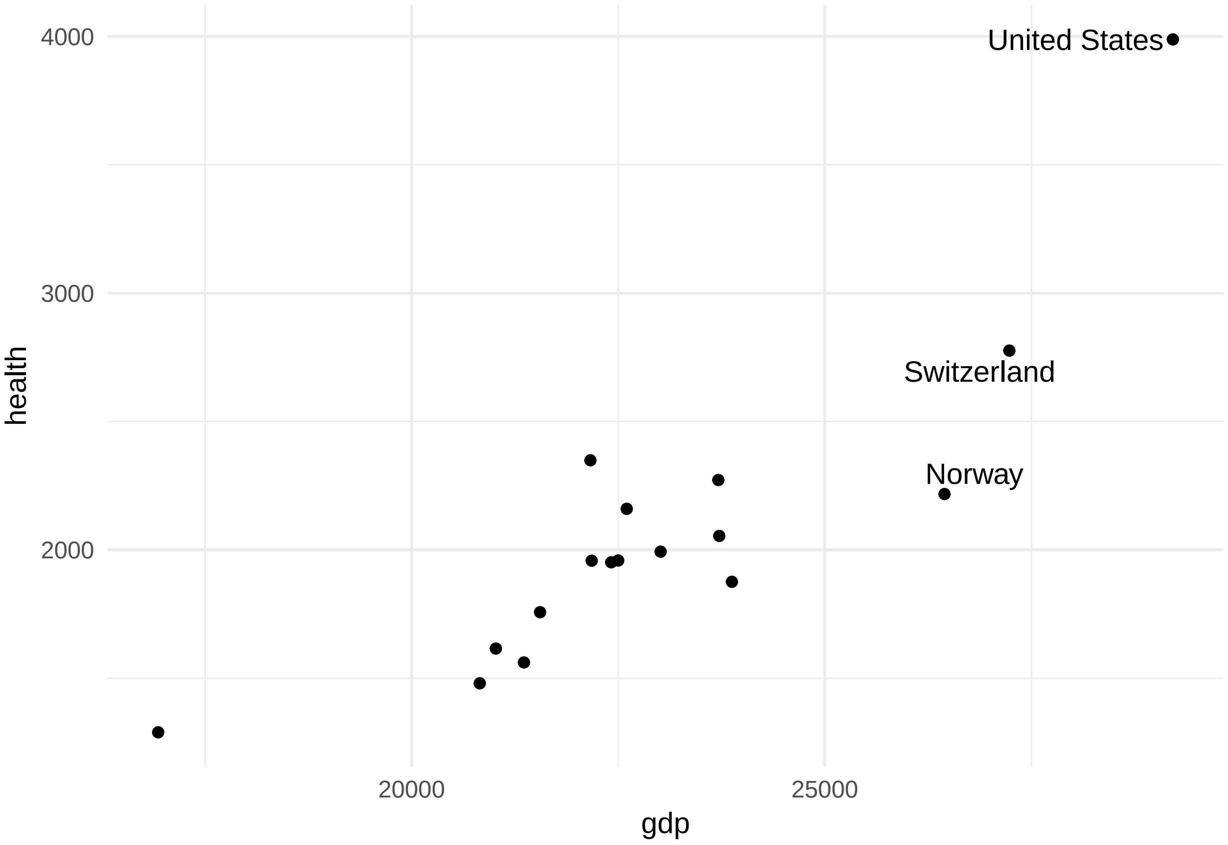
A bit more complex. Let's make some variables with the titles and texts first, to keep graph code clean.

```
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)
```

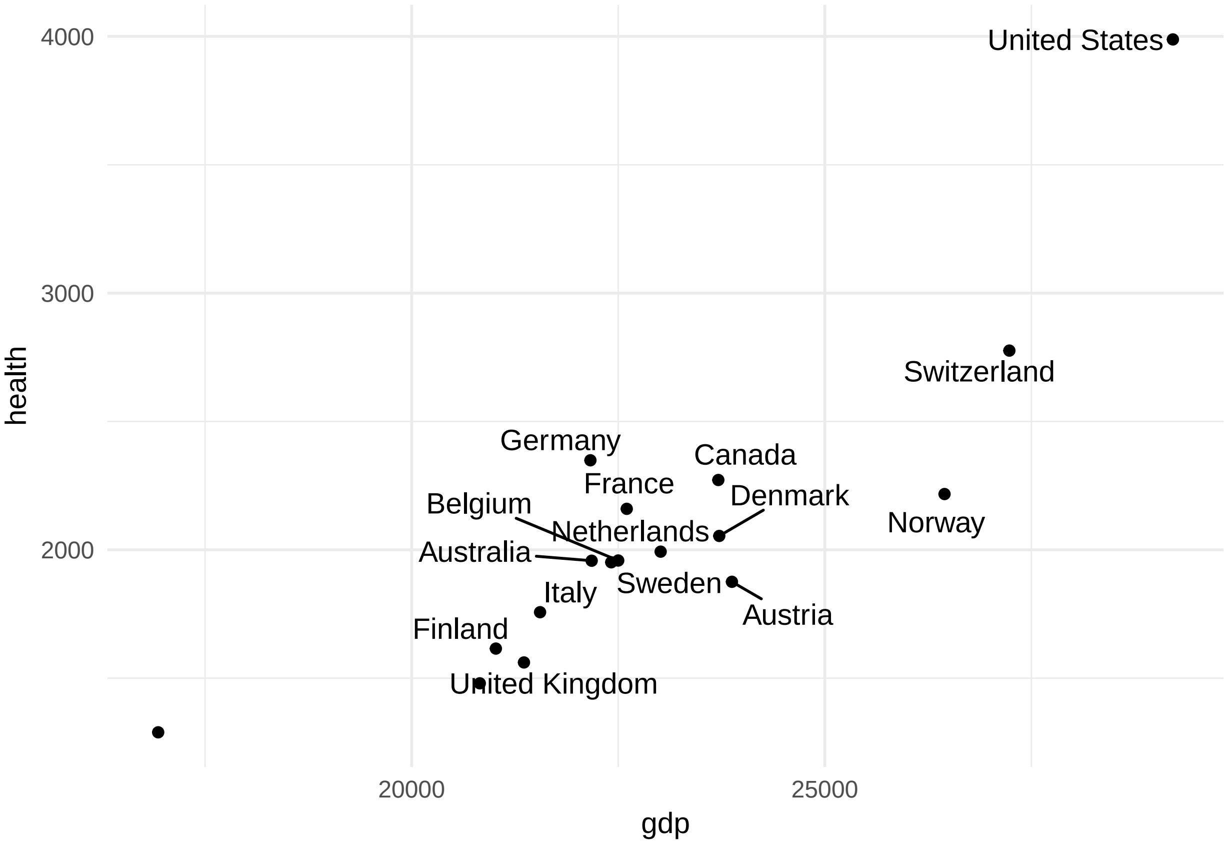


```
p <- ggplot(data = by_country,
            mapping = aes(x = gdp, y = health))
p + geom_point() +
  geom_text_repel(data = subset(by_country, gdp > 25000),
                  mapping = aes(label = country))
```



Or

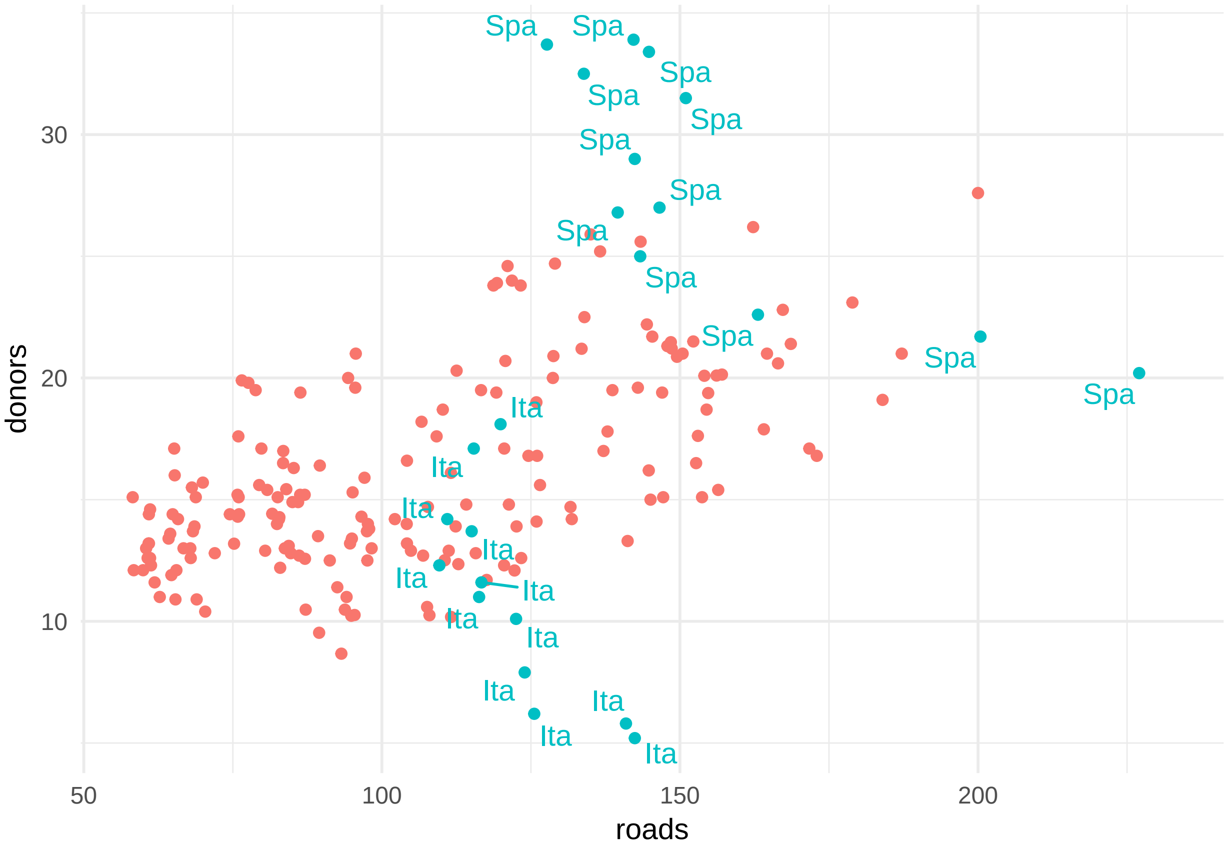
```
p <- ggplot(data = by_country,
            aes(x = gdp, y = health))
p + geom_point() +
  geom_text_repel(data = subset(by_country,
                                gdp > 25000 | health > 1500 | country %in% "Belgium"),
                  mapping = aes(label = country))
```



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).
```

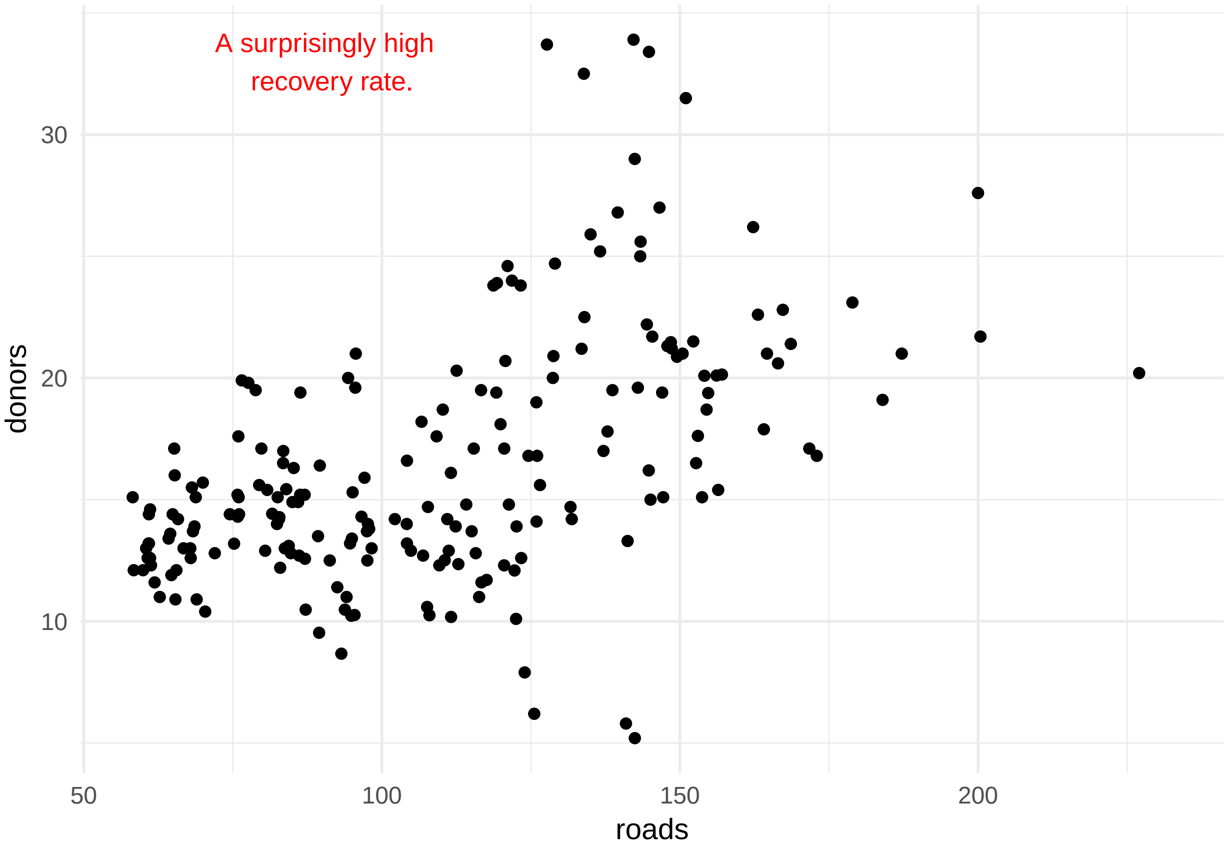


Write and draw in the Plot Area

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")
```

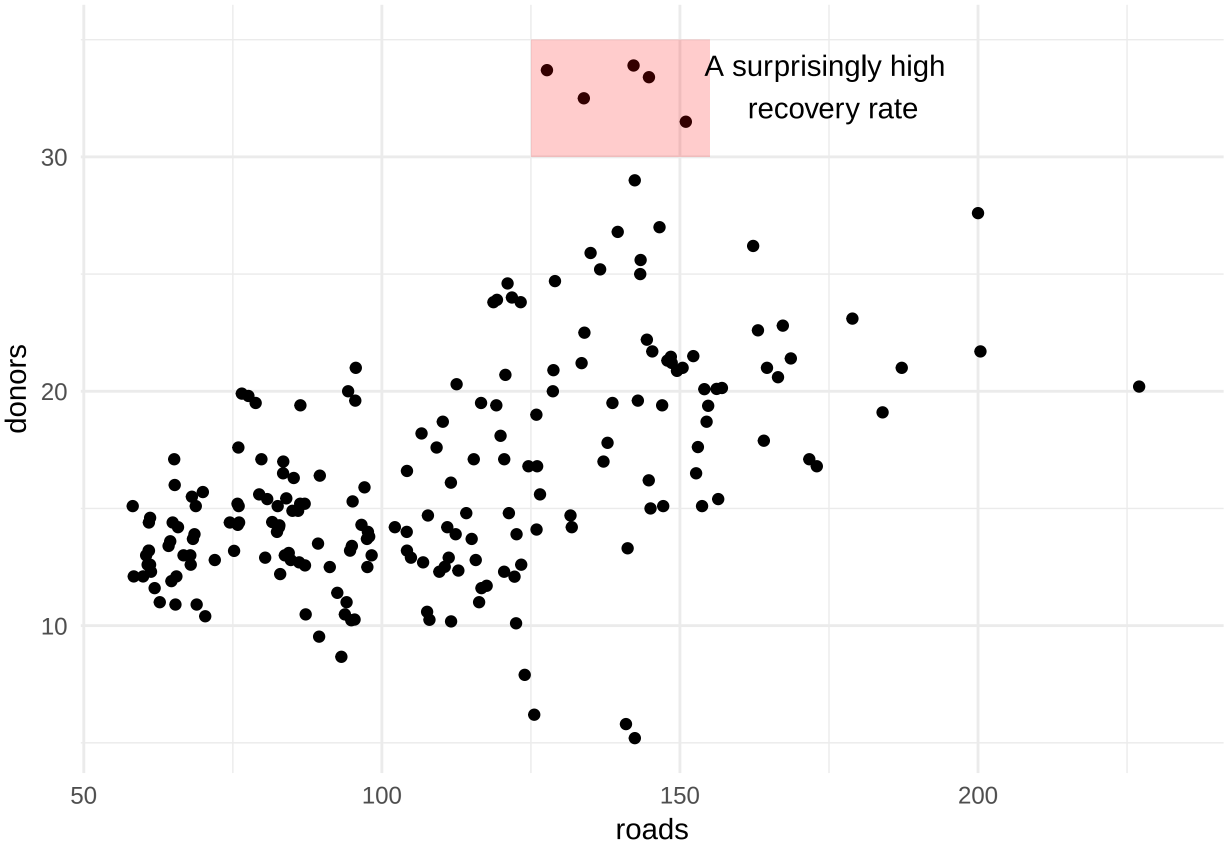
```
## Warning: Removed 34 rows containing missing values (geom_point).
```



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 makred 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.