Data Visualization

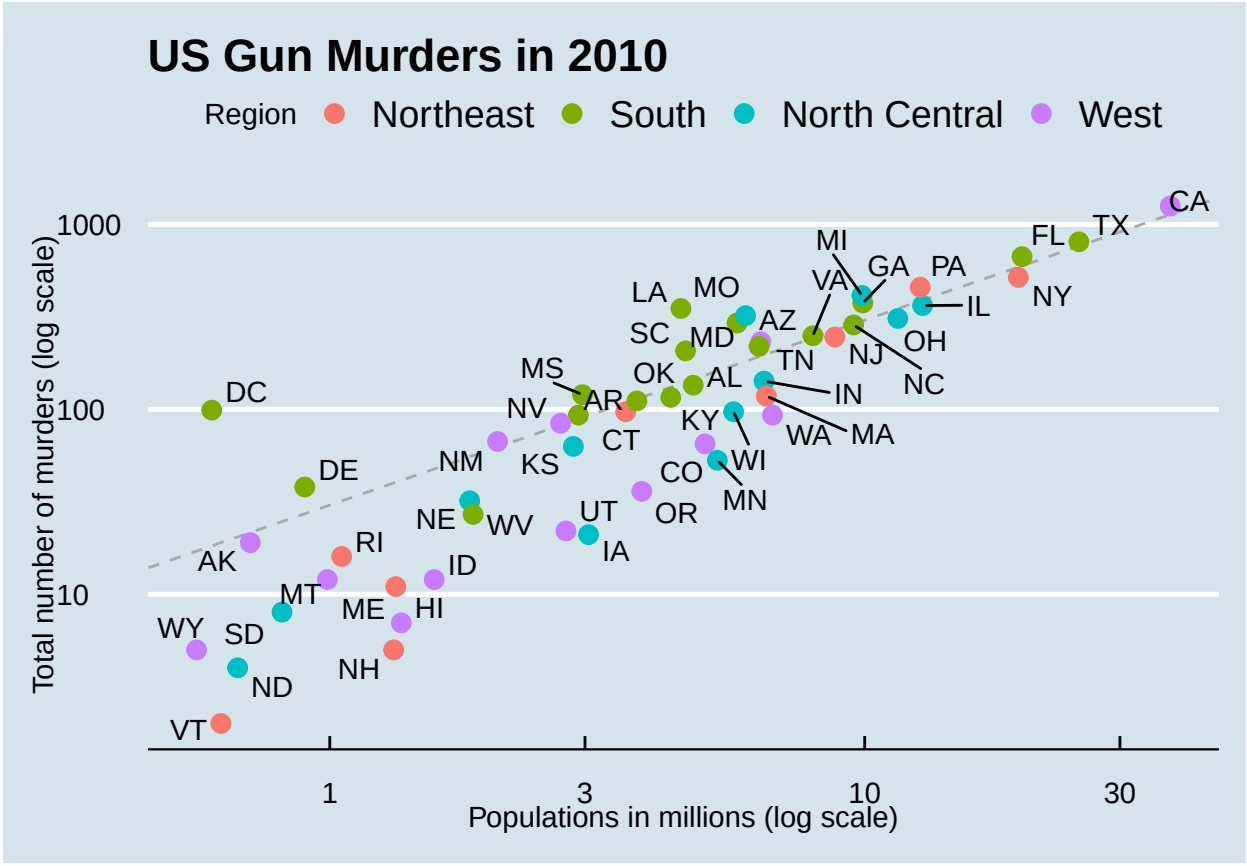
Data Visualization and Exploratory Data Analysis

Looking at the numbers and character strings that define a dataset is rarely useful. To convince yourself, print and stare at this data table:

```
library(tidyverse)
library(dslabs)
data(murders)
head(murders)
```

```
##        state abb region population total
## 1    Alabama  AL  South    4779736   135
## 2     Alaska  AK   West     710231    19
## 3    Arizona  AZ   West    6392017   232
## 4   Arkansas  AR  South    2915918    93
## 5 California  CA   West   37253956  1257
## 6   Colorado  CO   West    5029196    65
```

What do you learn from staring at this table? How quickly can you determine which states have the largest populations? Which states have the smallest? How large is a typical state? Is there a relationship between population size and total murders? How do murder rates vary across regions of the country? For most human brains it is quite difficult to extract this information just from looking at the numbers. In contrast, the answer to all the questions above are readily available from examining this plot



We are reminded of the saying "a picture is worth a thousand words". Data visualization provides a powerful way to communicate a data-driven finding. In some cases, the visualization is so convincing that no follow-up analysis is required. We also note that many widely used data analysis tools were initiated by discoveries made via exploratory data analysis (EDA). EDA is perhaps the most important part of data analysis, yet is often overlooked.

With the talks New Insights on Poverty and The Best Stats You've Ever Seen, Hans Rosling forced us to to notice the unexpected with a series of plots related to world health and economics. In his videos, he used animated graphs to show us how the world was changing and that old narratives are no longer true. We will use this data as an example to learn about `ggplot2` and data visualization.

It is also important to note that mistakes, biases, systematic errors and other unexpected problems often lead to data that should be handled with care. Failure to discover these problems often leads to flawed analyses and false discoveries. As an example, consider that measurement devices sometimes fail and that most data analysis procedures are not designed to detect these.

Yet, these data analysis procedures will still give you an answer. The fact that it can be hard or impossible to notice an error just from the reported results, makes data visualization particularly important.

Today we will learn the basics of the `ggplot2` package - the software we will use to learn the basics of data visualization and exploratory data analysis. We will use motivating examples and start by reproducing the murders by state example to learn the basics of `ggplot2`. Then we will cover world health and economics and infectious disease trends in the United States.

Note that there is much more to data visualization than what we cover here. More references include:

- ER Tufte (1983) The visual display of quantitative information. Graphics Press.
- ER Tufte (1990) Envisioning information. Graphics Press.
- ER Tufte (1997) Visual explanations. Graphics Press.

- A Gelman, C Pasarica, R Dodhia (2002) Let's practice what we preach: Turning tables into graphs. The American Statistician 56:121-130
- NB Robbins (2004) Creating more effective graphs. Wiley
- Rob Kabacoff (2018) Data Visualization with R

We will cover the basics of interactive graphics later in this course. If you want to check out interactive graphs now, below are some useful resources for learning more.

- https://shiny.rstudio.com/
- https://d3js.org/

A first introduction to `ggplot2`

We will be using the `ggplot2` package. We can load it, along with `dplyr`, as part of the tidyverse:

```
library(tidyverse)
```

One reason `ggplot2` is generally more intuitive for beginners is that it uses a *grammar of graphics*, the *gg* in `ggplot2`. This is analogous to the way learning grammar can help a beginner construct hundreds of different sentences by learning just a a handful of verbs, nouns and adjectives without having to memorize each specific sentence. Similarly, by learning a handful of `ggplot2` building blocks and its grammar, you will be able to create hundreds of different plots.

Another reason `ggplot2` makes it easier for beginners is that its default behavior is carefully chosen to satisfy the great majority of cases and are aesthetically pleasing. As a result, it is possible to create informative and elegant graphs with relatively simple and readable code.

One limitation is that ggplot is designed to work exclusively with data tables in which rows are observations and columns are variables. However, a substantial percentage of datasets that beginners work with are, or can be converted into, this format. An advantage of this approach is that assuming that our data follows this format simplifies the code and learning the grammar.

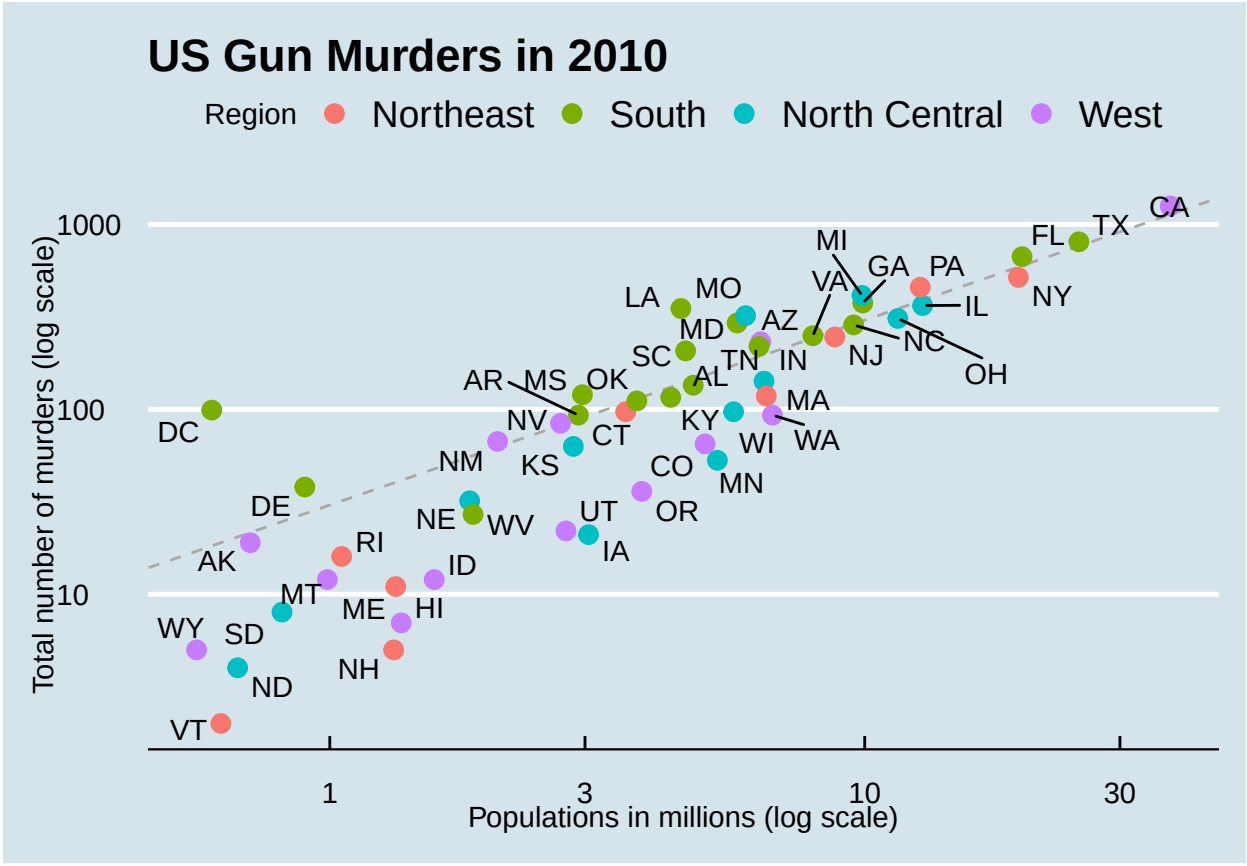
The Cheat Sheet

To use `ggplot2` you will have to learn several functions and arguments. These are hard to memorize so we highly recommend you have the a ggplot2 cheat sheet handy.

The components of a graph

We construct a graph that summarizes the US murders dataset.

```
library(dslabs)
data(murders)
```



We can clearly see how much states vary across population size and the total number of murders. Not surprisingly, we also see a clear relationship between murder totals and population size. A state falling on the dashed grey line has the same murder rate as the US average. The four geographic regions are denoted with color and depicts how most southern states have murder rates above the average.

This data visualization shows us pretty much all the information in the data table. The code needed to make this plot is relatively simple. We will learn to create the plot part by part.

The first step in learning `ggplot2` is to be able to break a graph apart into components. Let's break down this plot and introduce some of the `ggplot2` terminology. The three main components to note are:

1. **Data**: The US murders data table is being summarized. We refer to this as the **data** component.
2. **Geometry**: The plot above is a scatter plot. This is referred to as the **geometry** component. Other possible geometries are barplots, histograms, smooth densities, qqplots, and boxplots.
3. **Aesthetic mapping**: The x-axis values are used to display population size, the y-axis values are used to display the total number of murders, text is used to identify the states, and colors are used to denote the four different regions. These are the **aesthetic mappings** component. How we define the mapping depends on what **geometry** we are using.

We also note that:

4. The range of the x-axis and y-axis appears to be defined by the range of the data. They are both on log-scales. We refer to this as the **scale** component.
5. There are labels, a title, a legend, and we use the style of The Economist magazine for this particular plot.

We will now construct the plot piece by piece.

Creating a blank slate `ggplot` object

The first step in creating a `ggplot2` graph is to define a `ggplot` object. We do this with the function `ggplot` which initializes the graph. If we read the help file for this function we see that the first argument is used to specify which data is associated with this object:

```
ggplot(data = murders)
```

We can also pipe the data. So this line of code is equivalent to the one above:

```
murders %>% ggplot()
```


Note that it renders a plot, in this case a blank slate since no geometry has been defined. The only style choice we see is a grey background.

What has happened above is that the object was created and because it was not assigned, it was automatically evaluated. But note that we can define an object, for example like this:

```
p <- ggplot(data = murders)
class(p)
```

```
## [1] "gg"     "ggplot"
```

To render the plot associated with this object we simply print the object `p`. The following two lines of code produce the same plot we see above:

```
print(p)
p
```

Layers

In ggplot we create graphs by adding *layers*. Layers can define geometries, compute summary statistics, define what scales to use, or even change styles. To add layers, we use the the symbol `+`. In general a line of code will look like this:

DATA %>% `ggplot()` + LAYER 1 + LAYER 2 + ... + LAYER N

Usually, the first added layer defines the geometry. We want to make a scatter plot. So what geometry do we use?

Geometry Taking a quick look at the cheat sheet we see that the function used to create plots with this geometry is `geom_point`.

We will see that geometry function names follow this pattern: `geom` and the name of the geometry connected by an underscore. For `geom_point` to know what to do, we need to provide data and a mapping. We have already connected the object `p` with the `murders` data table and if we add as a layer `geom_point` we will default to using this data. To find out what mappings are expected we read the **Aesthetics** section of the `geom_point` help file:

> Aesthetics
>
> geom_point understands the following aesthetics:
>
> x
>
> y
>
> alpha
>
> colour

and, as expected, we see that at least two arguments are required: `x` and `y`.

`aes`

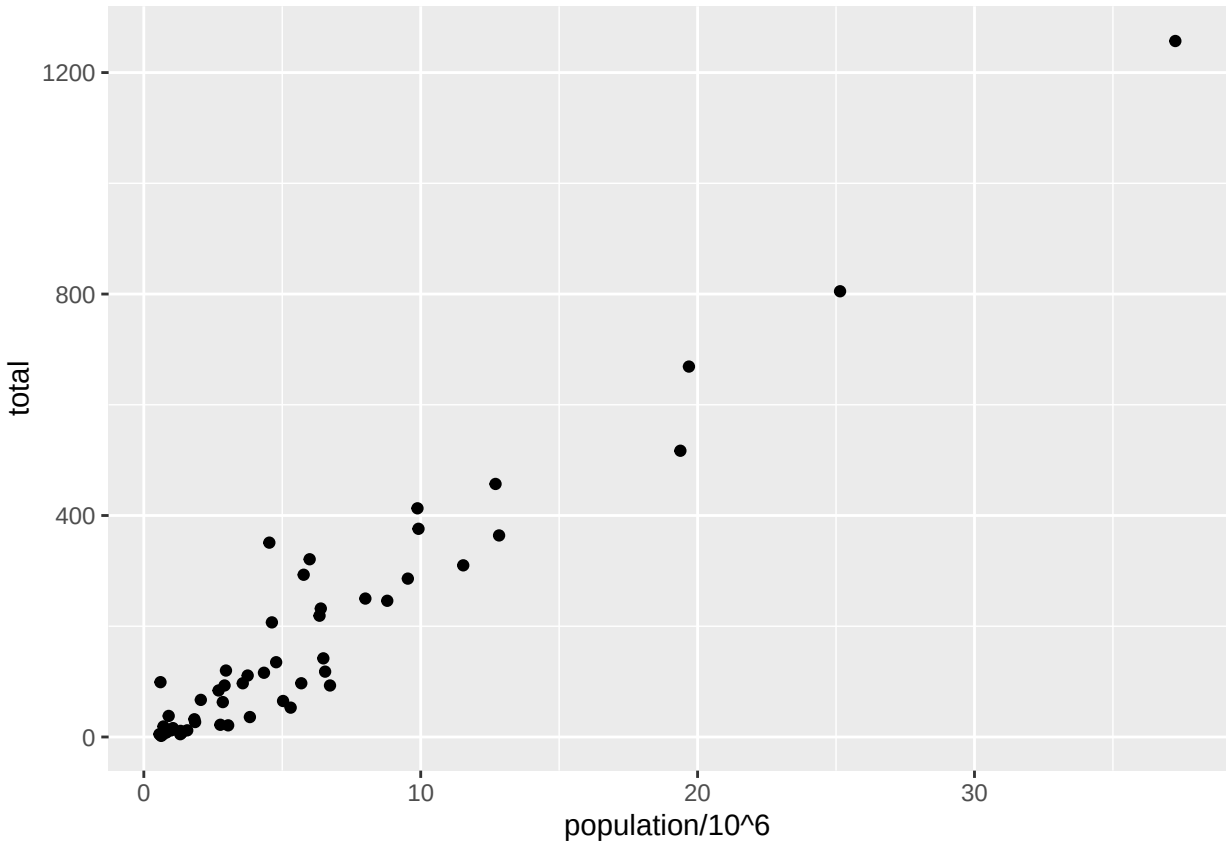
`aes` will be one of the functions that you will most use. The function connects data with what we see on the graph. We refer to this connection as the **aesthetic mappings**. The outcome of this function is often used as the argument of a geometry function. This example produces a scatter plot of total murders versus population in millions:

```
murders %>% ggplot() +
            geom_point(aes(x = population/10^6, y = total))
```

Note that we can drop the `x =` and `y =` if we wanted to as these are the first and second expected arguments as seen on the help page.

Also note that we can add a layer to the `p` object that was defined above as `p <- ggplot(data = murders)`:

```
p <- murders %>% ggplot()

p + geom_point(aes(population/10^6, total))
```



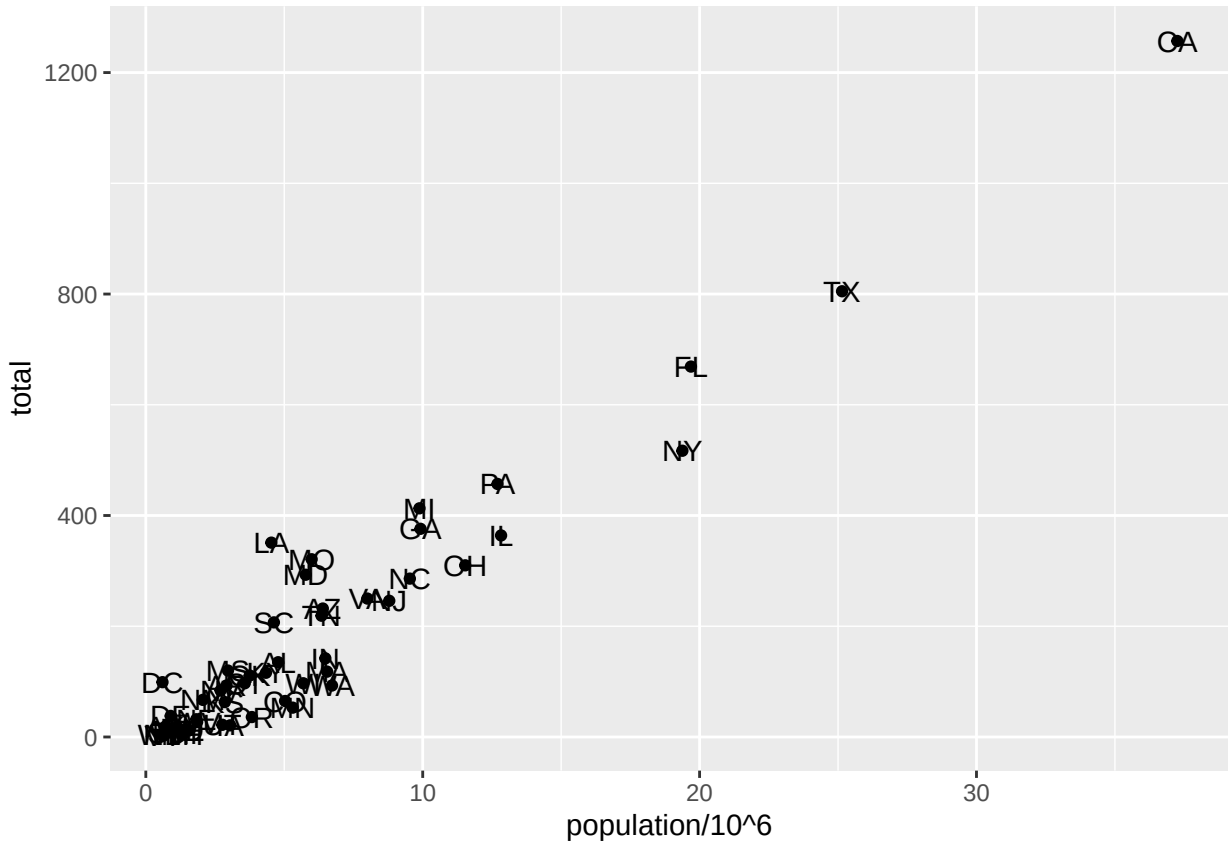
Note that the scale and labels are defined by default when adding this layer. Also notice that we use the variable names from the object component: `population` and `total`.

Keep in mind that the behavior of recognizing the variables from the data component is quite specific to `aes`. With most functions, if you try to access the values of `population` or `total` outside of `aes` you receive an error.

Adding other layers A second layer in the plot we wish to make involves adding a label to each point to identify the state. The `geom_label` and `geom_text` functions permit us to add text to the plot, without and with a rectangle behind the text respectively.

Because each state (each point) has a label we need an aesthetic mapping to make the connection. By reading the help file we learn that we supply the mapping between point and label through the `label` argument of `aes`. So the code looks like this:

```
p + geom_point(aes(population/10^6, total)) +
    geom_text(aes(population/10^6, total, label = abb))
```



We have successfully added a second layer to the plot.

As an example of the unique behavior of `aes` mentioned above, note that this call

```
p_test <- p + geom_text(aes(population/10^6, total, label = abb))
```

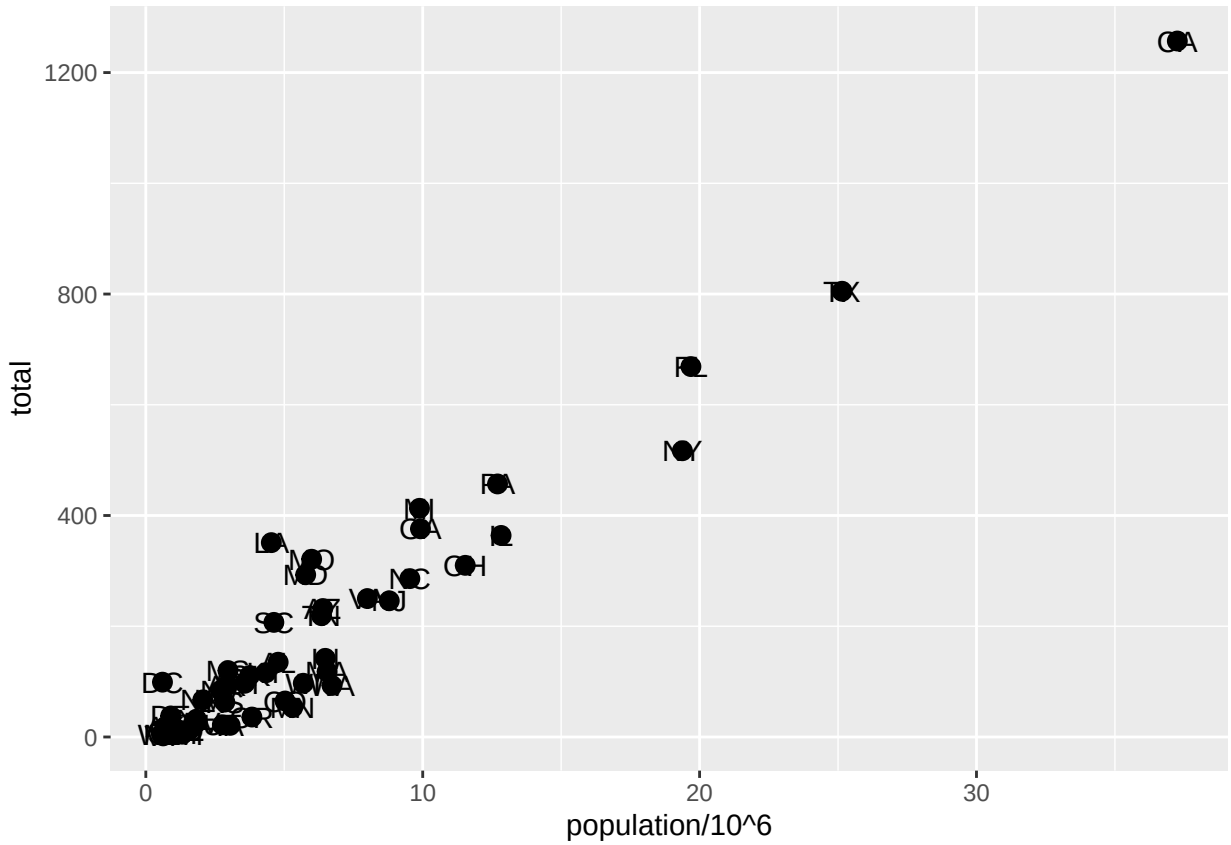
is fine, this call

```
p_test <- p + geom_text(aes(population/10^6, total), label = abb)
```

will give you an error as `abb` is not found once it is outside of the `aes` function and `geom_text` does not know where to find `abb` as it is not a global variable.

Tinkering with other arguments Note that each geometry function has many arguments other than `aes` and `data`. They tend to be specific to the function. For example, in the plot we wish to make, the points are larger than the default ones. In the help file we see that `size` is an aesthetic and we can change it like this:

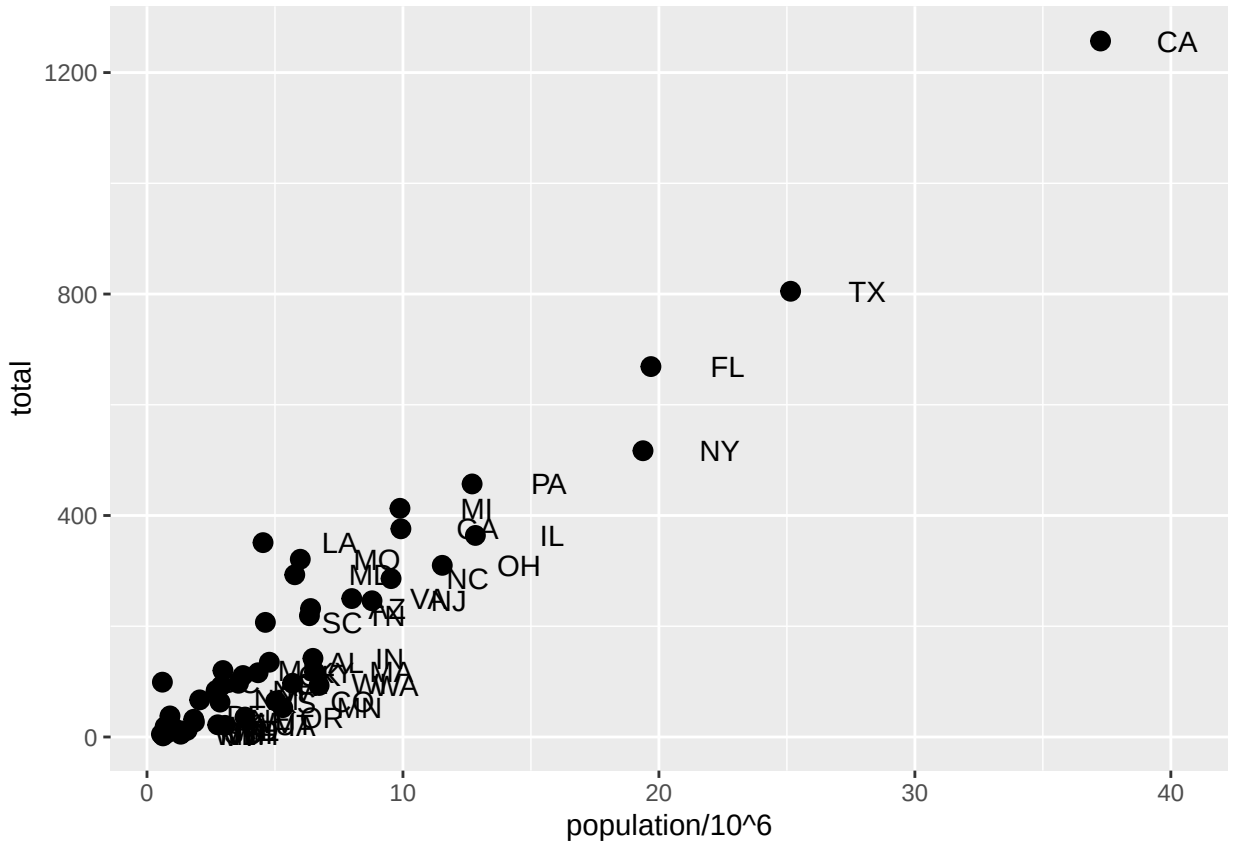
```
p + geom_point(aes(population/10^6, total), size = 3) +
    geom_text(aes(population/10^6, total, label = abb))
```



Note that `size` is **not** a mapping, it affects all the points so we do not need to include it inside `aes`.

Now that the points are larger, it is hard to see the labels. If we read the help file for `geom_text` we learn of the `nudge_x` argument which moves the text slightly to the right:

```
p + geom_point(aes(population/10^6, total), size = 3) +
    geom_text(aes(population/10^6, total, label = abb), nudge_x = 3)
```



This is preferred as it makes it easier to read the text.

Global aesthetic mappings

Note that in the previous line of code, we define the mapping `aes(population/10^6, total)` twice, once in each geometry. We can avoid this by using a *global* aesthetic mapping. We can do this when we define the blank slate `ggplot` object. Remember that the function `ggplot` contains an argument that permits us to define aesthetic mappings:

```
args(ggplot)
```

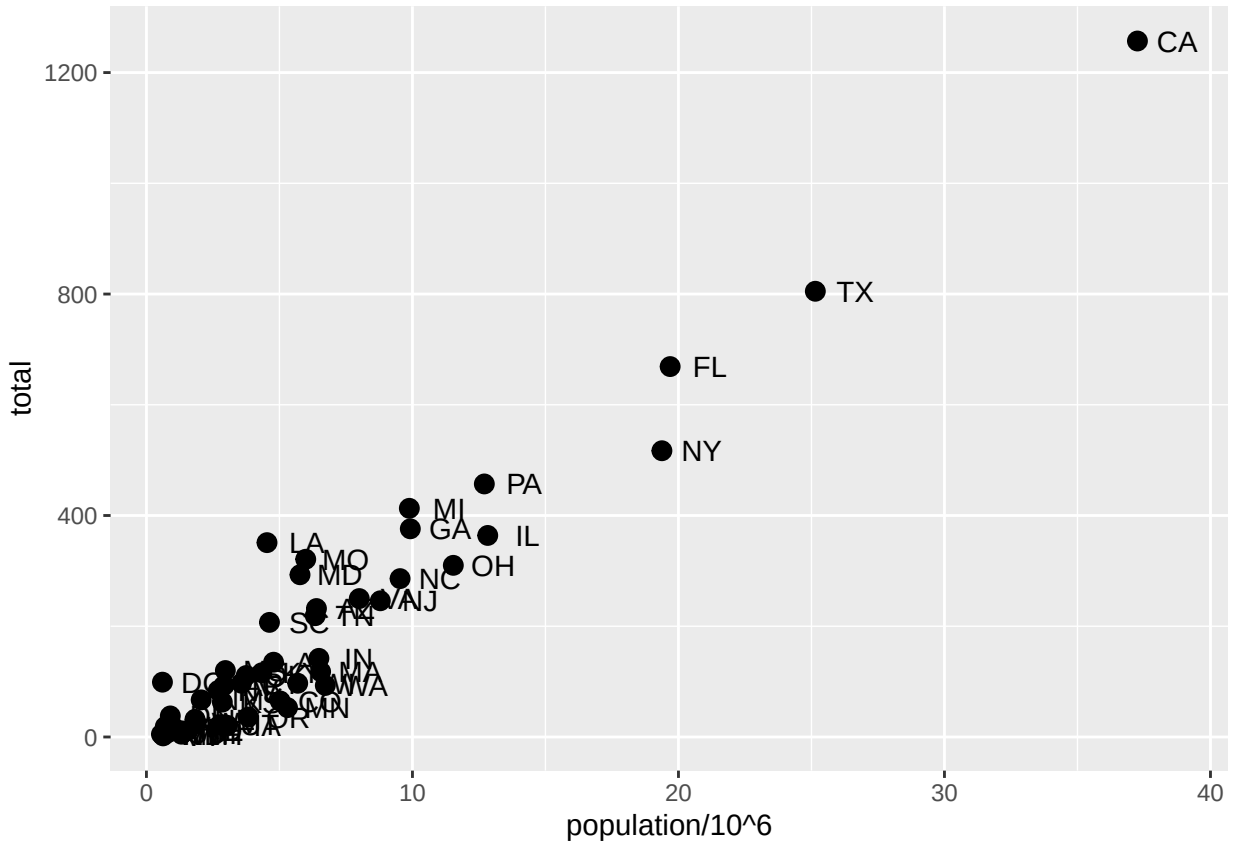
```
## function (data = NULL, mapping = aes(), ..., environment = parent.frame())
## NULL
```

If we define a mapping in `ggplot`, then all the geometries that are added as layers will default to this mapping. We redefine `p`:

```
p <- murders %>%
     ggplot(aes(x = population/10^6, y = total, label = abb))
```

and then we can simply use code like this:

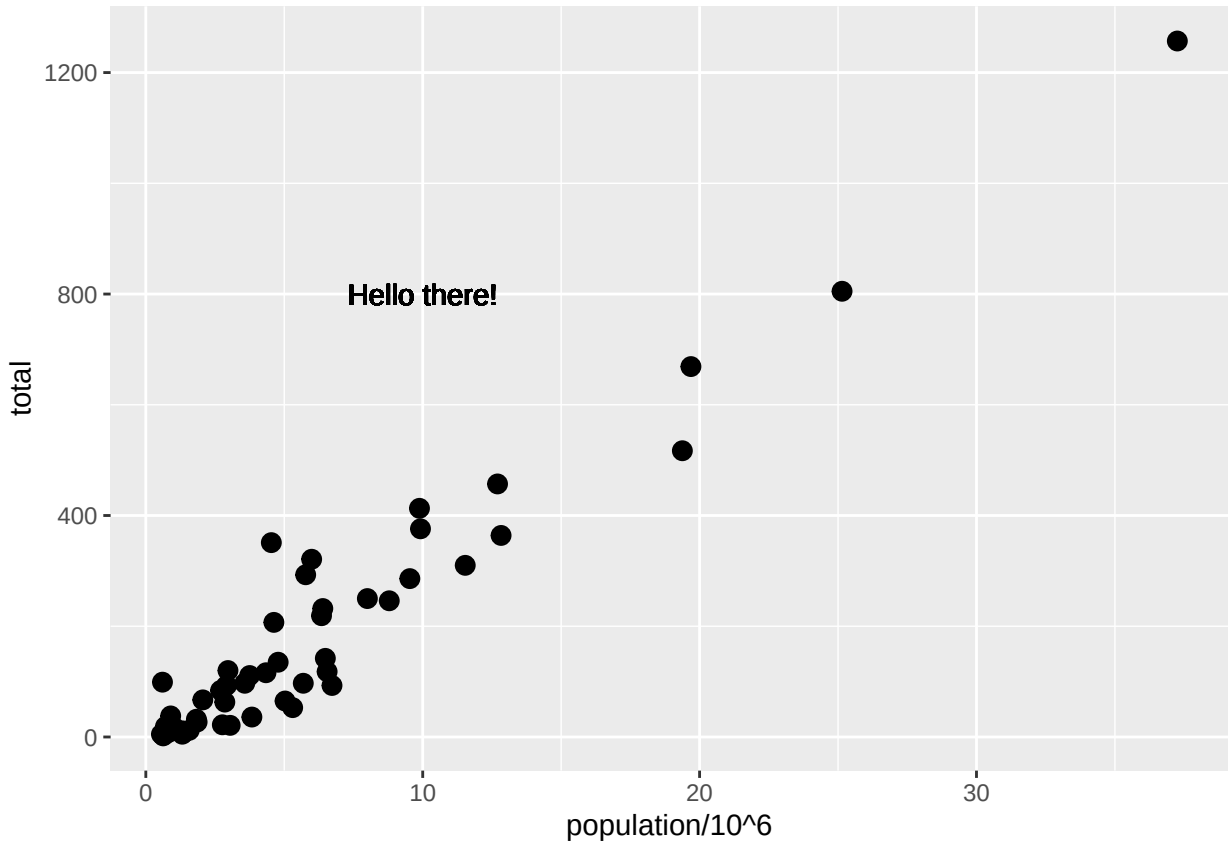
```
p + geom_point(size = 3) +
    geom_text(nudge_x = 1.5)
```



We keep the `size` and `nudge_x` argument in `geom_point` and `geom_text` respectively because we only want to increase the size of points and nudge only the labels. Also note that the `geom_point` function does not need a `label` argument and therefore ignores it.

Local aesthetic mappings overide global ones If we need to, we can override the global mapping by defining a new mapping within each layer. These *local* definitions override the *global*. Here is an example:

```
p + geom_point(size = 3) +
    geom_text(aes(x = 10, y = 800, label = "Hello there!"))
```

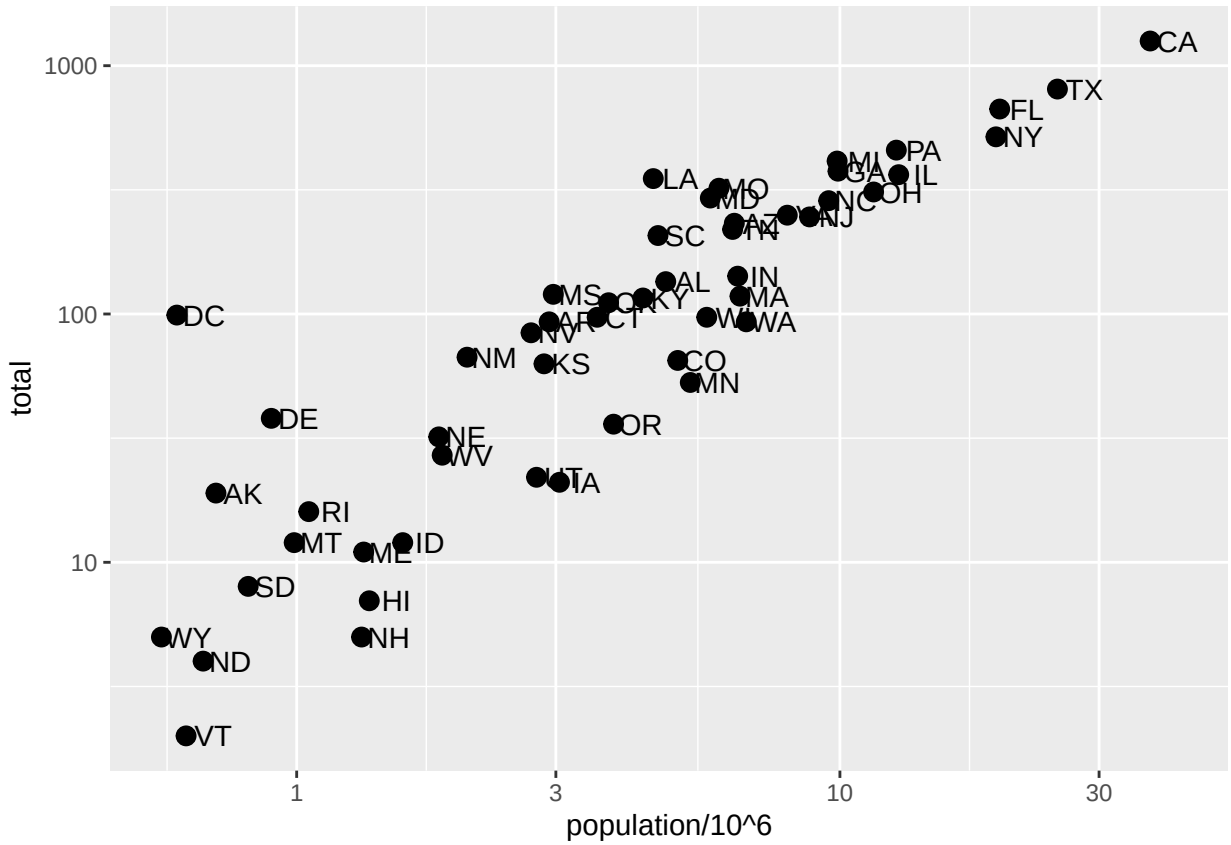


Clearly, the second call to `geom_text` does not use `population` and `total` on the `x` and `y` axis.

Scales

Recall that our desired scales are in log-scale. This is not the default so this change needs to be added through a *scales* layer. A quick look at the cheat sheet reveals `scale_x_continuous` is needed to edit the behavior of scales. We use it like this:

```
p + geom_point(size = 3) +
    geom_text(nudge_x = 0.05) +
    scale_x_continuous(trans = "log10") +
    scale_y_continuous(trans = "log10")
```



Because we are in the log-scale now, the *nudge* must be made smaller.

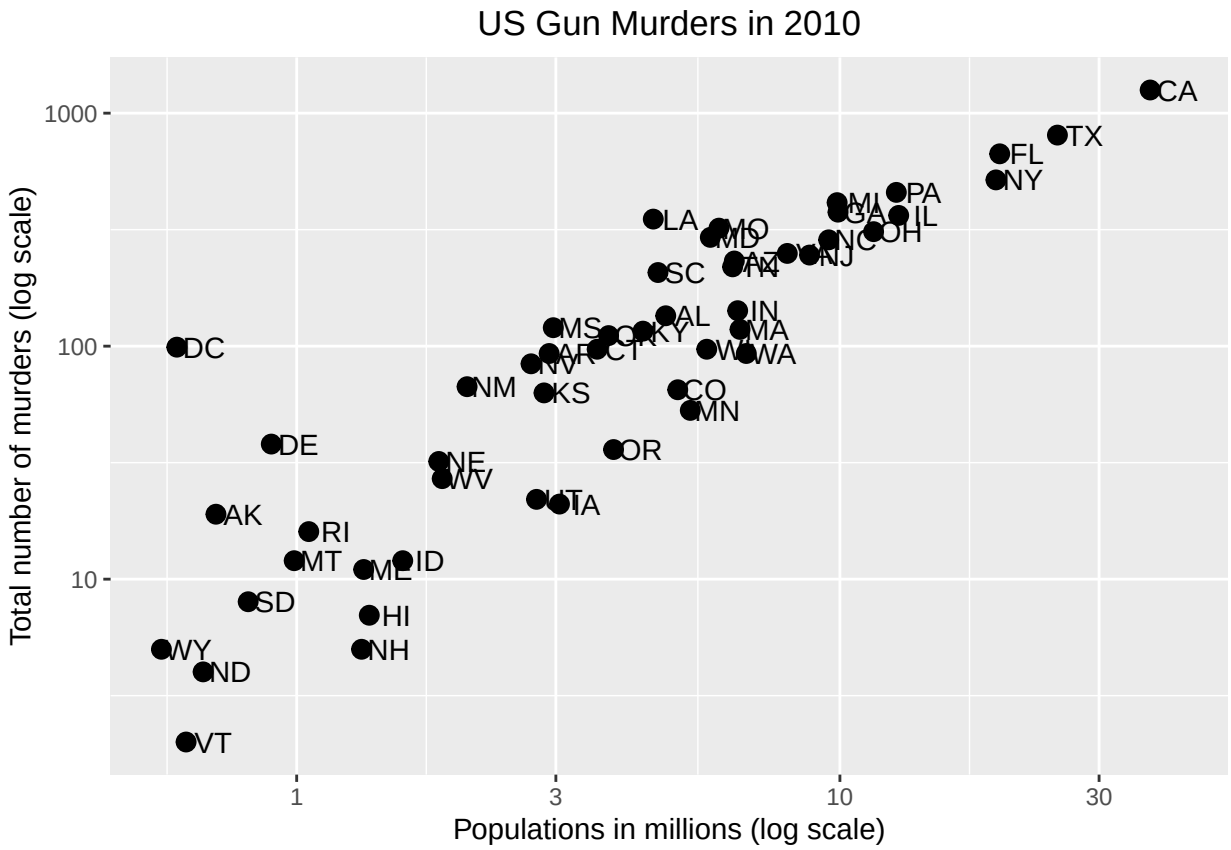
This particular transformation is so common that `ggplot` provides specialized functions:

```
p + geom_point(size = 3) +
    geom_text(nudge_x = 0.05) +
    scale_x_log10() +
    scale_y_log10()
```

Labels and Titles

Similarly, the cheat sheet quickly reveals that to change labels and add a title we use the following functions: `xlab`, `ylab` and `ggtitle`.

```
p + geom_point(size = 3) +
    geom_text(nudge_x = 0.05) +
    scale_x_log10() +
    scale_y_log10() +
    xlab("Populations in millions (log scale)") +
    ylab("Total number of murders (log scale)") +
    ggtitle("US Gun Murders in 2010") +
    theme(plot.title = element_text(hjust = 0.5))
```



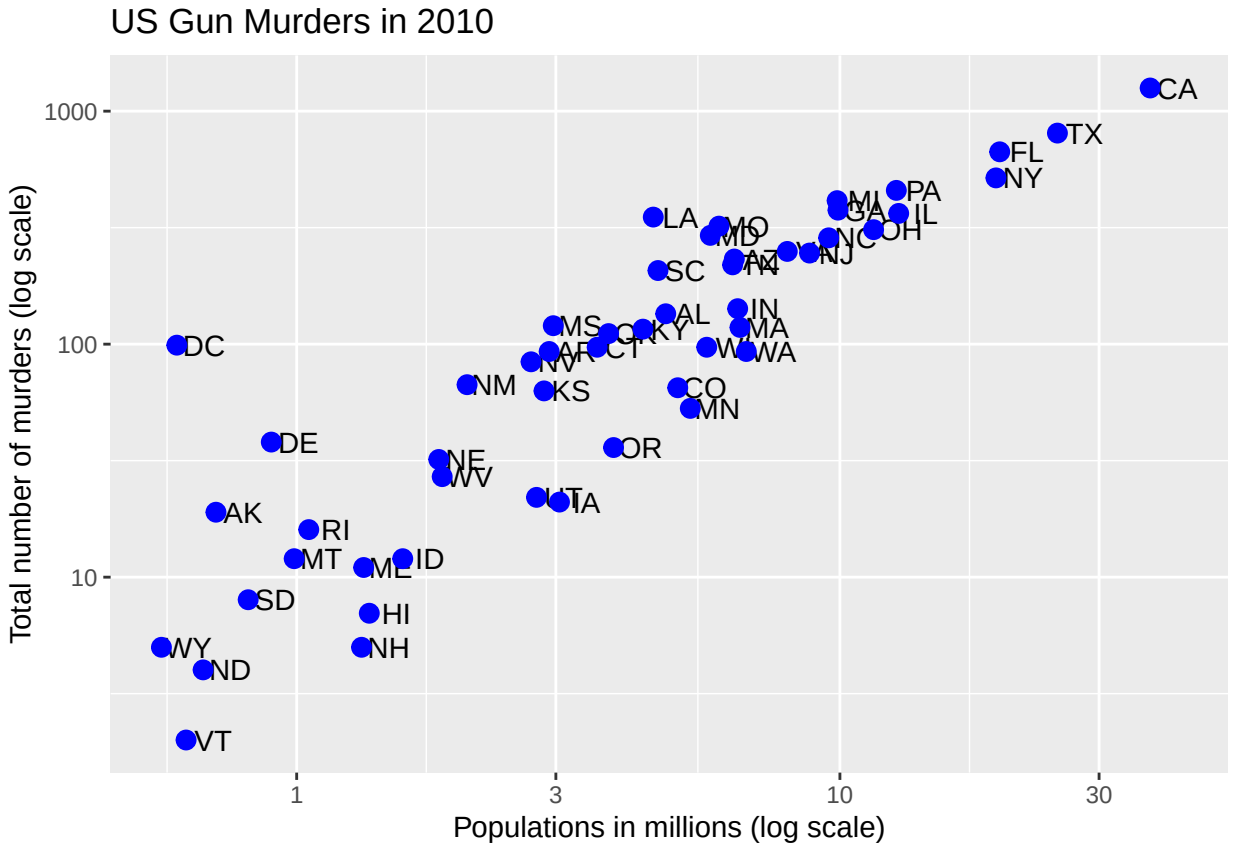
We are almost there! All we have to do is add color, a legend and optional changes to the style.

Categories as colors Note that we can change the color of the points using the `color` argument in the `geom_point` function. To facilitate exposition we will redefine `p` to be everything except the points layer:

```
p <-  murders %>%
      ggplot(aes(population/10^6, total, label = abb)) +
      geom_text(nudge_x = 0.05) +
      scale_x_log10() +
      scale_y_log10() +
      xlab("Populations in millions (log scale)") +
      ylab("Total number of murders (log scale)") +
      ggtitle("US Gun Murders in 2010")
```

and then test out what happens by adding different calls to `geom_point`. We can make all the points blue by adding the `color` argument:

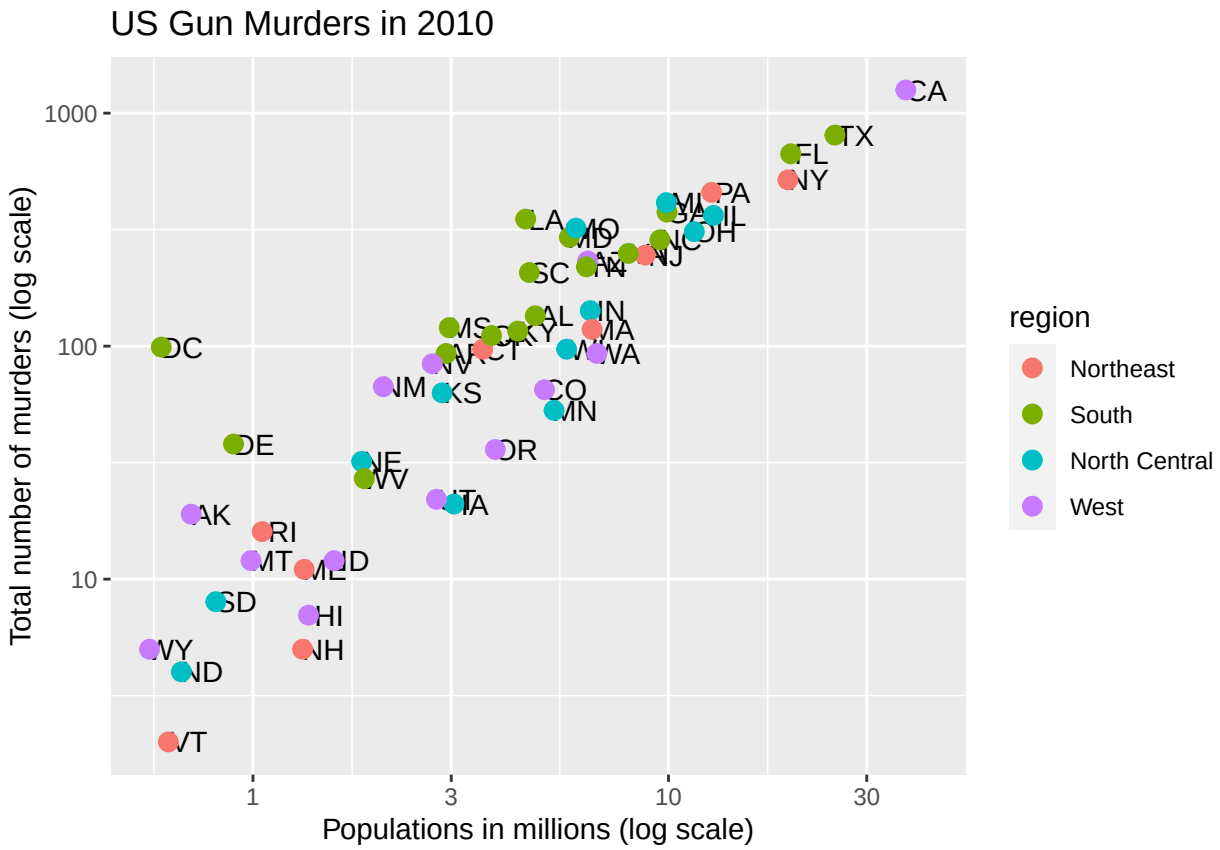
```
p + geom_point(size = 3, color ="blue")
```



This, of course, is not what we want. We want to assign color depending on the geographical region. A nice default behavior of `ggplot2` is that if we assign a categorical variable to color, it automatically assigns a different color to each category. It also adds a legend!

To map each point to a color, we need to use `aes` since this is a mapping. So we use the following code:

```
p + geom_point(aes(color = region), size = 3)
```



The `x` and `y` mappings are inherited from those already defined in `p`. So we do not redefine them. We also move `aes` to the first argument since that is where the mappings are expected in this call.

Here we see yet another useful default behavior: `ggplot2` has automatically added a legend that maps color to region.

Adding a line

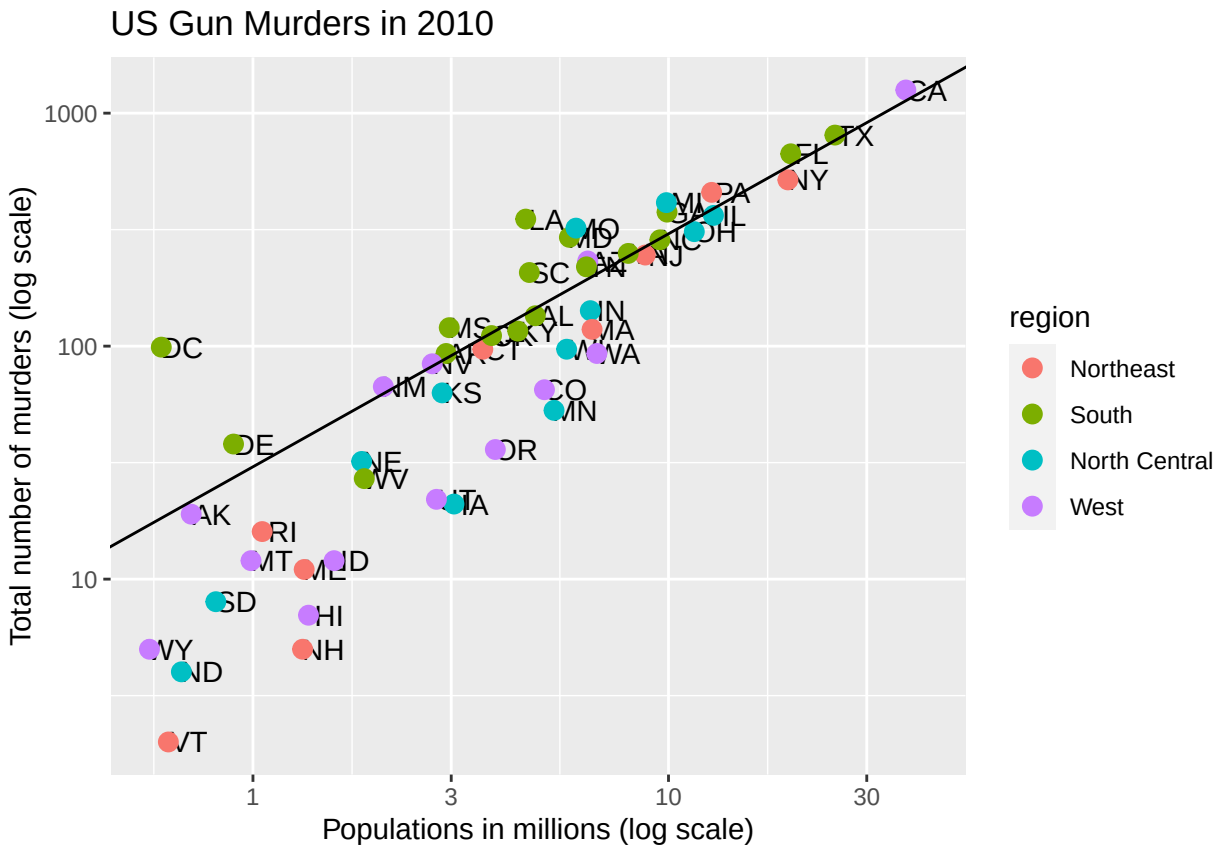
We want to add a line that represents the average murder rate for the entire country. Note that once we determine the per million rate to be r, this line is defined by the formula: $y = rx$ with y and x our axes: total murders and population in millions respectively. In the log-scale this line turns into: $\log(y) = \log(r) + \log(x)$. So in our plot it's a line with slope 1 and intercept $\log(r)$. To compute this value we use our `dplyr` skills:

```
r <- murders %>%
     summarize(murder_rate = sum(total) /  sum(population) * 10^6) %>% .$murder_rate
r
```

```
## [1] 30.34555
```

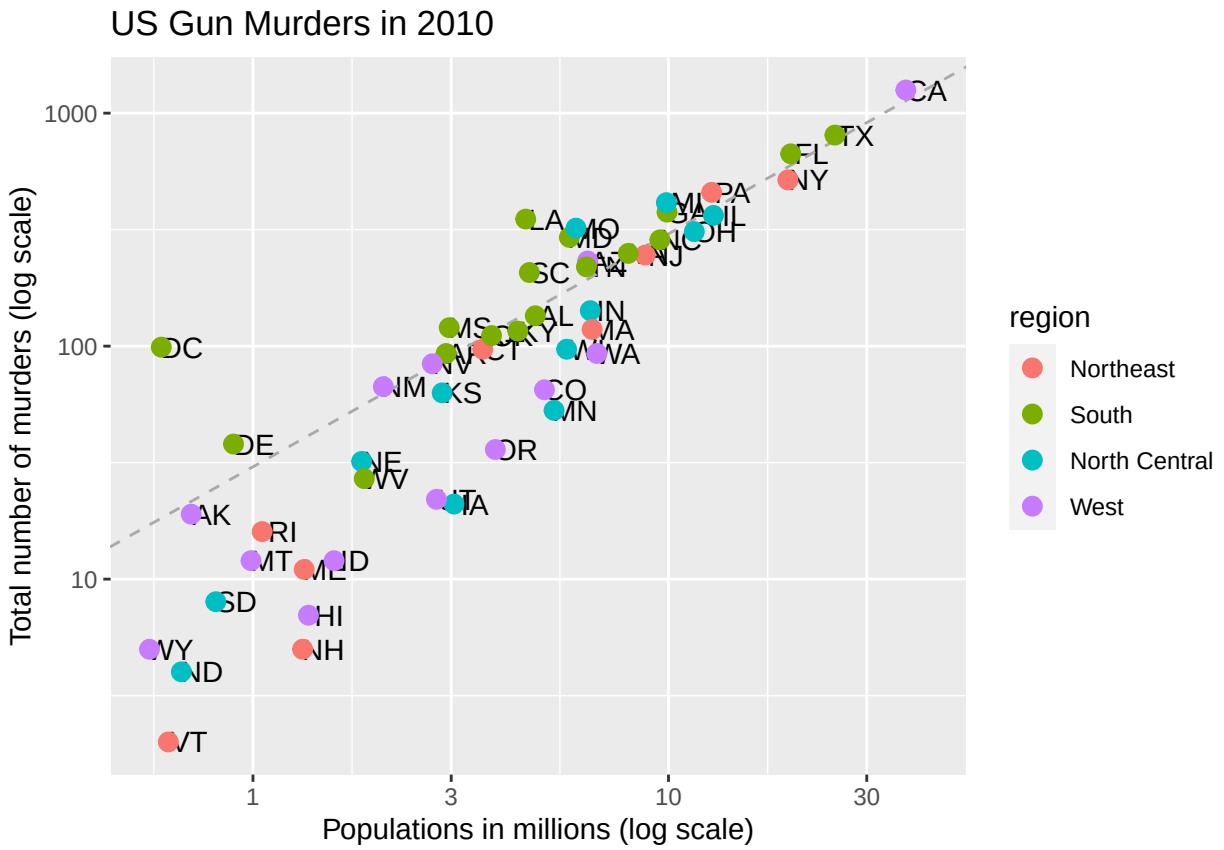
To add a line we use the `geom_abline` function. `ggplot` uses `ab` in the name to remind us we are supplying the intercept (`a`) and slope (`b`). The default line has slope 1 and intercept 0 so we only have to define the intercept:

```
p + geom_point(aes(col=region), size = 3) +
    geom_abline(intercept = log10(r))
```



We can change the line type and color of the lines using arguments. We also draw it first so it doesn't go over our points.

```
p <- p + geom_abline(intercept = log10(r), lty = 2, color = "darkgrey") +
         geom_point(aes(color = region), size = 3)
p
```



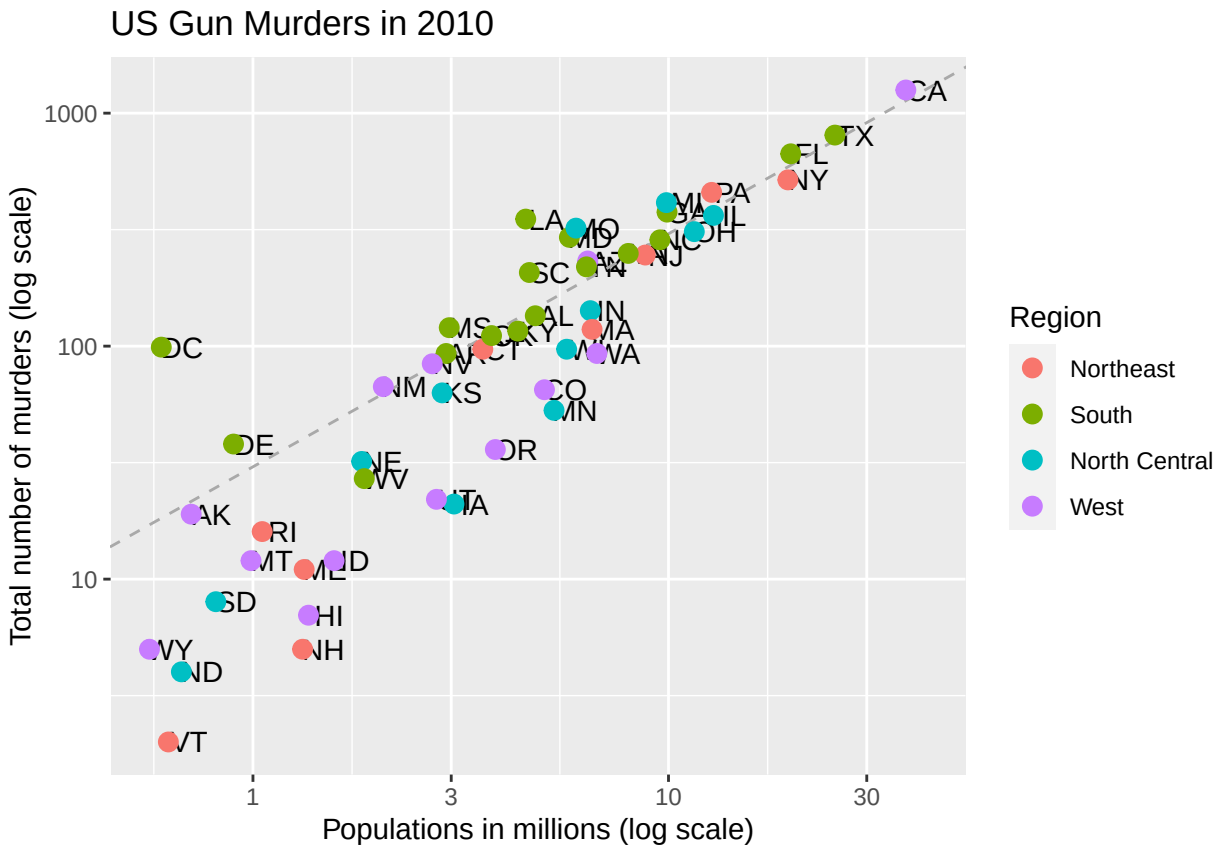
Note that we redefined `p`.

Other adjustments

The default plots created by `ggplot` are already very useful. But often, we need to make minor tweaks to the default behavior. Although it is not always obvious how to make these even with the cheat sheet, `ggplot2` is very flexible.

For example, note that we can make changes to the legend via the `scale_color_discrete` function. For example, in our plot the word *region* is not capitalized. We can change that like this:

```
p <- p + scale_color_discrete(name = "Region")
p
```



Add-on packages

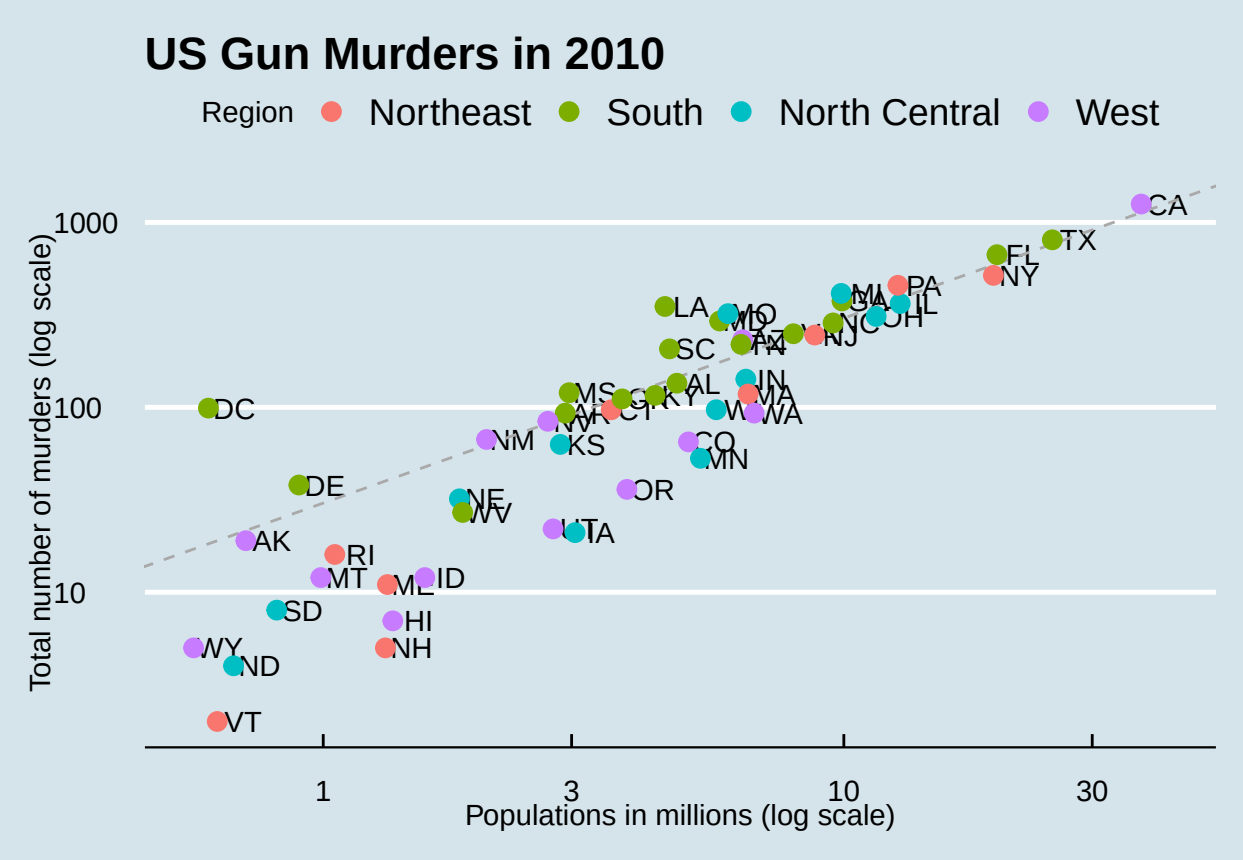
The power of `ggplot2` is augmented further due to the availability of add-on packages. The remaining changes required to put the finishing touches on our plot require the `ggthemes` and `ggrepel` packages.

The style of a `ggplot` graph can be changed using the `theme` functions. Several themes are included as part of the `ggplot2` package. In fact, for most of the plots in this course we use a function in the `dslabs` package that automatically sets a default theme:

```
ds_theme_set()
```

Many other themes are added by the package `ggthemes`. Among those are the `theme_economist` theme that we used. After installing the package, you can change the style of the plot by adding a layer:

```
library(ggthemes)
p + theme_economist()
```



You can see how some of the other themes look like by simply changing the function. For example you might try the `theme_fivethirtyeight()` theme instead.

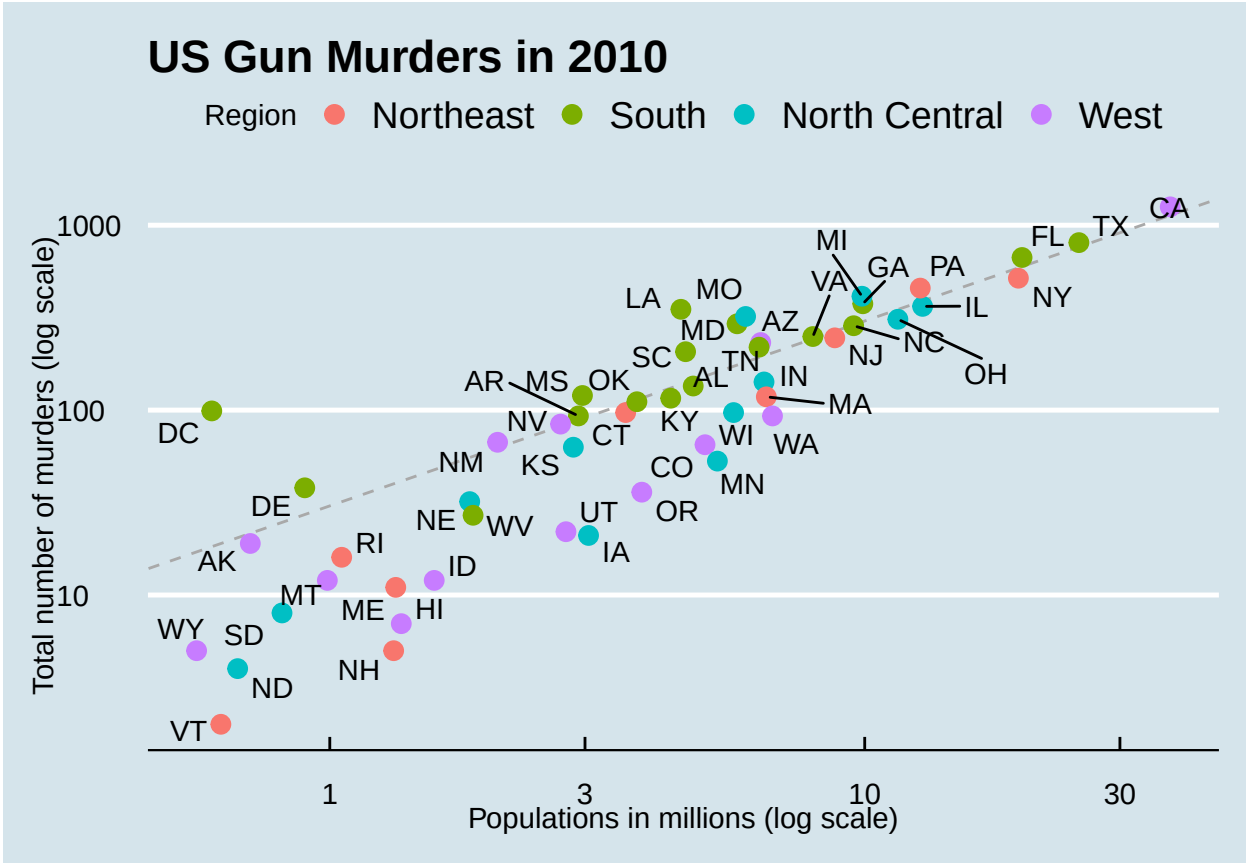
The final difference has to do with the position of the labels. Note that in our plot, some of the labels fall on top of each other. The add-on package `ggrepel` includes a geometry that adds labels ensuring that they don't fall on top of each other. We simply change `geom_text` with `geom_text_repel`.

Putting it all together

So now that we are done testing we can write one piece of code that produces our desired plot from scratch.

```
library(ggthemes)
library(ggrepel)

### First define the slope of the line
r <- murders %>%
     summarize(murder_rate= sum(total) /  sum(population) * 10^6) %>% .$murder_rate

## Now make the plot
murders %>% ggplot(aes(population/10^6, total, label = abb)) +
  geom_abline(intercept = log10(r), lty = 2, color = "darkgrey") +
  geom_point(aes(col=region), size = 3) +
  geom_text_repel() +
  scale_x_log10() +
  scale_y_log10() +
  xlab("Populations in millions (log scale)") +
  ylab("Total number of murders (log scale)") +
```

```
ggtitle("US Gun Murders in 2010") +
scale_color_discrete(name = "Region") +
theme_economist()
```



Data Visualization Case Study: Trends in World Health and Economics

In this section we will demonstrate how relatively simple `ggplot` code can create insightful and aesthetically pleasing plots that help us better understand trends in world health and economics. We later augment the code somewhat to perfect the plots and describe some general principles for data visualization.

Example 1: Life Expectancy and Fertility Rates

Hans Rosling was the co-founder of the Gapminder Foundation, an organization dedicated to educating the public by using data to dispel common myths about the so-called developing world. The organization uses data to show how actual trends in health and economics contradict the narratives that emanate from sensationalist media coverage of catastrophes, tragedies and other unfortunate events. As stated on the Gapminder Foundation's website:

> Journalists and lobbyists tell dramatic stories. That's their job. They tell stories about extraordinary events and unusual people. The piles of dramatic stories pile up in peoples' minds into an over-dramatic worldview and strong negative stress feelings: "The world is getting worse!", "It's we vs. them!",

> "Other people are strange!", "The population just keeps growing!" and "Nobody cares!"

Hans Rosling conveyed actual data-based trends in a dramatic way of his own, using effective data visualization. This section is based on two talks that exemplify this approach to education: New Insights on Poverty and The Best Stats You've Ever Seen. Specifically, in this section, we set out to answer the following two questions using data:

1. Is it a fair characterization of today's world to say it is divided into western rich nations and the developing world in Africa, Asia and Latin America?
2. Has income inequality across countries worsened during the last 40 years?

To answer these questions we will be using the `gapminder` dataset provided in `dslabs`. This dataset was created using a number of spreadsheets available from the Gapminder Foundation. You can access the table like this:

```
library(dslabs)
data(gapminder)
head(gapminder)
```

```
##               country year infant_mortality life_expectancy fertility
## 1             Albania 1960           115.40           62.87      6.19
## 2             Algeria 1960           148.20           47.50      7.65
## 3              Angola 1960           208.00           35.98      7.32
## 4 Antigua and Barbuda 1960               NA           62.97      4.43
## 5           Argentina 1960            59.87           65.39      3.11
## 6             Armenia 1960               NA           66.86      4.55
##   population          gdp continent          region
## 1    1636054           NA    Europe Southern Europe
## 2   11124892  13828152297    Africa Northern Africa
## 3    5270844           NA    Africa   Middle Africa
## 4      54681           NA  Americas       Caribbean
## 5   20619075 108322326649  Americas   South America
## 6    1867396           NA      Asia    Western Asia
```

Hans Rosling's Quiz As done in the *New Insights on Poverty* video, we start by testing our knowledge regarding differences in child mortality across different countries.

For each of the six pairs of countries below, which country do you think had the highest child mortality in 2015? Which pairs do you think are most similar?

1. Sri Lanka or Turkey
2. Poland or South Korea
3. Malaysia or Russia
4. Pakistan or Vietnam
5. Thailand or South Africa

When answering these questions without data, the non-European countries are typically picked as having higher mortality rates: Sri Lanka over Turkey, South Korea over Poland, and Malaysia over Russia. It is also common to assume that countries considered to be part of the developing world, Pakistan, Vietnam, Thailand and South Africa, have similarly high mortality rates.

To answer these questions **with data** we can use `dplyr`. For example, for the first comparison we see that Turkey has the higher rate.

```
library(tidyverse)
gapminder %>% filter(year == 2015 & country %in% c("Sri Lanka","Turkey")) %>%
              select(country, infant_mortality)
```

```
##     country infant_mortality
## 1 Sri Lanka              8.4
## 2    Turkey             11.6
```

We can use this code on all comparisons and find the following:

Country_1	Infant_Mortality_1	Country_2	Infant_Mortality_2
Sri Lanka	8.4	Turkey	11.6
Poland	4.5	South Korea	2.9
Malaysia	6.0	Russia	8.2
Pakistan	65.8	Vietnam	17.3
Thailand	10.5	South Africa	33.6

We see that the European countries have higher rates: Poland has a higher rate than South Korea, and Russia has a higher rate than Malaysia. We also see that Pakistan has a much higher rate than Vietnam and South Africa a much higher rate than Thailand. It turns out that most people do worse if they are guessing, which implies we are more than ignorant, we are misinformed.

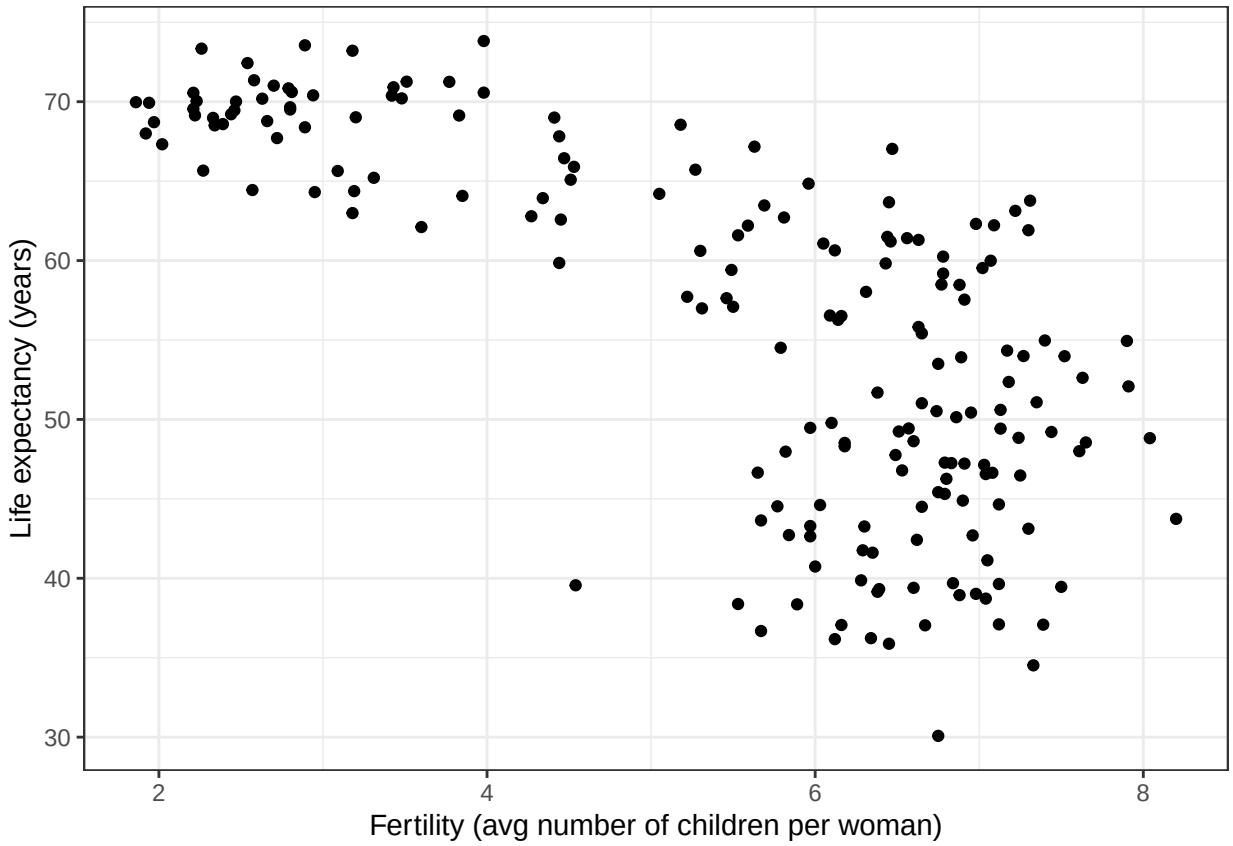
Life expectancy and fertility rates

The reason for this stems from the preconceived notion that the world is divided into two groups: the western world (Western Europe and North America), characterized by long life spans and small families, versus the developing world (Africa, Asia, and Latin America) characterized by short life spans and and large families. But, does the data support this dichotomous view of two groups?

The necessary data to answer this question is also available in our `gapminder` table. Using our newly learned data visualization skills we will be able to answer this question.

The first plot we make to see what data have to say about this world view is a scatter plot of life expectancy versus fertility rates (average number of children per woman). We will start by looking at data from about 50 years ago, when perhaps this view was cemented in our minds.

```
ds_theme_set()

gapminder %>% filter(year == 1962) %>%
  ggplot(aes(fertility, life_expectancy)) +
  geom_point() +
  ylab("Life expectancy (years)") +
  xlab("Fertility (avg number of children per woman)")
```

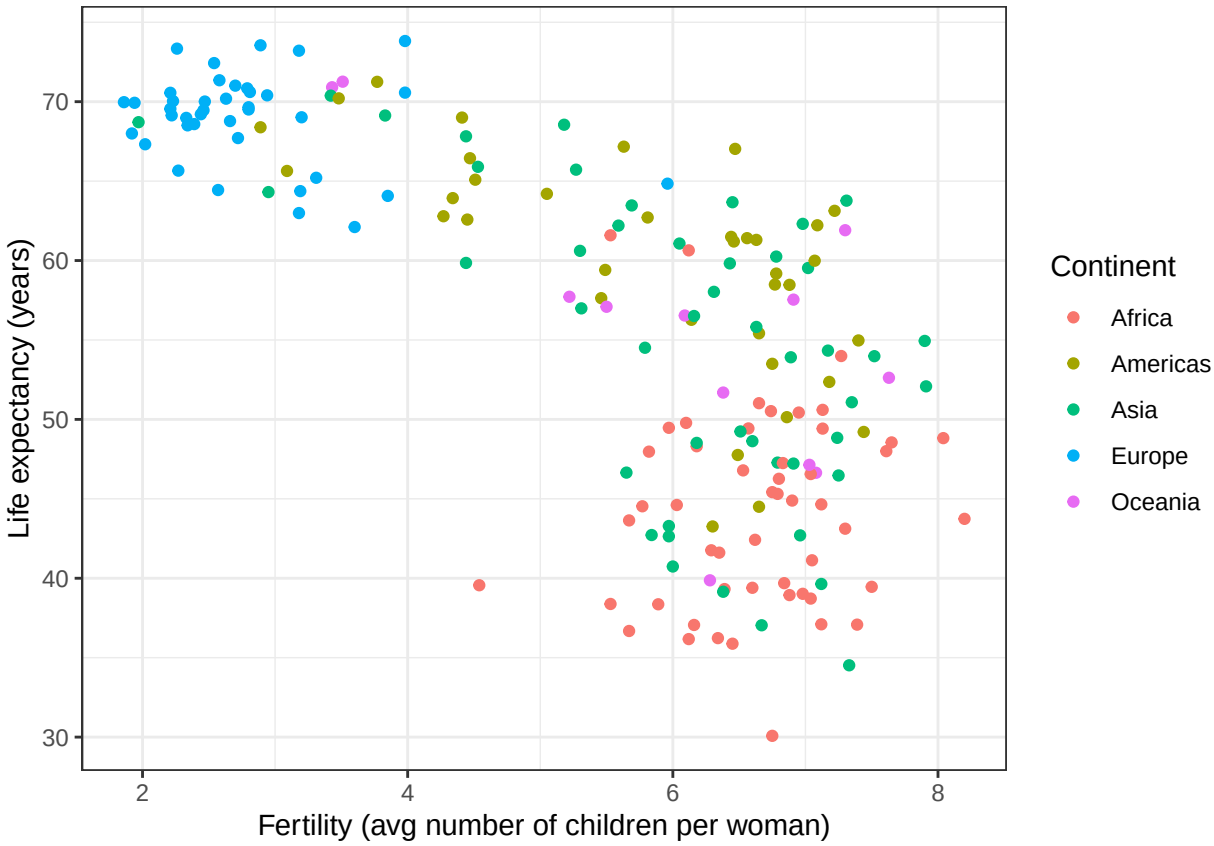


Most points fall into two distinct categories:

1. Life expectancy around 70 years and 3 or less children per family
2. Life expediencies lower then 65 years and with more than 5 children per family.

To confirm that indeed these countries are from the regions we expect, we can use color to represent continent.

```
filter(gapminder, year == 1962) %>%
  ggplot(aes(fertility, life_expectancy, color = continent)) +
  geom_point() +
  xlab("Fertility (avg number of children per woman)") +
  ylab("Life expectancy (years)") +
  scale_color_discrete(name = "Continent")
```



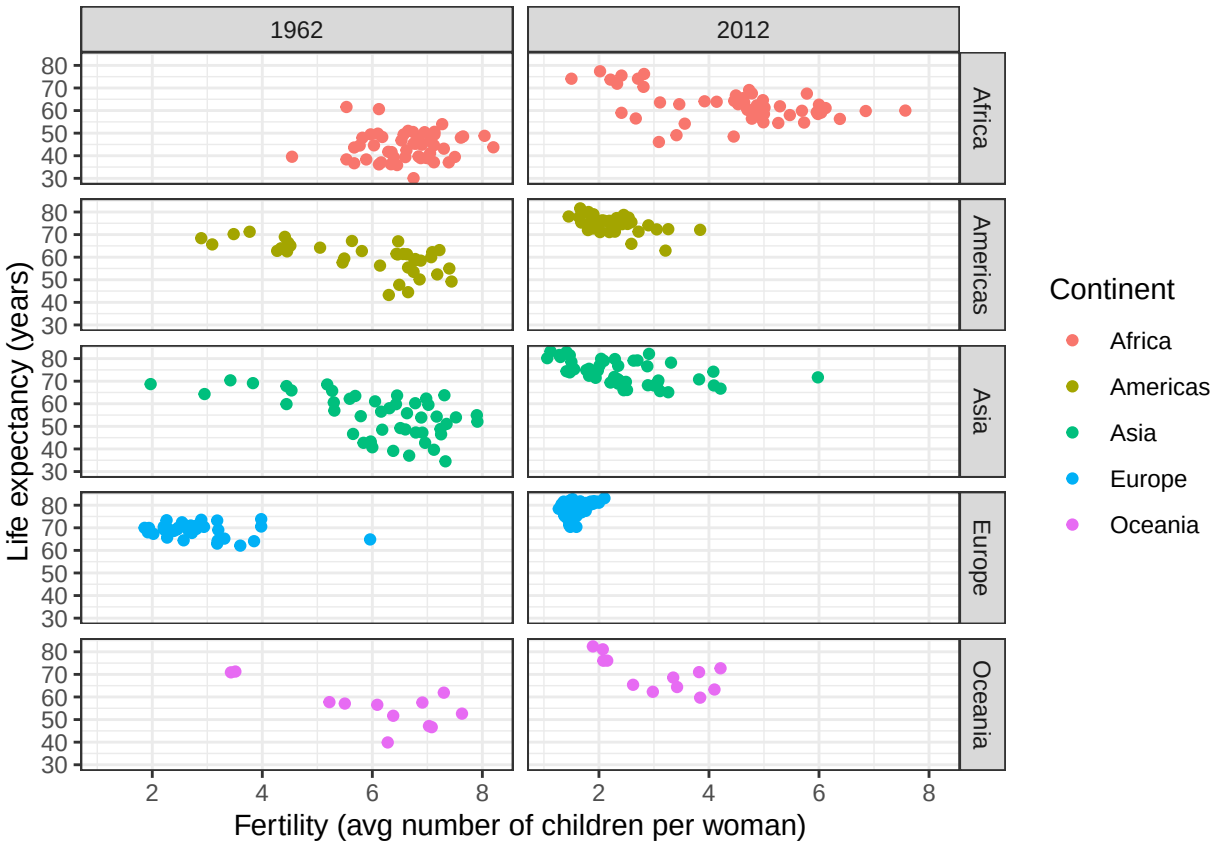
So in 1962, "the west versus developing world" view was grounded in some reality. But is this still the case 50 years later?

Faceting

We could easily plot the 2012 data in the same way we did for 1962. But to compare, side by side plots are preferable. In `ggplot` we can achieve this by *faceting variables*: we stratify the data by some variable and make the same plot for each stratum.

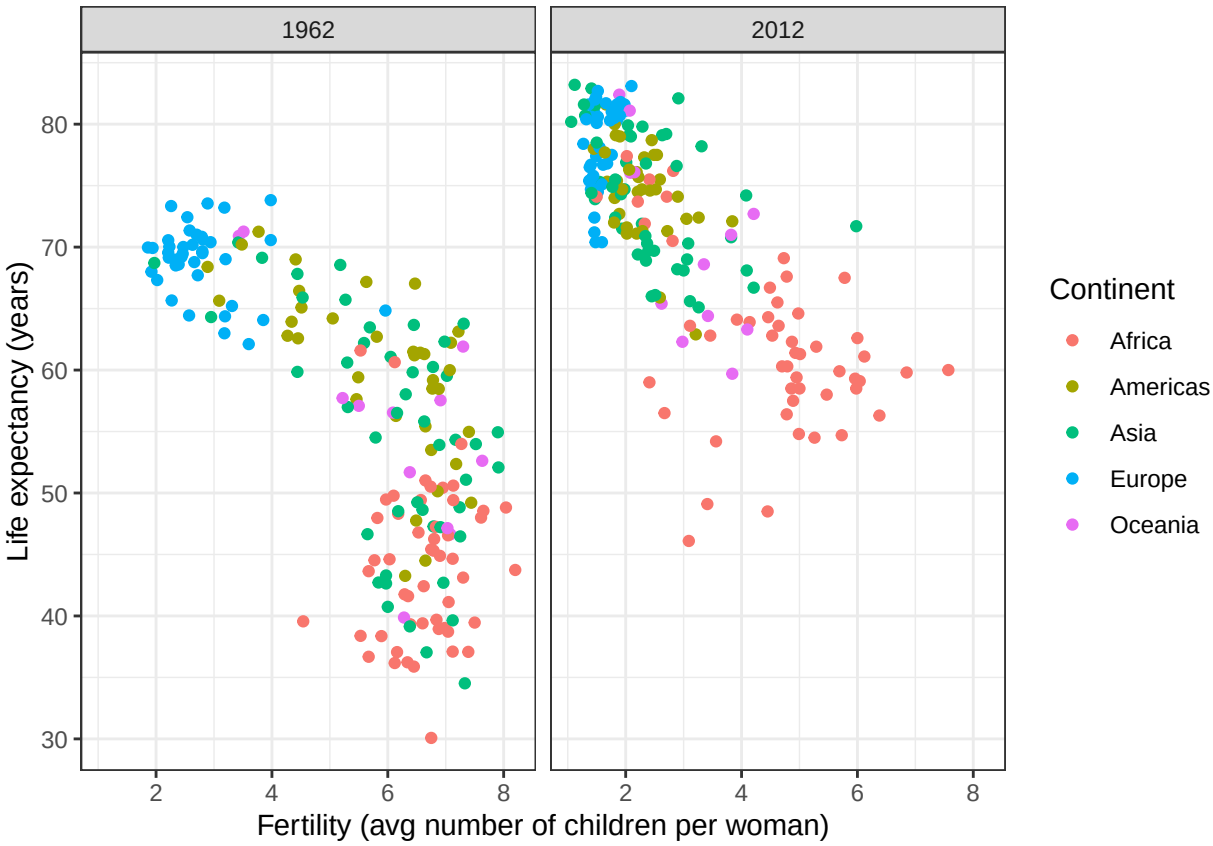
To achieve faceting we add a layer with the function `facet_grid`, which automatically separates the plots. This function lets you facet by up to two variables using columns to represent one variable and rows to represent the other. The function expects the row and column variables separated by a `~`. Here is an example of a scatter plot with `facet_grid` added as the last layer:

```
gapminder %>% filter(year %in% c(1962, 2012)) %>%
  ggplot(aes(fertility, life_expectancy, color = continent)) +
  geom_point() +
  facet_grid(continent~year) +
  xlab("Fertility (avg number of children per woman)") +
  ylab("Life expectancy (years)") +
  scale_color_discrete(name = "Continent")
```



We see a plot for each continent/year pair. However, this is just an example, and more than what we want, which is simply to compare 1962 and 2012. In this case, there is just one variable and we use `.` to let facet know that we are not using one of the variables:

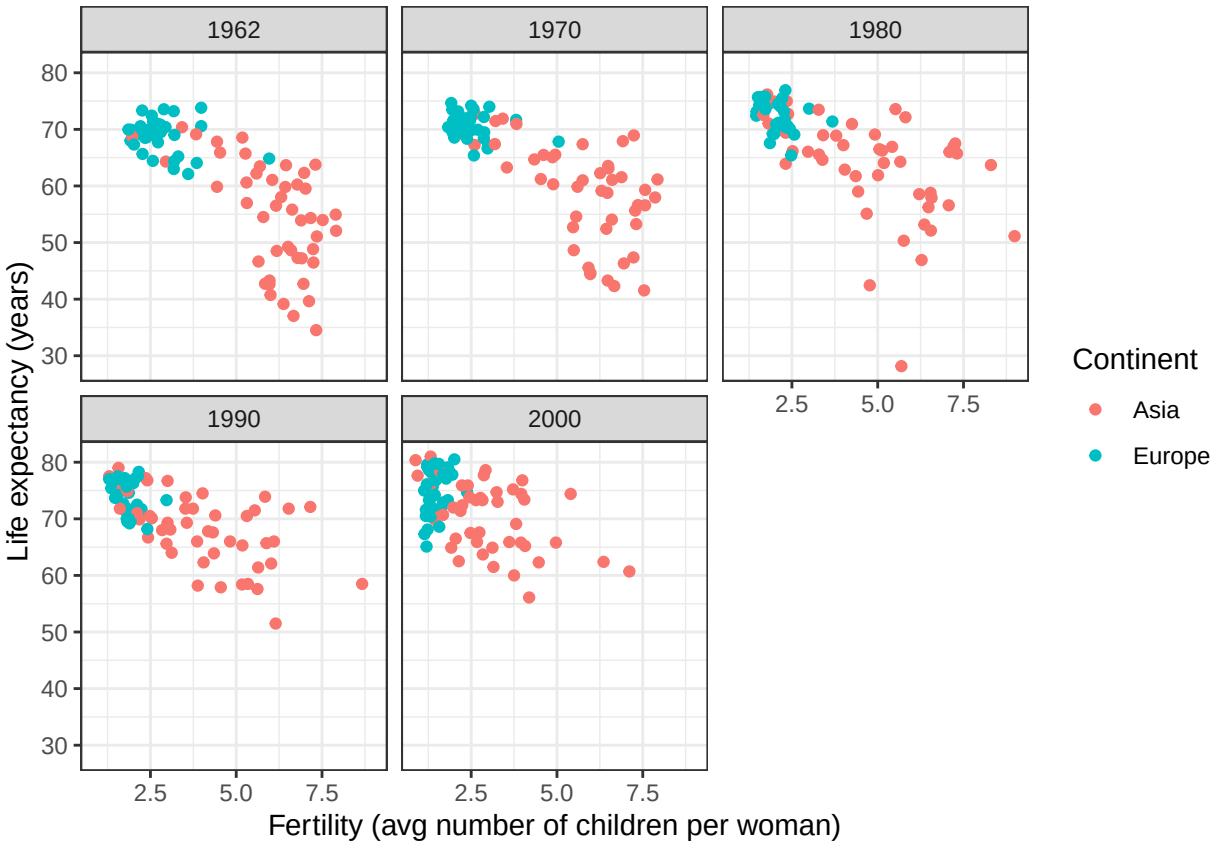
```
filter(gapminder, year %in% c(1962, 2012)) %>%
  ggplot(aes(fertility, life_expectancy, color = continent)) +
  geom_point() +
  facet_grid(. ~ year) +
  xlab("Fertility (avg number of children per woman)") +
  ylab("Life expectancy (years)") +
  scale_color_discrete(name = "Continent")
```



This plot clearly shows that the majority of countries have moved from the *developing world* cluster to the *western world* one. In 2012, the western versus developing world view no longer makes sense. This is particularly clear when comparing Europe to Asia, which includes several countries that have made great improvements.

`facet_wrap` To explore how this transformation happened through the years, we can make the plot for several years. For example we can add 1970, 1980, 1990, and 2000. If we do this, we will not want all the plots on the same row, the default behavior of `facet_grid`, as they will become too thin to show the data. Instead we will want to use multiple rows and columns. The function `facet_wrap` permits us to do this, as it automatically wraps the series of plots so that each display has viewable dimensions:

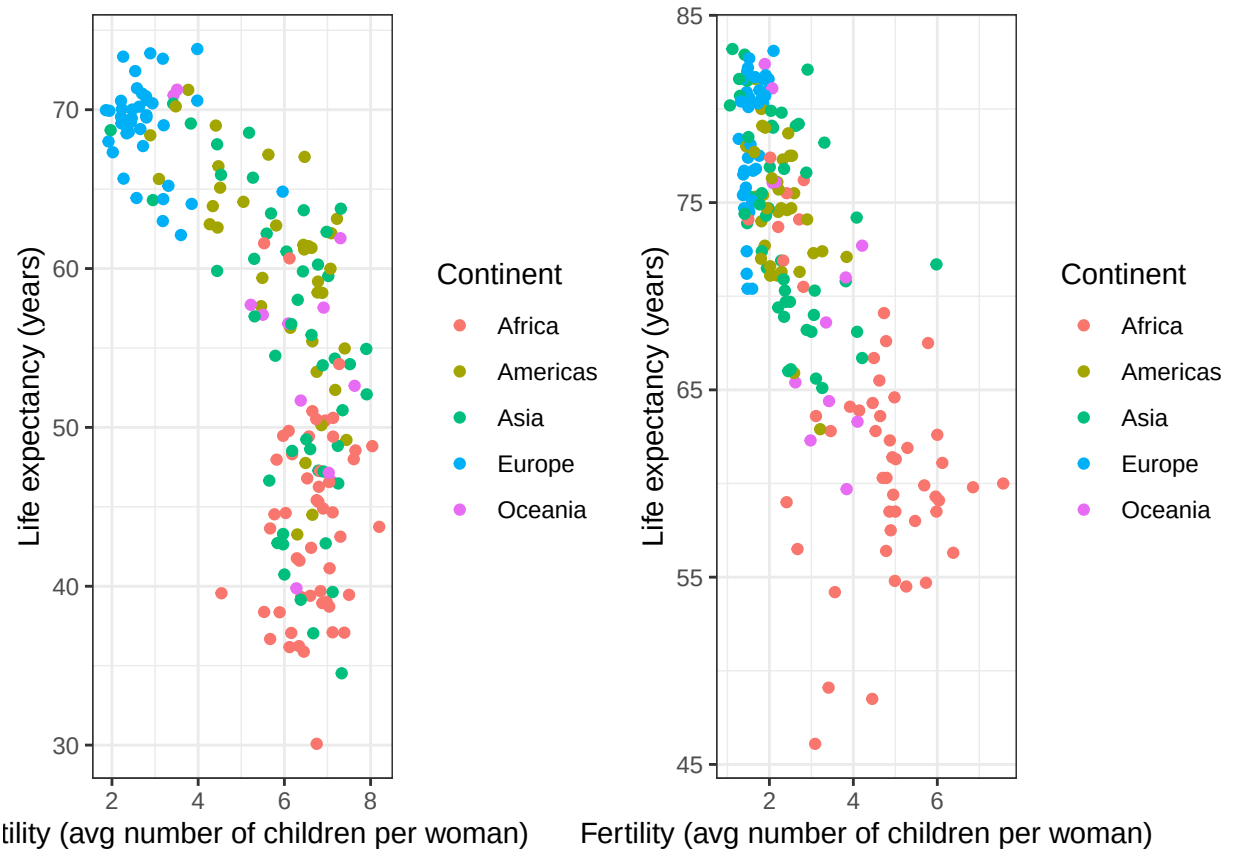
```
years <- c(1962, 1970, 1980, 1990, 2000)
continents <- c("Europe", "Asia")
gapminder %>%
  filter(year %in% years & continent %in% continents) %>%
  ggplot(aes(fertility, life_expectancy, color = continent)) +
  geom_point() +
  facet_wrap(. ~ year)  +
  xlab("Fertility (avg number of children per woman)") +
  ylab("Life expectancy (years)") +
  scale_color_discrete(name = "Continent")
```



This plot clearly shows how most Asian countries have improved at a much faster rate than European ones.

Fixed scales for better comparisons

Note that the default choice of the range of the axes is an important one. When not using `facet`, this range is determined by the the data shown in the plot. When using `facet`, this range is determined by the data shown in all plots and therefore kept fixed across plots. This makes comparisons across plots much easier. For example, in the plot above we see that life expectancy has increased and the fertility has decreased across most countries. We see this because the cloud of points moves. This is not the case if we don't adjust the scales:



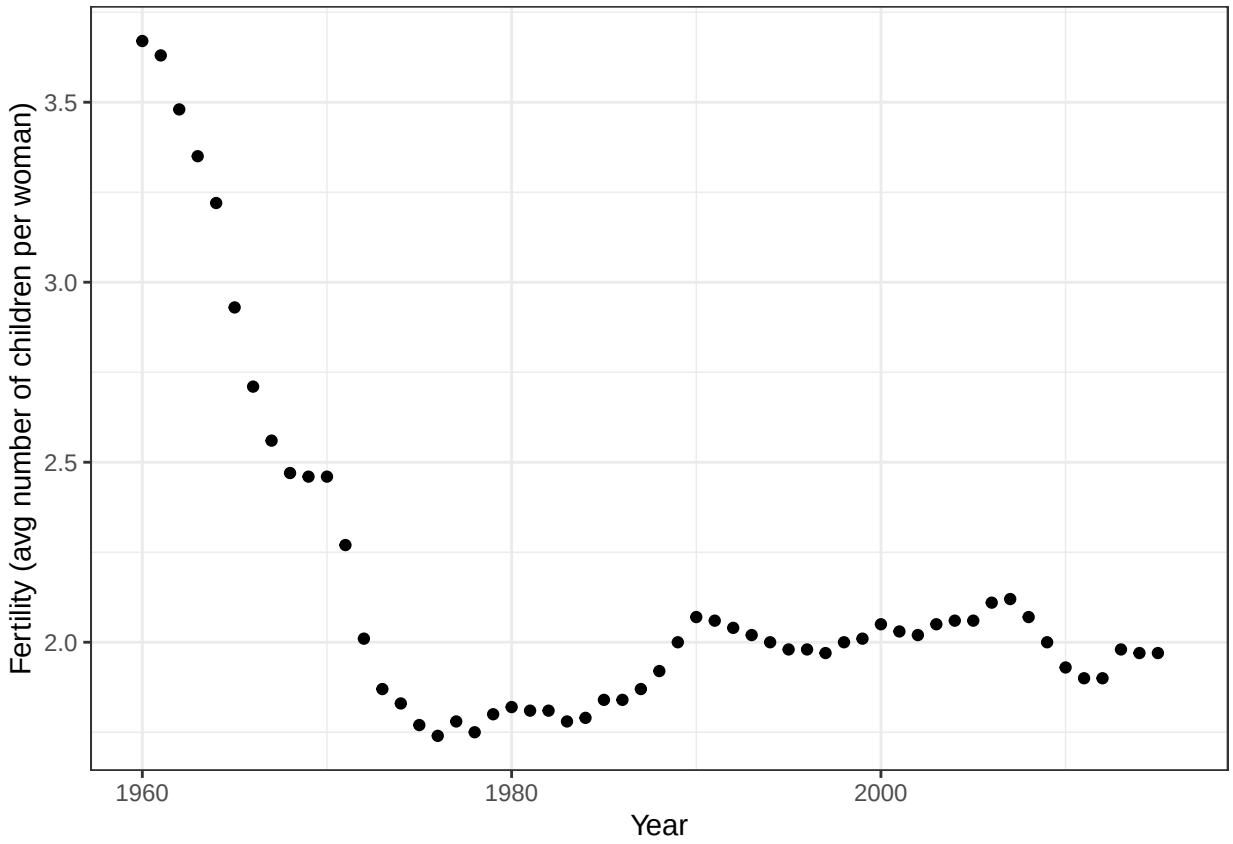
In the plot above we have to pay special attention to the range to notice that the plot on the right has larger life expectancy.

Time Series Plots

The visualizations above effectively illustrate that data no longer support the western versus developing world view. Once we see these plots new questions emerge. For example, which countries are improving more, which ones less? Was the improvement constant during the last 50 years or was there more accelerated improvement during certain periods? For a closer look that may help answer these questions, we introduce *time series plots*.

Time series plots have time on the x-axis and an outcome or measurement of interest on the y-axis. For example, here is a trend plot for the United States fertility rates:

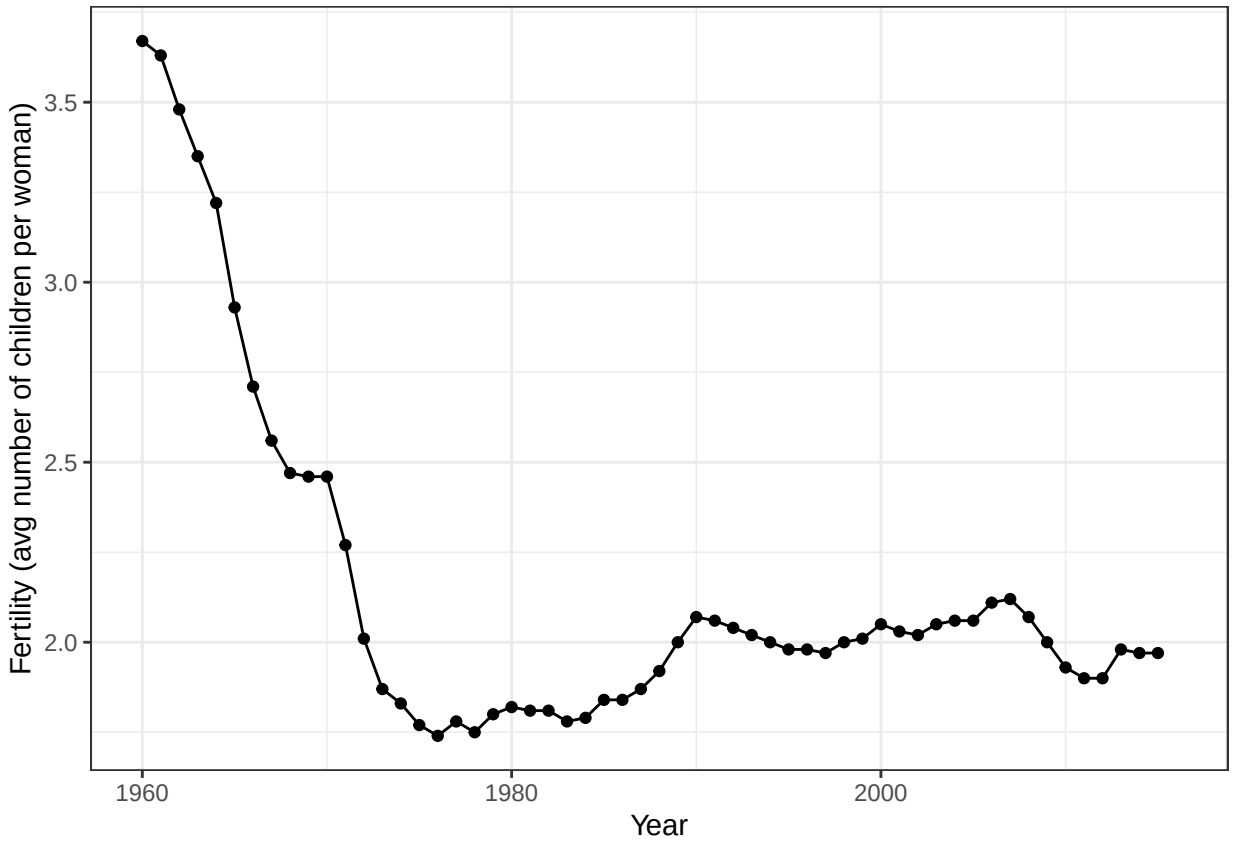
```
gapminder %>% filter(country == "United States") %>%
  ggplot(aes(year, fertility)) +
  geom_point() +
  xlab("Year") +
  ylab("Fertility (avg number of children per woman)")
```



We see that the trend is not linear at all. Instead we see a sharp drop during the 60s and 70s to below 2. Then the trend comes come back to 2 and stabilizes during the 90s.

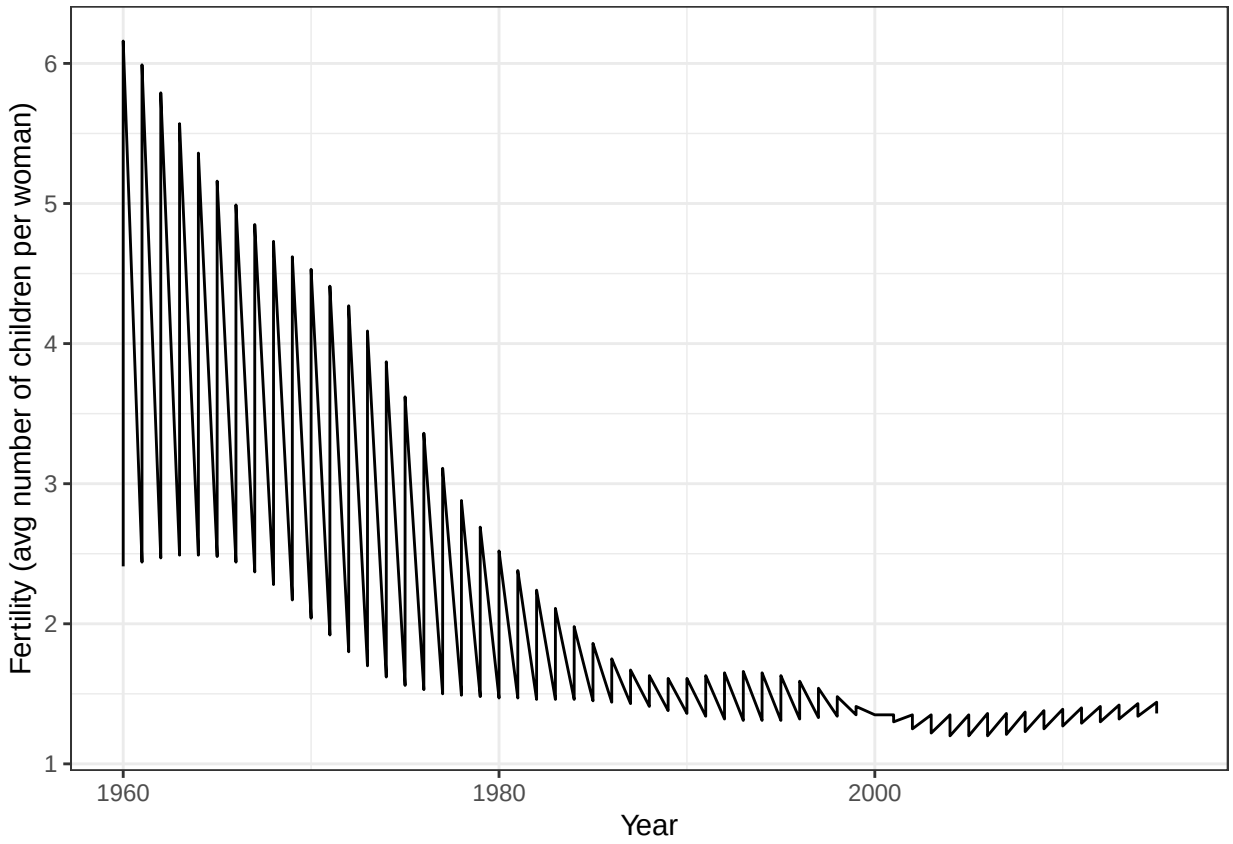
When the points are regularly and densely spaced, as they are here, we create curves by joining the points with lines, to convey that these data are from a single country. To do this we use the `geom_line` function instead of `geom_point`.

```
gapminder %>% filter(country == "United States") %>%
  ggplot(aes(year, fertility)) +
  geom_line() +
  geom_point() +
  xlab("Year") +
  ylab("Fertility (avg number of children per woman)")
```



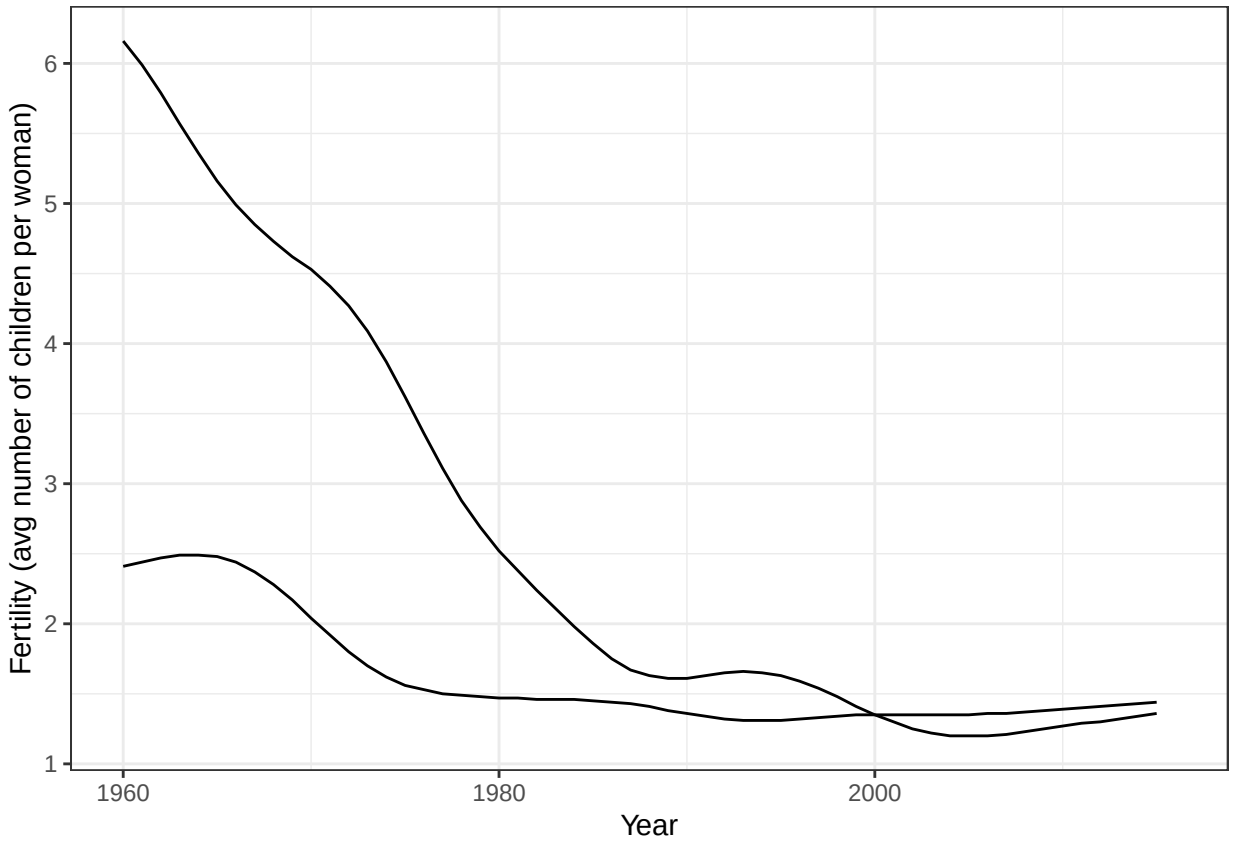
This is particularly helpful when we look at two countries. If we subset the data to include two countries, one from Europe and one from Asia, then copy the code above:

```
countries <- c("South Korea", "Germany")
gapminder %>% filter(country %in% countries) %>%
  ggplot(aes(year,fertility)) +
  geom_line() +
  xlab("Year") +
  ylab("Fertility (avg number of children per woman)")
```



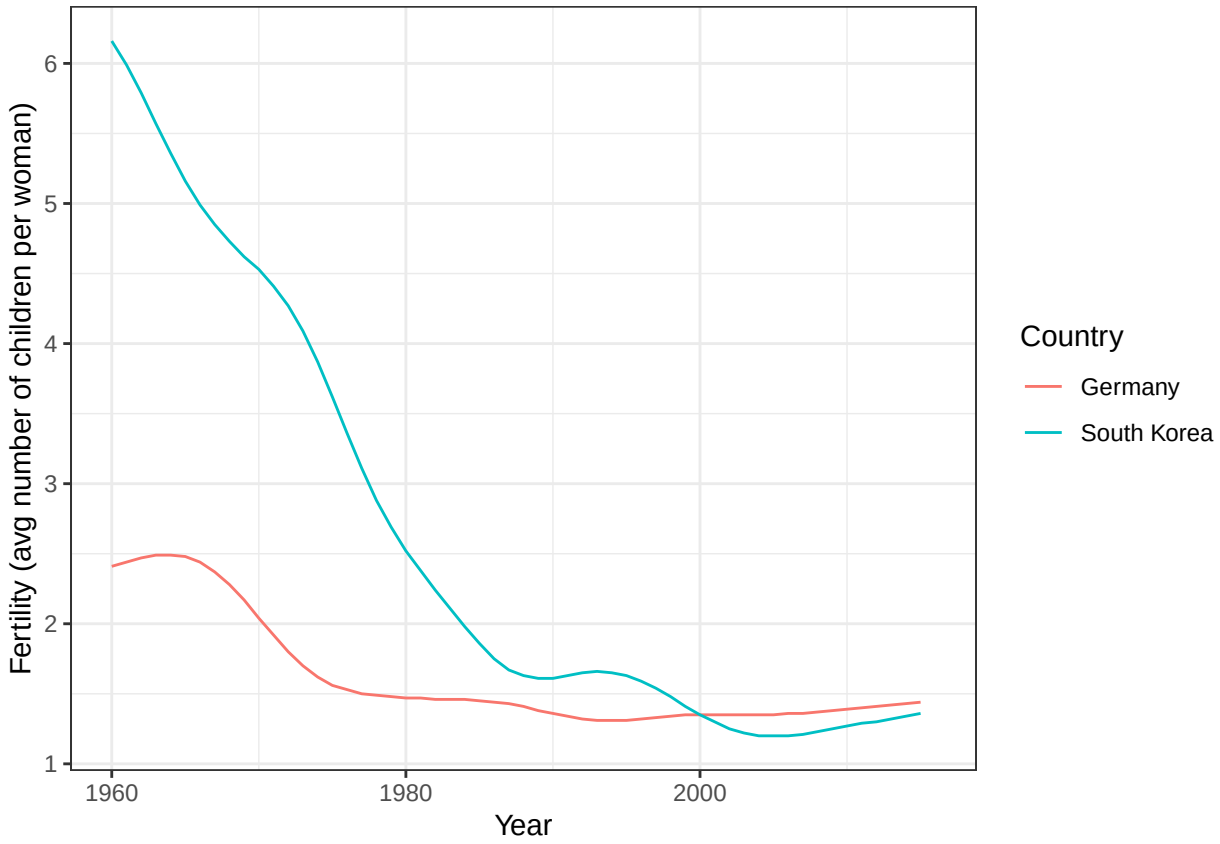
Note that this is **not** the plot that we want. Rather than a line for each country, the points for both countries are joined. This is actually expected since we have not told `ggplot` anything about wanting two separate lines. To let `ggplot` know that there are two curves that need to be made separately, we assign each point to a `group`, one for each country:

```
countries <- c("South Korea", "Germany")
gapminder %>% filter(country %in% countries) %>%
  ggplot(aes(year, fertility, group = country)) +
  geom_line() +
  xlab("Year") +
  ylab("Fertility (avg number of children per woman)")
```



But which line goes with which country? We can assign colors to make this distinction. A useful side-effect of using the `color` argument to assign different colors to the different countries is that the data is automatically grouped:

```
countries <- c("South Korea", "Germany")
gapminder %>% filter(country %in% countries) %>%
  ggplot(aes(year, fertility, color = country)) +
  geom_line() +
  xlab("Year") +
  ylab("Fertility (avg number of children per woman)") +
  scale_color_discrete(name = "Country")
```



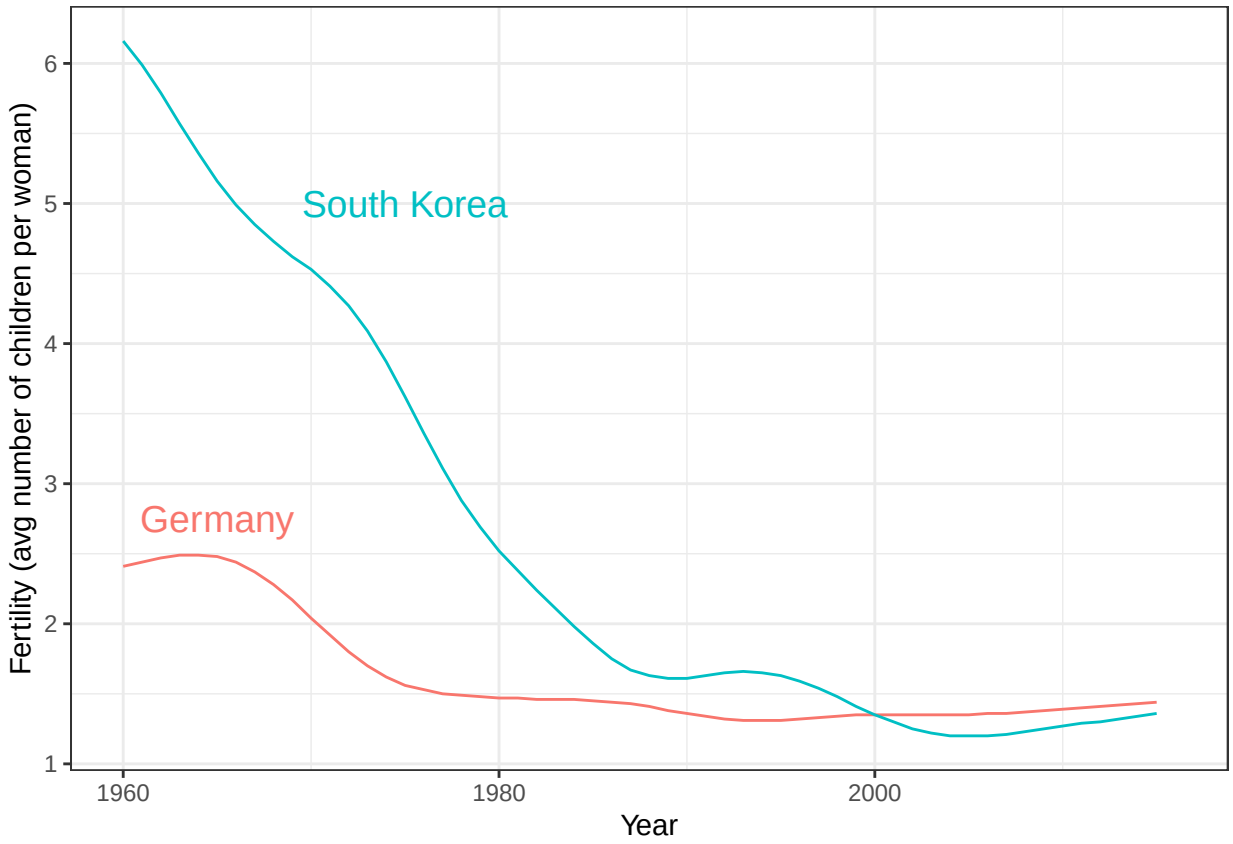
The plot clearly shows how South Korea's fertility rate dropped drastically during the 60s and 70s and by 1990 had a similar rate to Germany.

We prefer labels over legends For trend plots we recommend labeling the lines rather than using legends as the viewer can quickly see which line is which country. This suggestion actually applies to most plots: labeling is usually preferred over legends.

We demonstrate how we can do this using the fertility data. We define a data table with the label locations and then use a second mapping just for these labels:

```
labels <- data.frame(country = countries, x = c(1975, 1965), y = c(5, 2.75))

gapminder %>% filter(country %in% countries) %>%
  ggplot(aes(year, fertility, color = country)) +
  geom_line() +
  geom_text(data = labels, aes(x, y, label = country), size = 5) +
  theme(legend.position = "none") +
  xlab("Year") +
  ylab("Fertility (avg number of children per woman)")
```

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



Example 2: Income Distribution

Another commonly held notion is that wealth distribution across the world has become worse during the last several decades. When general audiences are asked if poor countries have become poorer and rich countries become richer, the majority answer yes. By using stratification, histograms, smooth densities, and boxplots we will be able to understand if this is in fact the case. We will also learn how transformations can sometimes help provide more informative summaries and plots.

Transformations

The `gapminder` data table includes a column with the countries gross domestic product (GDP). GDP measures the market value of goods and services produced by a country in a year. The GDP per person is often used as a rough summary of how rich a country is. Here we divide this quantity by 365 to obtain the more interpretable measure *dollars per day*. Using current US dollars as a unit, a person surviving on an income of less than $2 a day is defined to be living in *absolute poverty*. We add this variable to the data table:

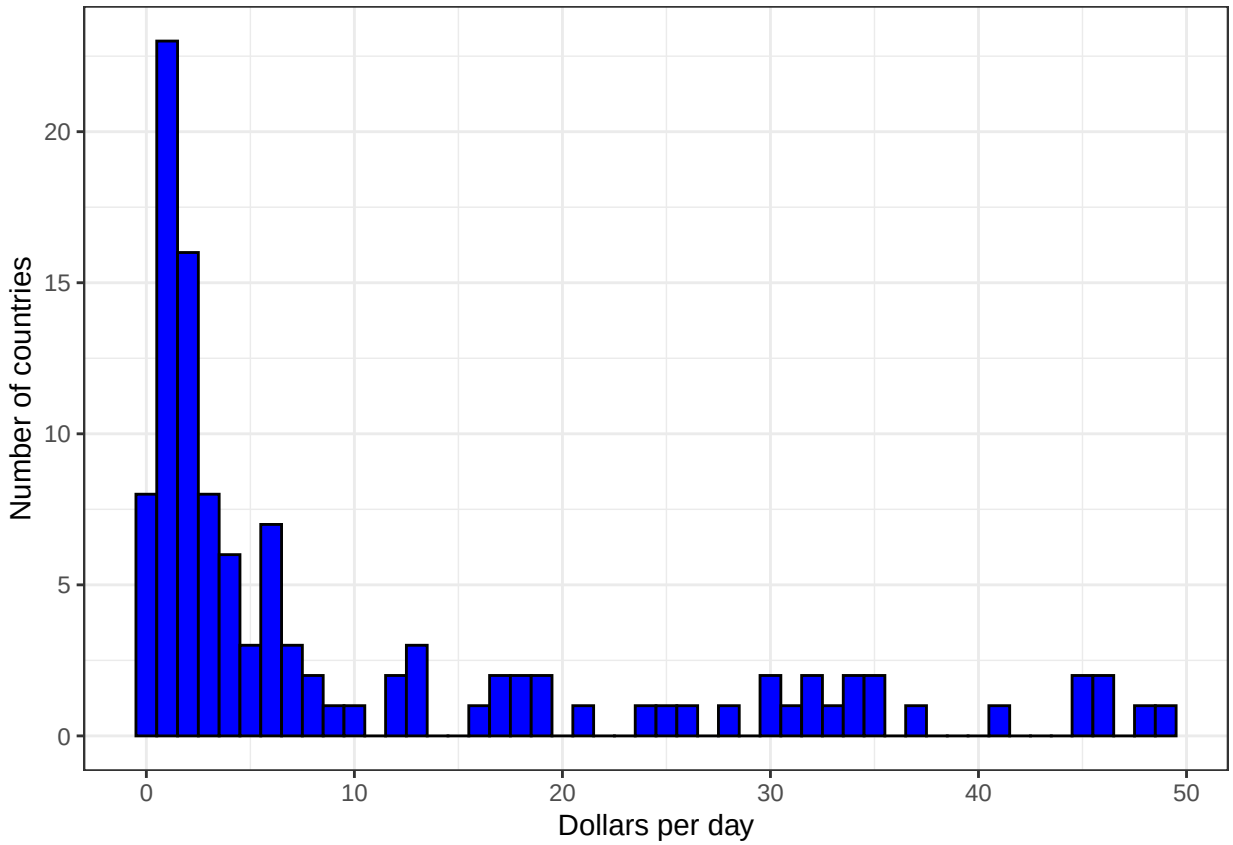
```
gapminder <- gapminder %>%
  mutate(dollars_per_day = gdp/population/365)
```

Note that the GDP values are adjusted for inflation and represent current US dollars, so these values are meant to be comparable across the years. Also note that these are country averages and that within each country there is much variability. All the graphs and insights described below relate to country averages and not to individuals.

Country income distribution Here is a histogram of per day incomes from 1970:

```
past_year <- 1970
gapminder %>%
```

```
  filter(year == past_year & !is.na(gdp)) %>%
  ggplot(aes(dollars_per_day)) +
  geom_histogram(binwidth = 1, color = "black", fill = "blue") +
  xlab("Dollars per day") +
  ylab("Number of countries")
```



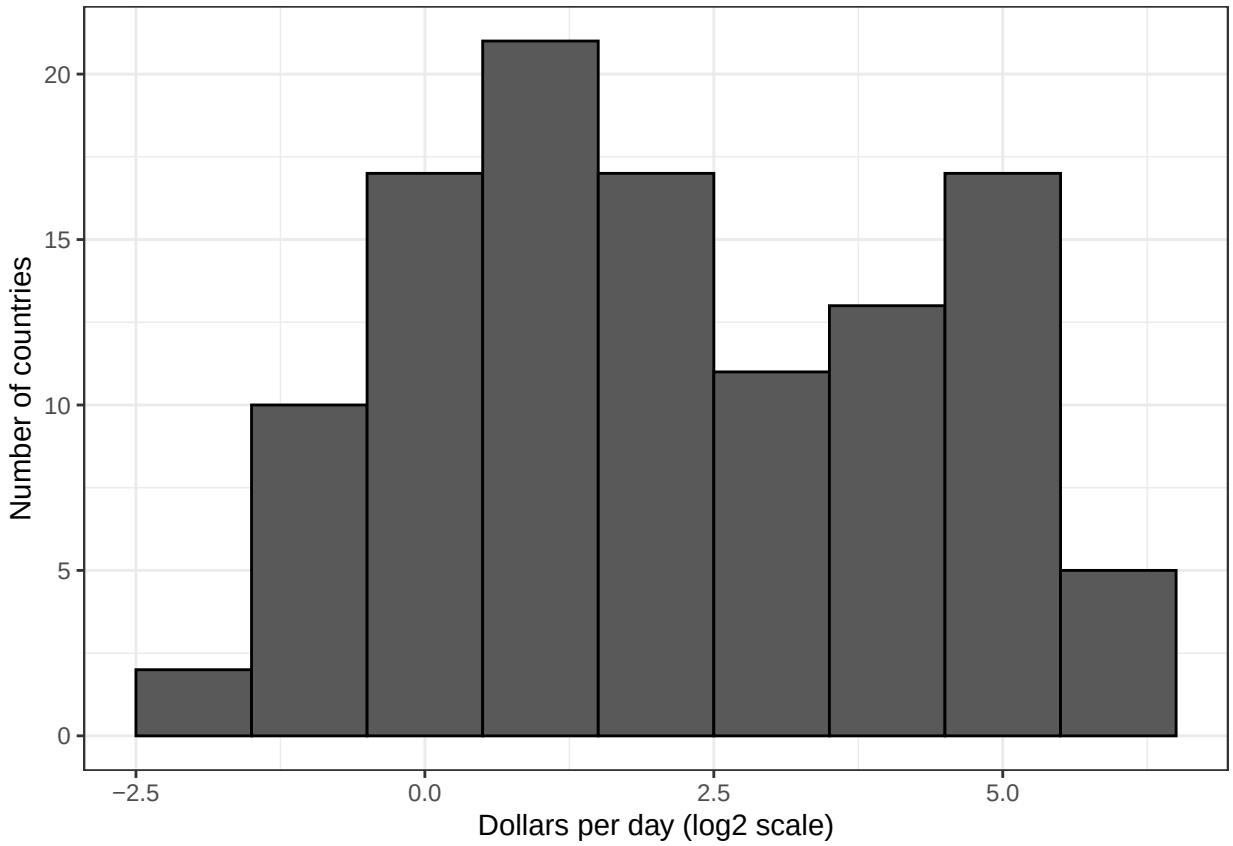
We use the `color = "black"` argument to draw a boundary and clearly distinguish the bins.

In this plot we see that for the majority of countries, averages are below $10 a day. However, the majority of the x-axis is dedicated to the 35 countries with averages above $10. So the plot is not very informative about countries with values below $10 a day.

It might be more informative to quickly be able to see how many countries have average daily incomes of about $1 (extremely poor), $2 (very poor), $4 (poor), $8 (middle), $16 (well off), $32 (rich), $64 (very rich) per day. These changes are multiplicative and log transformations change multiplicative changes into additive ones: when using base 2, a doubling of a value turns into an increase by 1.

Here is the distribution if we apply a log base 2 transformation:

```
gapminder %>%
  filter(year == past_year & !is.na(gdp)) %>%
  ggplot(aes(log2(dollars_per_day))) +
  geom_histogram(binwidth = 1, color = "black") +
  xlab("Dollars per day (log2 scale)") +
  ylab("Number of countries")
```



In a way this provides a *close up* of the mid to lower income countries.

Which base? In the case above we used base 2 in the log transformations. Other common choices are base e (the natural log) and base 10.

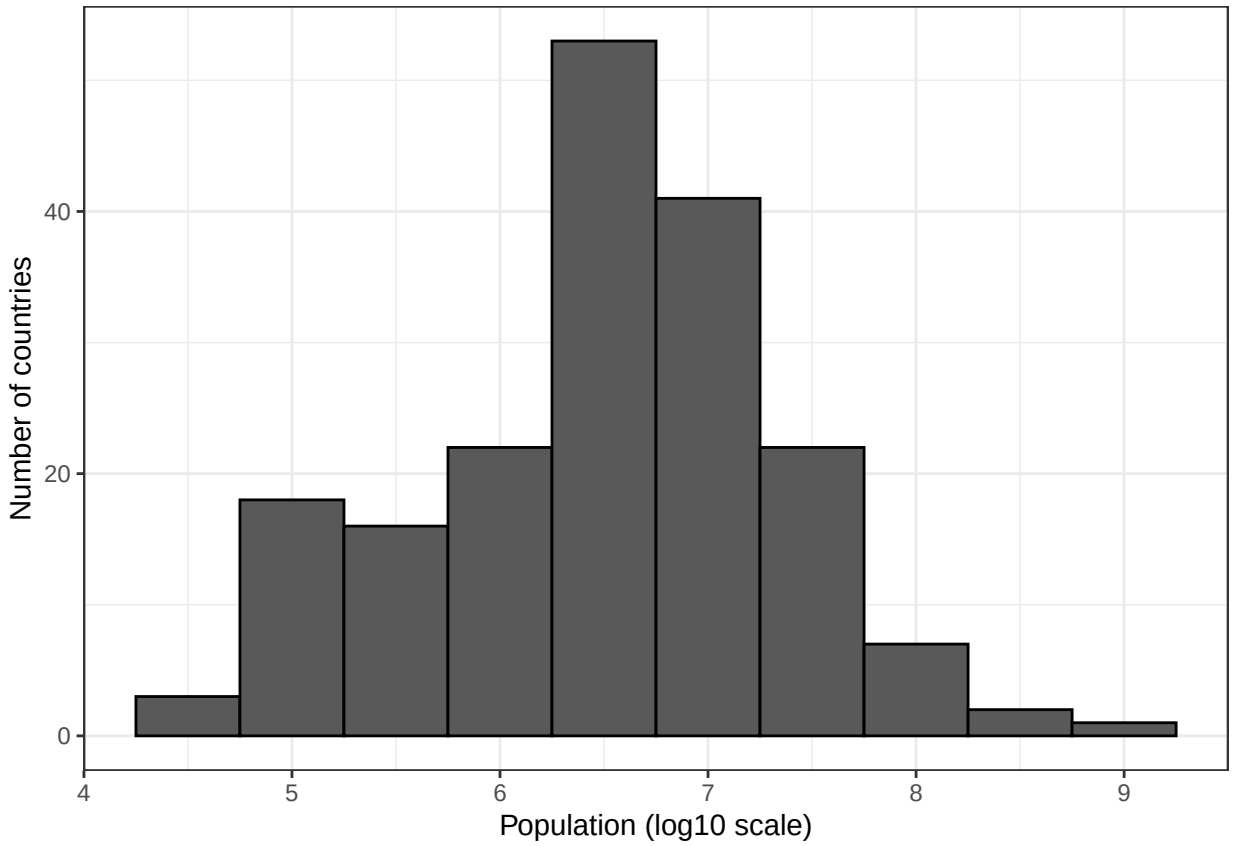
In general, we do not recommend using the natural log for data exploration and visualization. This is because while $2^2, 2^3, 2^4, \ldots$ or $10^1, 10^2, \ldots$ are easy to compute in our heads, the same is not true for $e^2, e^3, \ldots$.

In the dollars per day example, we used base 2 instead of base 10 because the resulting range is easier to interpret. The range of the values being plotted is (0.3269426, 48.8852142).

In base 10 this turns into a range that includes very few integers: just 0 and 1. With base two, our range includes -2, -1, 0, 1, 2, 3, 4 and 5. It is easier to compute 2^x and 10^x when x is an integer and between -10 and 10, so we prefer to have more small integers on the scale. Another consequence of a limited range is that choosing the binwidth is more challenging. With log base 2, we know that a binwidth of 1 will translate to a bin with range x to $2x$.

As an example in which base 10 makes more sense consider population sizes. A log base 10 makes more sense since the range for these is about 1,000 to 10 billion. Here is the histogram of the transformed values:

```
gapminder %>% filter(year == past_year) %>%
  ggplot(aes(log10(population))) +
  geom_histogram(binwidth = 0.5, color = "black") +
  xlab("Population (log10 scale)") +
  ylab("Number of countries")
```



Here we quickly see that country populations range between ten thousand and ten billion.

Transform the values or the scale? There are two ways we can use log transformations in plots. We can log the values before plotting them or use log scales in the axes. Both approaches are useful and have different strengths. If we log the data we can more easily interpret intermediate values in the scale. For example, if we see

—-1—-x—-2———3—-

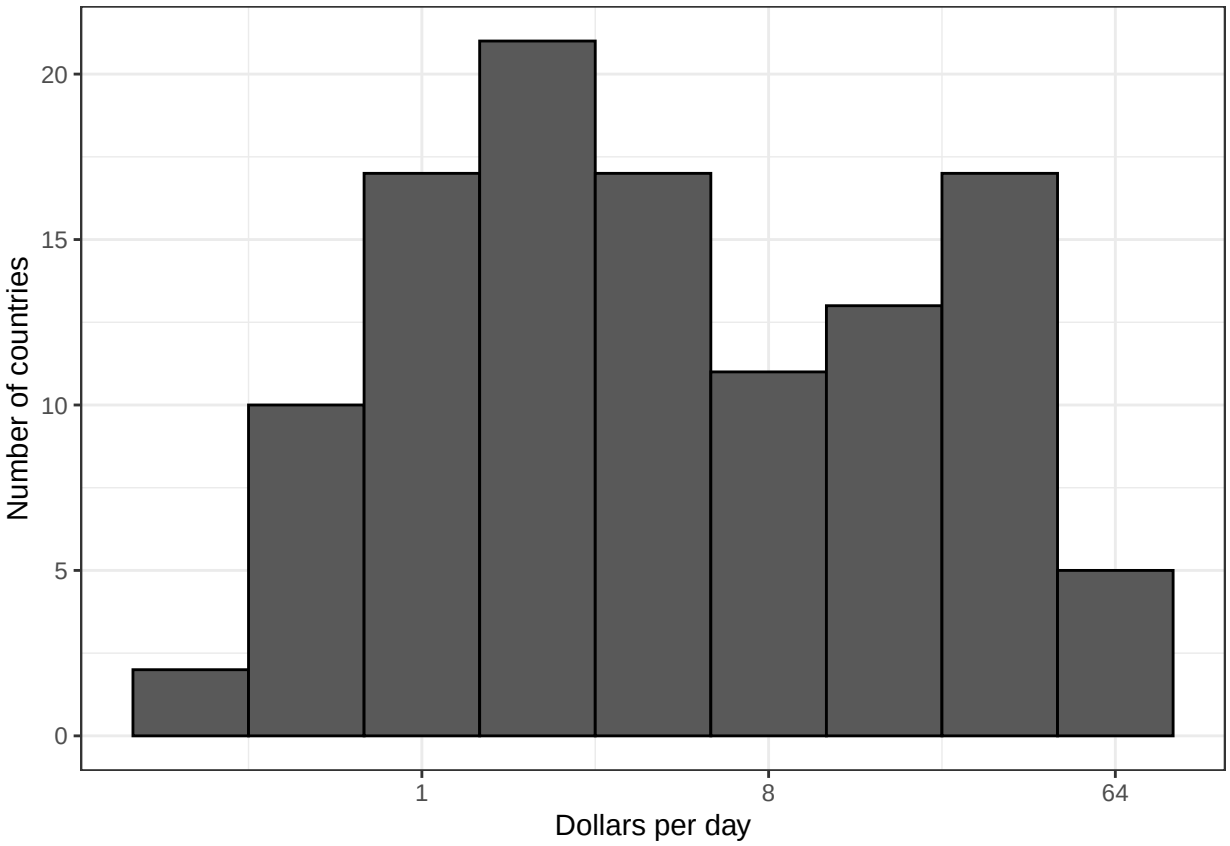
for log transformed data we know that the value of x is about 1.5. If the scales are logged

—-1—-x—-10——100—

then, to determine `x`, we need to compute $10^{1.5}$, which is not easy to do in our heads. However, the advantage of showing logged scales is that the original values are displayed in the plot, which are easier to interpret. For example, we would see "32 dollars a day" instead of "5 log base 2 dollar a day".

As we learned earlier, if we want to scale the axis with logs we can use the `scale_x_ccontinuous` function. So instead of logging the values first, we apply this layer:

```
gapminder %>%
  filter(year == past_year & !is.na(gdp)) %>%
  ggplot(aes(dollars_per_day)) +
  geom_histogram(binwidth = 1, color = "black") +
  scale_x_continuous(trans = "log2") +
  xlab("Dollars per day") +
  ylab("Number of countries")
```



Note that the log base 10 transformation has it's own function: `scale_x_log10()`, but currently base 2 does not. Although we could easily define our own.

Note that there are other transformations avaiable through the `trans` argument. As we learn later on, the square root (`sqrt`) transformation, for example, is useful when considering counts. The logistic transformation (`logit`) is useful when plotting proportions between 0 and 1. The `reverse` transformation is useful when we want smaller values to be on the right or on top.

Modes

The above plot show two "bumps". In statistics, these bumps are sometimes referred to as *modes*. The mode of a distribution is the value with the highest frequency. The mode of the normal distribution is the average. When a distribution, like the one above, doesn't monotonically decrease from the mode we call the locations where it goes up and down again local modes and say that the distribution has multiple modes.

The histogram above suggests that the 1970 country income distribution has two modes: one at about 2 dollars per day (1 in the log 2 scale) and another at about 32 dollars per day (5 in the log 2 scale). This *bimodality* is consistent with a dichotomous world made up of countries with average incomes less than $8 (3 in the log 2 scale) a day and countries above that.

Stratify and boxplot

The histogram showed us that the income distribution values show a dichotomy. However, the histogram does not show us if the two groups of countries are *west* versus the *developing* world.

To see distributions by geographical region we first stratify the data into regions, and then examine the distribution for each. Because of the number of regions,

```
levels(gapminder$region)
```

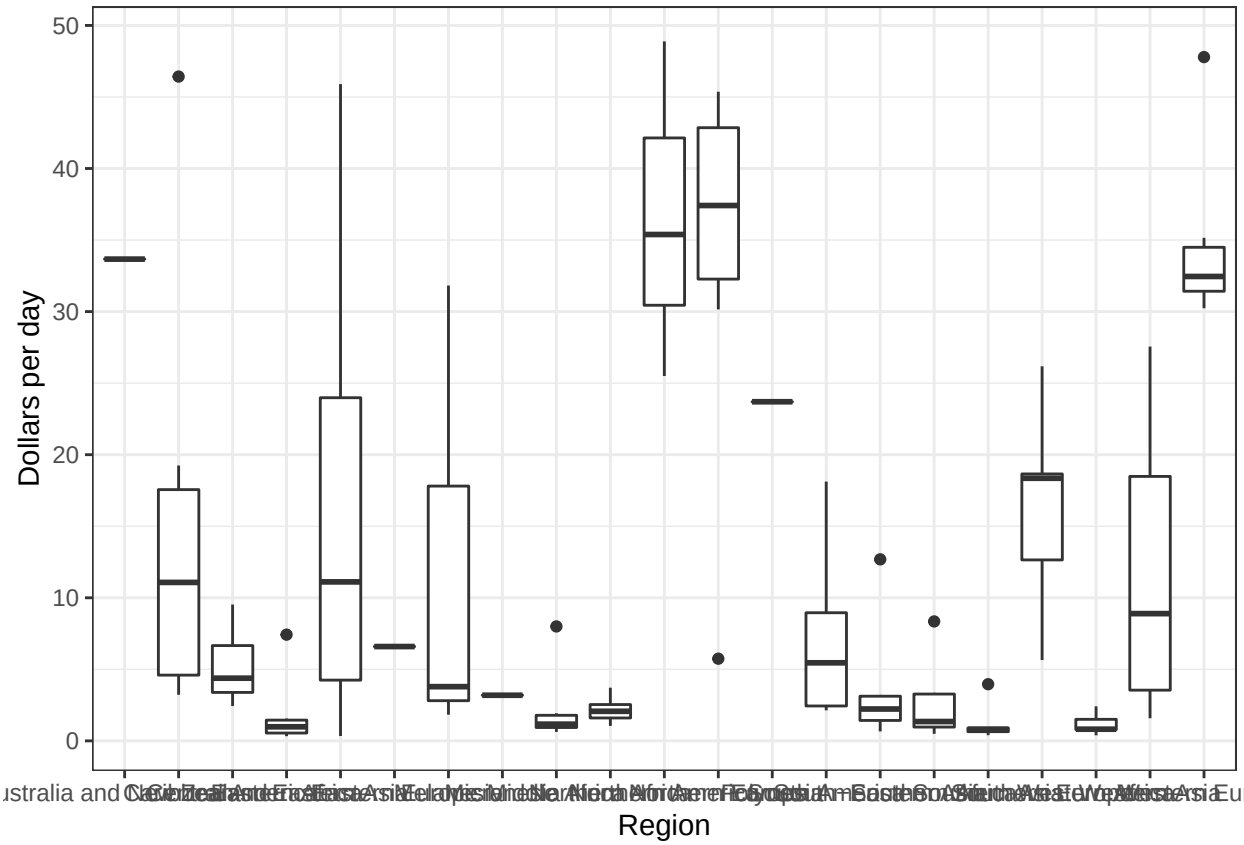
```
##  [1] "Australia and New Zealand" "Caribbean"
##  [3] "Central America"           "Central Asia"
##  [5] "Eastern Africa"            "Eastern Asia"
##  [7] "Eastern Europe"            "Melanesia"
##  [9] "Micronesia"                "Middle Africa"
## [11] "Northern Africa"           "Northern America"
## [13] "Northern Europe"           "Polynesia"
## [15] "South America"             "South-Eastern Asia"
## [17] "Southern Africa"           "Southern Asia"
## [19] "Southern Europe"           "Western Africa"
## [21] "Western Asia"              "Western Europe"
```

```
length(levels(gapminder$region))
```

```
## [1] 22
```

looking at histograms or smooth densities for each will not be useful. Instead, we can stack boxplots next to each other:

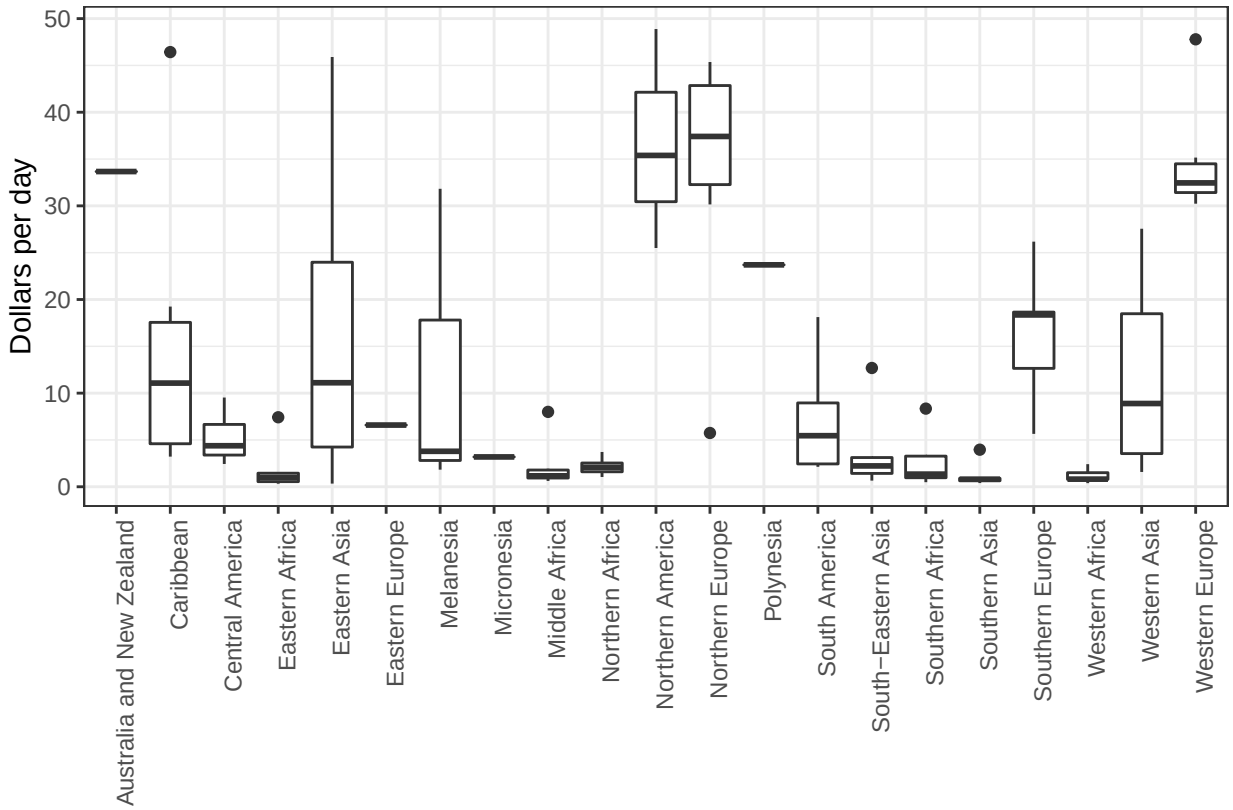
```
p <- gapminder %>%
  filter(year == past_year & !is.na(gdp)) %>%
  ggplot(aes(region, dollars_per_day))  +
  xlab("Region") +
  ylab("Dollars per day")
p + geom_boxplot()
```



Note that we can't read the region names because the default gpplot behavior is to write the labels horizontally and, here, we run of of room. We can easily fix this by rotating the labels. Consulting the cheat sheet we find we can rotate the names by changing the `theme` through `element_text`. The `hjust=1` justifies the text

so that it is next to the axis.

```
p + geom_boxplot() +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))  +
  xlab("") +
  ylab("Dollars per day")
```



We can already see that there is indeed a west versus the rest dichotomy.

Do not order alphabetically There are a few more adjustments we can make to this plot help uncover this reality. First, it helps to order the regions in the boxplots from poor to rich rather than alphabetically. This can be achieved using the `reorder` function. This function let's us change the order of the levels of a factor variable based on a summary computed on a numeric vector. A character vector gets coerced into a factor. Here is an example. Note how the order of levels changes:

```
fac <- factor(c("Asia", "Asia", "West", "West", "West"))
levels(fac)
```

```
## [1] "Asia" "West"
```

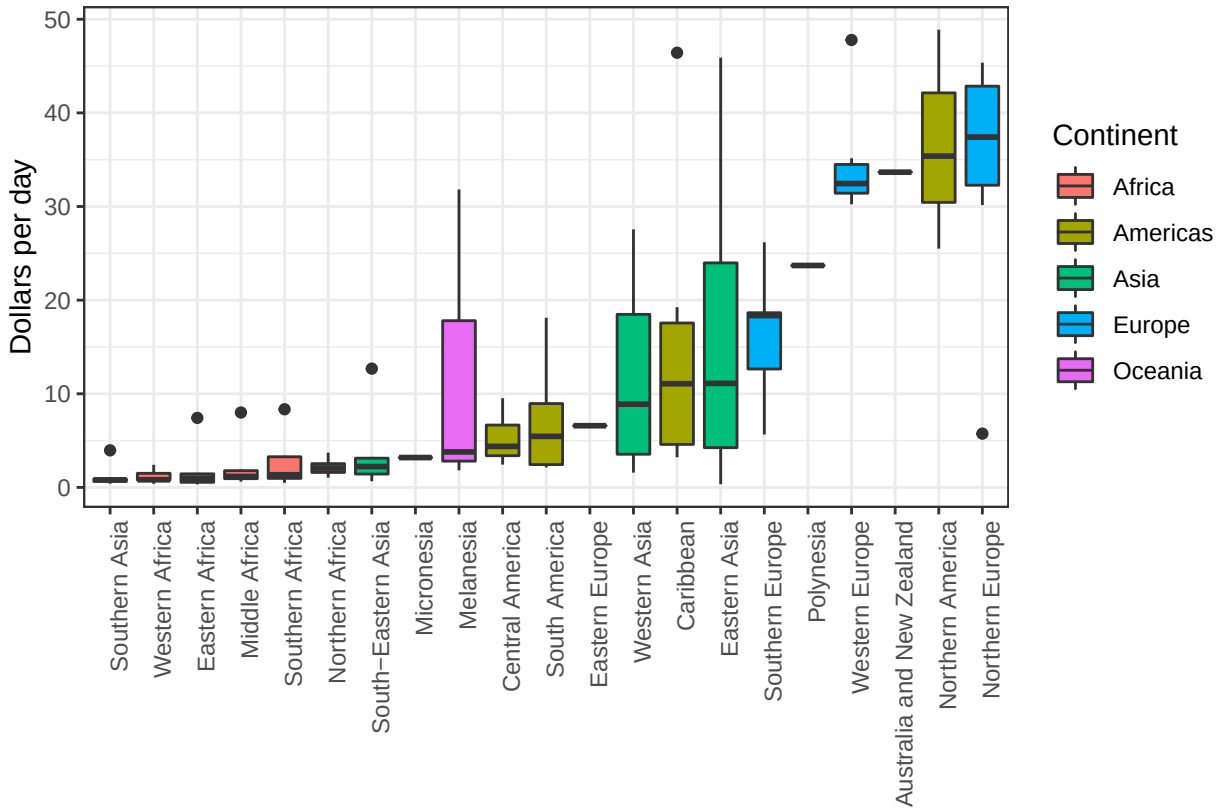
```
value <- c(10, 11, 12, 6, 4)
fac <- reorder(fac, value, FUN = mean)
levels(fac)
```

```
## [1] "West" "Asia"
```

Second, we can use color to distinguish the different continents, a visual cue that helps find specific regions. Here is the code:

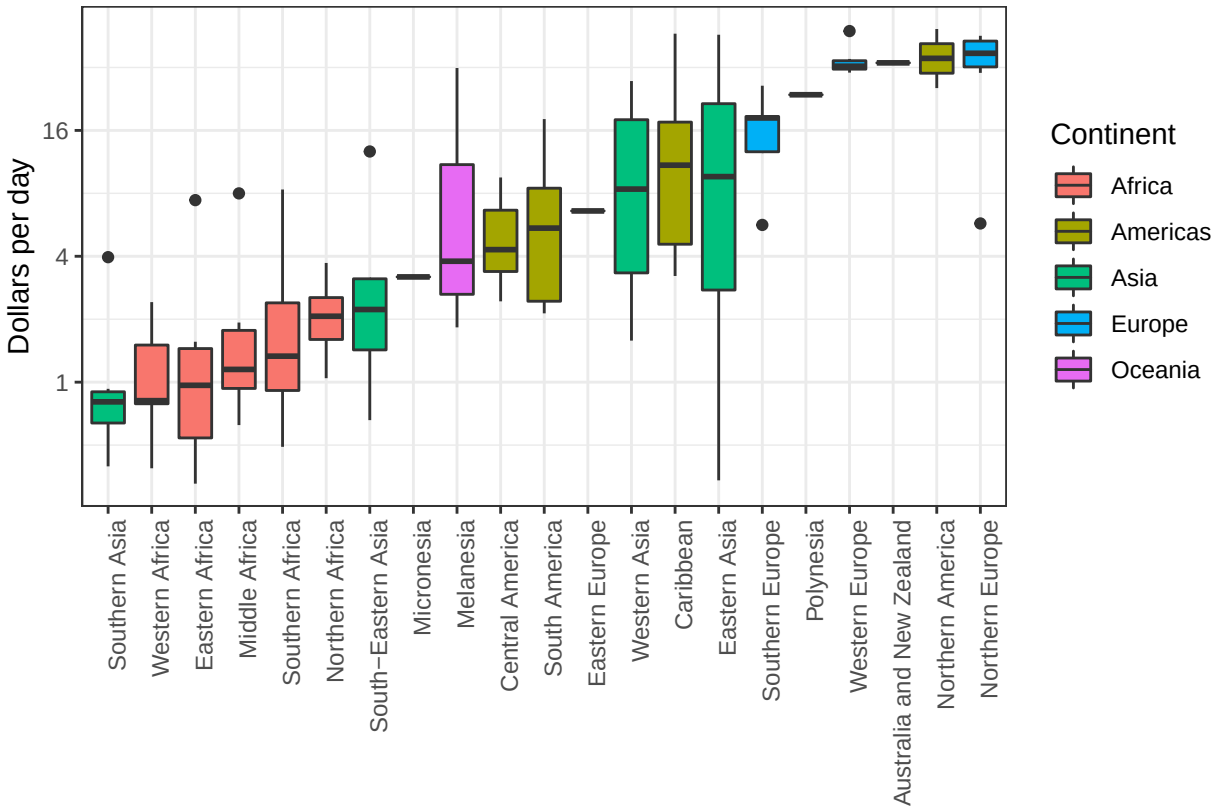
```
p <- gapminder %>%
  filter(year == past_year & !is.na(gdp)) %>%
```

```
  mutate(region = reorder(region, dollars_per_day, FUN = median)) %>%
  ggplot(aes(region, dollars_per_day, fill = continent)) +
  geom_boxplot() +
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  xlab("") +
  ylab("Dollars per day") +
  labs(fill = "Continent")
p
```



This plot shows two clear groups, with the rich group composed of North America, Northern and Western Europe, New Zealand and Australia. As with the histogram, if we remake the plot using a log scale

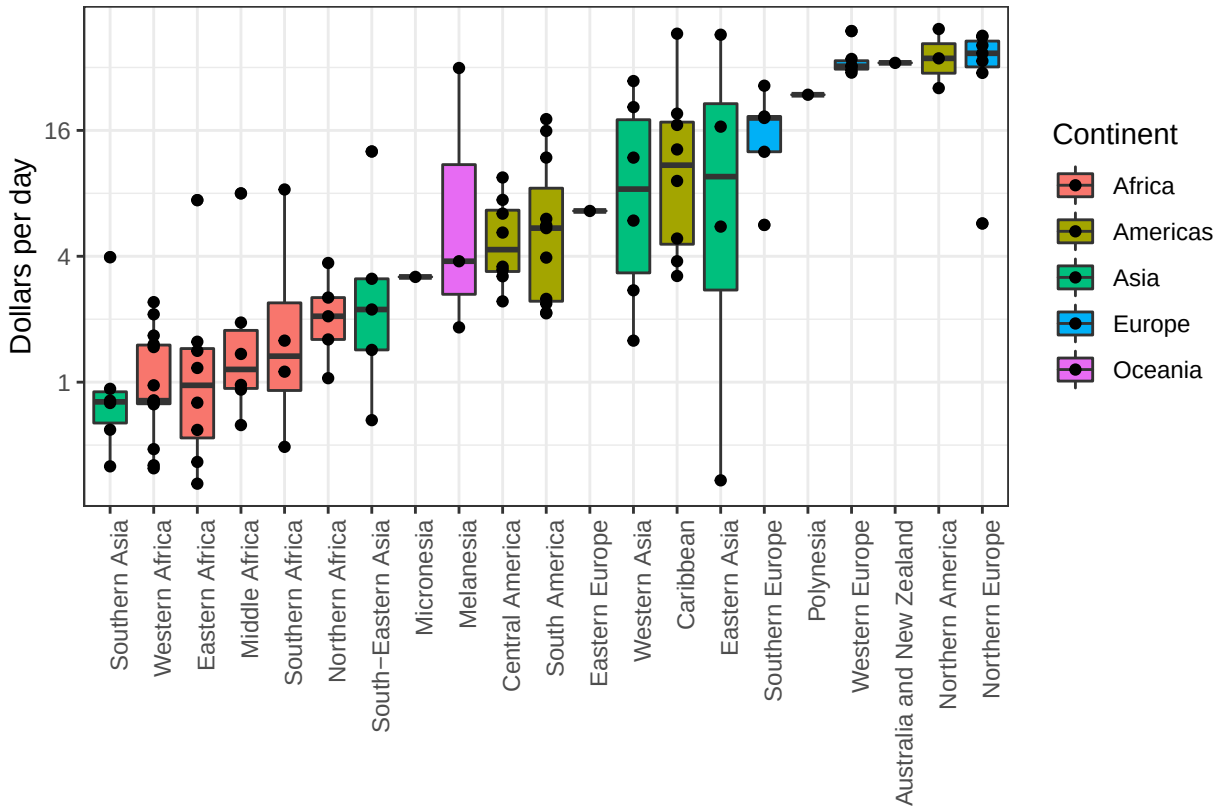
```
p + scale_y_continuous(trans = "log2")
```



we are able to better see differences within the devoloping world.

Show the data In many cases we do not show the data because it adds clutter to the plot and obfuscates the message. In the example above this is not the case and adding points actually lets us see all the data. We can add this layer using `geom_point()`.

```
p + scale_y_continuous(trans = "log2") +
  geom_point() +
  xlab("") +
  ylab("Dollars per day") +
  labs(fill = "Continent")
```



Comparing Distributions

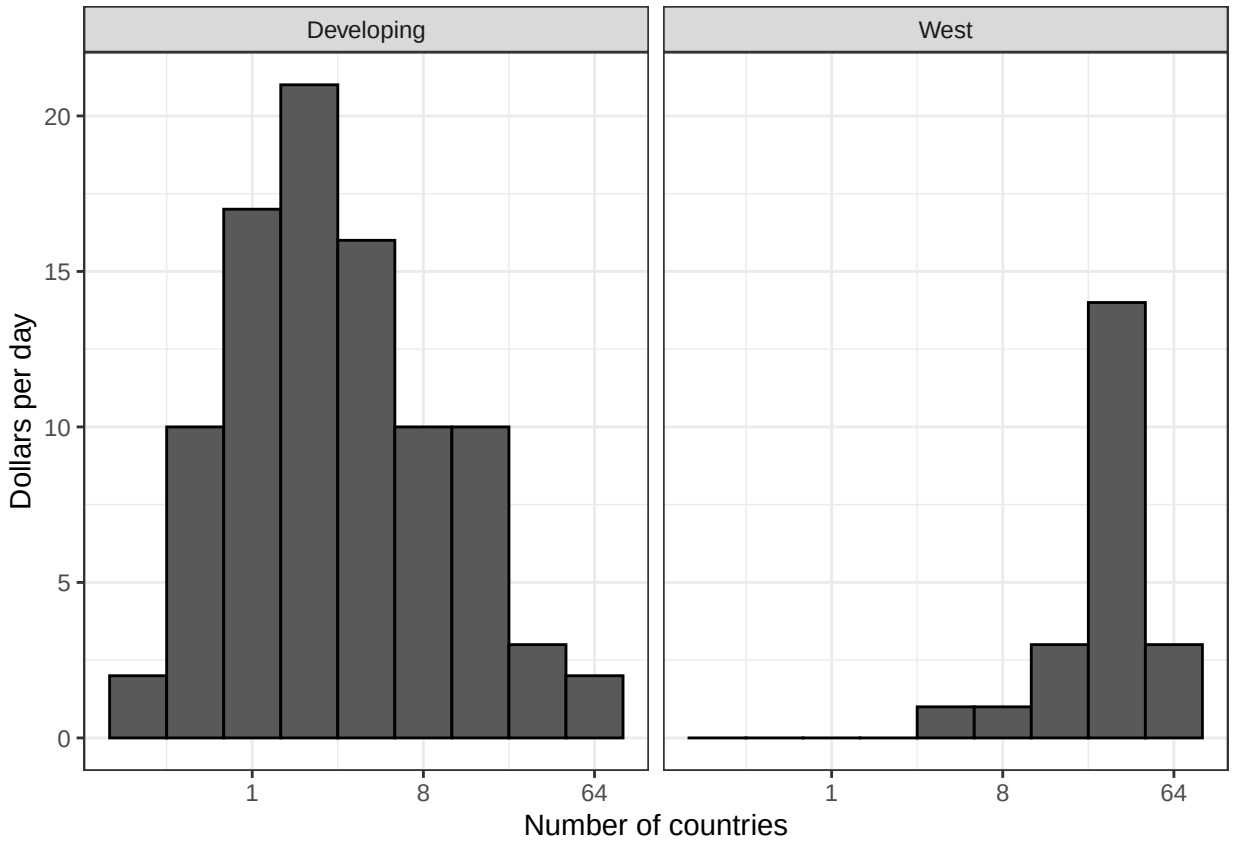
The exploratory data analysis above has revealed two characteristics about average income distribution in 1970. Using a histogram we found a bimodal distribution with the modes relating to poor and rich countries. Then by stratifying by region and examining boxplots we found that the rich countries were mostly in Europe and Northern America, along with Australia and New Zealand. We define a vector with these regions:

```
west <- c("Western Europe","Northern Europe","Southern Europe",
          "Northern America","Australia and New Zealand")
```

Now we want to focus on comparing the differences in distributions across time.

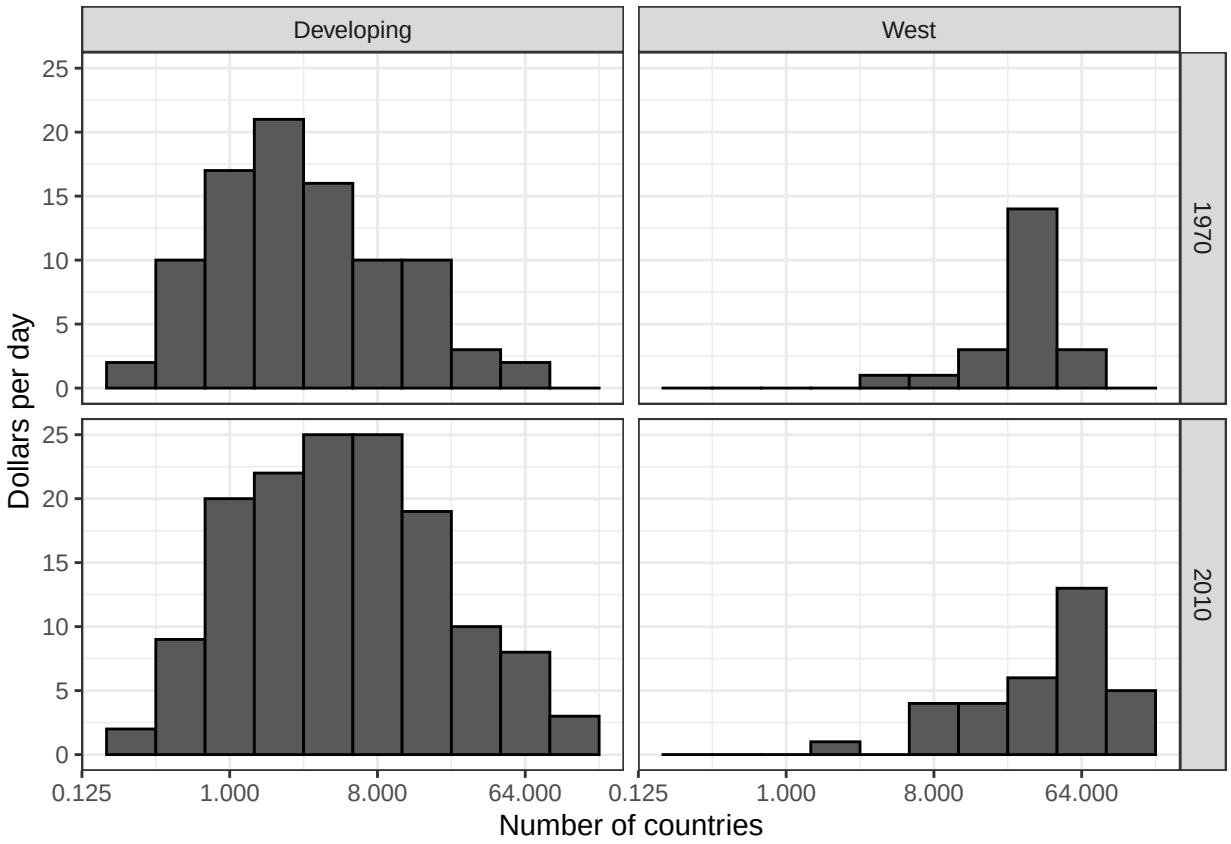
We start by confirming that the bimodality observed in 1970 is explained by a west versus devoloping world dichotomy. We do this by creating histograms for the groups previously identified. Note that we create the two groups with an `ifelse` inside a `mutate` and that we use `facet_grid` to make a histogram for each group:

```
gapminder %>%
  filter(year == past_year & !is.na(gdp)) %>%
  mutate(group = ifelse(region %in% west, "West", "Developing")) %>%
  ggplot(aes(dollars_per_day)) +
  geom_histogram(binwidth = 1, color = "black") +
  scale_x_continuous(trans = "log2") +
  facet_grid(. ~ group)  +
  xlab("Number of countries") +
  ylab("Dollars per day")
```



Now we are ready to see if the separation is worse today than it was 40 years ago. We do this by faceting by both region and year:

```
past_year <- 1970
present_year <- 2010
gapminder %>%
  filter(year %in% c(past_year, present_year) & !is.na(gdp)) %>%
  mutate(group = ifelse(region %in% west, "West", "Developing")) %>%
  ggplot(aes(dollars_per_day)) +
  geom_histogram(binwidth = 1, color = "black") +
  scale_x_continuous(trans = "log2") +
  facet_grid(year ~ group)  +
  xlab("Number of countries") +
  ylab("Dollars per day")
```



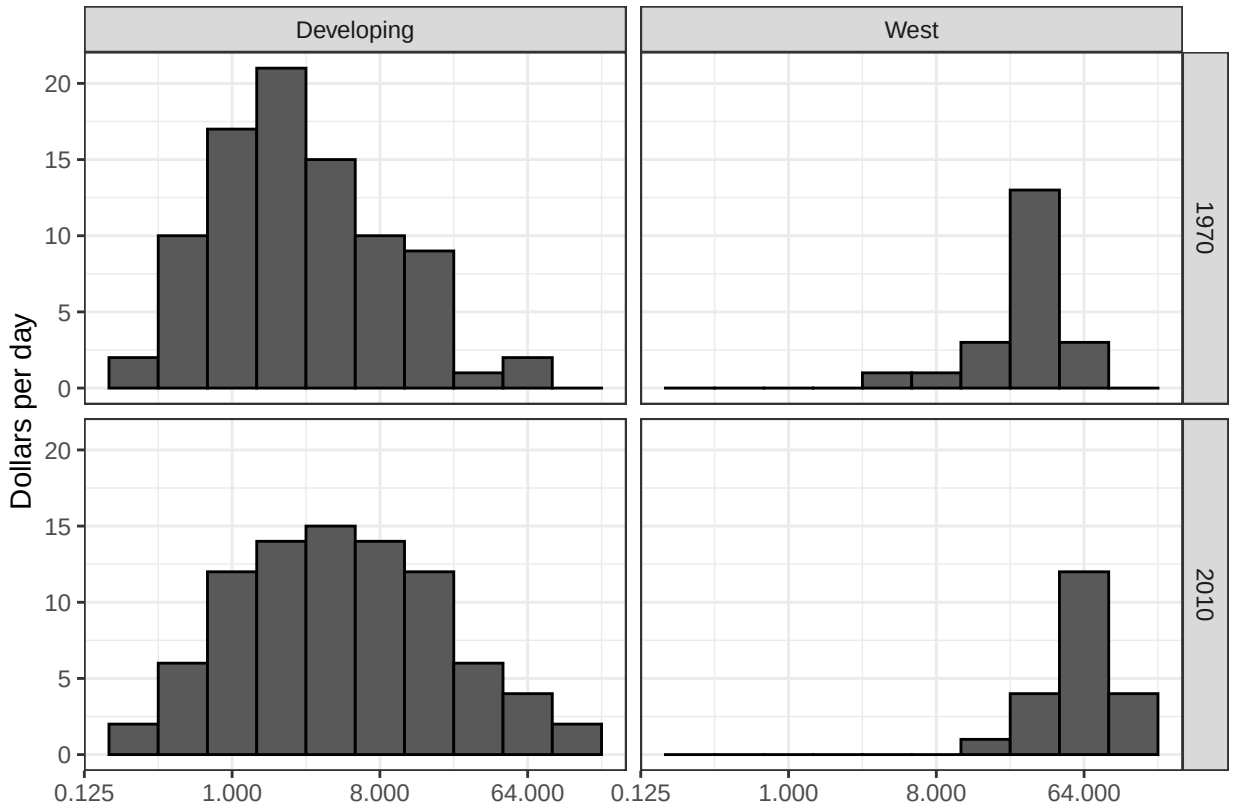
Before we interpret the findings of this plot, we note that there are more countries represented in the 2010 histograms than in 1970: the total counts are larger. One reason for this is that several countries were founded after 1970. For example, the Soviet Union turned into several countries including Russia and Ukraine during the 1990s. Another reason is that data was available for more countries during 2010.

We remake the plots using only countries with data available for both years. In the data wrangling chapter we will learn `tidyverse` tools that permit us to write efficient code for this, but here is simple code using the `intersect` function:

```
country_list_1 <- gapminder %>%
  filter(year == past_year & !is.na(dollars_per_day)) %>% .$country

country_list_2 <- gapminder %>%
  filter(year == present_year & !is.na(dollars_per_day)) %>% .$country

country_list <- intersect(country_list_1, country_list_2)
```

These 108 account for 86 % of the world population, so this subset should be representative.

Let's remake the plot but only for this subset by simply adding `country %in% country_list` to the filter func-



tion:

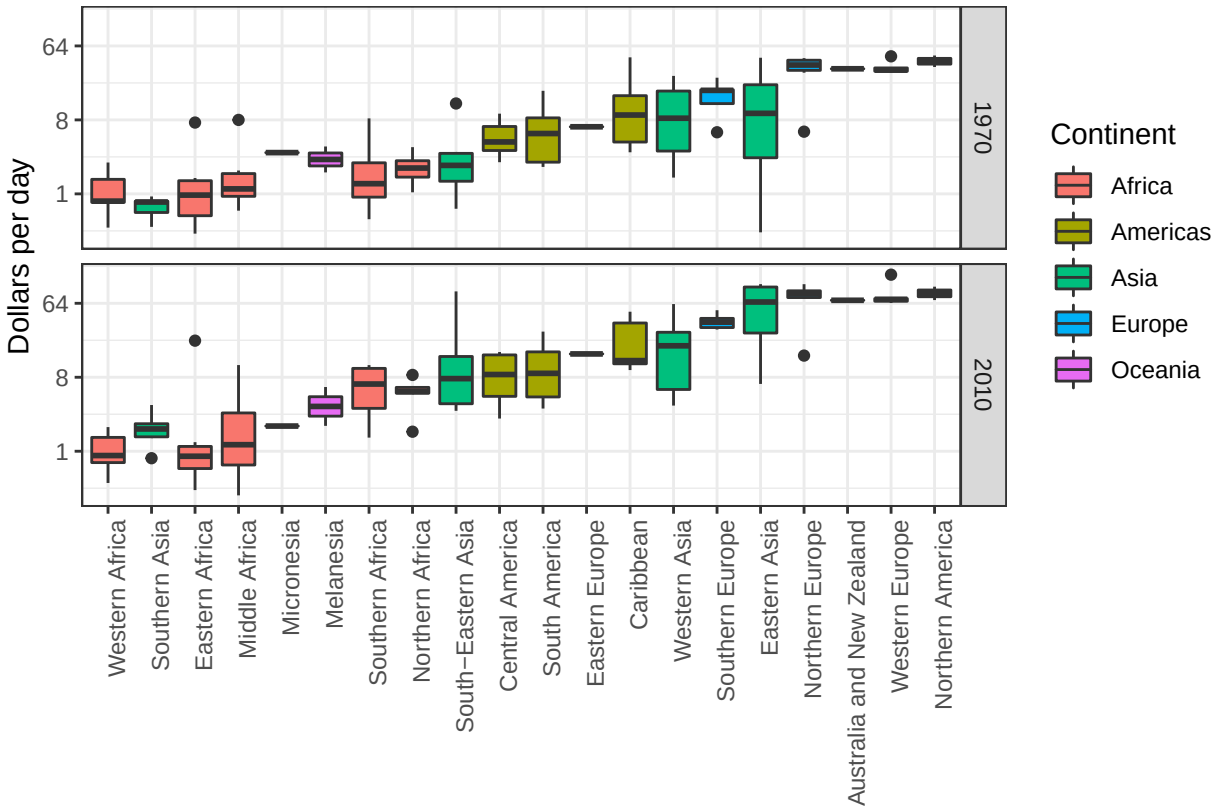
We now see that while the rich countries have become a bit richer, percentage-wise, the poor countries appear to have improved more. In particular we see that the proportion of *developing* countries earning more than $16 a day increases substantially.

To see which specific regions improved the most we can remake the boxplots we made above but now adding 2010

```
p <- gapminder %>%
  filter(year %in% c(past_year, present_year) & country %in% country_list) %>%
  mutate(region = reorder(region, dollars_per_day, FUN = median)) %>%
  ggplot() +
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  xlab("") +
  ylab("Dollars per day") +
  scale_y_continuous(trans = "log2") +
  labs(fill = "Continent")
```

and then using facet to compare the two years:

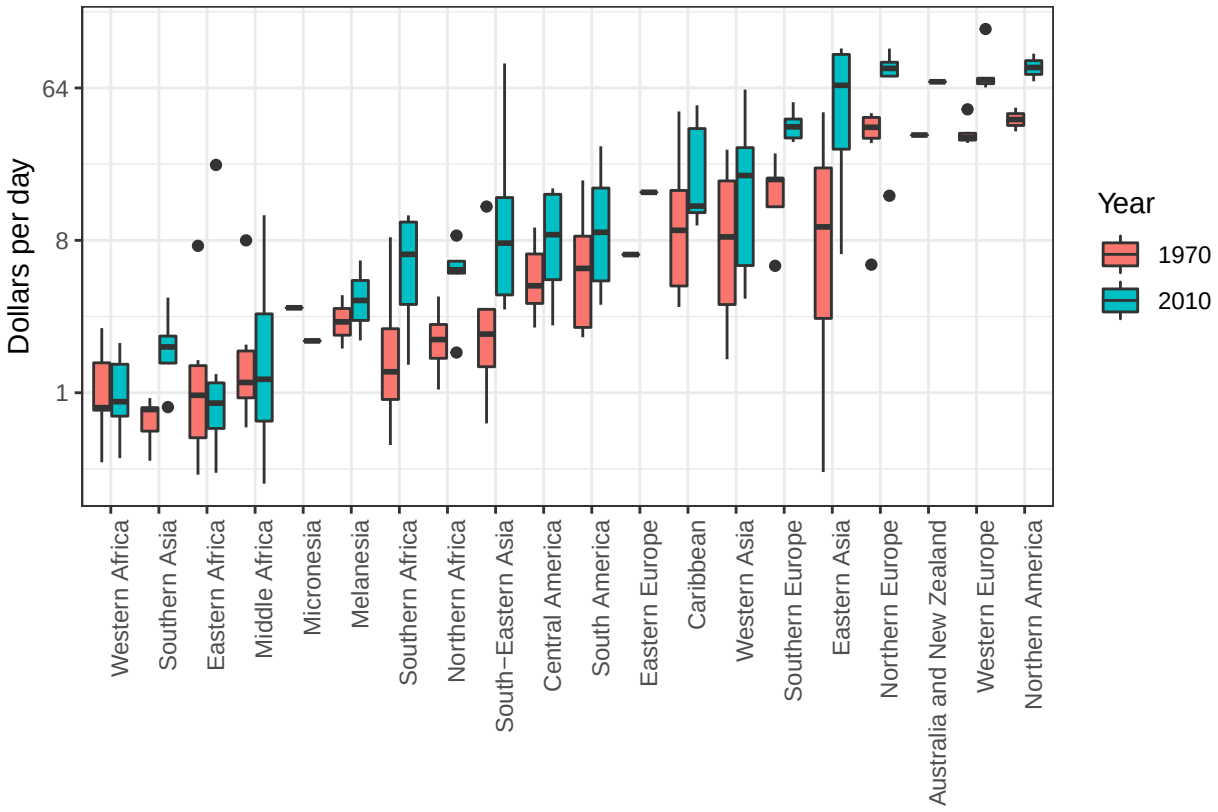
```
p + geom_boxplot(aes(region, dollars_per_day, fill = continent)) +
  facet_grid(year~.)
```



Here, we pause to introduce another powerful ggplot2 feature. Because we want to compare each region before and after, it would be convenient to have the 1970 boxplot next to the 2010 boxplot for each region. In general, comparisons are easier when data are plotted next to each other.

So, instead of faceting, we keep the data from each year together, but ask ggplot to color (or fill) them depending on the year. ggplot automatically separates them and puts the two boxplots next to each other. Because year is a number, we turn it into a factor because ggplot automatically assigns a color to each category of a factor:

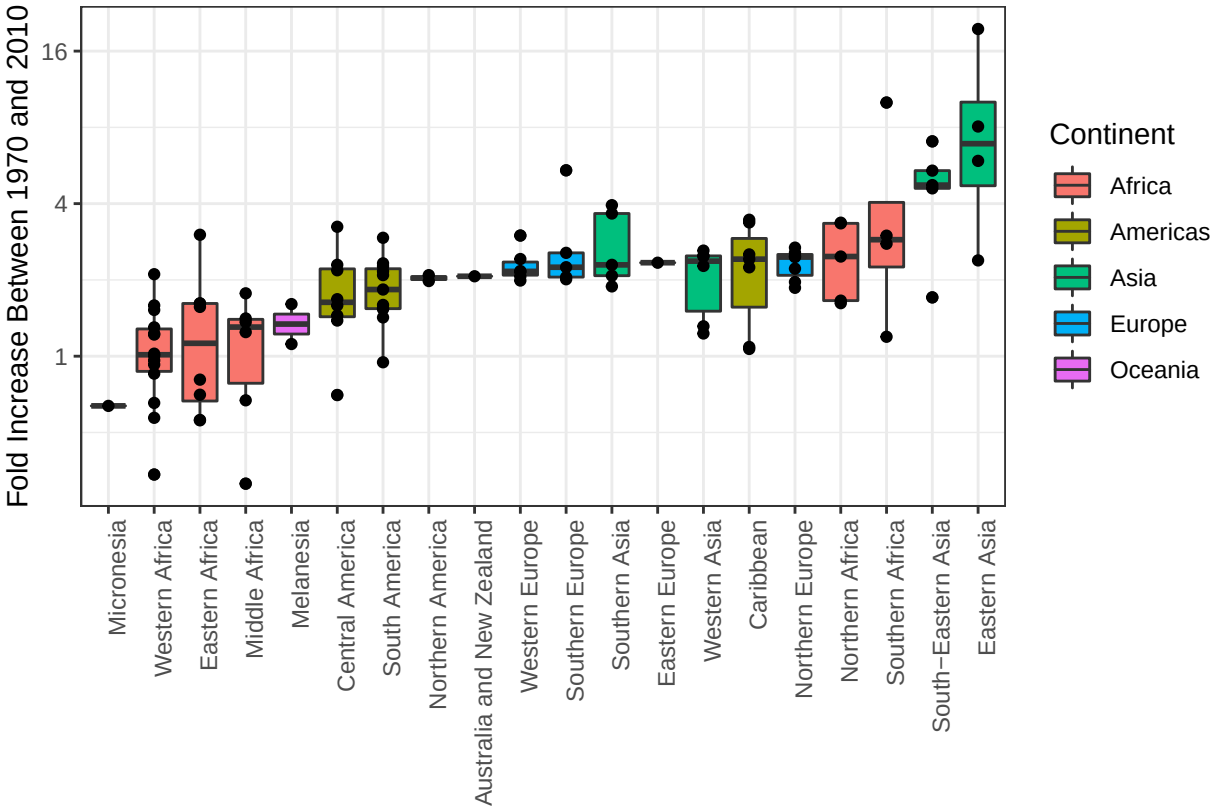
```
p + geom_boxplot(aes(region, dollars_per_day, fill = factor(year))) +
    labs(fill = "Year")
```



Finally, we point out that if what we are most interested is in comparing before and after values, it might make more sense to plot the ratios, or difference in the log scale. We are still not ready to learn to code this but here is what the plot would look like:

```
##      country                   region continent year dollars_per_day
## 1    Algeria           Northern Africa    Africa past       3.7172265
## 2  Argentina             South America  Americas past      18.1207496
## 3  Australia Australia and New Zealand   Oceania past      33.6671656
## 4    Austria            Western Europe    Europe past      30.2348264
## 5    Bahamas                 Caribbean  Americas past      46.4254181
## 6 Bangladesh             Southern Asia      Asia past       0.7950843
```

```
##      country                   region continent       past   present
## 1    Algeria           Northern Africa    Africa  3.7172265  6.018638
## 2  Argentina             South America  Americas 18.1207496 28.871158
## 3  Australia Australia and New Zealand   Oceania 33.6671656 69.602975
## 4    Austria            Western Europe    Europe 30.2348264 73.114157
## 5    Bahamas                 Caribbean  Americas 46.4254181 50.493566
## 6 Bangladesh             Southern Asia      Asia  0.7950843  1.499445
```



To make the y-axis label general we could use the `paste` function:

```
ylab(paste("Fold Increase Between", past_year, "and", present_year))
```

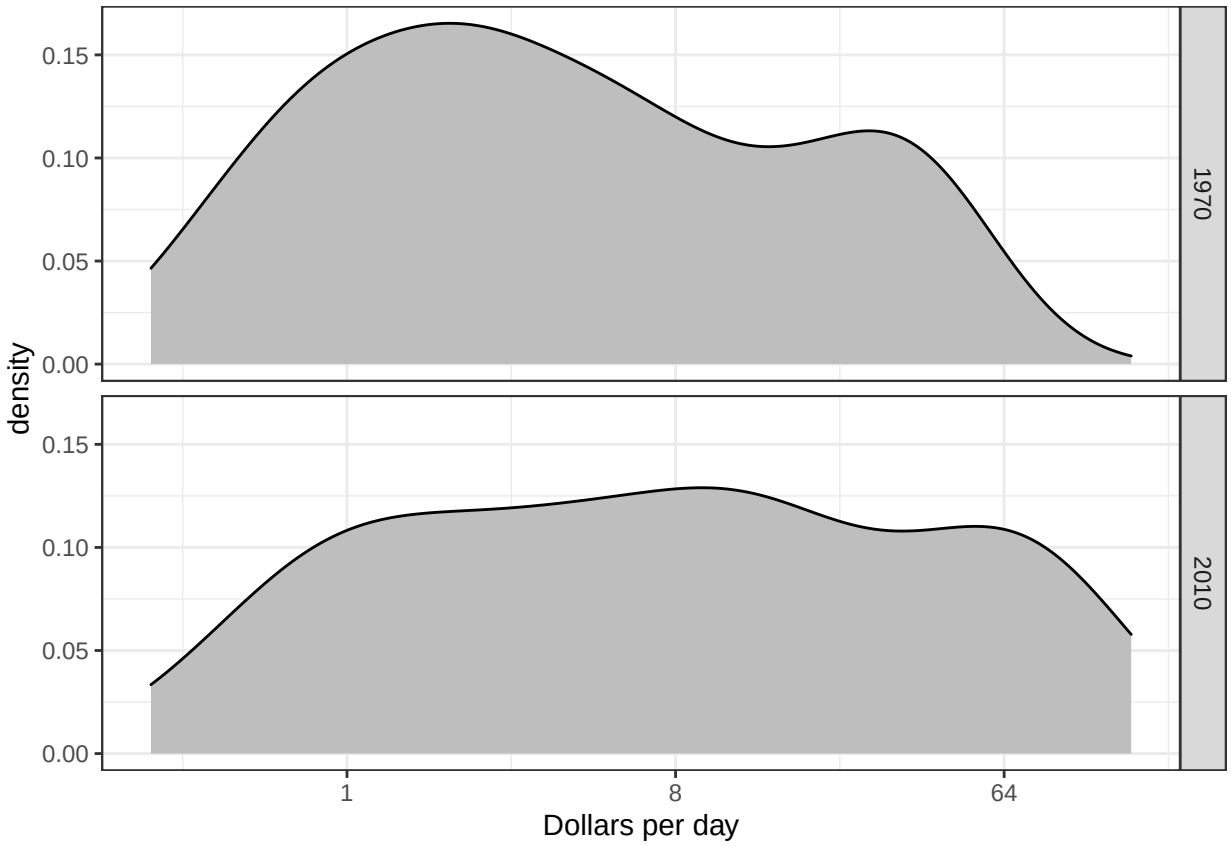
```
## $y
## [1] "Fold Increase Between 1970 and 2010"
##
## attr(,"class")
## [1] "labels"
```

Smooth Density Plots

We have used data exploration to discover that income gap between rich and poor countries has closed considerably during the last 40 years. We used a series of histograms and boxplots to see this. Here we suggest a succinct way to convey this message with just one plot. We will use smooth density plots.

Let's start by noting that density plots for income distribution in 1970 and 2010 deliver the message that the gap is closing:

```
gapminder %>%
  filter(year %in% c(past_year, present_year) & country %in% country_list) %>%
  ggplot(aes(dollars_per_day)) +
  geom_density(fill = "grey") +
  scale_x_continuous(trans = "log2") +
  xlab("Dollars per day") +
  facet_grid(year~.)
```



In the 1970 plot we see two clear modes, poor and rich countries. In 2010 it appears that some of the poor countries have shifted towards the right, closing the gap.

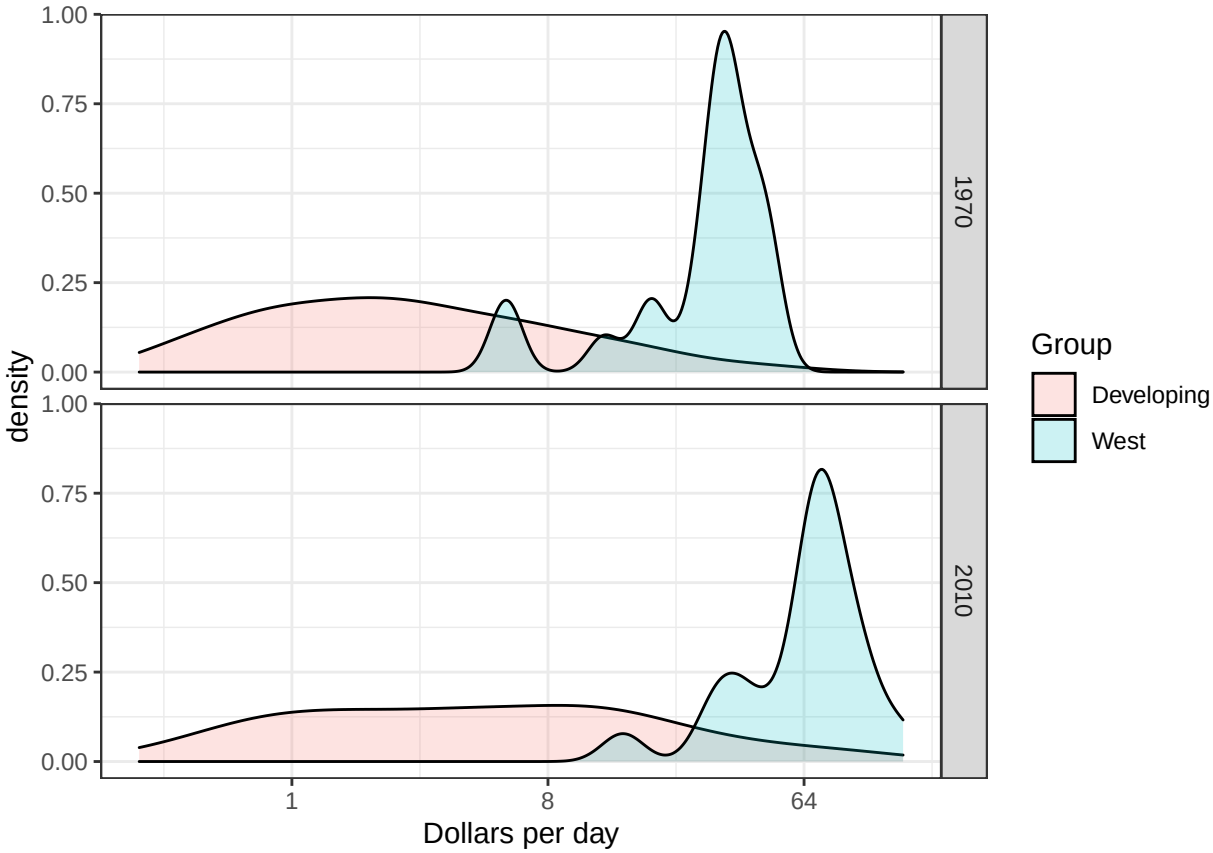
The next message we need to covey is that the reason for this change in distribution is that poor countries became richer rather than some rich countries becoming poorer. To do this all we need to do is assign a color to the groups we identified during data exploration.

However, before we can do this we need to learn how to make these smooth densities in a way that preserves information of how many countries are in each group. To understand why we need this, note the discrepancy in the size of each group:

group	n
Developing	87
West	21

but when we overlay two densities, the default is to have the area represented by each distribution add up to 1 regardless of the size of each group:

```
gapminder %>%
  filter(year %in% c(past_year, present_year) & country %in% country_list) %>%
  mutate(group = ifelse(region %in% west, "West", "Developing")) %>%
  ggplot(aes(dollars_per_day, fill = group)) +
  scale_x_continuous(trans = "log2") +
  geom_density(alpha = 0.2) +
  facet_grid(year ~ .) +
  xlab("Dollars per day") +
  labs(fill = "Group")
```



which make it appear as if there are the same number of countries in each group. To change this, we will need to learn to access computed variables with the `geom_density` function.

Accessing Computed Variables

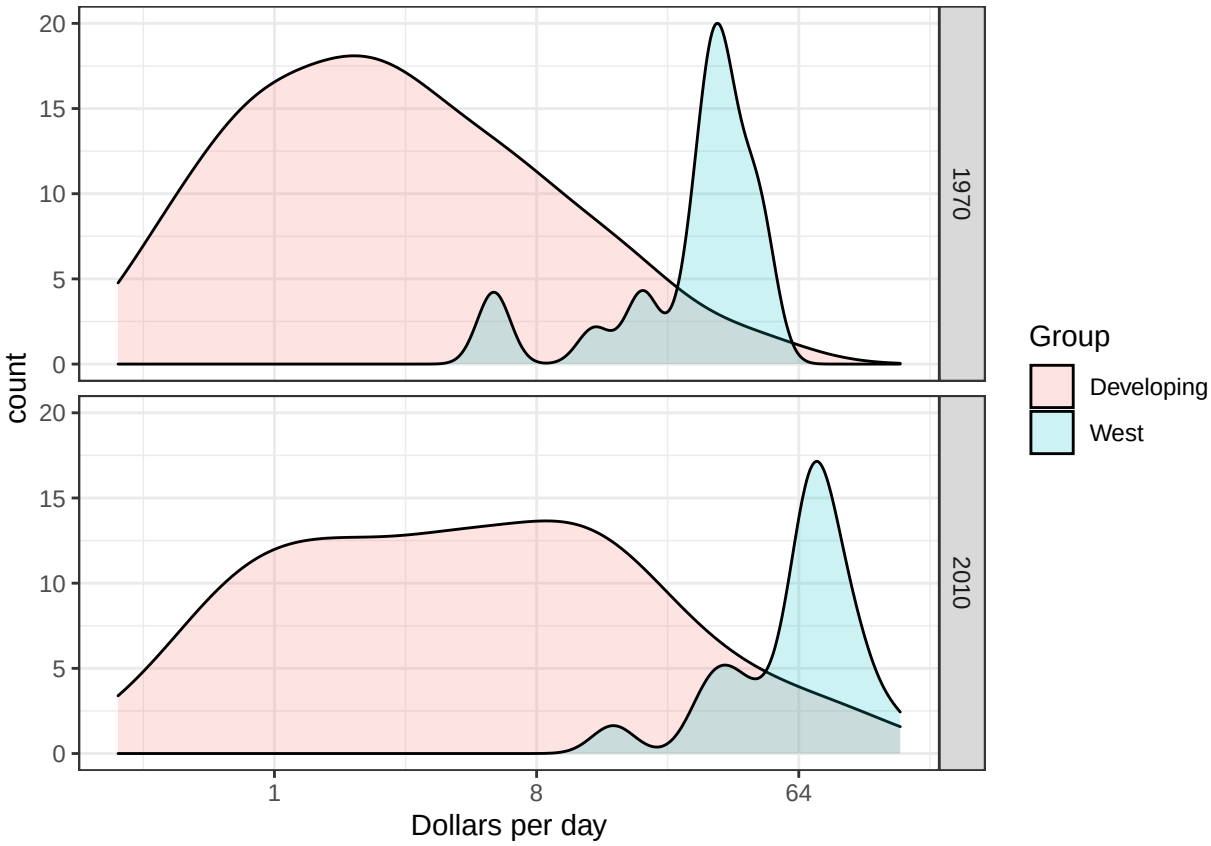
To have the areas of these densities be proportional to the size of the groups, we can simply multiply the y-axis values by the size of the group. From the `geom_density` help file we see that the function computes a variable called `count` that does exactly this. We want this variable to be on the y-axis rather than the density.

In ggplot we access these variables by surrounding `count` by `..` (two periods). So we will use the following mapping:

```
aes(x = dollars_per_day, y = ..count..)
```

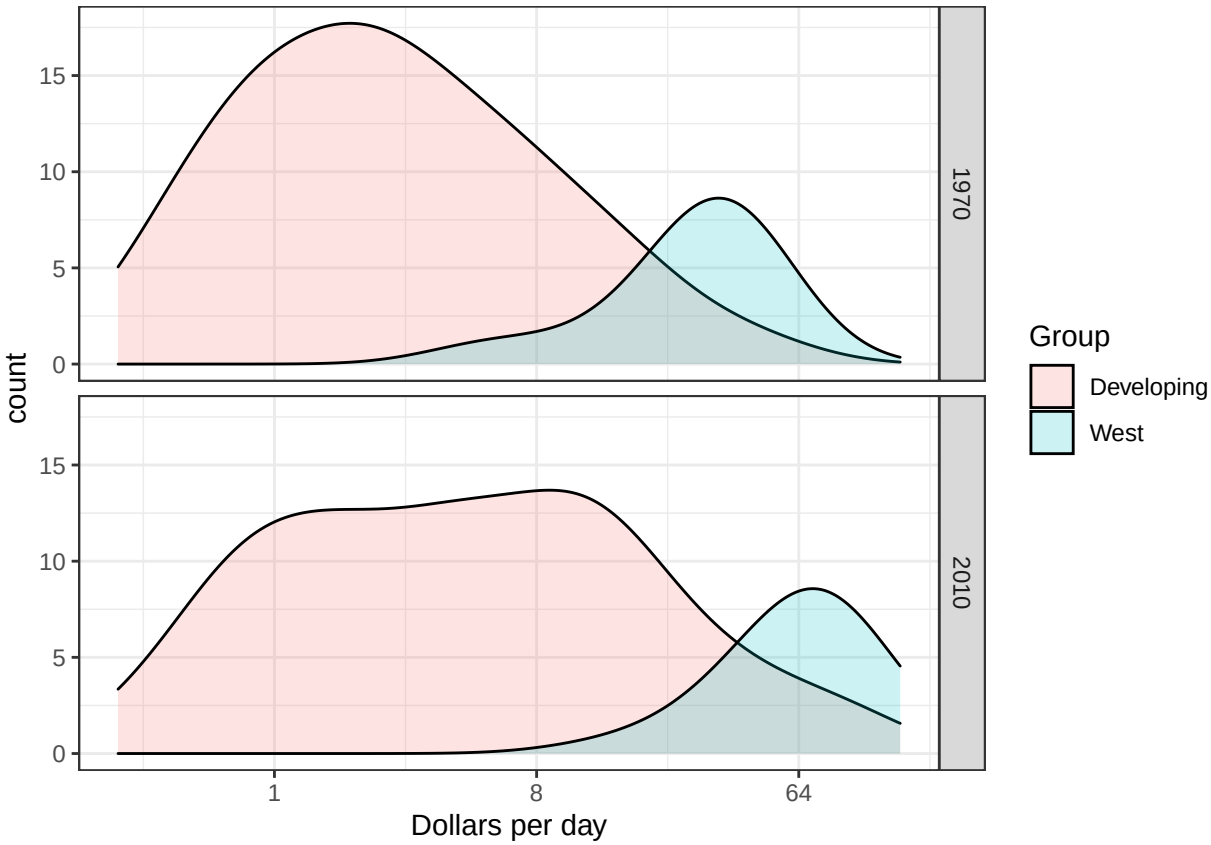
We can now create the desired plot by simply changing the mapping in the previous code chunk:

```
p <- gapminder %>%
  filter(year %in% c(past_year, present_year) & country %in% country_list) %>%
  mutate(group = ifelse(region %in% west, "West", "Developing")) %>%
  ggplot(aes(dollars_per_day, y = ..count.., fill = group)) +
  scale_x_continuous(trans = "log2") +
  xlab("Dollars per day") +
  labs(fill = "Group")

p + geom_density(alpha = 0.2) + facet_grid(year ~ .)
```



If we want the densities to be smoother, we use the `bw` argument. We tried a few and decided on 0.75:

```
p + geom_density(alpha = 0.2, bw = 0.75) + facet_grid(year ~ .)
```



This plot now shows what is happening very clearly. The developing world distribution is changing. A third mode appears consisting of the countries that most closed the gap.

'case_when'

We can actually make this figure somewhat more informative. From the exploratory data analysis we noticed that many of the countries that most improved were from Asia. We can easily alter the plot to show key regions separately.

We introduce the `case_when` function useful for defining groups:

```
gapminder <- gapminder %>%
  mutate(group = case_when(
    region %in% west ~ "West",
    region %in% c("Eastern Asia", "South-Eastern Asia") ~ "East Asia",
    region %in% c("Caribbean", "Central America", "South America") ~ "Latin America",
    continent == "Africa" & region != "Northern Africa" ~ "Sub-Saharan Africa",
    TRUE ~ "Others"))
```

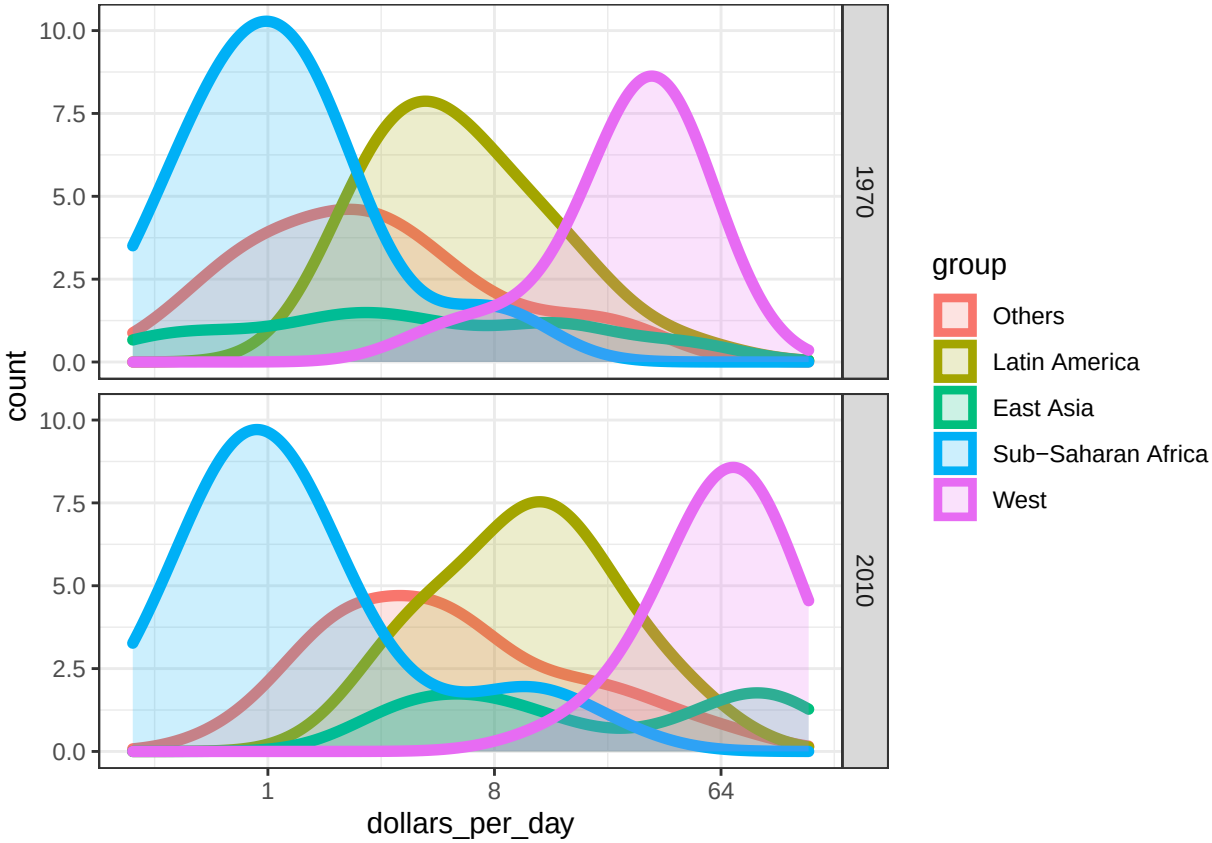
We turn this `group` variable into a factor to control the order of the levels:

```
gapminder <- gapminder %>%
  mutate(group = factor(group,
                        levels = c("Others", "Latin America",
                                   "East Asia","Sub-Saharan Africa",
                                   "West")))
```

We pick this particular order for a reason that becomes clear later.

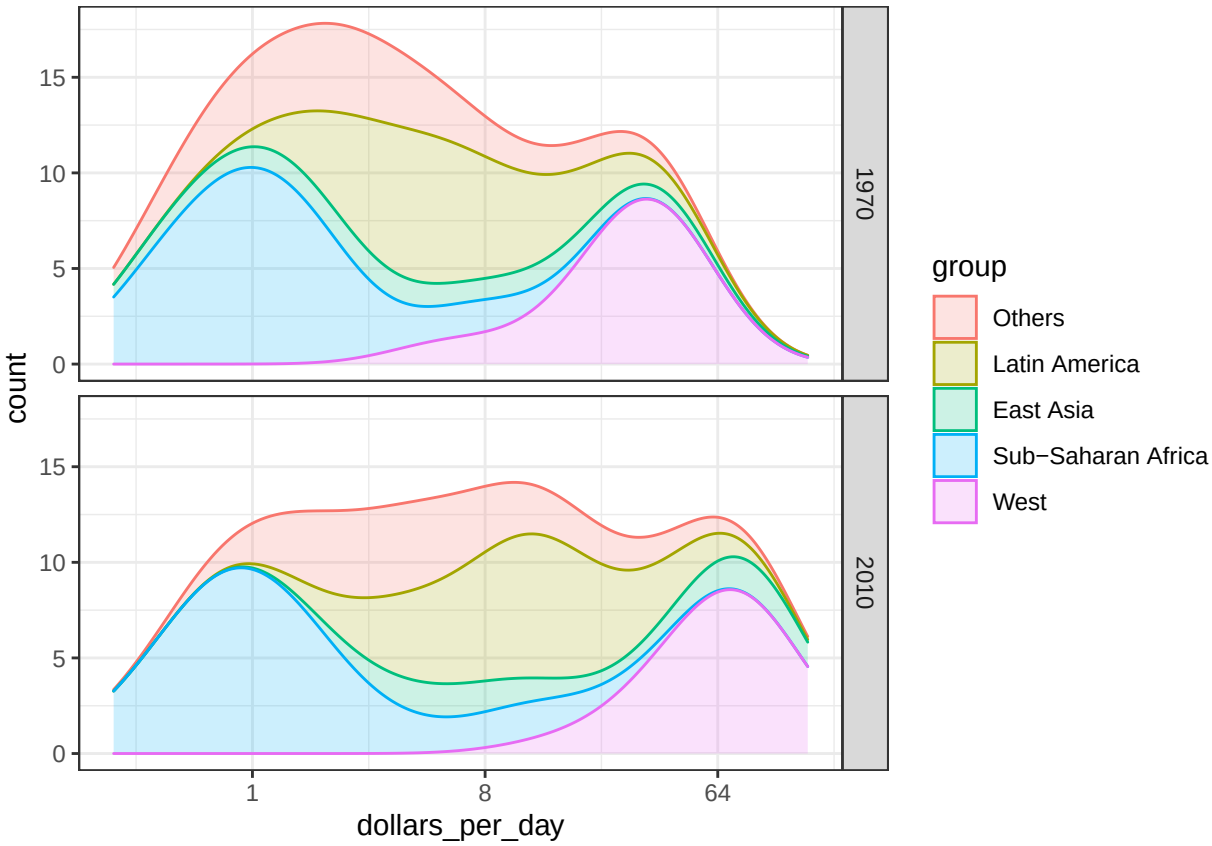
We can now easily plot the densities for each. We use `color` and `size` to clearly see the tops:

```
p <- gapminder %>%
    filter(year %in% c(past_year, present_year) & country %in% country_list) %>%
  ggplot(aes(dollars_per_day, y = ..count.., fill = group, color = group)) +
  scale_x_continuous(trans = "log2")

p + geom_density(alpha = 0.2, bw = 0.75, size = 2) + facet_grid(year ~ .)
```



The plot is cluttered and somewhat hard to read. A clearer picture is sometimes achieved by stacking the densities on top of each other:

```
p + geom_density(alpha = 0.2, bw = 0.75, position = "stack") + facet_grid(year ~ .)
```

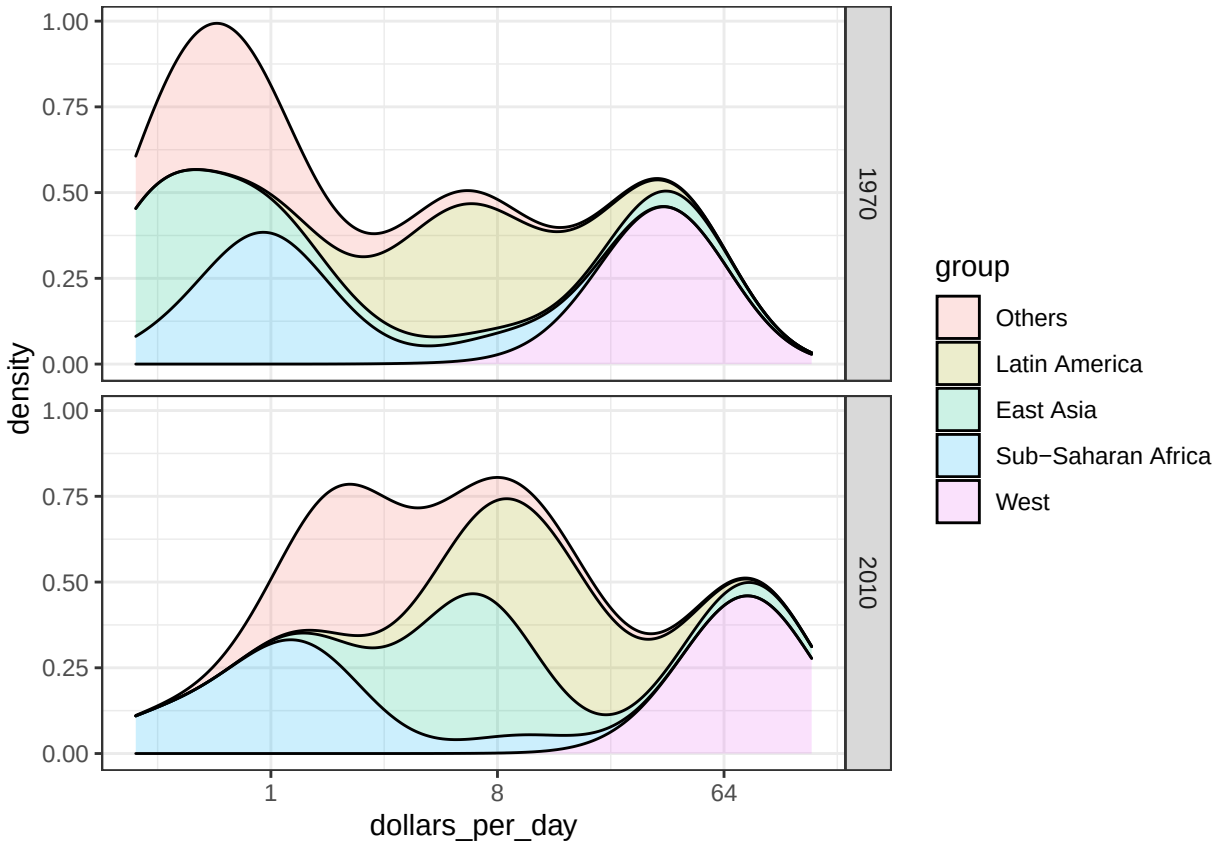


Here we can clearly see how the distributions for East Asia, Latin America and Others shift markedly to the right. While Sub-Saharan Africa remains stagnant.

Note that we order the levels of the group so that The West density be plotted first, then Sub-Saharan Africa. *Having the two extremes be plotted first let's us see the remaining bimodality better.*

Weighted densities As a final point, we note that these distributions weigh every country the same. So if most of the population is improving, but living in a very large country, such as China, we might not appreciate this. We can actually weight the smooth densities using the `weight` mapping argument. Here we weight by population:

```
gapminder %>%
    filter(year %in% c(past_year, present_year) & country %in% country_list) %>%
  group_by(year) %>%
  mutate(weight = population/sum(population)) %>%
  ungroup() %>%
  ggplot(aes(dollars_per_day, fill = group, weight = weight)) +
  scale_x_continuous(trans = "log2") +
  geom_density(alpha = 0.2, bw = 0.75, position = "stack") + facet_grid(year ~ .)
```



This particular figure shows very clearly how the income distribution gap is closing with most of the poor remaining in Sub-Saharan Africa.

Ecological Fallacy

Throughout this section we have been comparing regions of the world. We have seen that on average, some regions do better than others. Now we focus on describing the importance of variability within the groups.

Here we will focus on the relationship between country child survival rates and average income. We start by comparing these quantities across regions. We define a few more regions:

```
gapminder <- gapminder %>%
  mutate(group = case_when(
    region %in% west ~ "The West",
    region %in% "Northern Africa" ~ "Northern Africa",
    region %in% c("Eastern Asia", "South-Eastern Asia") ~ "East Asia",
    region == "Southern Asia"~ "Southern Asia",
    region %in% c("Central America", "South America", "Caribbean") ~ "Latin America",
    continent == "Africa" & region != "Northern Africa" ~ "Sub-Saharan Africa",
    region %in% c("Melanesia", "Micronesia", "Polynesia") ~ "Pacific Islands"))
```

We then compute these quantities for each region.

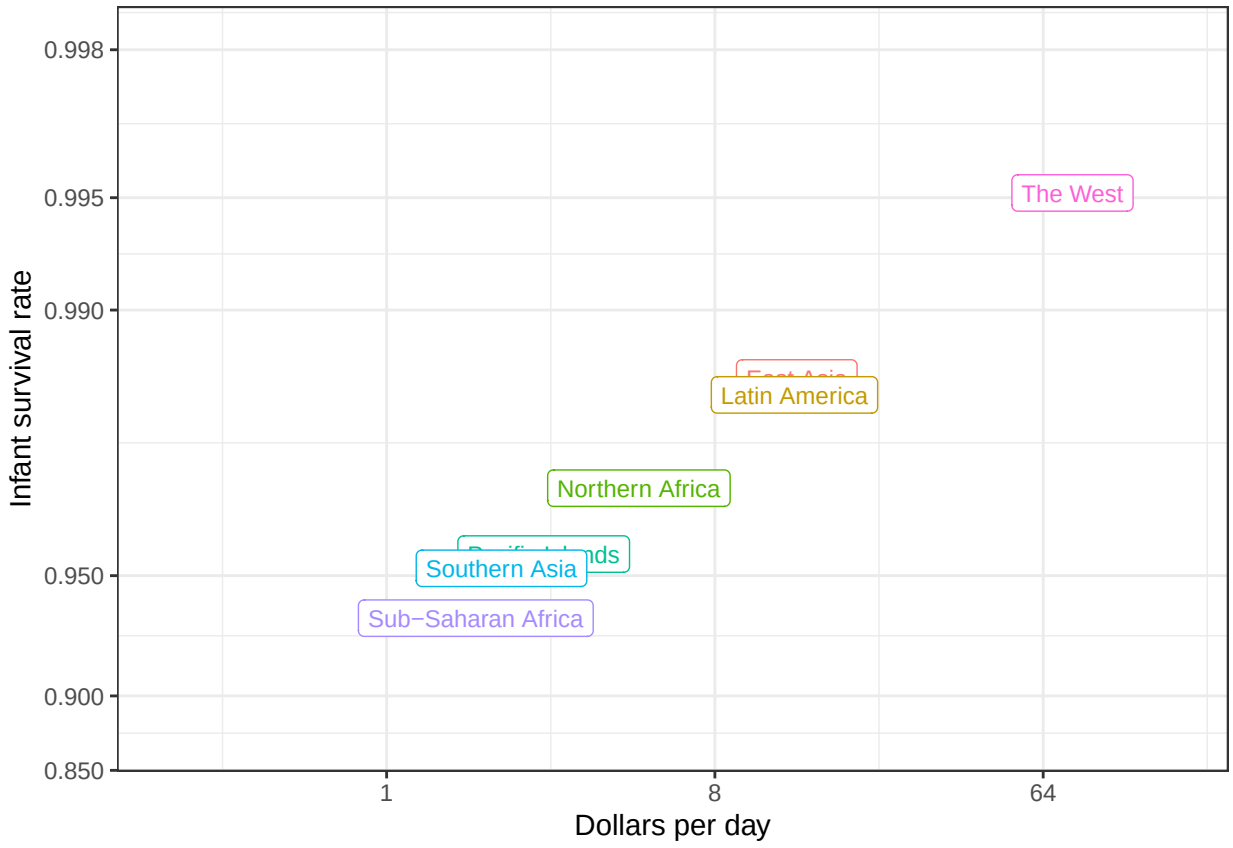
```
surv_income <- gapminder %>%
  filter(year %in% present_year & !is.na(gdp) & !is.na(infant_mortality) & !is.na(group)) %>%
  group_by(group) %>%
  summarize(income = sum(gdp)/sum(population)/365,
            infant_survival_rate = 1-sum(infant_mortality/1000*population)/sum(population))
```

```
surv_income %>% arrange(income)
```

```
## # A tibble: 7 x 3
##   group              income infant_survival_rate
##   <chr>               <dbl>                <dbl>
## 1 Sub-Saharan Africa   1.76                0.936
## 2 Southern Asia        2.07                0.952
## 3 Pacific Islands      2.70                0.956
## 4 Northern Africa      4.94                0.970
## 5 Latin America       13.2                 0.983
## 6 East Asia           13.4                 0.985
## 7 The West            77.1                 0.995
```

This shows a dramatic difference. While in the west less than 0.5% children die, in Sub-Saharan Africa the rate is higher than 6%! The relationship between these two variables is almost perfectly linear.

```
surv_income %>% ggplot(aes(income, infant_survival_rate, label = group, color = group)) +
  scale_x_continuous(trans = "log2", limit = c(0.25, 150)) +
  scale_y_continuous(trans = "logit", limit = c(0.875, .9981),
                     breaks = c(.85,.90,.95,.99,.995,.998)) +
  geom_label(size = 3, show.legend = FALSE) +
  xlab("Dollars per day") +
  ylab("Infant survival rate")
```



In this plot we introduce the use of the `limit` argument which lets us change the range of the axes. We are making the range larger than the data needs because we will later compare this plot to one with more variability and we want the ranges to be the same. We also introduce the `breaks` argument which lets us set the location of the axis tick labels. Finally we introduce a new transformation, the logistic transformation.

Logistic transformation The logistic or logit transformation for a proportion or rate p is defined as

$$f(p) = \log\left(\frac{p}{1-p}\right)$$

When p is a proportion or probability, the quantity that is being logged, $p/(1-p)$ is called the *odds*. In this case p is the proportion of children that survived. The odds tell us how many more children are expected to survive than to die. The log transformation makes this symmetric. If the rates are the same, then the log odds is 0. Fold increases or decreases turn into positive and negative increments respectively.

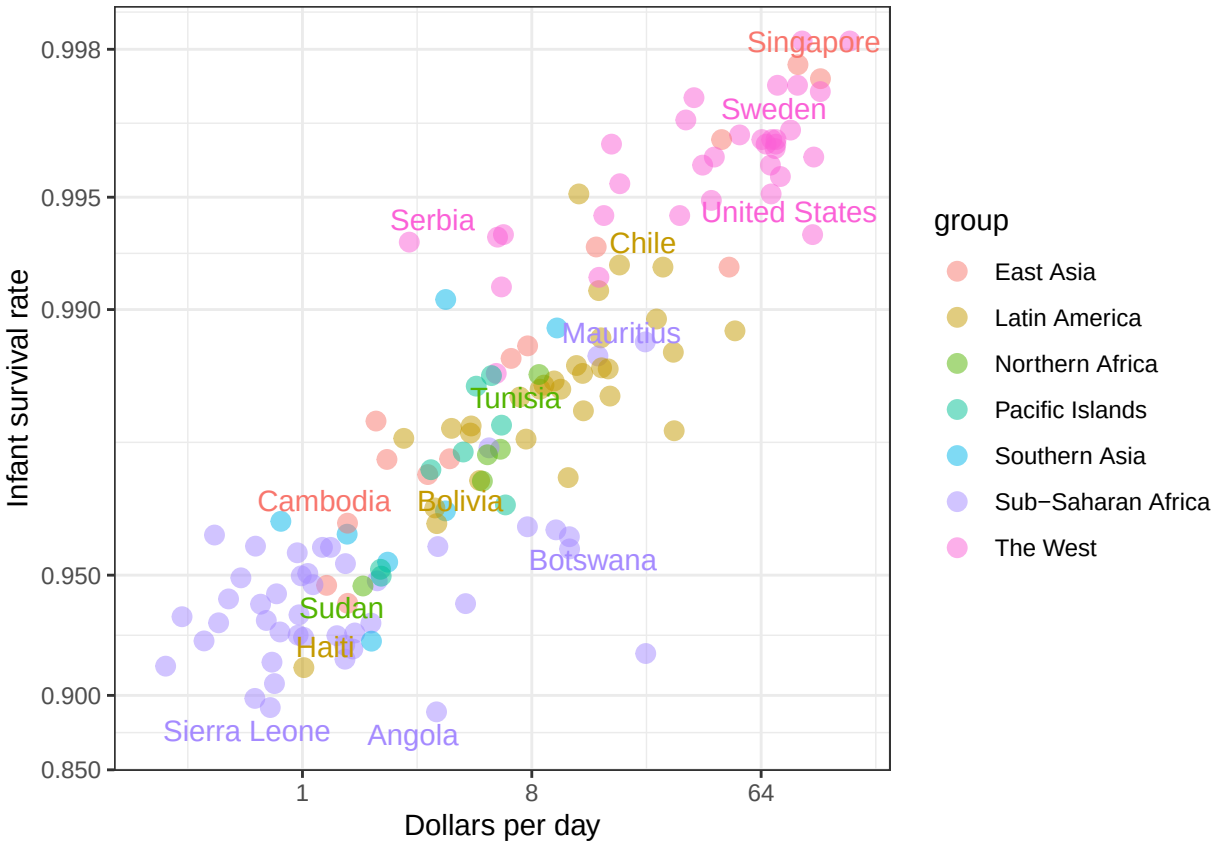
This scale is useful when we want to highlight differences near 0 or 1. For survival rates this is important because a survival rate of 90% is unacceptable, while a survival of 99% is relatively good. We would much prefer a survival rate closer to 99.9%. We want our scale to highlight these differences and the logit does this. Note that 99.9/0.1 is about 10 times bigger than 99/1 which is about 10 times larger than 90/10. And by using the log, these fold changes turn into constant increases.

Show the data

Now, back to our plot. Based on the plot above, do we conclude that a country with a low income is destined to have low survival rate? Do we conclude that all survival rates in Sub-Saharan Africa are all lower than in Southern Asia which in turn are lower than in the Pacific Islands, and so on?

Jumping to this conclusion based on a plot showing averages is referred to as the *ecological fallacy*. The almost perfect relationship between survival rates and income is only observed for the averages at the region level. Once we show all the data we see a somewhat more complicated story:

```
library(ggrepel)
highlight <- c("Sierra Leone", "Mauritius",  "Sudan", "Botswana", "Tunisia",
               "Cambodia","Singapore","Chile", "Haiti", "Bolivia",
               "United States","Sweden", "Angola", "Serbia")

gapminder %>% filter(year %in% present_year & !is.na(gdp) & !is.na(infant_mortality) & !is.na(group)) %
  ggplot(aes(dollars_per_day, 1 - infant_mortality/1000, color = group, label = country)) +
  scale_x_continuous(trans = "log2", limits=c(0.25, 150)) +
  scale_y_continuous(trans = "logit",limit=c(0.875, .9981),
                     breaks=c(.85,.90,.95,.99,.995,.998)) +
  geom_point(alpha = 0.5, size = 3) +
  geom_text_repel(size = 4, show.legend = FALSE,
    data = filter(gapminder, year %in% present_year & country %in% highlight)) +
  xlab("Dollars per day") +
  ylab("Infant survival rate") +
  labs("Region")
```



Specifically, we see that there is a large amount of variability. We see that countries from the same regions can be quite different and that countries with the same income can have different survival rates. For example, while on average, Sub-Saharan Africa had the worse health and economic outcomes, there is wide variability within that group. For example, note that Mauritius and Botswana are doing better than Angola and Sierra Leone with Mauritius comparable to Western countries.

Data Visualization Principles

We have already provided some rules to follow as we created plots for our examples. Here we aim to provide some general principles we can use as a guide for effective data visualization. Much of this section is based on a talk by Karl Broman titled "Creating effective figures and tables" including some of the figures which were made with code that Karl makes available on his GitHub repository, and class notes from Peter Aldhous' Introduction to Data Visualization course.

Following Karl's approach, we show some examples of plot styles we should avoid, explain how to improve them, and use these as motivation for a list of principles.We compare and contrast plots that follow these principles to those that don't.

The principles are mostly based on research related to how humans detect patterns and make visual comparisons. The preferred approaches are those that best fit the way our brains process visual information. When deciding on a visualization approach it is also important to keep our goal in mind. We may be comparing a viewable number of quantities, describing a distribution for categories or numeric values, comparing the data from two groups, or describing the relationship between two variables.

As a final note, we also note that for a data scientist it is important to adapt and optimize graphs to the audience. For example, an exploratory plot made for ourselves will be different than a chart intended to

communicate a finding to a general audience.

We will be using these libraries:

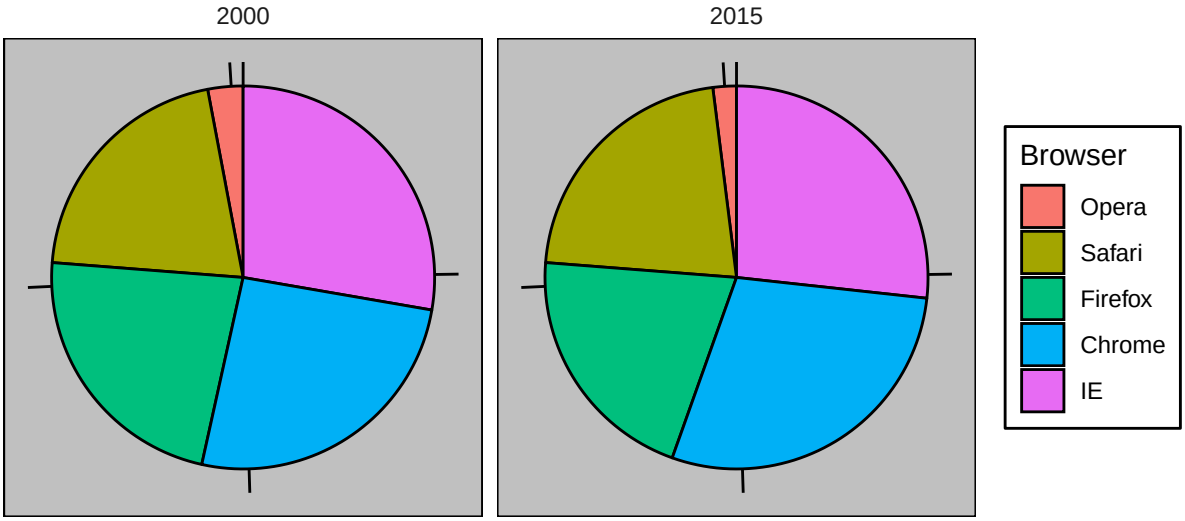
```
library(tidyverse)
library(gridExtra)
library(dslabs)
ds_theme_set()
```

Encoding data using visual cues

We start by describing some principles for encoding data. There are several approaches at our disposal including *position*, *aligned lengths*, *angles*, *area*, *brightness*, and *color hue*.

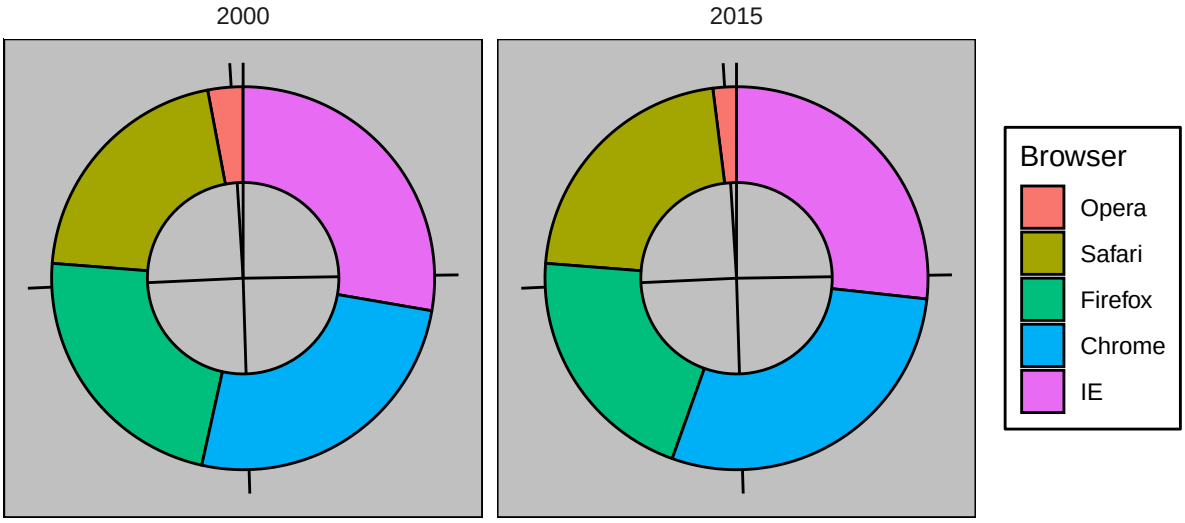
To illustrate how some of these strategies compare let's suppose we want to report the results from two hypothetical polls regarding browser preference taken in 2000 and then 2015. Here, for each year, we are simply comparing five quantities, five percentages.

A widely used graphical representation of percentages, popularized by Microsoft Excel, is the pie chart:



Here we are representing quantities with both areas and angles since both the angle and area of each pie slice is proportional to the quantity it represents. This turns out to be a suboptimal choice since, as demonstrated by perception studies, *humans are not good at precisely quantifying angles and are even worse when only area is available*.

The donut chart is an example of a plot that uses only area:



To see how hard it is to quantify angles, note that the rankings and all the percentages in the plots above changed from 2000 to 2015. Can you determine the actual percentages and rank the browsers' popularity? Can you see how the percentages changed from 2000 to 2015? It is not easy to tell from the plot.

In fact, the `pie` R function help file states:

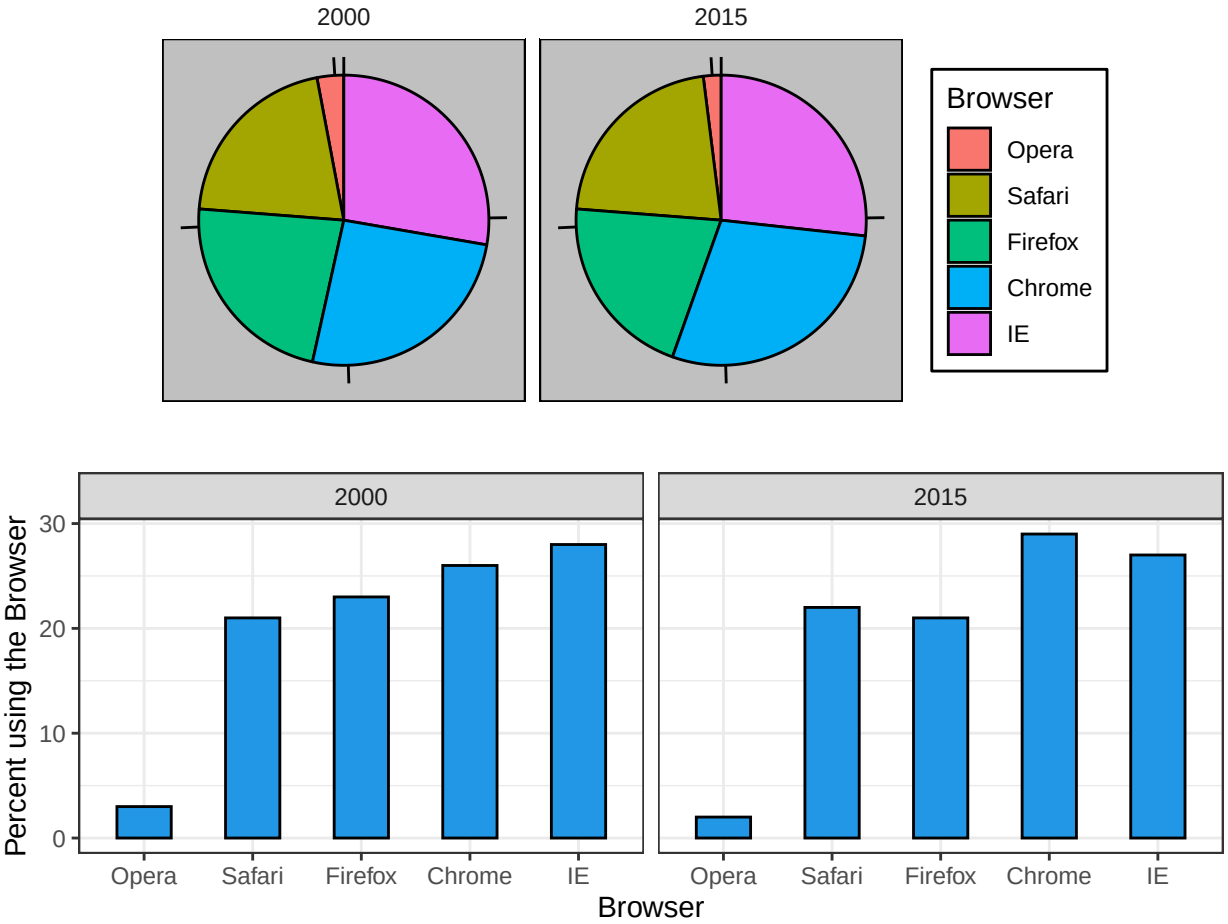
> "Note: Pie charts are a very bad way of displaying information. The eye is good at judging linear measures and bad at judging relative areas. A bar chart or dot chart is a preferable way of displaying this type of data."

In this case, simply showing the numbers is not only clearer, but it would save on print cost if making a paper version.

Browser	2000	2015
Opera	3	2
Safari	21	22
Firefox	23	21
Chrome	26	29
IE	28	27

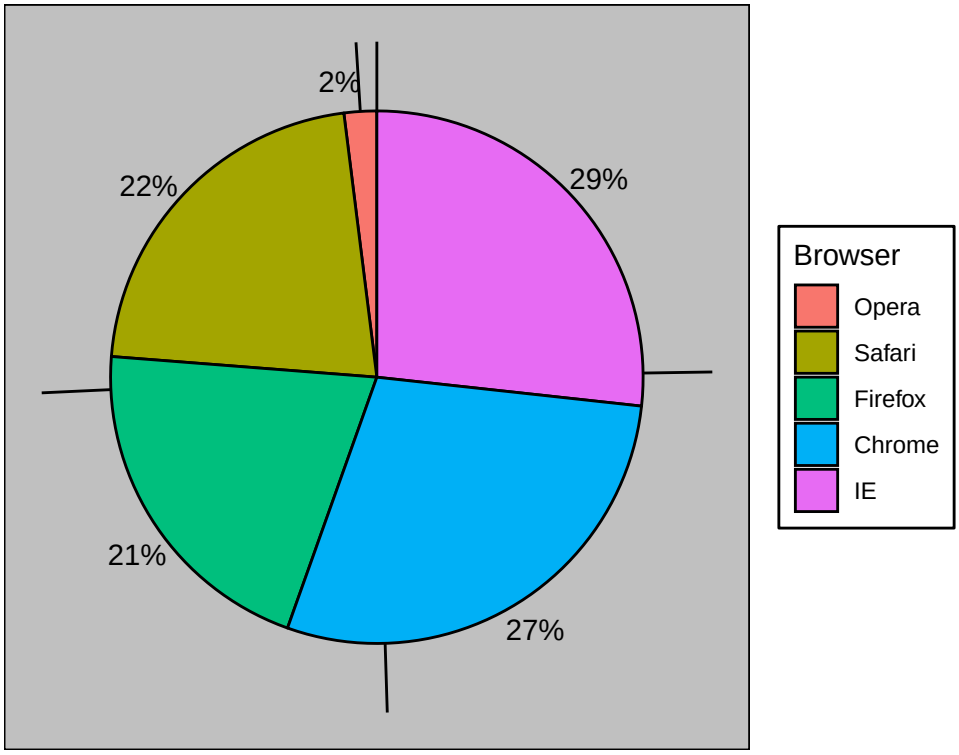
The preferred way to plot quantities is to use *length* and *position* since humans are much better at judging *linear measure*. The bar plot uses bars of length proportional to the quantities of interest. By adding horizontal lines at strategically chosen values, in this case at every multiple of 10, we ease the quantifying through the position of the top of the bars.

```
p2 <-browsers %>%
  ggplot(aes(Browser, Percentage)) +
  geom_bar(stat = "identity", width=0.5, fill=4, col = 1) +
  ylab("Percent using the Browser") +
  facet_grid(.~Year)
grid.arrange(p1, p2, nrow = 2)
```



Notice how much easier it is to see the differences in the barplot. In fact, we can now determine the actual percentages by following a horizontal line to the x-axis.

If for some reason you need to make a pie chart, do include the percentages as numbers to avoid having to infer them from the angles or area:



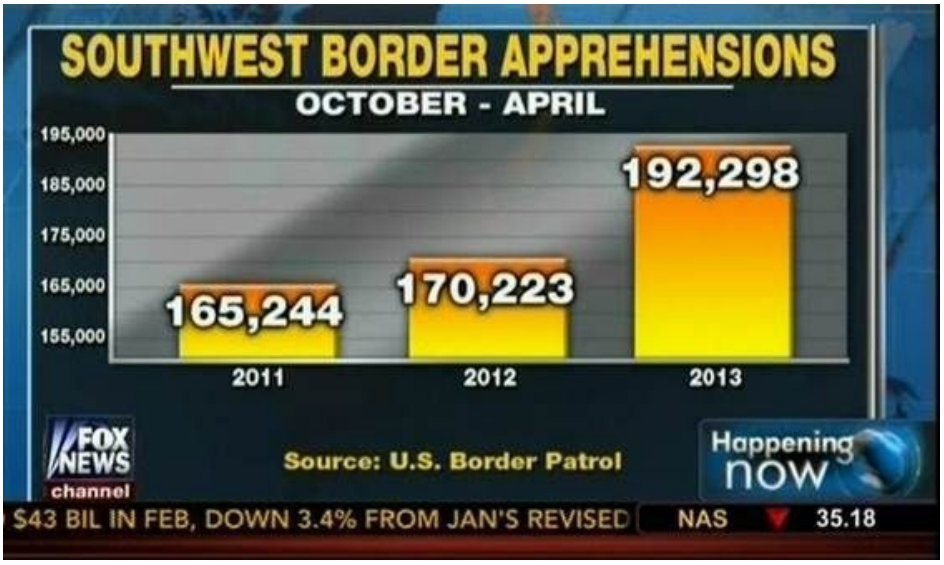
In general, position and length are the preferred ways to display quantities over angles which are preferred to area.

Brightness and color are even harder to quantify than angles and area but, as we will see later, they are sometimes useful when more than two dimensions are being displayed.

Know when to include 0

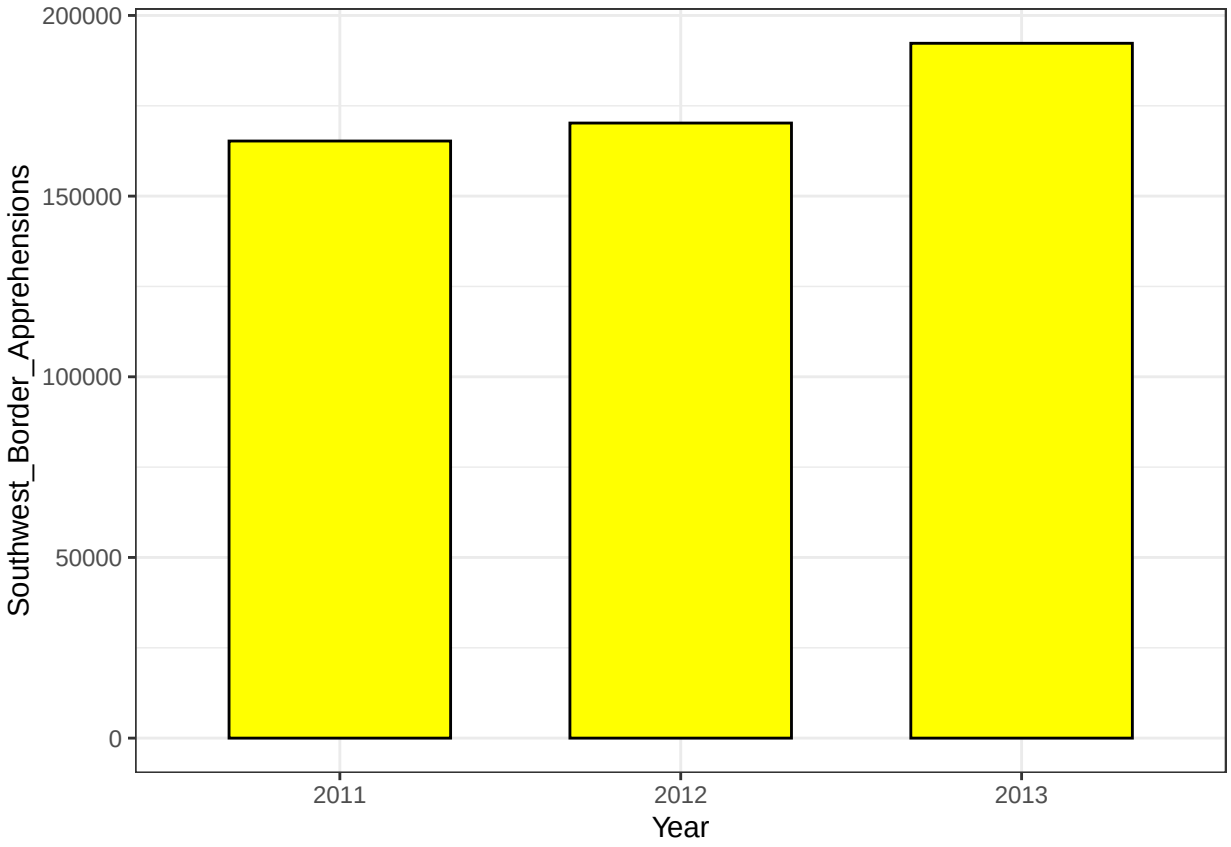
When using barplots it is dishonest not to start the bars at 0. This is because, by using a barplot, we are implying the length is proportional to the quantities being displayed. By avoiding 0, relatively small differences can be made to look much bigger than they actually are. This approach is often used by politicians or media organizations trying to exaggerate a difference.

Here is an illustrative example:

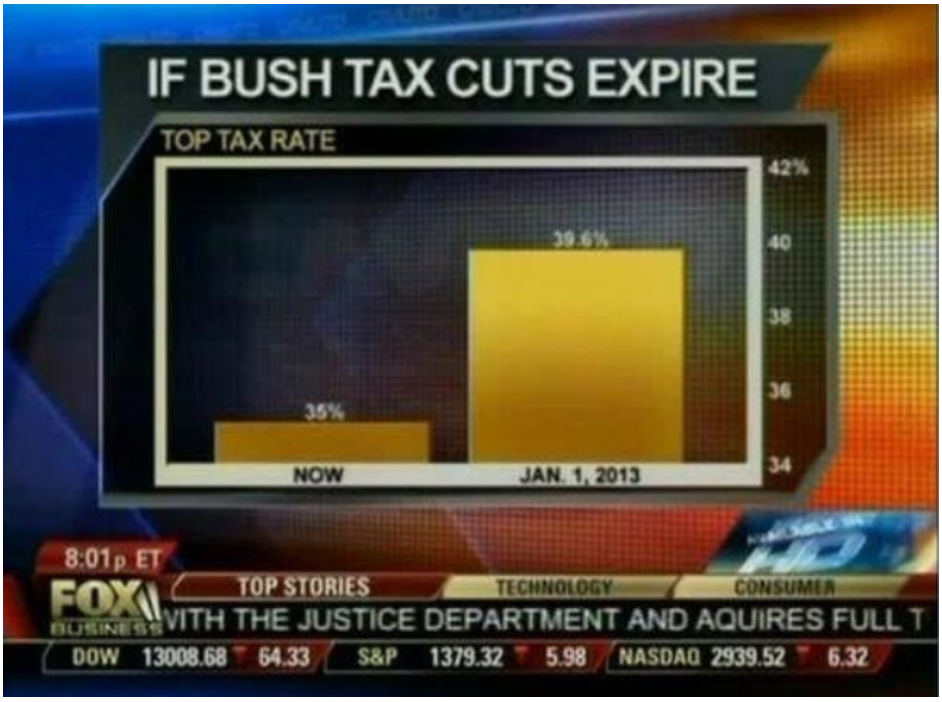


(Source: Fox News, via Peter Aldhous via Media Matters via Fox News) via Media Matters.

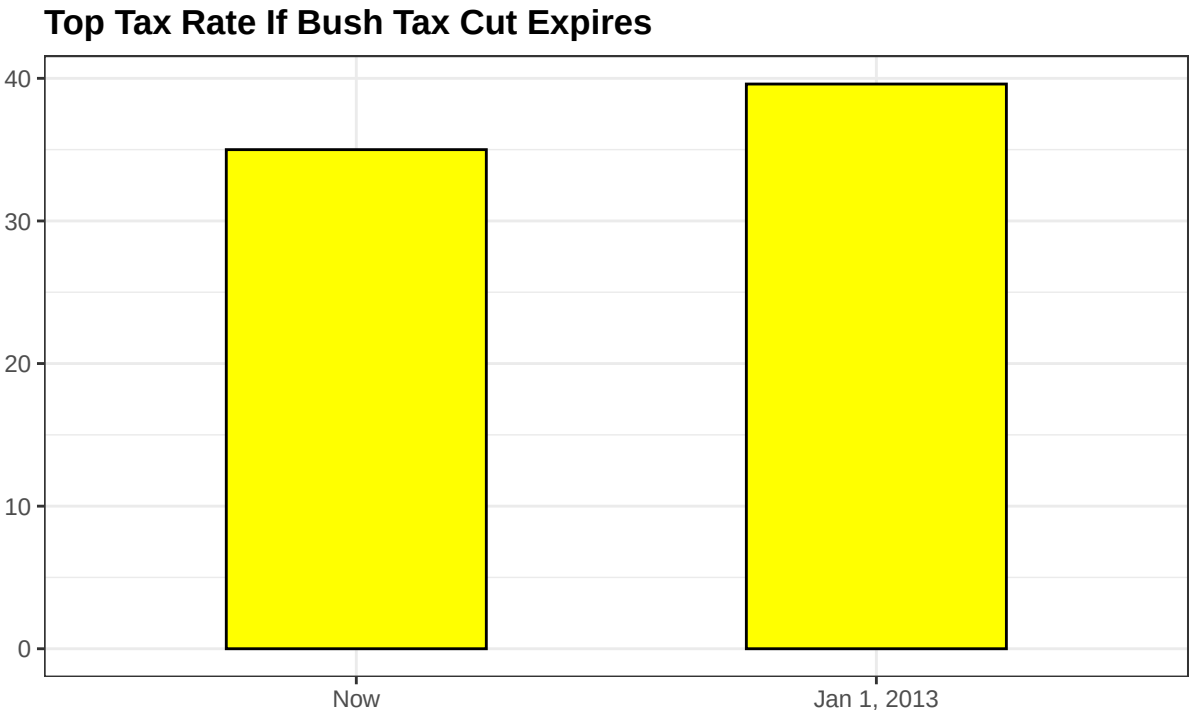
From the plot above, it appears that apprehensions have almost tripled when in fact they have only increased by about 16%. Starting the graph at 0 illustrates this clearly:



Here is another example, described in detail here, which makes a 4.6% increase look like a five fold change.

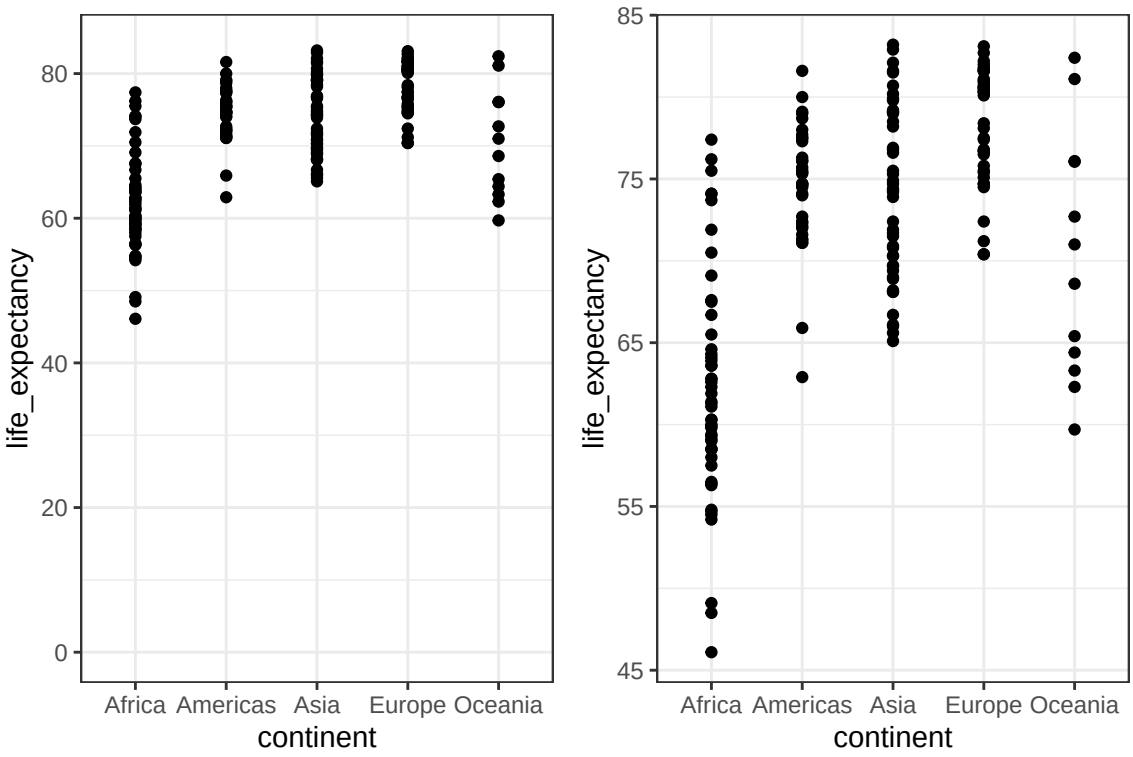


Here is the appropriate plot:



When using position rather than length, it is **not** necessary to include 0. This is particularly the case when we want to compare differences between groups relative to the variability seen within the groups.

Here is an illustrative example showing country average life expectancy stratified by continent in 2012:

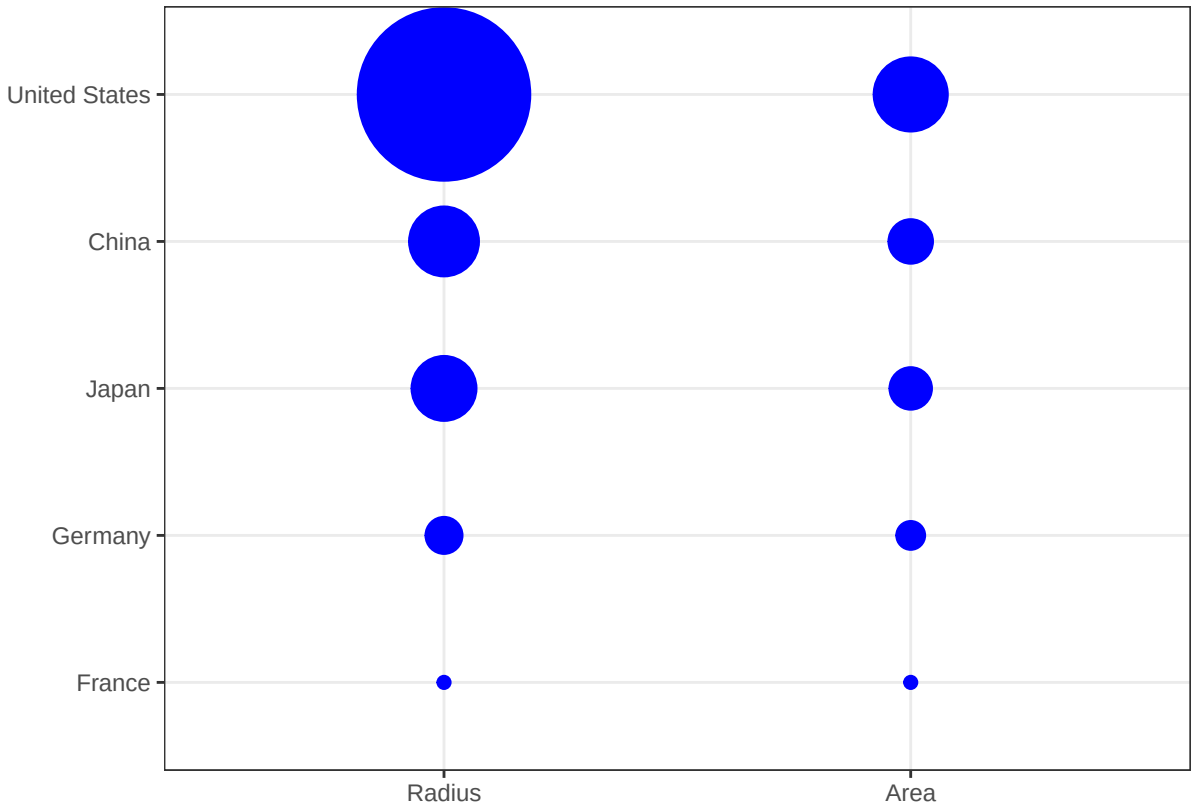


The space between 0 and 43 in the plot on the left adds no information and makes it harder to appreciate the between and within variability. Here, 0 should not be included.

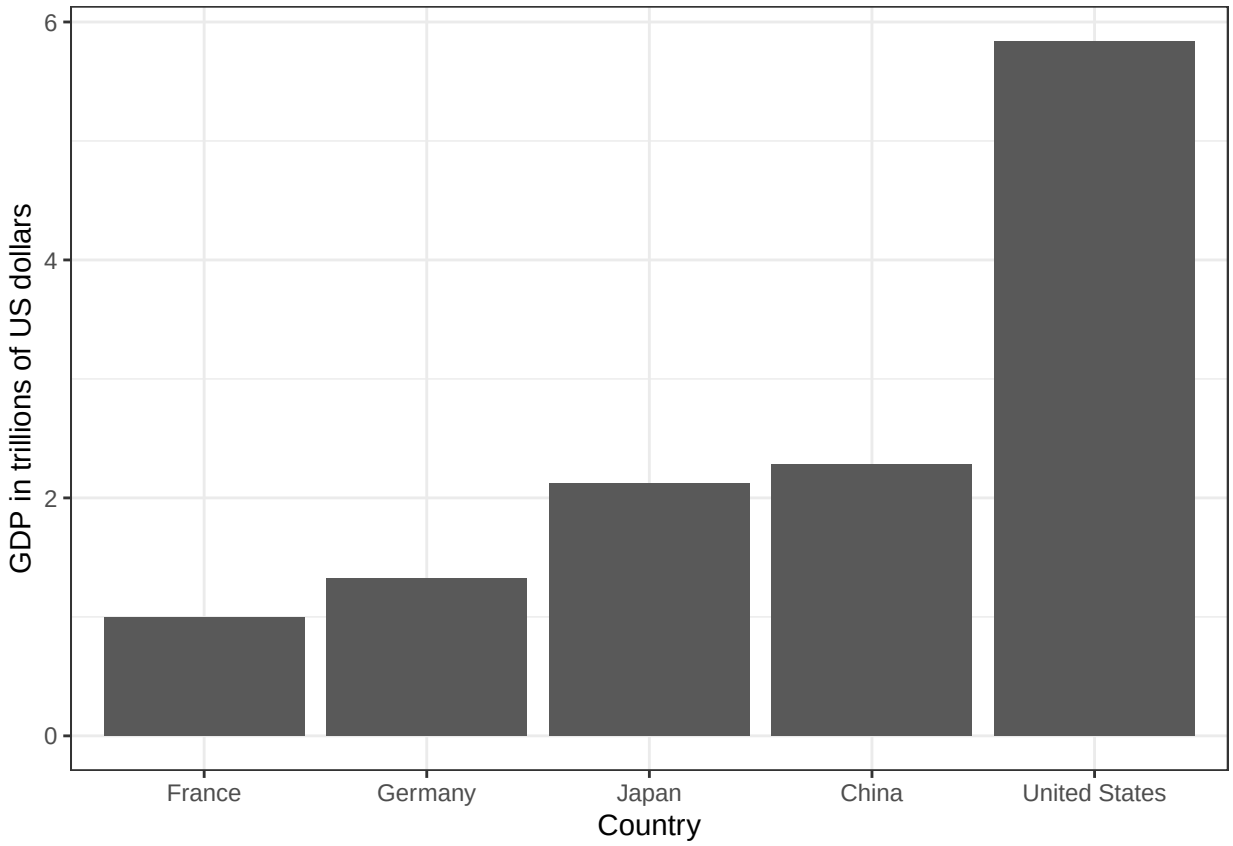
Do not distort quantities

During President Barack Obama's 2011 State of the Union Address the following chart was used to compare the US GDP to the GDP of four competing nations:

Note judging by the area of the circles the US appears to have an economy over five times larger than China and over 30 times larger than France. However, when looking at the actual numbers one sees that this is not the case. The actual ratios are 2.6 and 5.8 times bigger than China and France respectively. The reason for this distortion is that the radius, rather than the area, was made to be proportional to the quantity which implies that the proportion between the areas is squared: 2.6 turns into 6.5 and 5.8 turns into 34.1. Proportional to the radius compared to proportional to area:



Not surprisingly, ggplot defaults to using area rather than radius. Of course, in this case, we really should not be using area at all since we can use position and length:



Order by a meaningful value

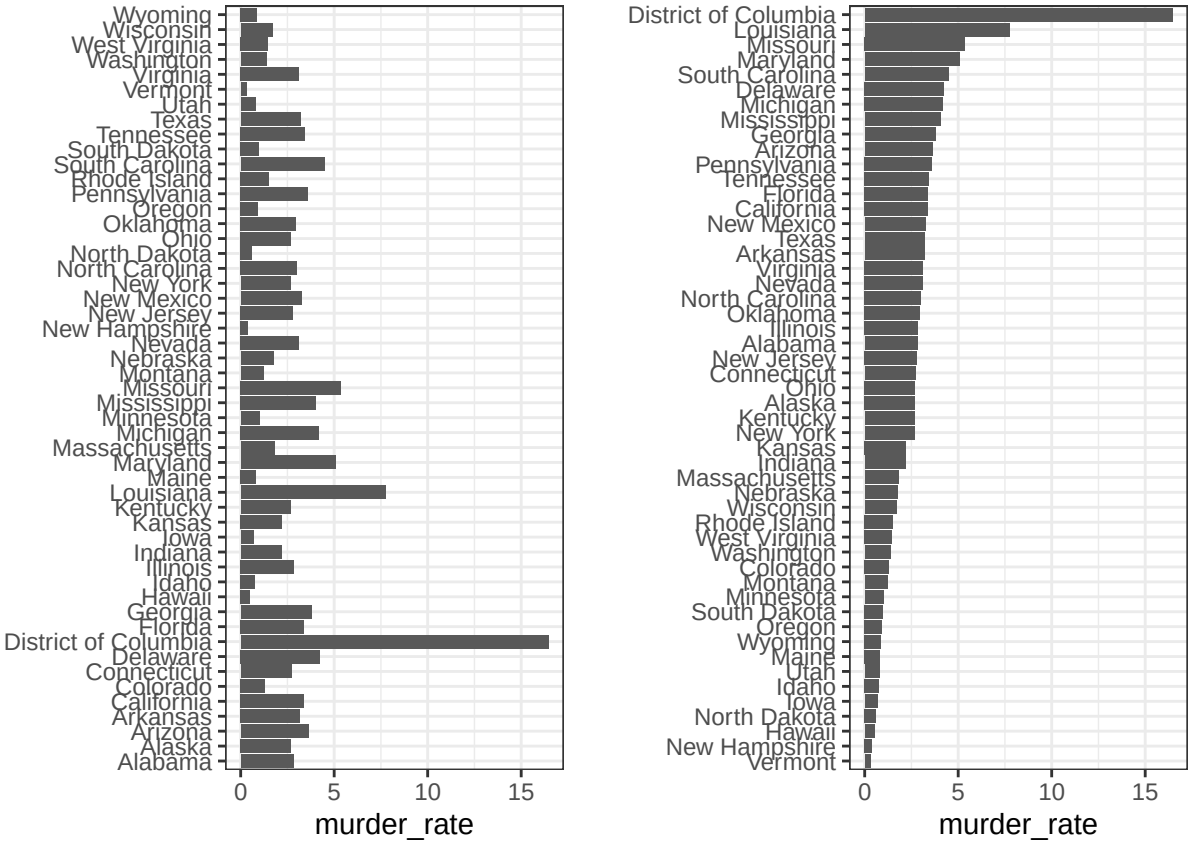
When one of the axes is used to show categories, as is done in barplots, the default ggplot behavior is to order the categories alphabetically when they are defined by character strings. If they are defined by factors, they are ordered by the factor levels. We rarely want to use alphabetical order. Instead we should order by a meaningful quantity.

In all the cases above, the barplots where ordered by the values being displayed. The exception was the graph showing barplots comparing browsers. In this case we kept the order the same across the barplots to ease the comparison. We ordered by the average value of 2000 and 2015. We previously learned how to use the `reorder` function, which helps achieve this goal.

To appreciate how the right order can help convey a message, suppose we want to create a plot to compare the murder rate across states. We are particularly interested in the most dangerous and safest states. Note the difference when we order alphabetically (the default) versus when we order by the actual rate:

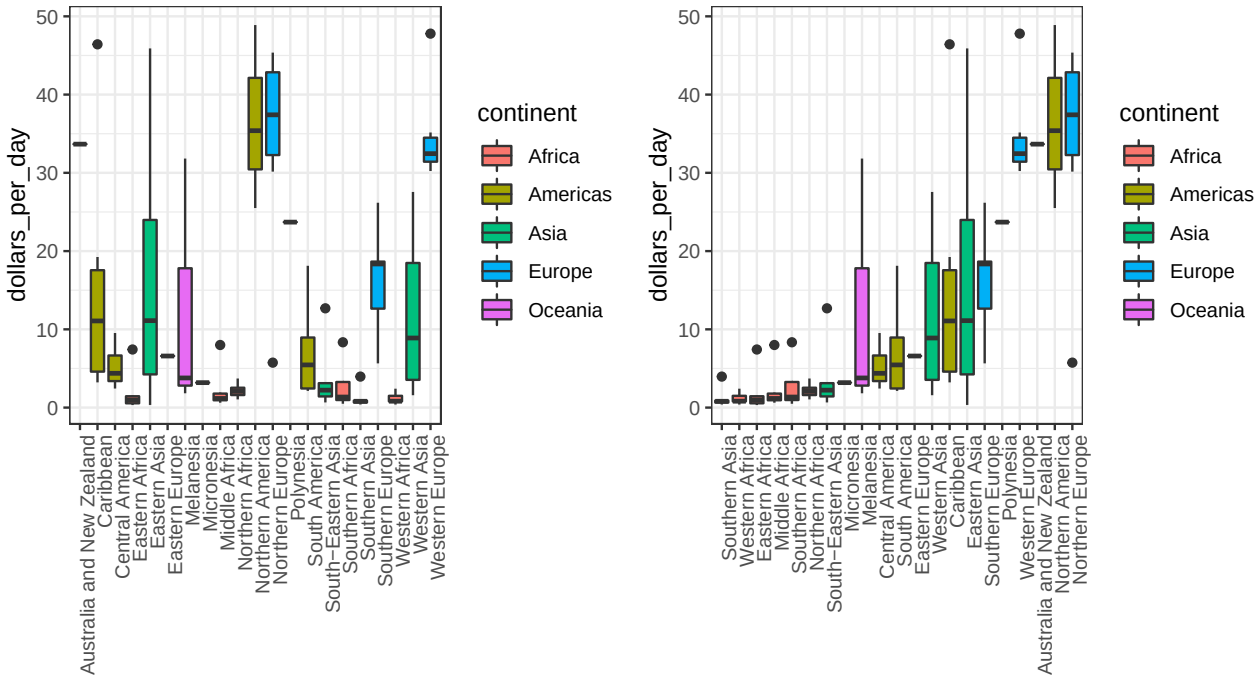
```
data(murders)
p1 <- murders %>% mutate(murder_rate = total / population * 100000) %>%
  ggplot(aes(state, murder_rate)) +
  geom_bar(stat="identity") +
  coord_flip() +
  xlab("")

p2 <- murders %>% mutate(murder_rate = total / population * 100000) %>%
  mutate(state = reorder(state, murder_rate)) %>%
  ggplot(aes(state, murder_rate)) +
  geom_bar(stat="identity") +
  coord_flip() +
```

```
  xlab("")
grid.arrange(p1, p2, ncol = 2)
```



Note that the `reorder` function lets us reorder groups as well.

Below is an example we saw earlier with and without reorder. The first orders the regions alphabetically while the second orders them by the group's median.

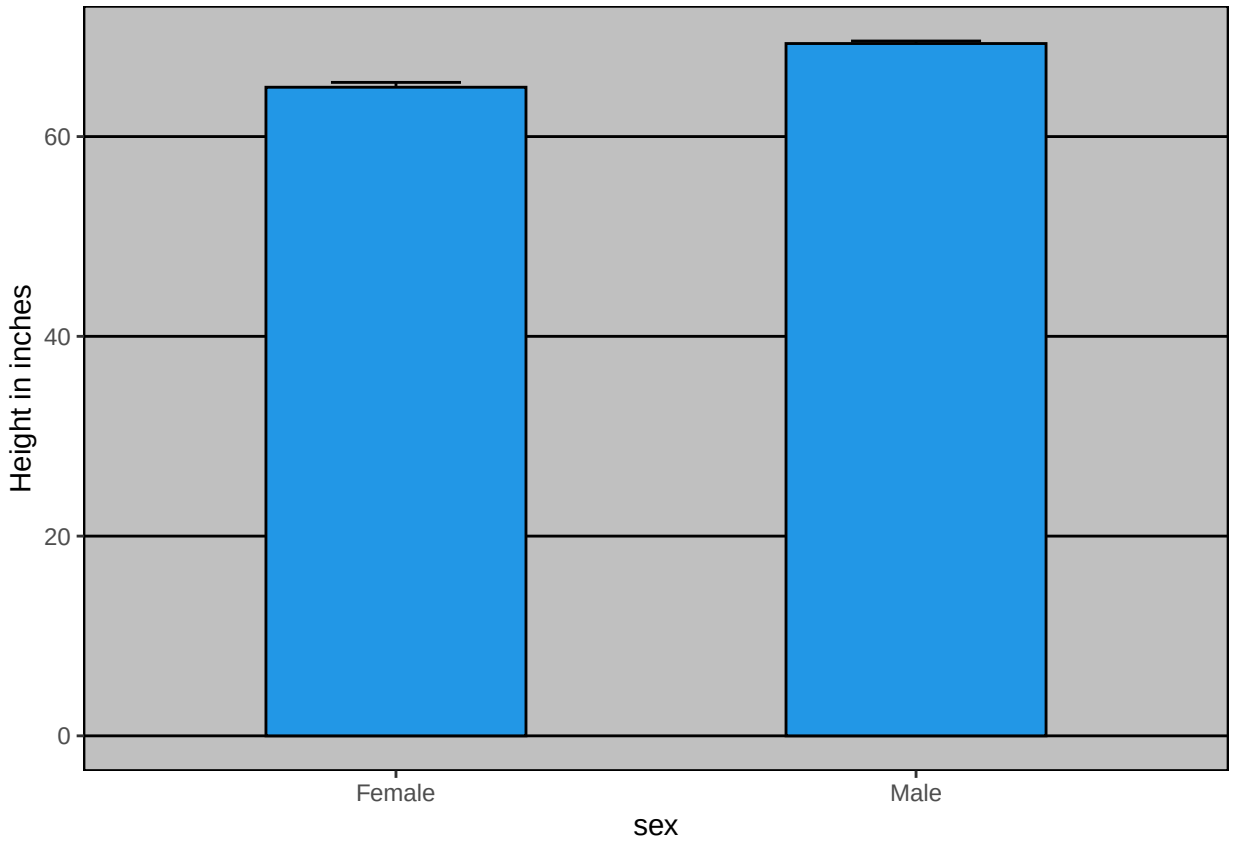


Show the data

We have focused on displaying single quantities across categories. We now shift our attention to displaying data, with a focus on comparing groups.

To motivate our first principle, we'll use our heights data. A commonly seen plot used for comparisons between groups, popularized by software such as Microsoft Excel, shows the average and standard errors (standard errors are defined in a later lecture, but don't confuse them with the standard deviation of the data).

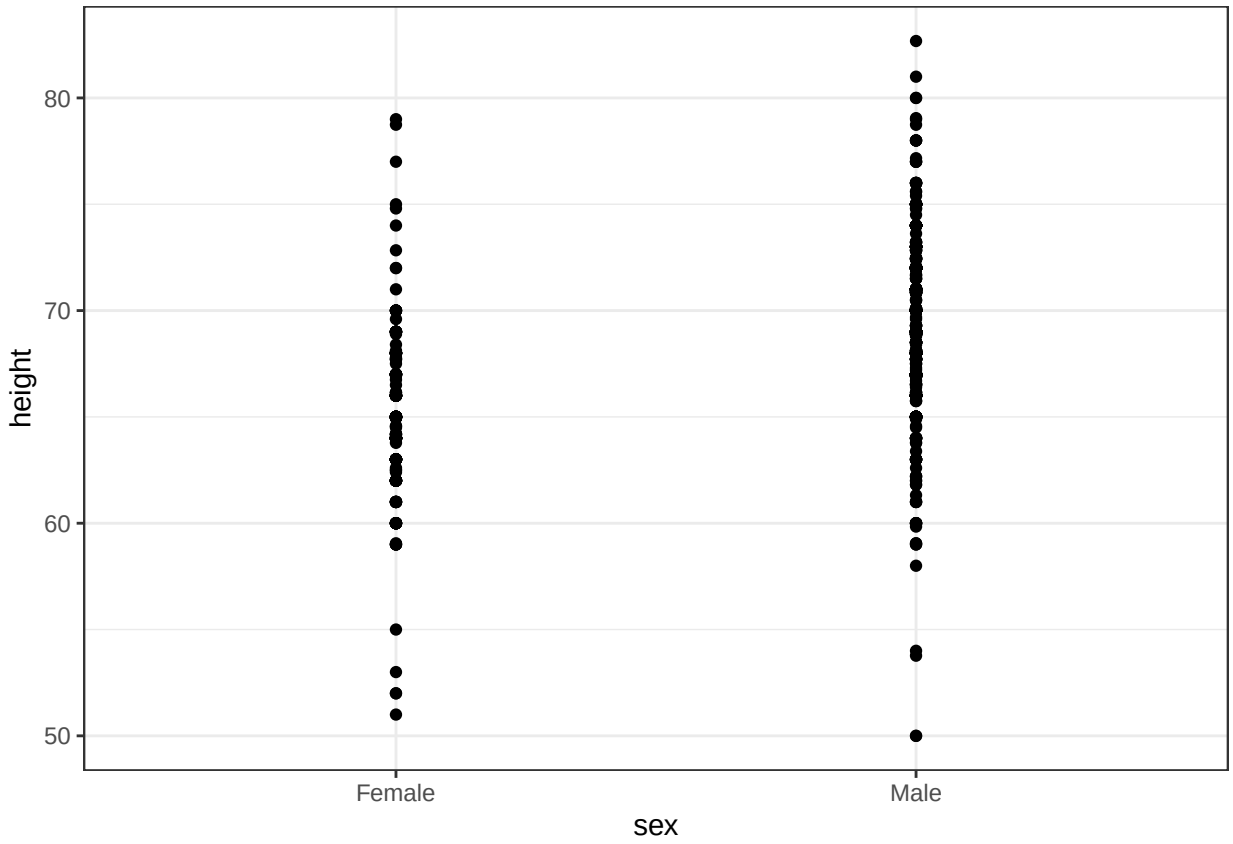
The plot looks like this:



The average of each group is represented by the top of each bar and the antennae expand to the average plus two standard errors. If all someone receives is this plot they will have little information on what to expect if they meet a group of human males and females. The bars go to 0, does this mean there are tiny humans measuring less than one foot? Are all males taller than the tallest females? Is there a range of heights? Someone can't answer these questions since we have provided almost no information on the height distribution.

This brings us to our first principle: **show the data**. This simple ggplot code already generates a more informative plot than the barplot by simply showing all the data points:

```
heights %>% ggplot(aes(sex, height)) + geom_point()
```



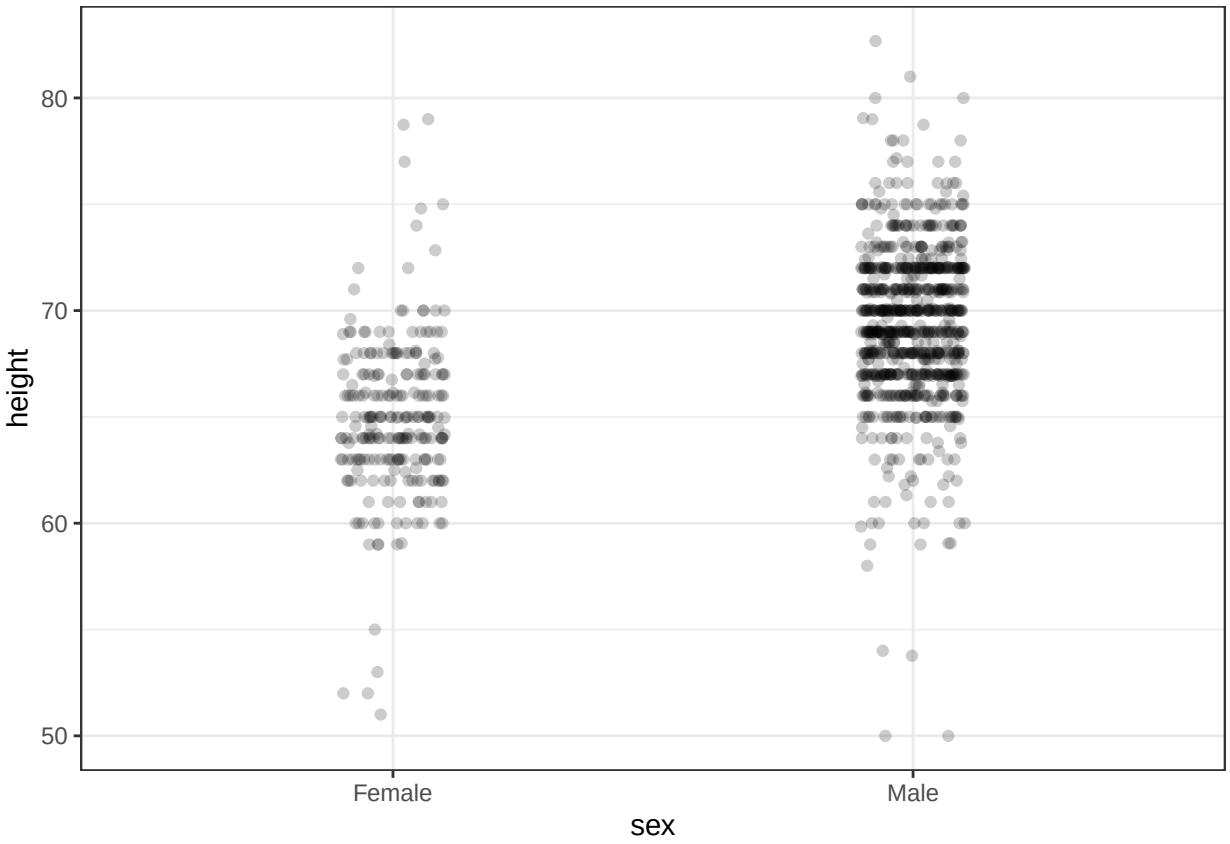
For example, we get an idea of the range of the data. However, this plot has limitations as well since we can't really see all the 238 and 812 points plotted for females and males respectively, and many points are plotted on top of each other. As we have described, visualizing the distribution is much more informative. But before doing this, we point out two ways we can improve a plot showing all the points.

The first is to add **jitter**: adding a small random shift to each point. In this case, adding horizontal jitter does not alter the interpretation, since the height of the points do not change, but we minimize the number of points that fall on top of each other and therefore get a better sense of how the data is distributed.

A second improvement comes from using **alpha blending**: making the points somewhat transparent. The more points fall on top of each other, the darker the plot which also helps us get a sense of how the points are distributed.

Here is the same plot with jitter and alpha blending:

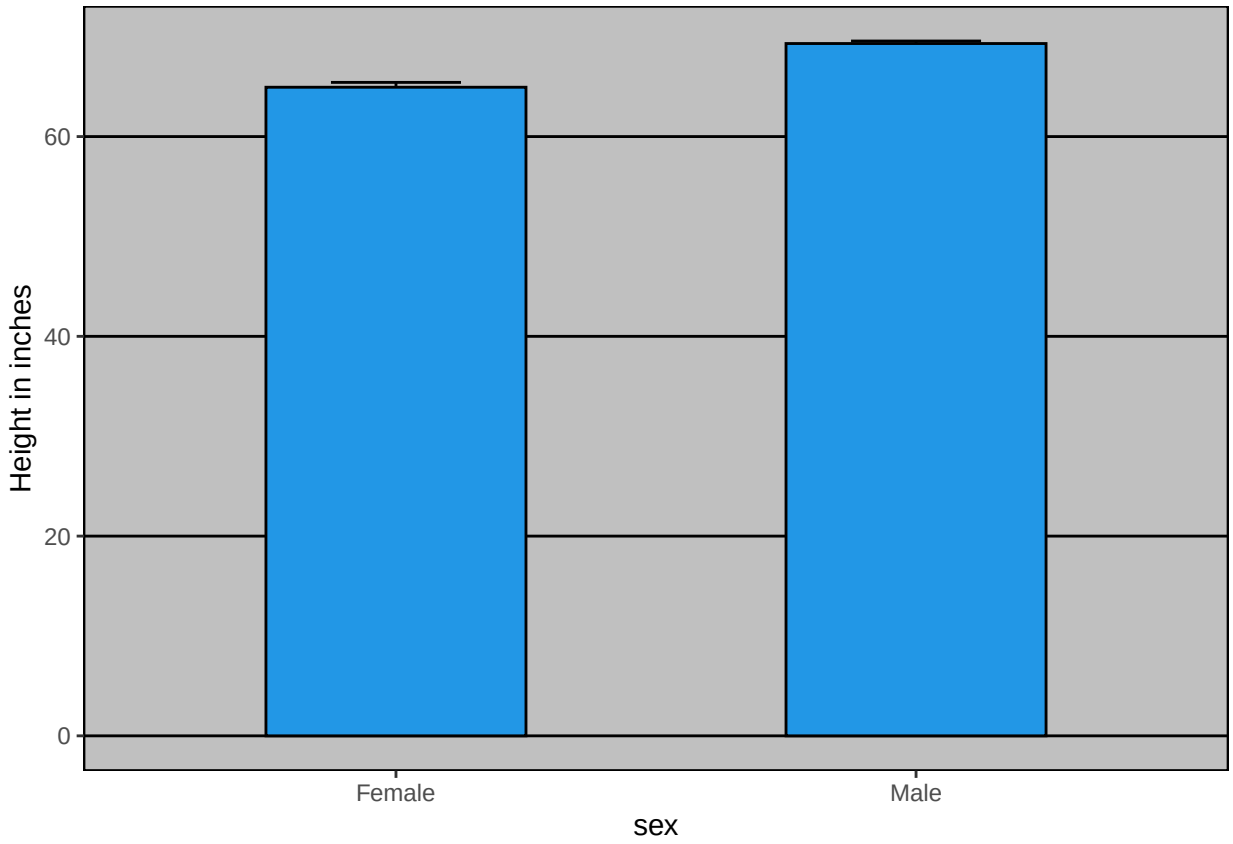
```
heights %>% ggplot(aes(sex, height)) +
  geom_jitter(width = 0.1, alpha = 0.2)
```



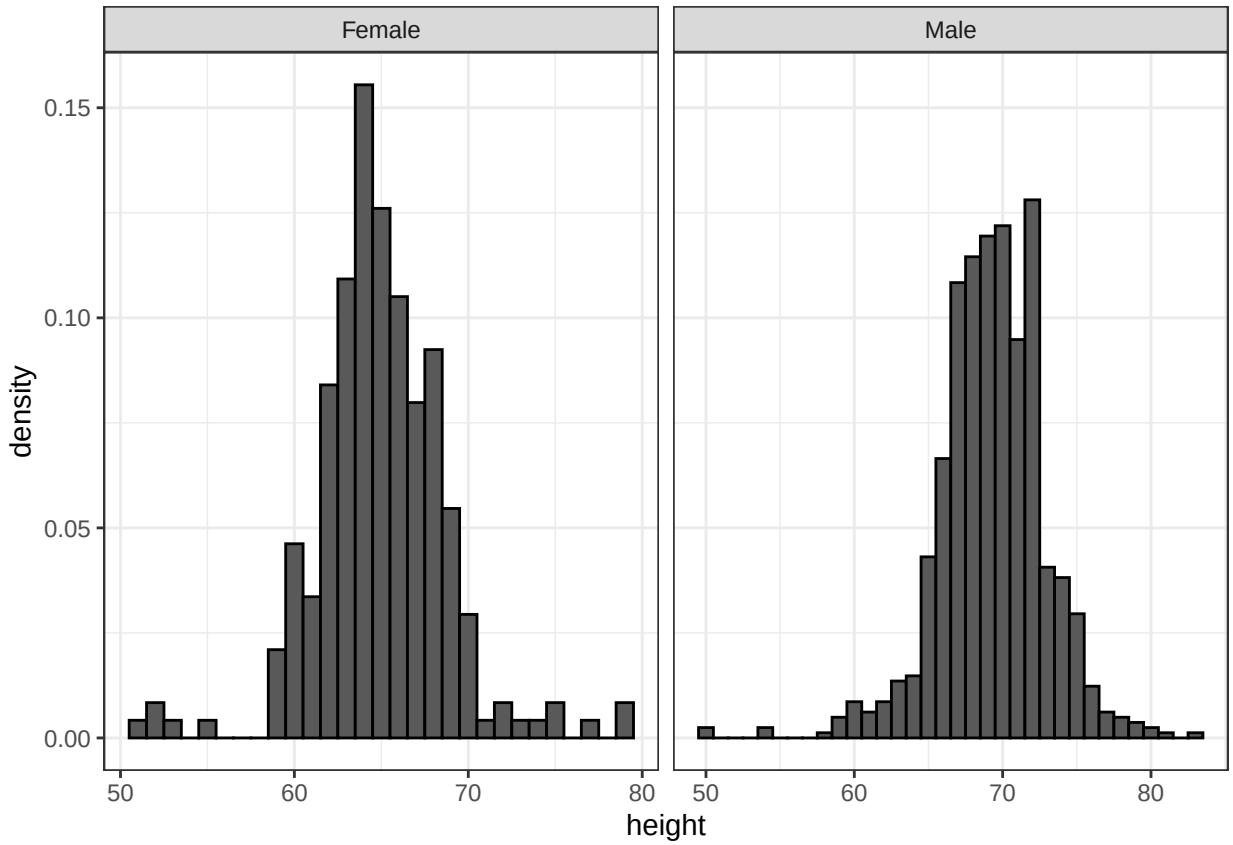
Now we start getting a sense that, on average, males are taller than females. We also note dark horizontal lines demonstrating that many reported values are rounded to the nearest integer. Since there are so many points it is more effective to show distributions, rather than show individual points. In our next example we show the improvements provided by distributions and suggest further principles.

Ease comparisons: Use common axes

Earlier we saw this plot used to compare male and female heights:

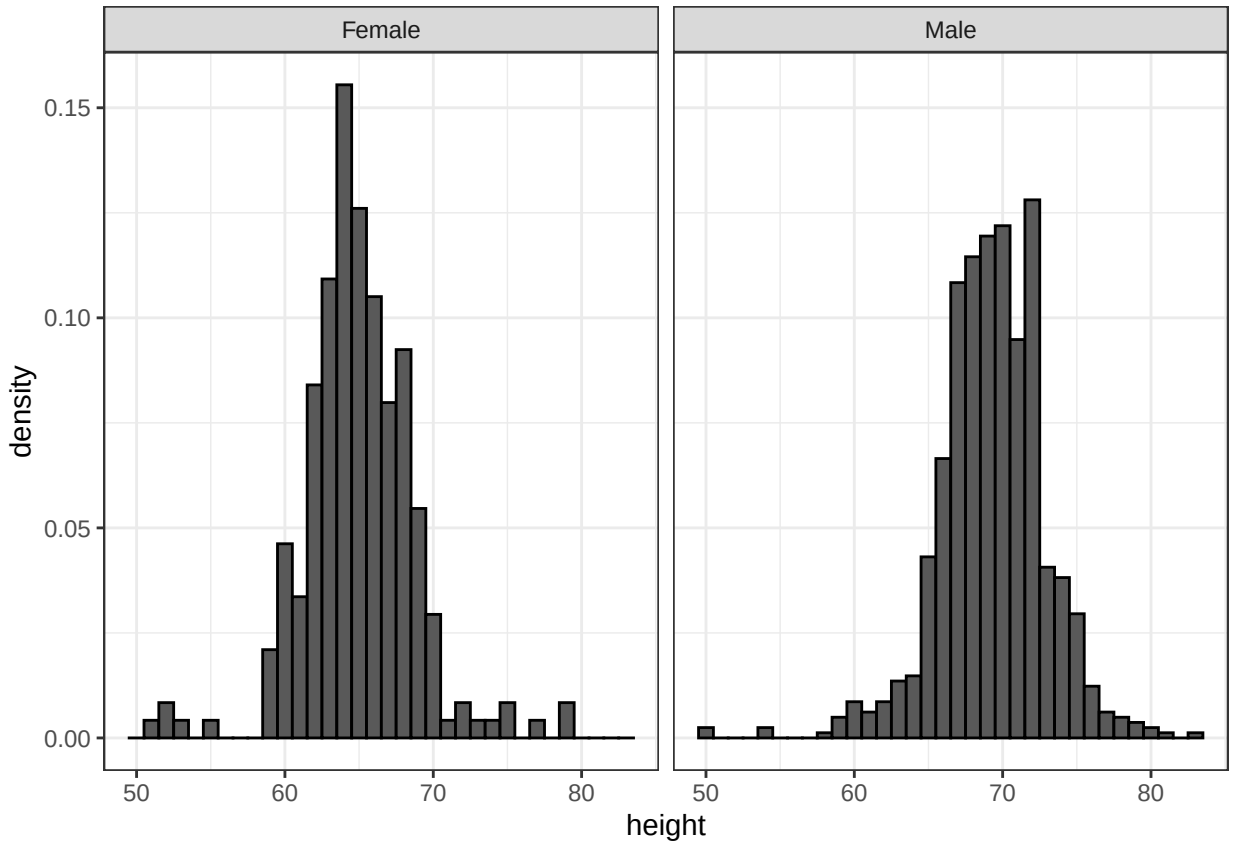


Since there are so many points it is more effective to show distributions, rather than show individual points. We therefore show histograms for each group:

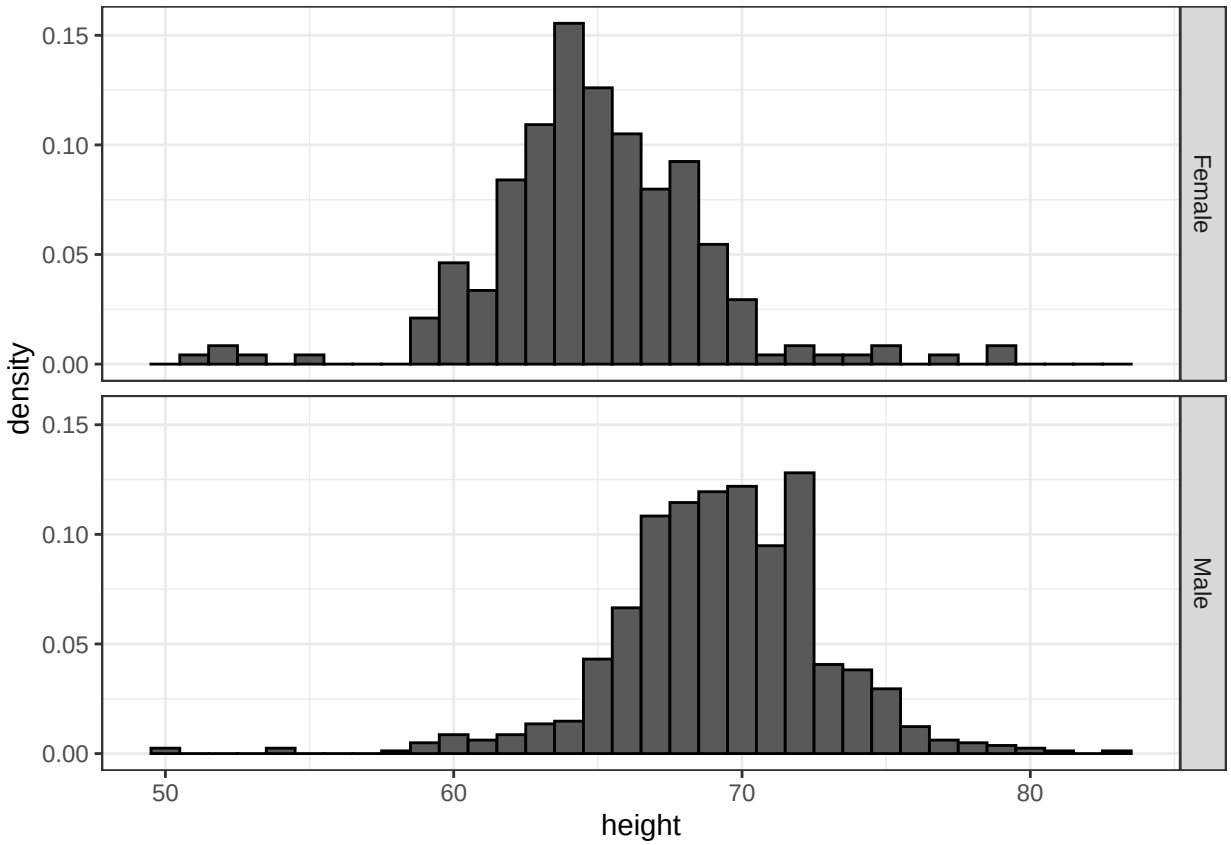


However, from this plot it is not immediately obvious that males are, on average, taller than females. We have to look carefully to notice that the x-axis has a higher range of values in the male histogram. An important principle here is to **keep the axes the same** when comparing data across to plots.

Note how the comparison becomes easier:

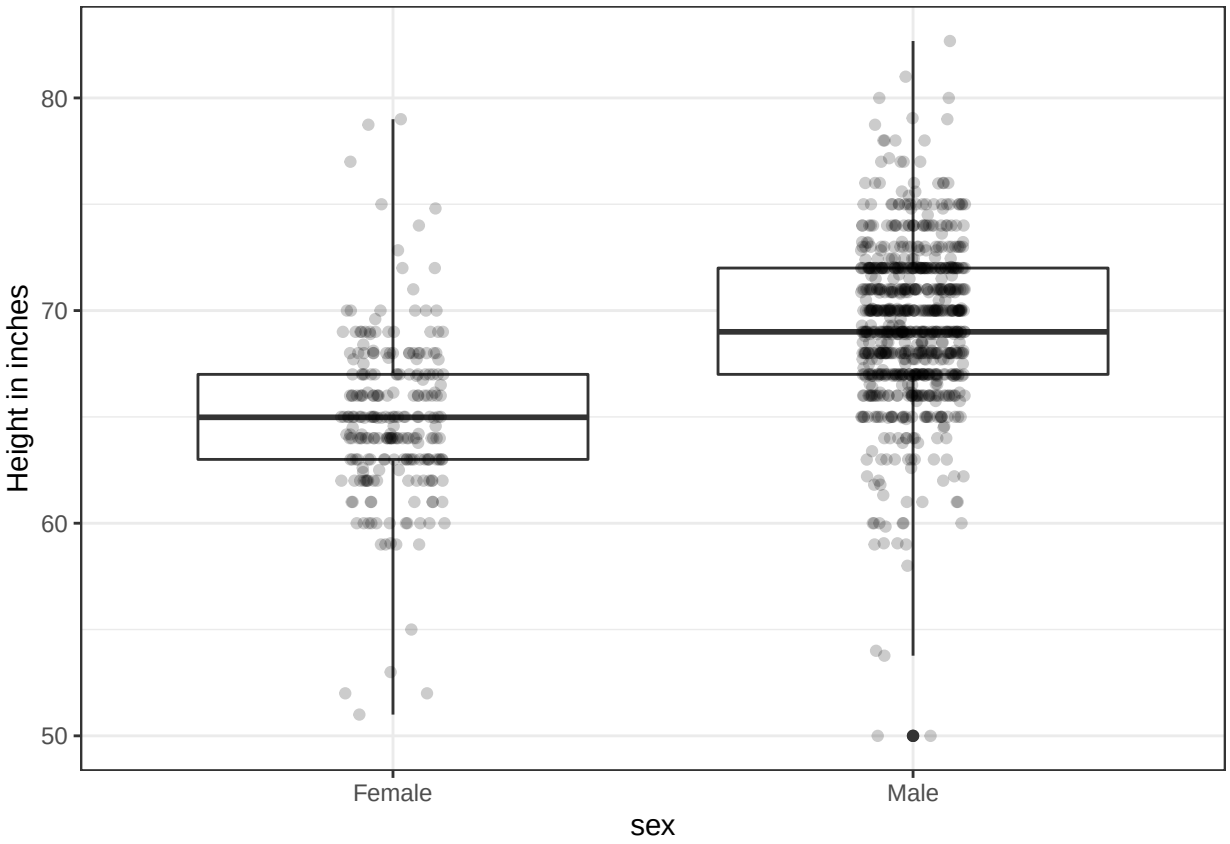


Align plots vertically to see horizontal changes and horizontally to see vertical changes. In these histograms, the visual cue related to decreases or increases in height are shifts to the left or right respectively: horizontal changes. Aligning the plots vertically helps us see this change when the axis are fixed:

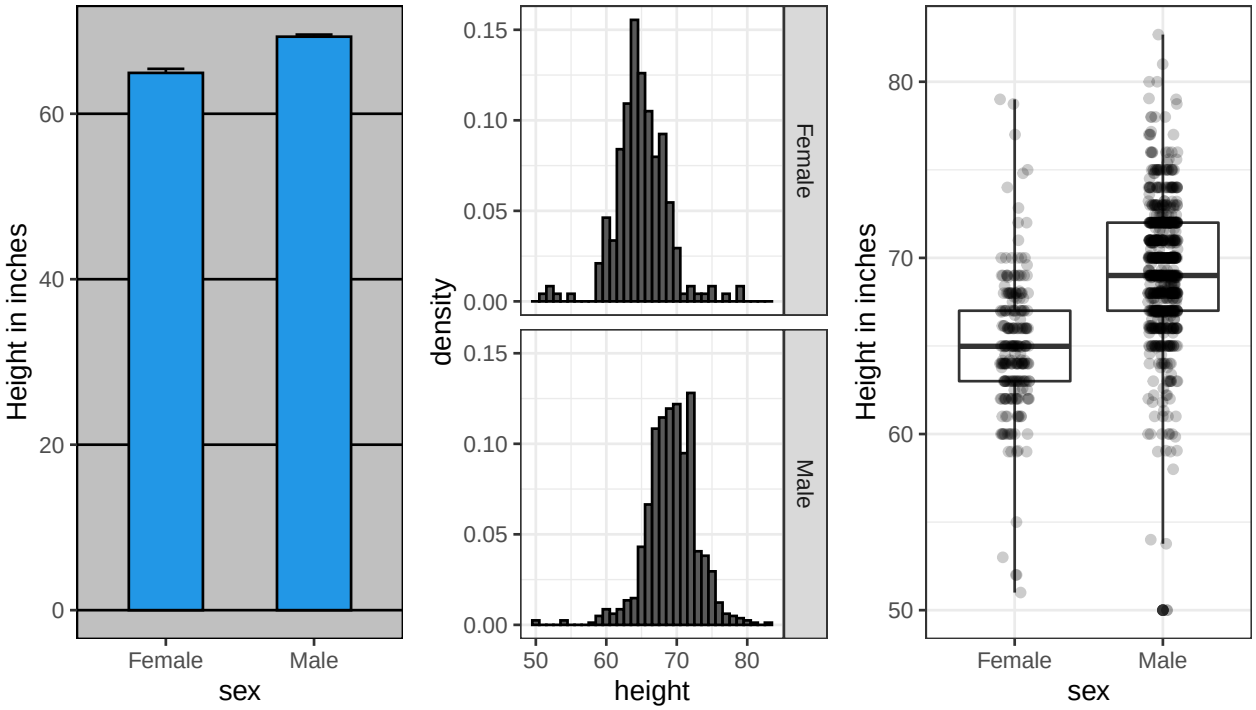


This plot makes it much easier to notice that men are, on average, taller. If instead of histograms we want the more compact summary provided by boxplots, then we align them horizontally, since, by default, boxplots move up and down with changes in height.

Following our *show the data* principle we overlay all the data points:



Now contrast and compare these three plots, based on exactly the same data:

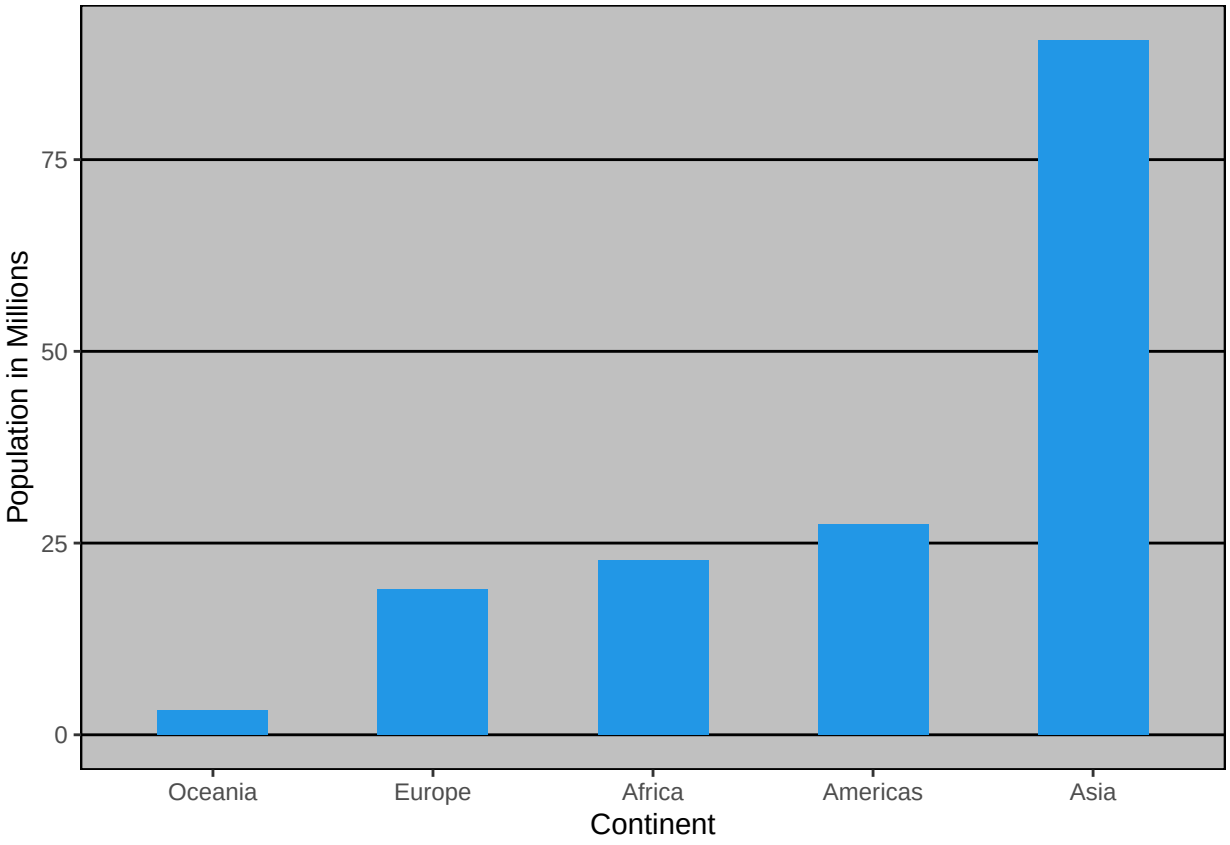


Note how much more we learn from the two plots on the right. Barplots are useful for showing one number, but not very useful when wanting to describe distributions.

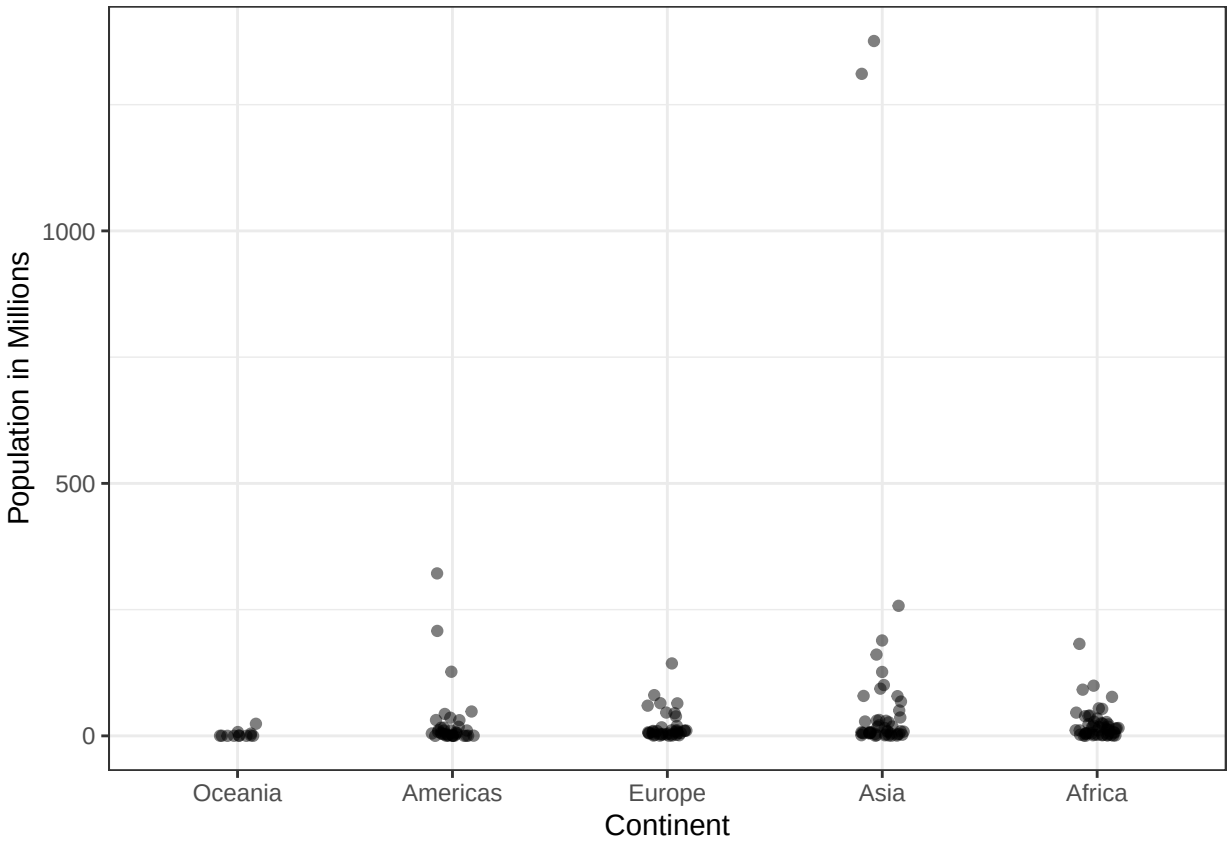
Consider transformations

We have motivated the use the log transformation in cases where the changes are multiplicative. Population size was an example in which we found a log transformation to yield a more informative transformation. The combination of incorrectly using barplots when a log transformation is merited can be particularly distorting.

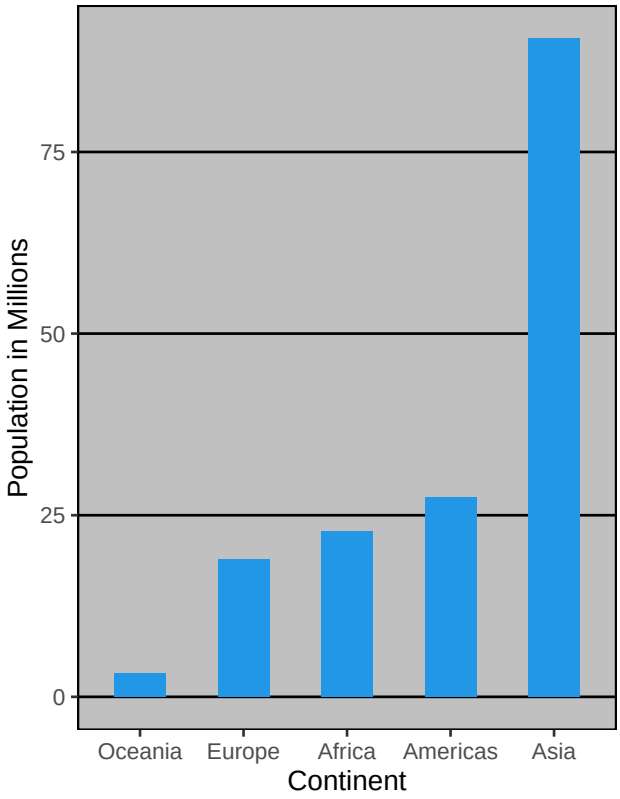
As an example, consider this barplot showing the average population sizes for each continent in 2015:

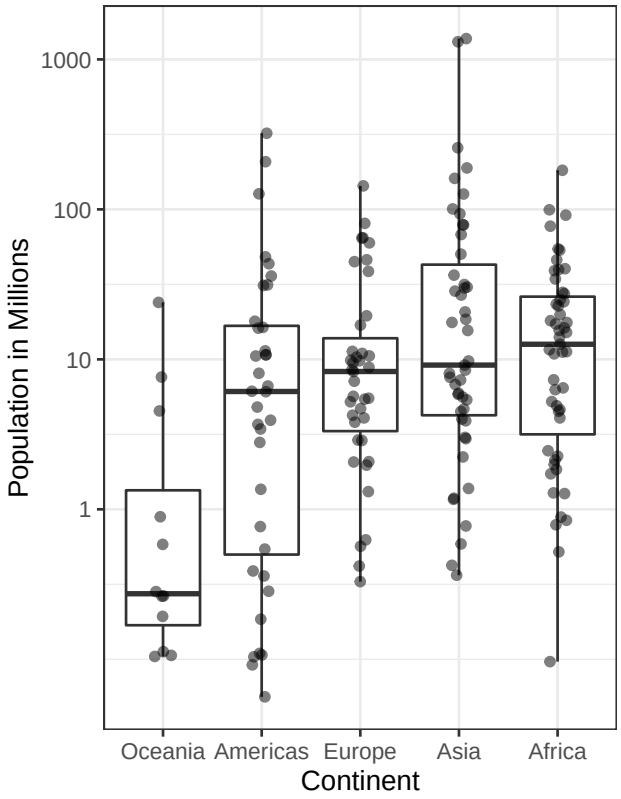


From this plot one would conclude that countries in Asia are much more populous than other continents. Following the *show the data* principle we quickly notice that this is due to two very large countries, which we assume are India and China:



Here, using a log transformation provides a much more informative plot. We compare the original barplot to a boxplot using the log scale transformation for the y-axis:



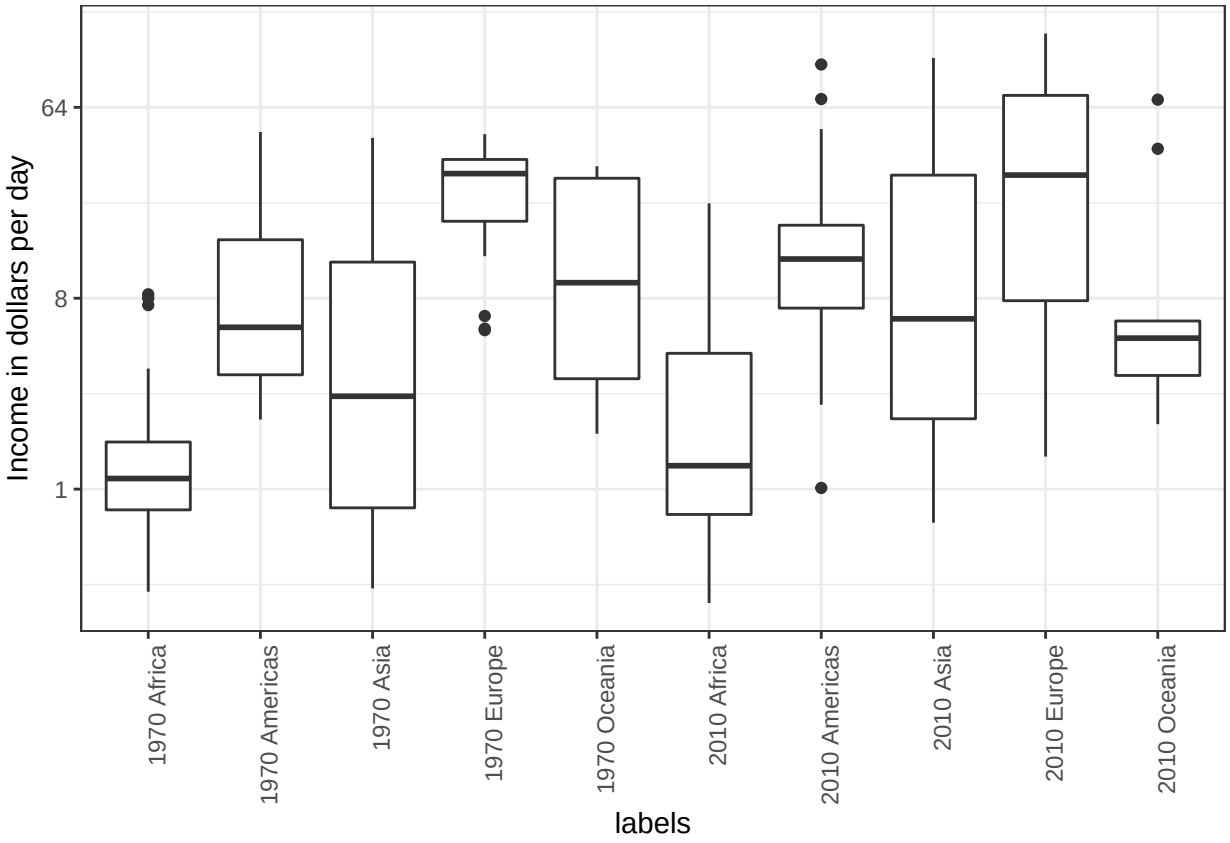


Note in particular that with the new plot we realize that countries in Africa actually have a larger median population size than those in Asia.

Other transformations you should consider are the logistic transformation, useful to better see fold changes in odds, and the square root transformation, useful for count data.

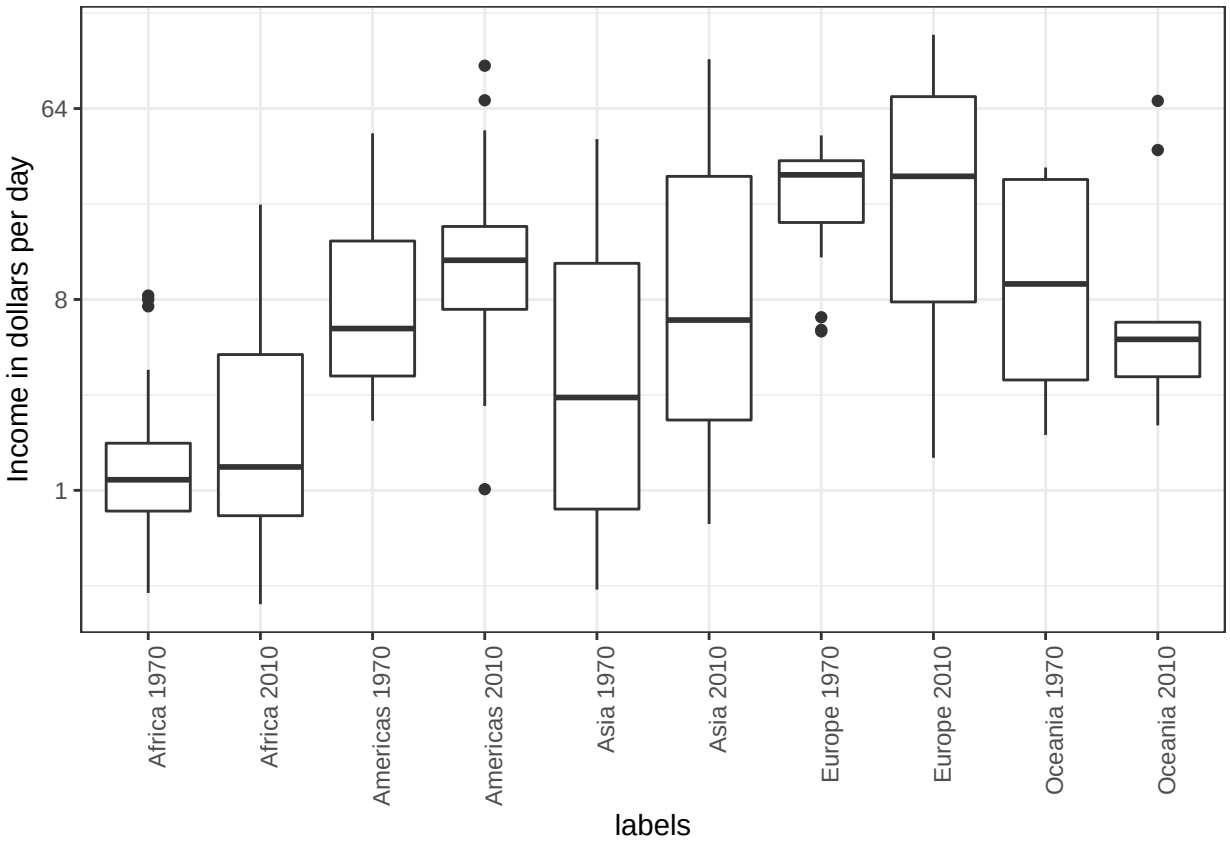
Adjacent visual cues

When comparing income data between 1970 and 2010 across region we made a figure similar to the one below. A difference is that here we look at continents instead of regions, but this is not relevant to the point we are making.



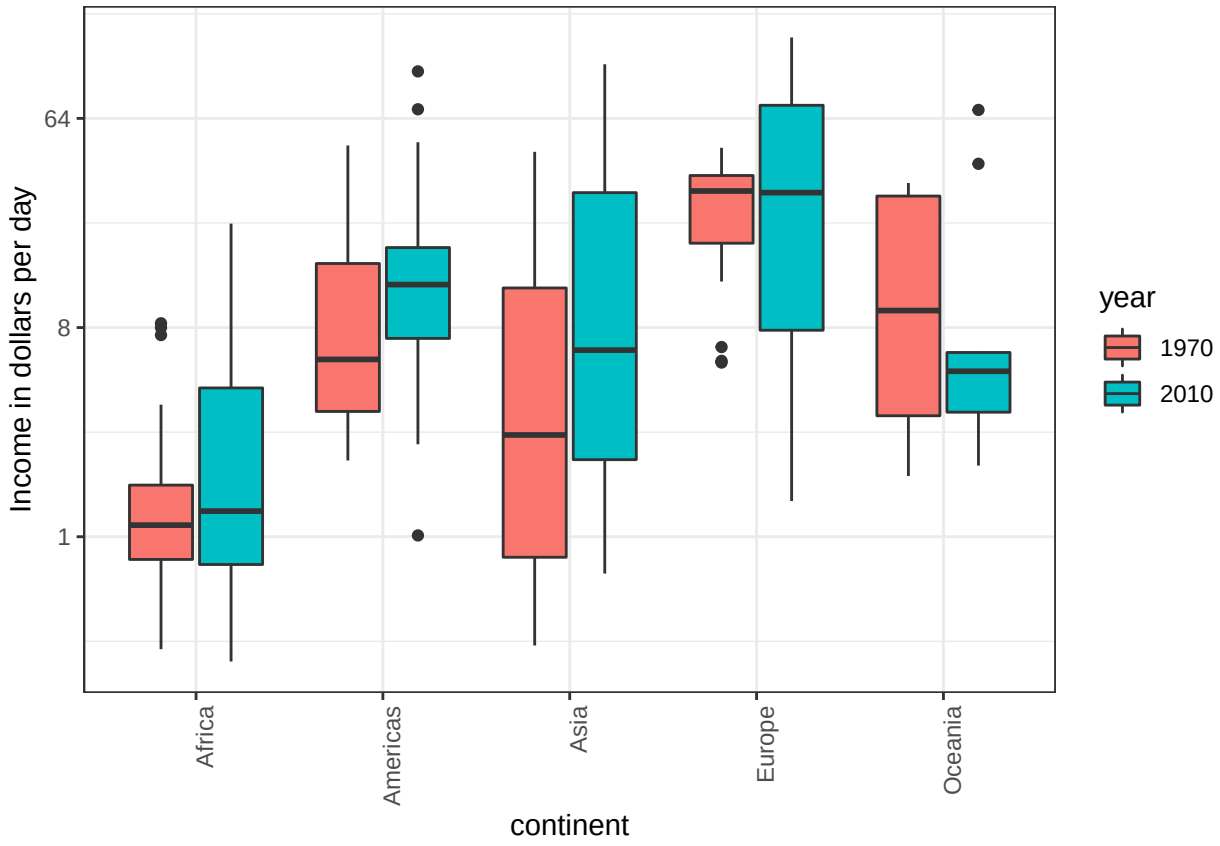
Note that, for each continent, we want to compare the distributions from 1970 to 2010. The default in ggplot is to order alphabetically so the labels with 1970 come before the labels with 2010, making the comparisons challenging.

Comparison is even easier when boxplots are next to each other:



Ease comparison: use color

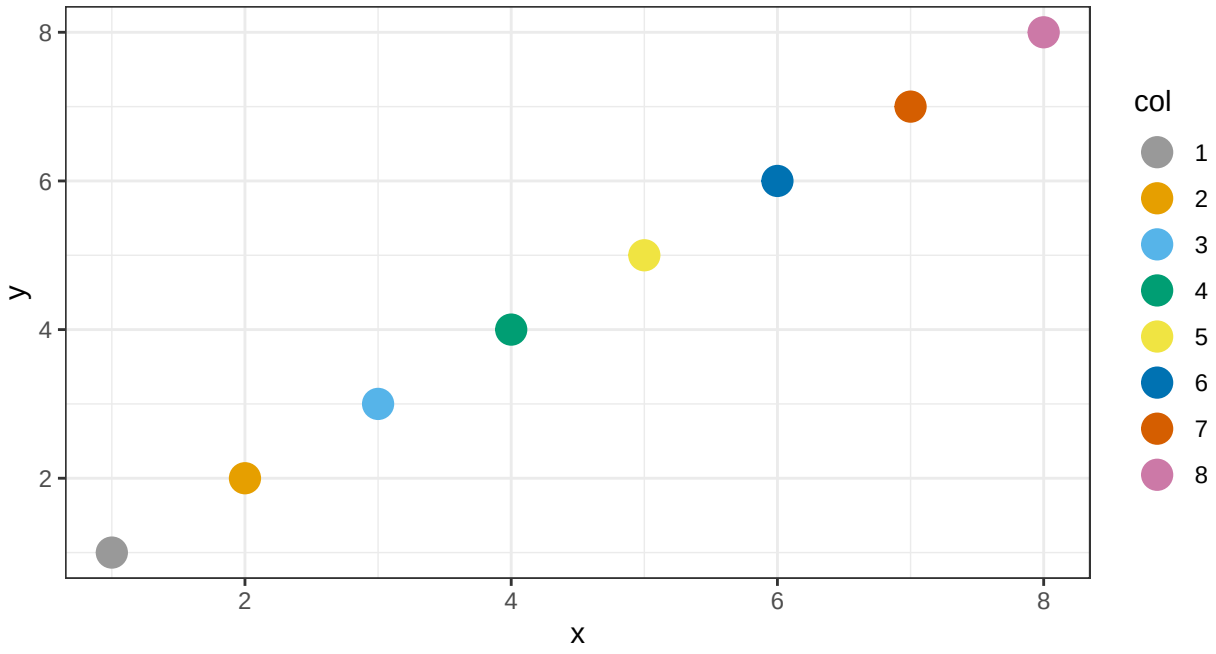
Comparison is easier when color is used to denote the two things compared.



Think of the color blind

About 10% of the population is color blind. Unfortunately, the default colors used in ggplot are not optimal for this group. However, ggplot does make it easy to change the color palette used in plots. Here is an example of how we can use a color blind friendly pallet. There are several resources that help you select colors, for example this one.

```
color_blind_friendly_cols <- c("#999999", "#E69F00", "#56B4E9",
                               "#009E73", "#F0E442", "#0072B2",
                               "#D55E00", "#CC79A7")
p1 <- data.frame(x=1:8, y=1:8, col = as.character(1:8)) %>%
  ggplot(aes(x, y, color = col)) + geom_point(size=5)
p1 + scale_color_manual(values=color_blind_friendly_cols)
```



Scatter plots

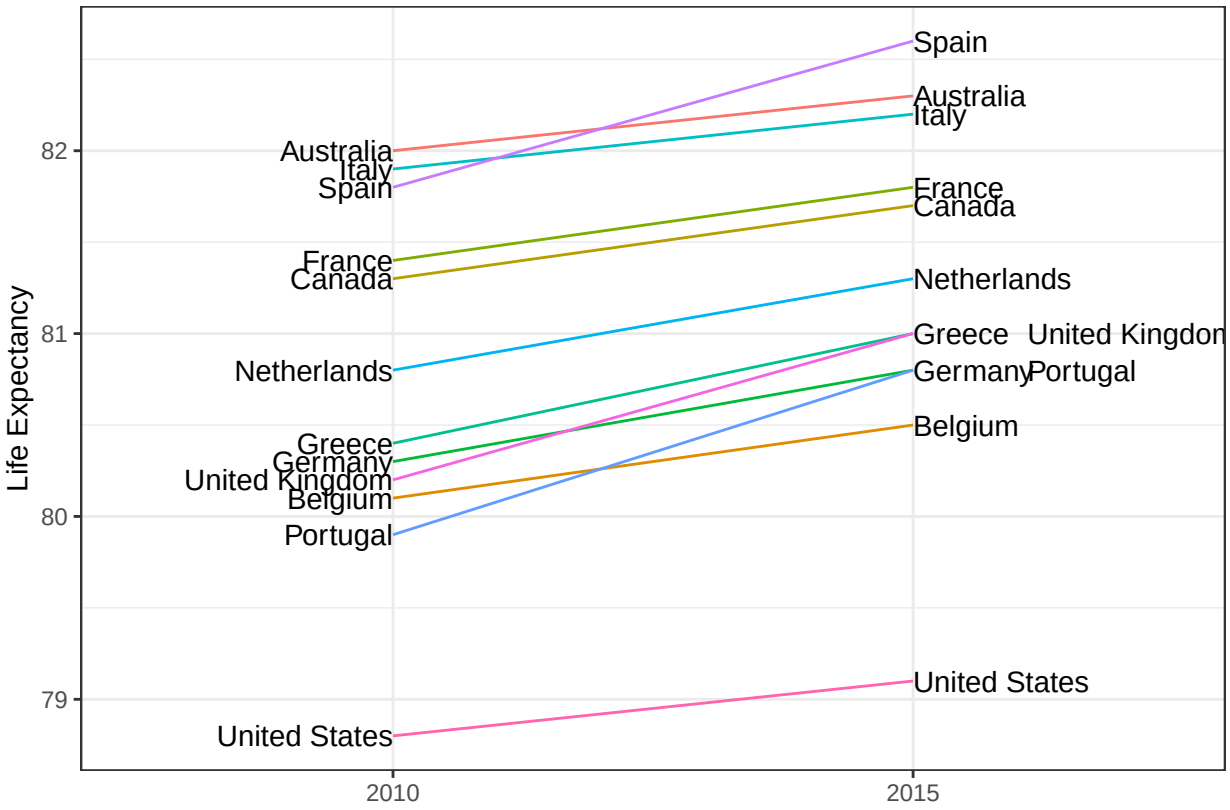
Use scatter plots to examine the relationship between two continuous variables. In every single instance in which we have examined the relationship between two continuous variables, total murders versus population size, life expectancy versus fertility rates, and child mortality versus income, we have used scatter plots. This is the plot we generally recommend.

Slope charts

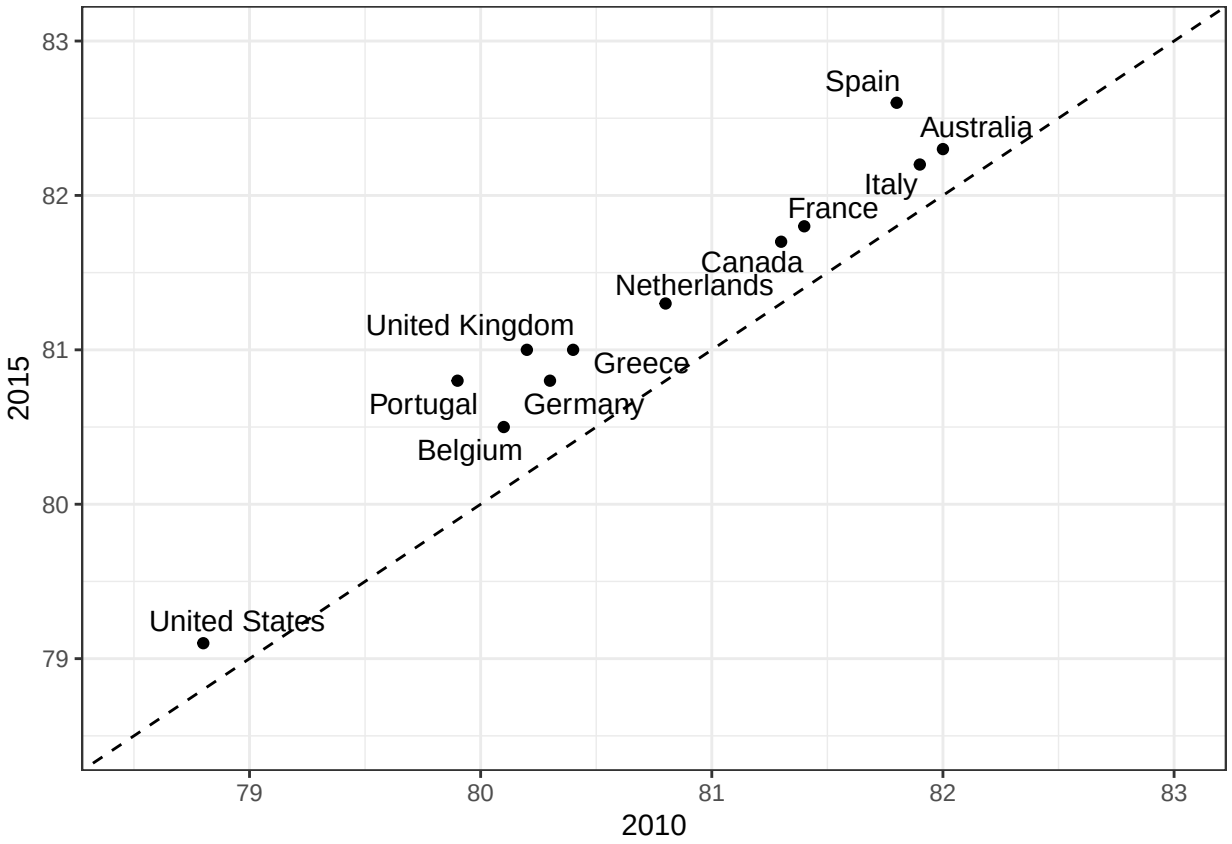
One exception where another type of plot may be more informative is when you are comparing variables of the same type but at different time points and for a relatively small number of comparisons. For example, comparing life expectancy between 2010 and 2015. In this case we might recommend a **slope chart**. There is not a geometry for slope chart in ggplot2 but we can construct one using `geom_lines`. We need to do some tinkering to add labels.

Here is a comparison for large western countries:

```
west <- c("Western Europe","Northern Europe","Southern Europe",
          "Northern America","Australia and New Zealand")
dat <- gapminder %>%
  filter(year%in% c(2010, 2015) & region %in% west &
           !is.na(life_expectancy) & population > 10^7)
dat %>%
  mutate(location = ifelse(year == 2010, 1, 2),
         location = ifelse(year == 2015 &
         country%in%c("United Kingdom","Portugal"),
         location+0.22, location),
         hjust = ifelse(year == 2010, 1, 0)) %>%
  mutate(year = as.factor(year)) %>%
  ggplot(aes(year, life_expectancy, group = country)) +
  geom_line(aes(color = country), show.legend = FALSE) +
  geom_text(aes(x = location, label = country, hjust = hjust),
            show.legend = FALSE) +
  xlab("") + ylab("Life Expectancy")
```



An advantage of the slope chart is that it permits us to quickly get an idea of changes based on the slope of the lines. Note that we are using **angle** as the visual cue. But we also have position to determine the exact values. Comparing the improvements is a bit harder with a scatter plot:

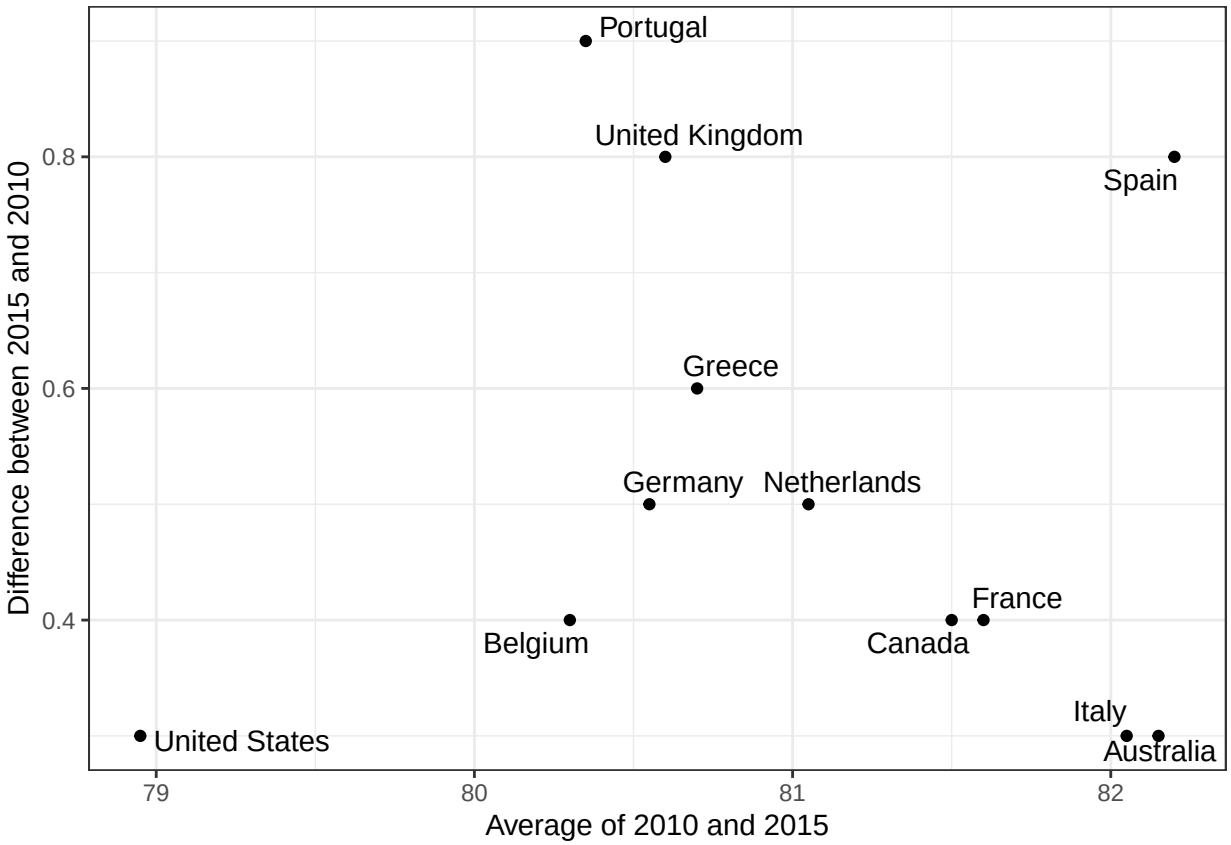


Use common charts

Note that in the scatter plot we have followed the principle *use common axes* since we are comparing these before and after. However, if we have many points the slope charts stop being useful as it becomes hard to see each line.

Bland-Altman plot

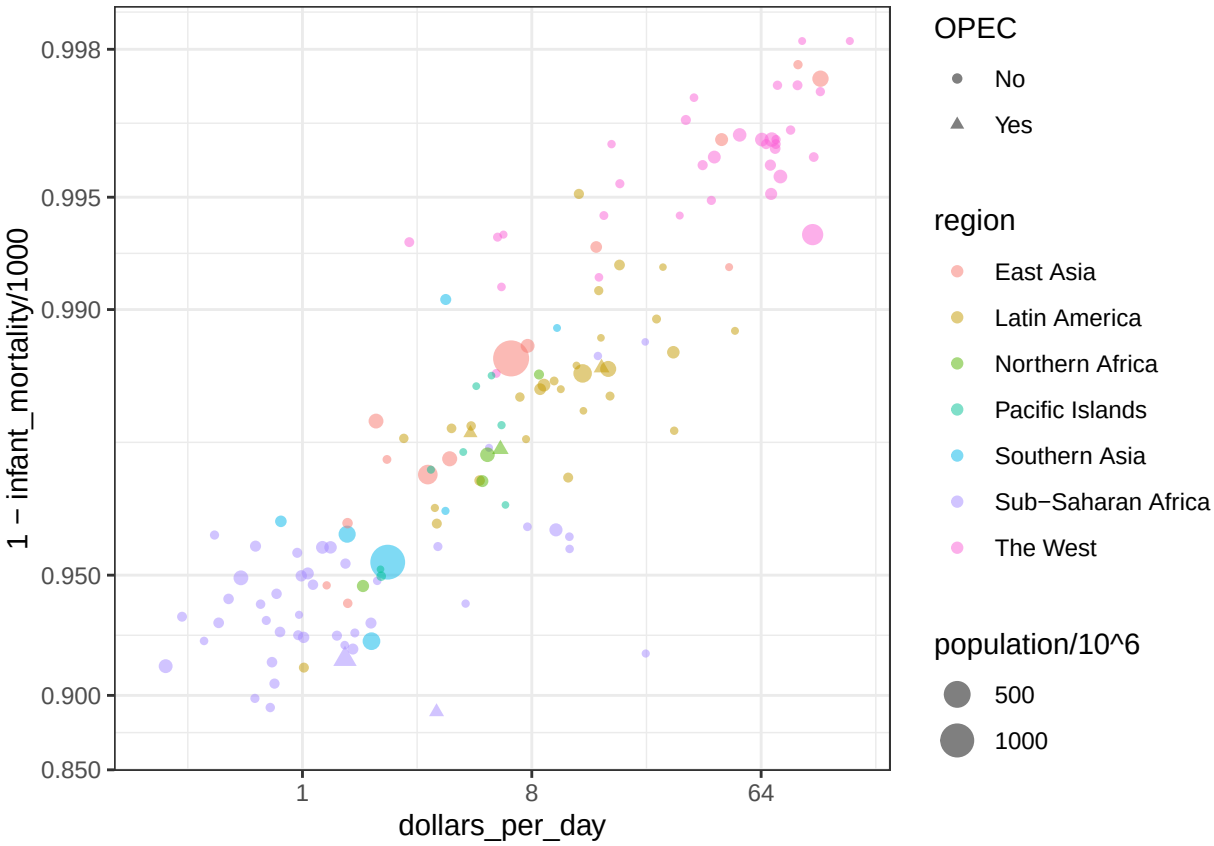
Since we are interested in the difference, it makes sense to dedicate one of our axes to it. The Bland-Altman plot, also known as the Tukey mean-difference plot and the MA-plot, shows the difference versus the average:



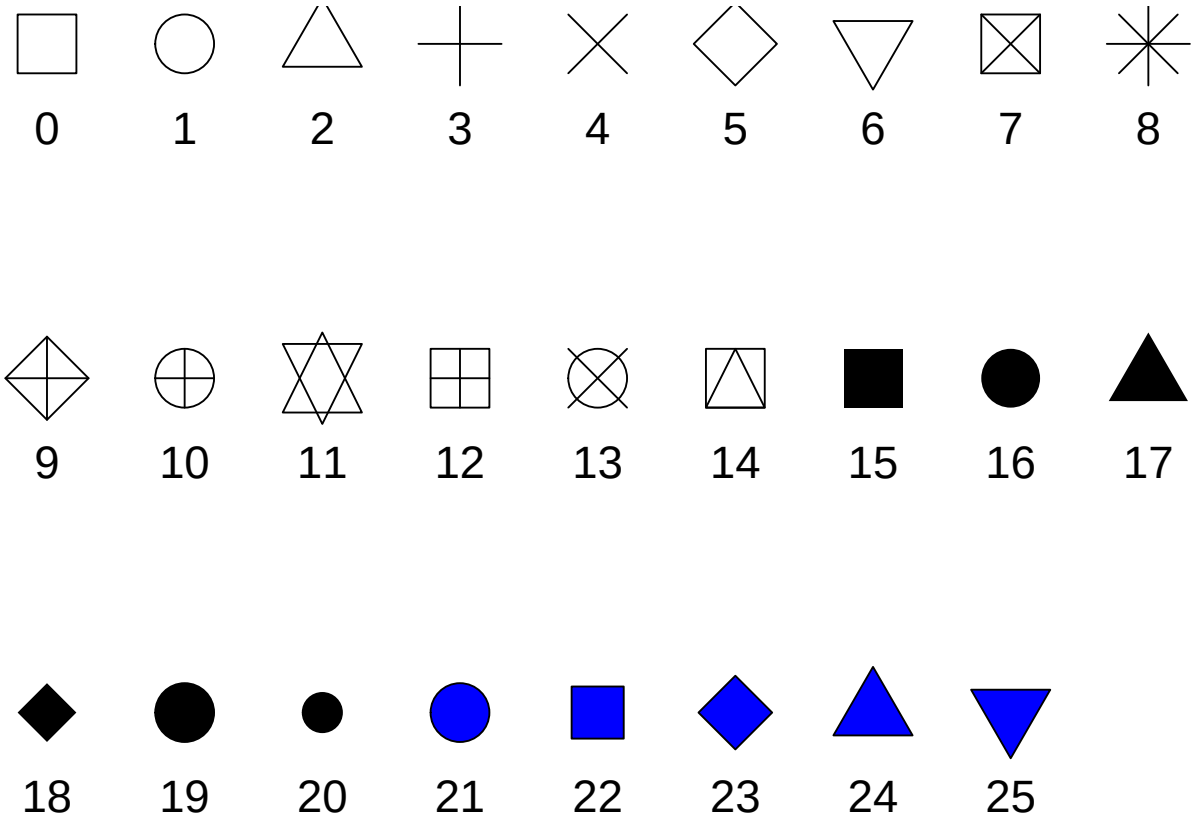
Here we quicky see which countries have improved the most as it is represented by the y-axis. We also get an idea of the overall value from the x-axis.

Encoding a third variable

We previously showed a scatter plot showing the relationship between infant survival and average income. Here is a version of this plot where we encode three variables: OPEC membership, region, and population:

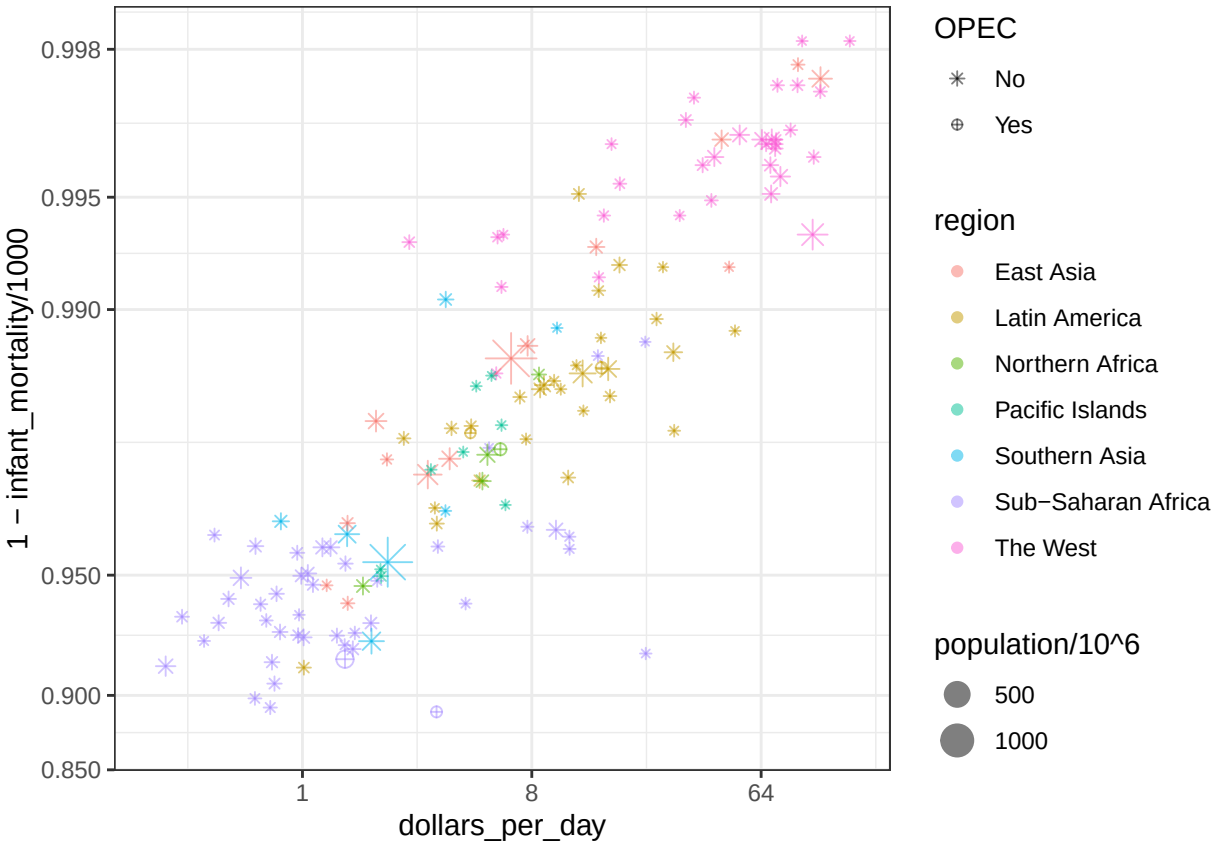


Note that we encode categorical variables with color, hue, and shape. The shape can be controlled with the `shape` argument. Below are the shapes available for use in R. Note that for the last five, the color goes inside.



The default shape values are a circle and a triangle for OPEC membership. We can manually customize these by adding the layer `scale_shape_manual(values = c(8, 10))`, where 8 and 10 are the numbers of the desired shapes from the list above.

```
dat %>%
  ggplot(aes(dollars_per_day, 1 - infant_mortality/1000,
             col = region, size = population/10^6,
             shape =  OPEC)) +
  scale_x_continuous(trans = "log2", limits=c(0.25, 150)) +
  scale_y_continuous(trans = "logit",limit=c(0.875, .9981),
                     breaks=c(.85,.90,.95,.99,.995,.998)) +
  geom_point(alpha = 0.5) +
  scale_shape_manual(values = c(8, 10))
```



For continuous variables we can use color, intensity or size. We now show an example of how we do this with a case study.

Case Study: Vaccines

Vaccines have helped save millions of lives. In the 19th century, before herd immunization was achieved through vaccination programs, deaths from infectious diseases, like smallpox and polio, were common. However, today, despite all the scientific evidence for their importance, vaccination programs have become somewhat controversial.

```
library(dslabs)
data(us_contagious_diseases)
str(us_contagious_diseases)
```

```
## 'data.frame':    16065 obs. of  6 variables:
##  $ disease        : Factor w/ 7 levels "Hepatitis A",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ state          : Factor w/ 51 levels "Alabama","Alaska",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ year           : num  1966 1967 1968 1969 1970 ...
##  $ weeks_reporting: num  50 49 52 49 51 51 45 45 45 46 ...
##  $ count          : num  321 291 314 380 413 378 342 467 244 286 ...
##  $ population     : num  3345787 3364130 3386068 3412450 3444165 ...
```

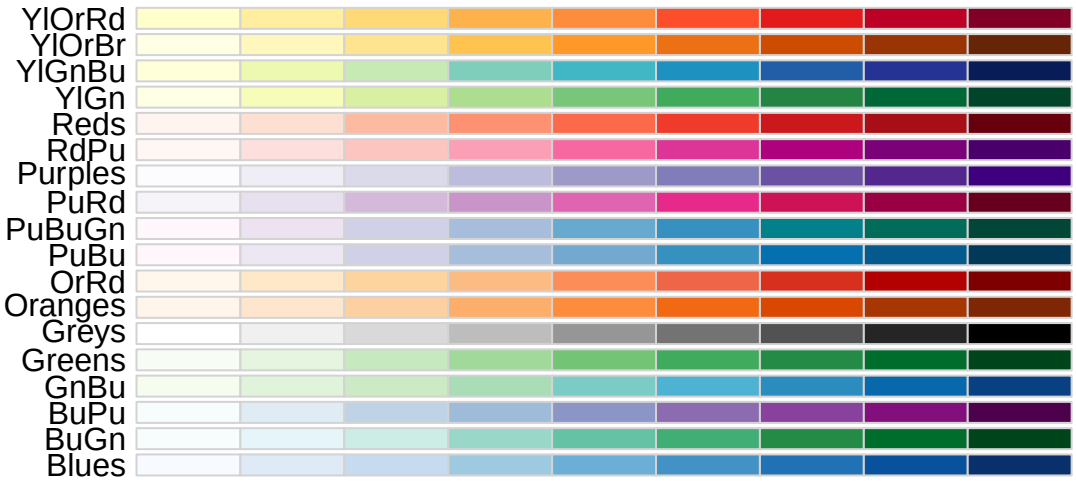
We create a temporary object `dat` that stores only the Measles data, includes a per 100,000 rate, orders states by average value of disease and removes Alaska and Hawaii since they only became states in the late 1950s.

```
the_disease <- "Measles"
dat <- us_contagious_diseases %>%
       filter(!state %in% c("Hawaii","Alaska") & disease == the_disease) %>%
       mutate(rate = (count / weeks_reporting) * 52 / (population / 100000))
```

Can we show data for all states in one plot? We have three variables to show: year, state and rate.

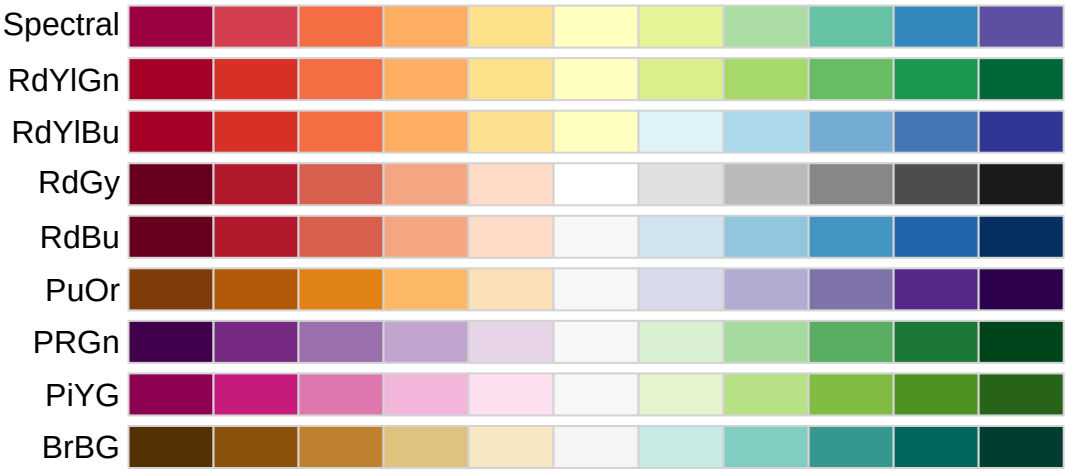
When choosing colors to quantify a numeric variable we chose between two options: *sequential* and *diverging*. Sequential colors are suited for data that goes from high to low. High values are clearly distinguished from low values. Here are some examples offered by the package `RColorBrewer`:

```
library(RColorBrewer)
display.brewer.all(type = "seq")
```



Diverging colors are used to represent values that diverge from a center. We put equal emphasis on both ends of the data range: higher than the center and lower than the center. An example of when we would use a divergent pattern would be if we were to show height in standard deviations away from the average. Here are some examples of divergent patterns:

```
library(RColorBrewer)
display.brewer.all(type="div")
```

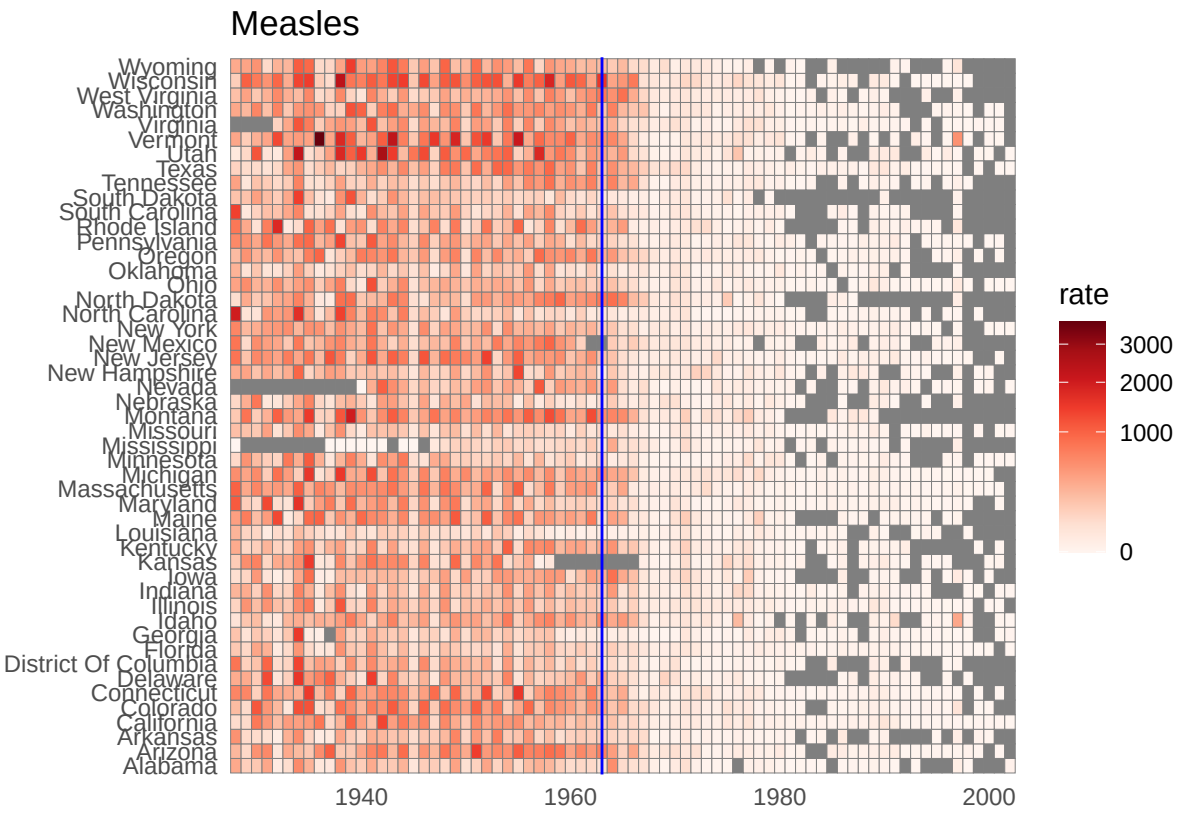


In our example we want to use a sequential palette since there is no meaningful center, just low and high rates.

geom tile

We use the geometry `geom_tile` to tile the region with colors representing disease rates. We use a square root transformation to avoid having the really high counts dominate the plot.

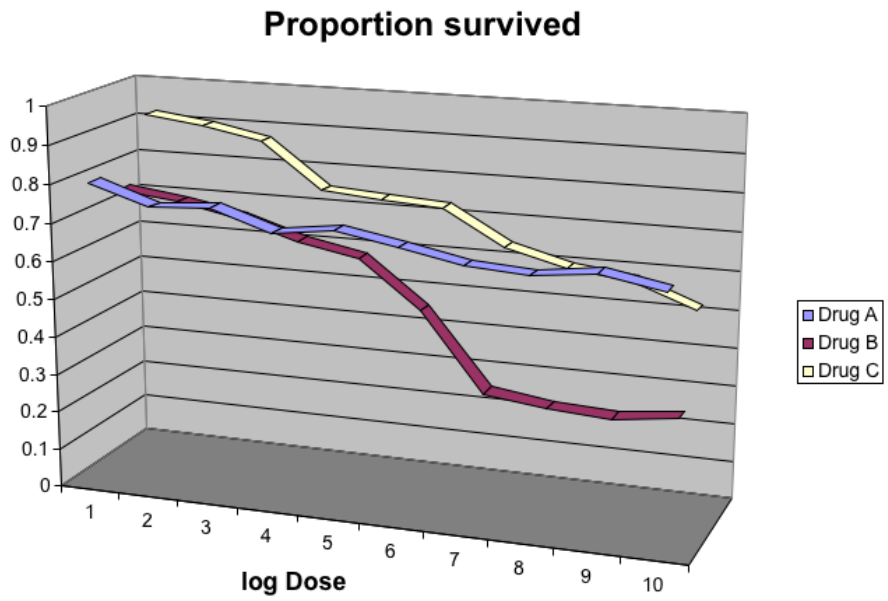
```
dat %>% ggplot(aes(year, state,  fill = rate)) +
        geom_tile(color = "grey50") +
        scale_x_continuous(expand = c(0,0)) +
        scale_fill_gradientn(colors = brewer.pal(9, "Reds"), trans = "sqrt") +
        geom_vline(xintercept = 1963, col = "blue") +
        theme_minimal() +  theme(panel.grid = element_blank()) +
        ggtitle(the_disease) +
        ylab("") +
        xlab("")
```



This plot makes a very striking argument for the contribution of vaccines. However, one limitation of this plot is that it uses color to represent quantity which we earlier explained makes it a bit harder to know exactly how high it is going. Position and length are better cues. If we are willing to lose state information, we can make a version of the plot that shows the values with position.

Avoid pseudo 3D plots

The figure below, taken from the scientific literature [CITE: DNA Fingerprinting: A Review of the Controversy Kathryn Roeder Statistical Science Vol. 9, No. 2 (May, 1994), pp. 222-247] shows three variables: dose, drug type and survival. Although your screen is flat and two dimensional, the plot tries to imitate three dimensions and assigns a dimension to each variable. The extra dimension is not only unnecessary, it makes what should be an easy plot to decipher nearly impossible to draw conclusions from.



Maps

Plots of maps can be very powerful, very informative and very aesthetically pleasing visualizations. However, they can also be misleading if not created correctly. Here we will introduce one R package for creating effective and informative maps and show how to correct a misleading set of maps.

There are several R packages available that can be used to create plots of maps. We will be focusing on one of the packages, `maps`, because of how intuitive it is, the data available, and because it works well with `ggplot2`. We will see several of the options available with this package, but you can read more about the `maps` package here and see more examples here. Other available R packages include `usmap`, `ggmap`, `ggspatial`, `sf`, `urbnmapr` and `rnaturalearth`, and we recommend exploring the examples (like this one) and documentation for these packages for additional types of map plots.

World Map

The `maps` package contains a lot of outlines of continents, countries, states, and counties. In order to map these using `ggplot2`, we must specify which type of map we want with `map_data()`. ggplot2 provides the `map_data()` as a function that turns a series of points along an outline (from the `maps` package) into a data frame of those points. Below we will see maps of the world, a few countries, the entire US, US states and US counties, but there are more options available with this package.

We can save map data as a data frame to see what data is available and then map with `ggplot`. Let's look at the "world" option.

```
# Pull out world map data frame
world_map <- map_data("world")
dim(world_map)
```

```
## [1] 99338     6
```

```
head(world_map)
```

```
##        long      lat group order region subregion
## 1 -69.89912 12.45200     1     1  Aruba      <NA>
```

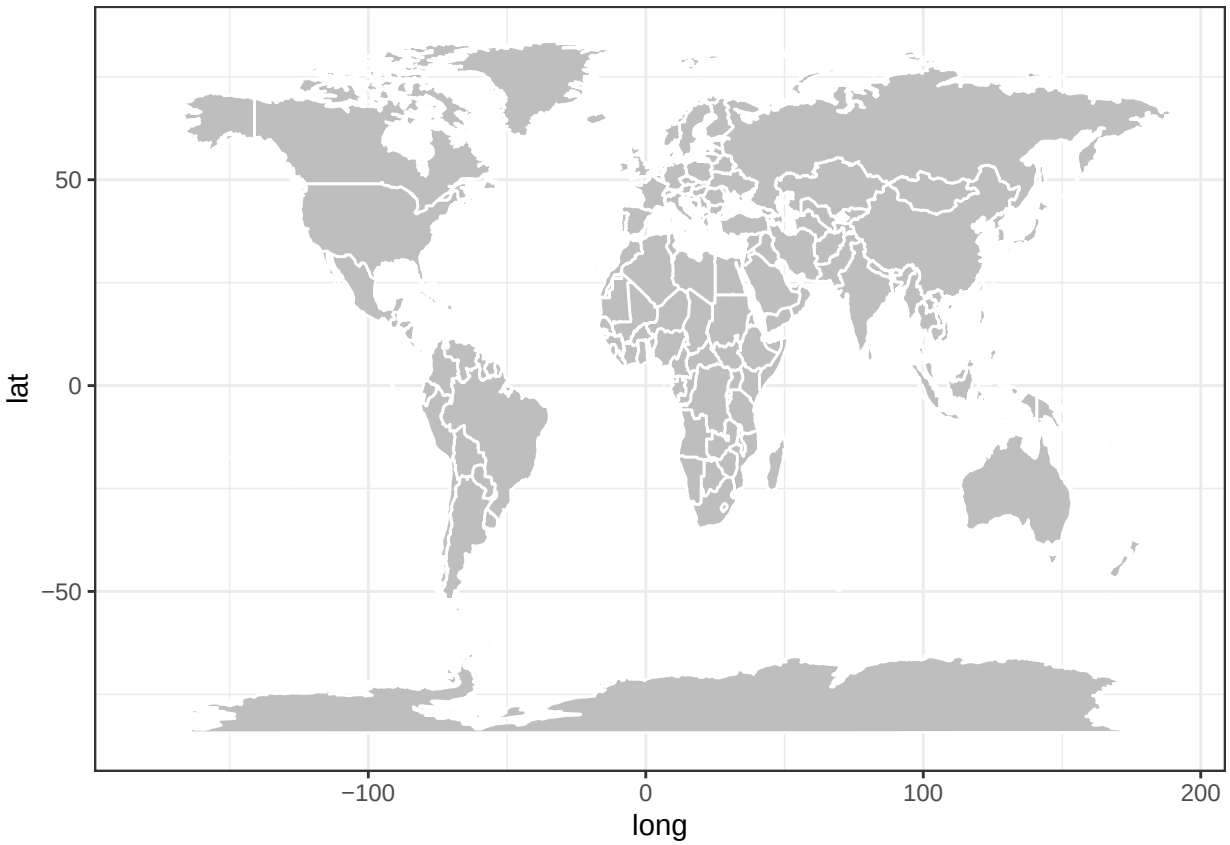
```
## 2 -69.89571 12.42300     1     2  Aruba       <NA>
## 3 -69.94219 12.43853     1     3  Aruba       <NA>
## 4 -70.00415 12.50049     1     4  Aruba       <NA>
## 5 -70.06612 12.54697     1     5  Aruba       <NA>
## 6 -70.05088 12.59707     1     6  Aruba       <NA>
```

We can see the latitude (`lat`) and longitude (`long`) for different regions of the world are available. Here, `region` corresponds to a country and `subregion` indicates additional information about that country if available. For example, the Virgin Islands fall into different political jurisdictions, so you'll see "British" and "US" under `subregion` for the Virgin Islands.

The column `order` shows in which order ggplot should "connect the dots" when plotting. `group` is important since functions in `ggplot2` can take a group argument which controls (amongst other things) whether adjacent points should be connected by lines. If they are in the same group, then they get connected, but if they are in different groups then they don't. Essentially, having two points in different groups means that ggplot "lifts the pen" when going between them. As we'll see below, including `group = group` in the `aes()` argument will plot the outlines of countries. If we don't include `group = group`, the plot will have many lines connecting different parts of different countries and will be impossible to interpret.

For this plot we also use `fill` (to denote the color to fill the countries with) and `color` (the outline color for each country). We will see more options as we create more complex maps.

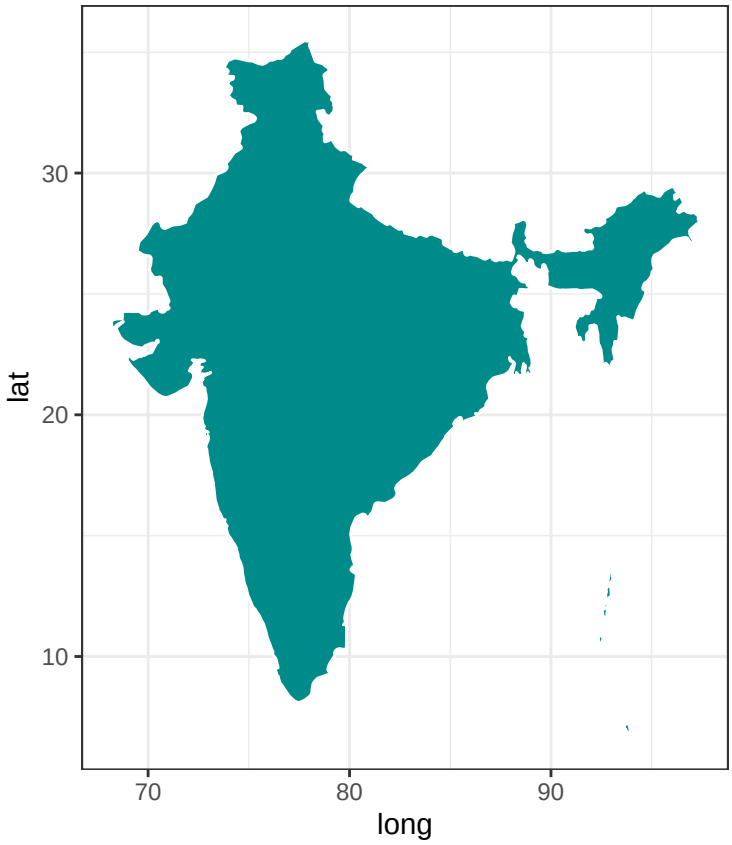
```
# Basic example of a world map
world_map %>% ggplot(aes(x = long, y = lat, group = group)) +
              geom_polygon(fill = "gray", color = "white")
```



We can also map one specific country, or a few countries. If plotting one country, the second argument of `map_data` should include the name of the country. Note that the name of the country can be typed using all lowercase or with the first letter capitalized. We also introduce `coord_fixed` to make plots even better

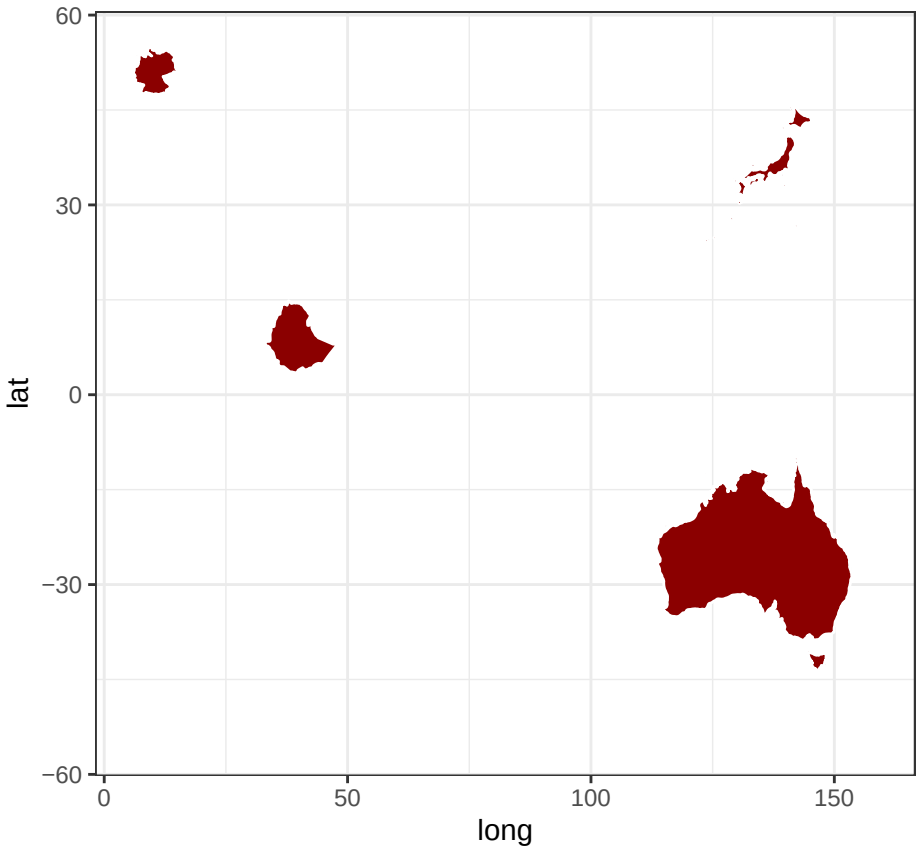
looking. This fixes the relationship between one unit in the y direction and one unit in the x direction. Then, even if you change the outer dimensions of the plot (i.e. by changing the window size or the size of the pdf file you are saving it to for example), the aspect ratio remains unchanged.

```
india_map <- map_data("world", "India")

india_map %>% ggplot(aes(x = long, y = lat, group = group)) +
              geom_polygon(fill = "cyan4", color = "white") +
              coord_fixed(1.2)
```



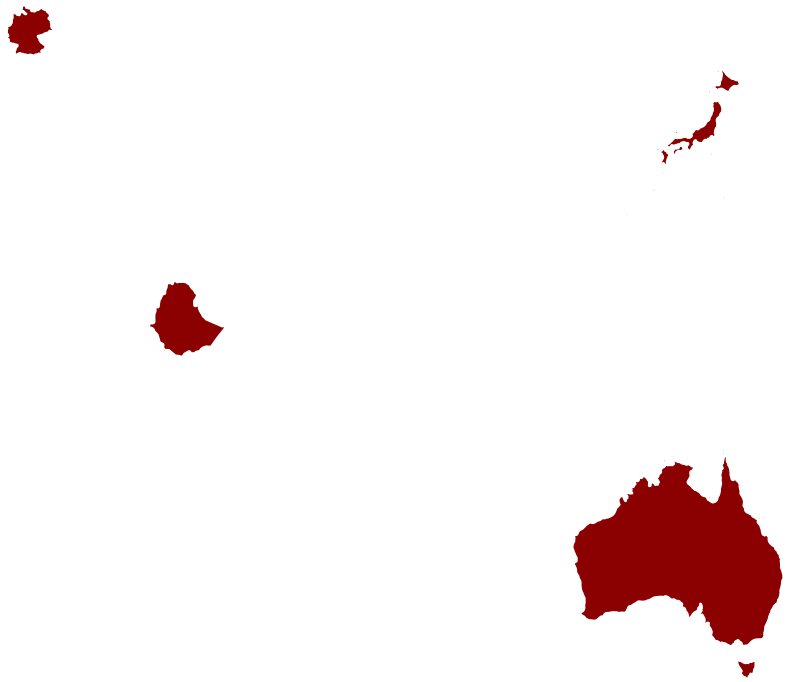
If we want to plot more than one country, we need to supply a string of country names as the second argument. Here we introduce `theme_set(theme_bw())`. As we've seen before, the default background color for `ggplot2` is a light gray with white grid lines. If you would prefer a different color background you can use different themes. `theme_set(theme_bw())` makes the background color white and the grid lines light gray. This just has to do with preference.

```
theme_set(theme_bw())
multiple_countries_map <- map_data("world", c("Japan", "Ethiopia", "Australia", "Germany"))
multiple_countries_map %>% ggplot(aes(x = long, y = lat, group = group)) +
                           geom_polygon(fill = "darkred", color = "white") +
                           coord_fixed(1.3)
```



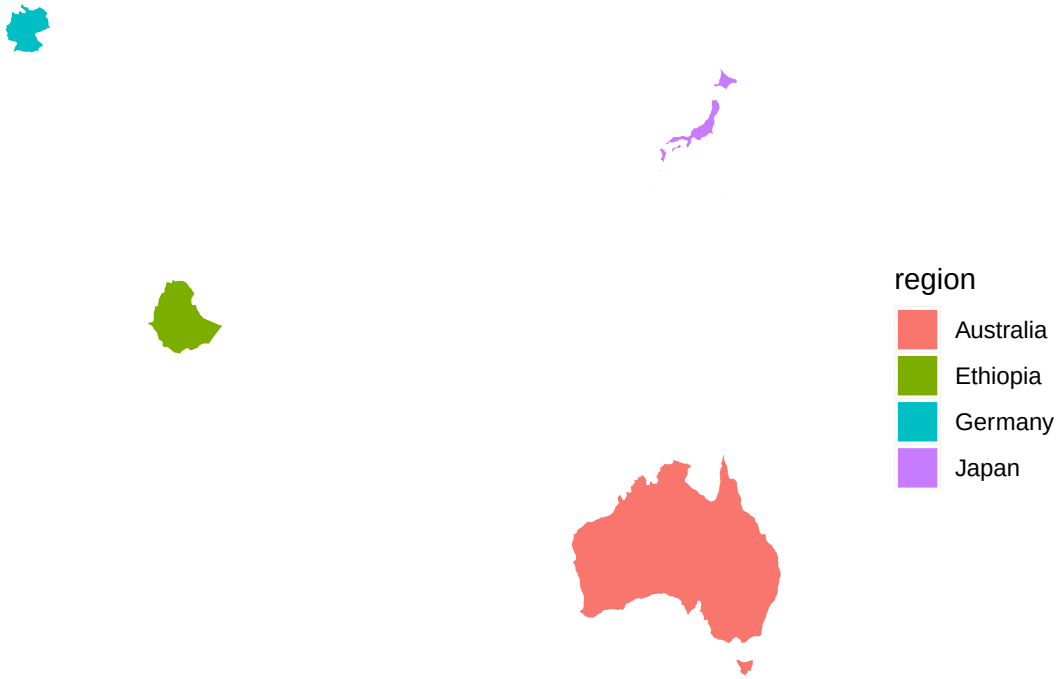
If you want to remove the longitude and latitude axes as well as the grid lines, you can use the `theme` layer and add the following code to the plot. Since we set a theme for all plots above using `theme_set(theme_bw())`, we need to reset the background to the ggplot default using `theme_set(theme_grey())` first.

```
theme_set(theme_grey()) # back to the default background
multiple_countries_map %>% ggplot(aes(x = long, y = lat, group = group)) +
                           geom_polygon(fill = "darkred", color = "white") +
                           theme(panel.grid.major = element_blank(),
                                 panel.background = element_blank(),
                                 axis.title = element_blank(),
                                 axis.text = element_blank(),
                                 axis.ticks = element_blank()) +
                           coord_fixed(1.3)
```



We can also add `fill = region` to the `aes()` function to fill each country with a different color and add a legend.

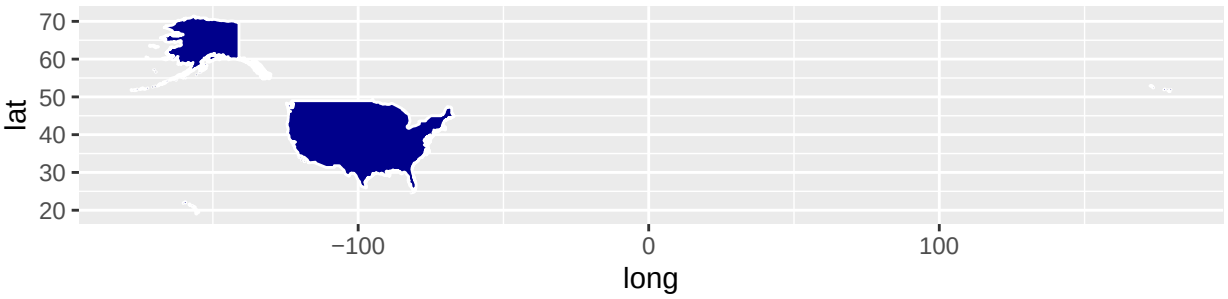
```
theme_set(theme_grey()) # back to the default background
multiple_countries_map %>% ggplot(aes(x = long, y = lat, group = group, fill = region)) +
                           geom_polygon(color = "white") +
                           theme(panel.grid.major = element_blank(),
                                 panel.background = element_blank(),
                                 axis.title = element_blank(),
                                 axis.text = element_blank(),
                                 axis.ticks = element_blank()) +
                           coord_fixed(1.3)
```



US Map

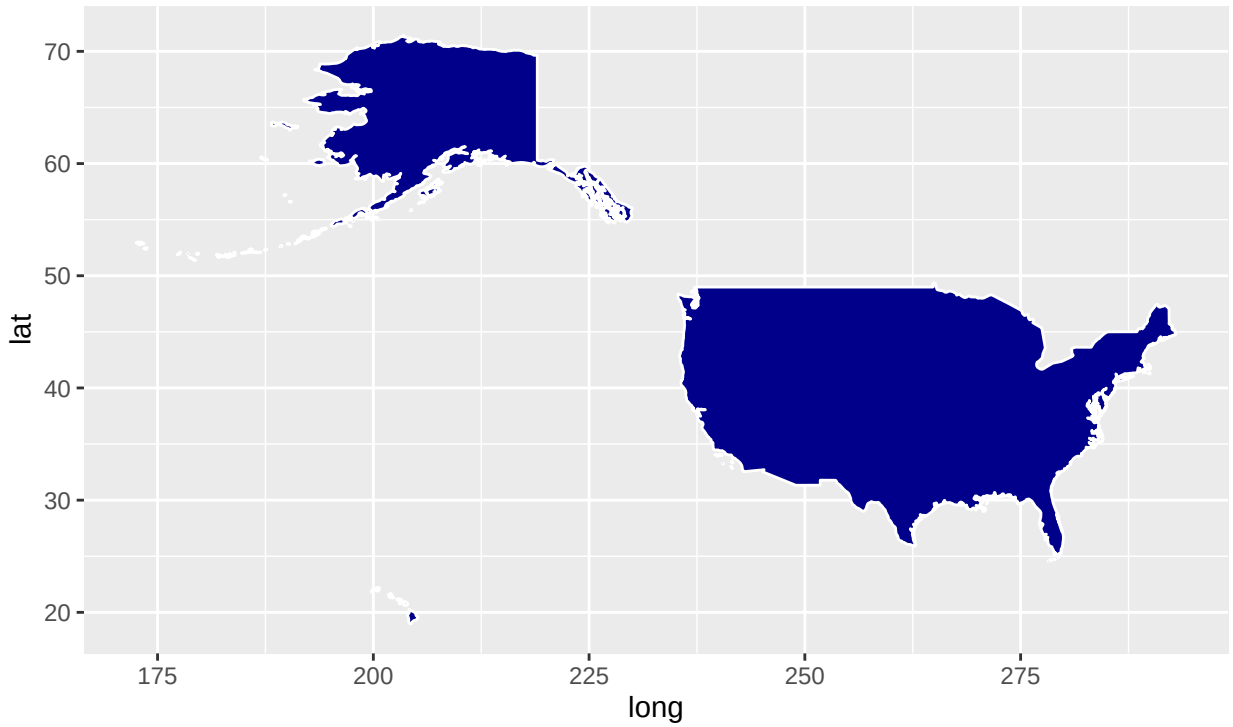
There are multiple ways of plotting the US with the `maps` package. You could use the `world` data and specify `usa` as the country:

```
us_map <- map_data("world", "usa")
us_map %>% ggplot(aes(x = long, y = lat, group = group)) +
           geom_polygon(fill = "blue4", color = "white") +
           coord_fixed(1.3)
```



But this is quite difficult to see because it includes Guam, a US territory (it's difficult to see but this is the set of white dots on the right side of the plot). A better alternative (assuming you don't want to include territories and just US states) would be to use `world2` data and specify `usa`:

```
better_us_map <- map_data("world2", "USA")
better_us_map %>% ggplot(aes(x = long, y = lat, group = group)) +
                  geom_polygon(fill = "blue4", color = "white") +
                  coord_fixed(1.3)
```

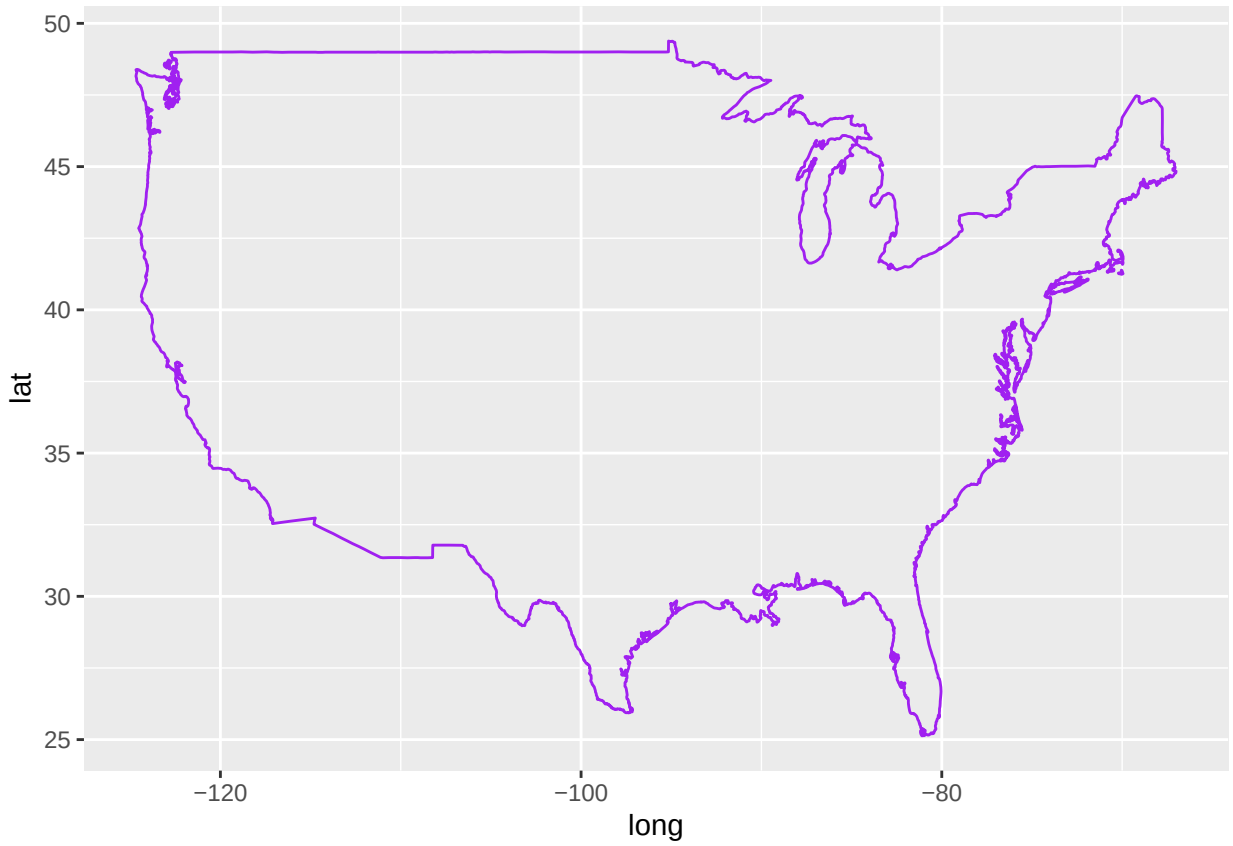


`world2` centers the plot on the Pacific Ocean, allowing us to see Hawaii and bigger versions of Alaska and the mainland states (also known as the lower 48 or contiguous 48).

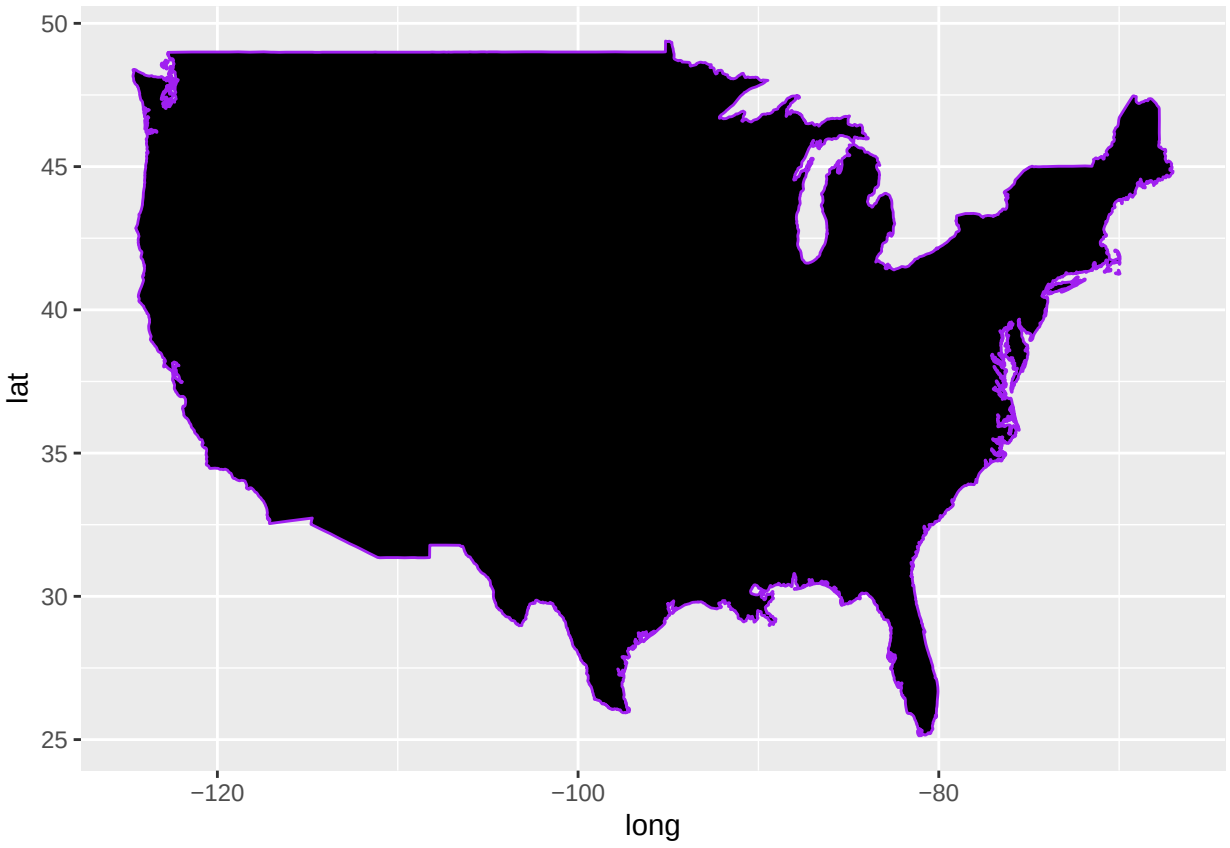
If you don't need to plot Alaska and Hawaii, an even better option is to not use `world` or `world2` and instead use `usa`; the `maps` package provides data specifically for the mainland states of the US. If we use `usa` instead, we get a much better plot of the US (without Alaska and Hawaii).

We can also experiment with color a bit and have the outline color set to purple and the fill color set to `NA`, which leaves it blank or transparent to the background color.

```
usa <- map_data("usa")
usa %>% ggplot(aes(x = long, y = lat, group = group)) +
        geom_polygon(color = "purple", fill = NA)
```



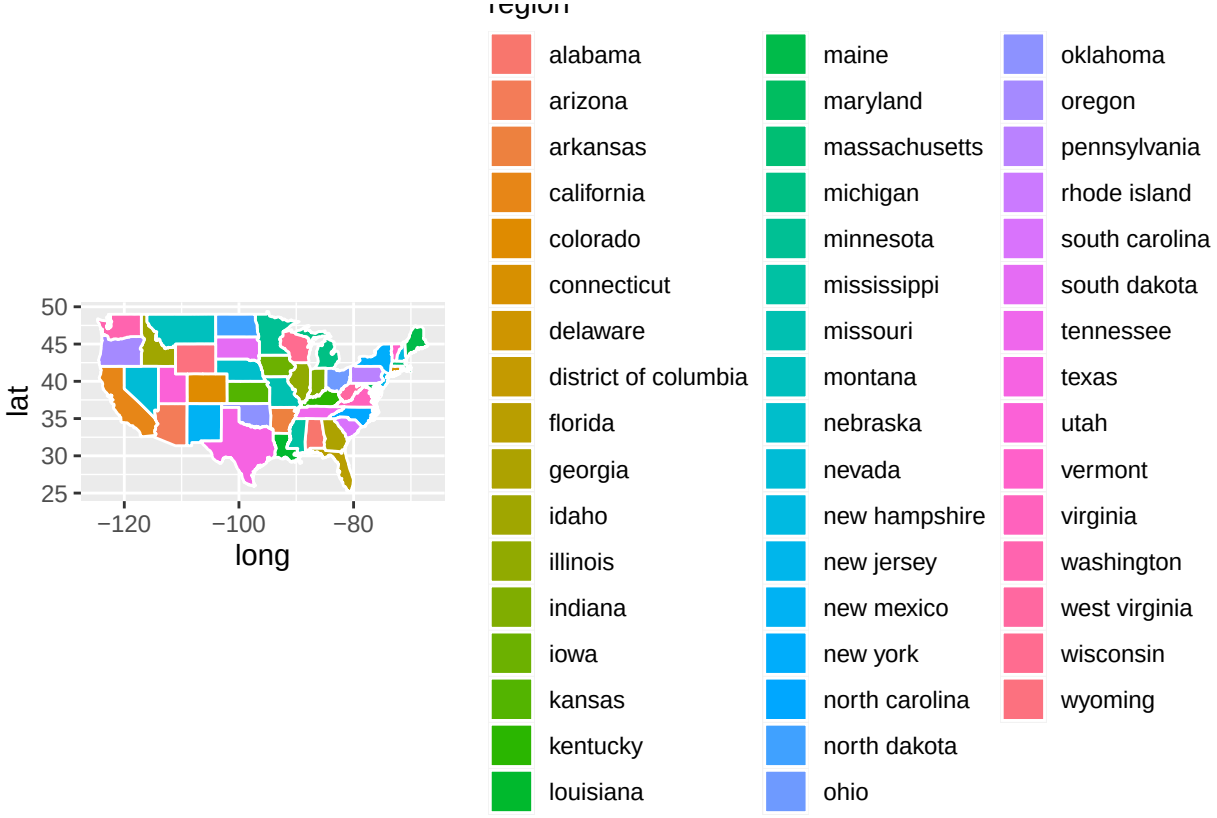

```
usa %>% ggplot(aes(x = long, y = lat, group = group)) +
        geom_polygon(color = "purple", fill = "black")
```



US States

If we want to show individual states, we should instead use the `state` option. But we see a problem - a massive legend is made with one color per state. This is a problem because it eats up most of the plot area. We could make the plot bigger to accommodate the legend, but it would still be difficult for someone without much knowledge of the states to figure out which color corresponds to which state. A better option would be to remove the legend and add labels to the states themselves.

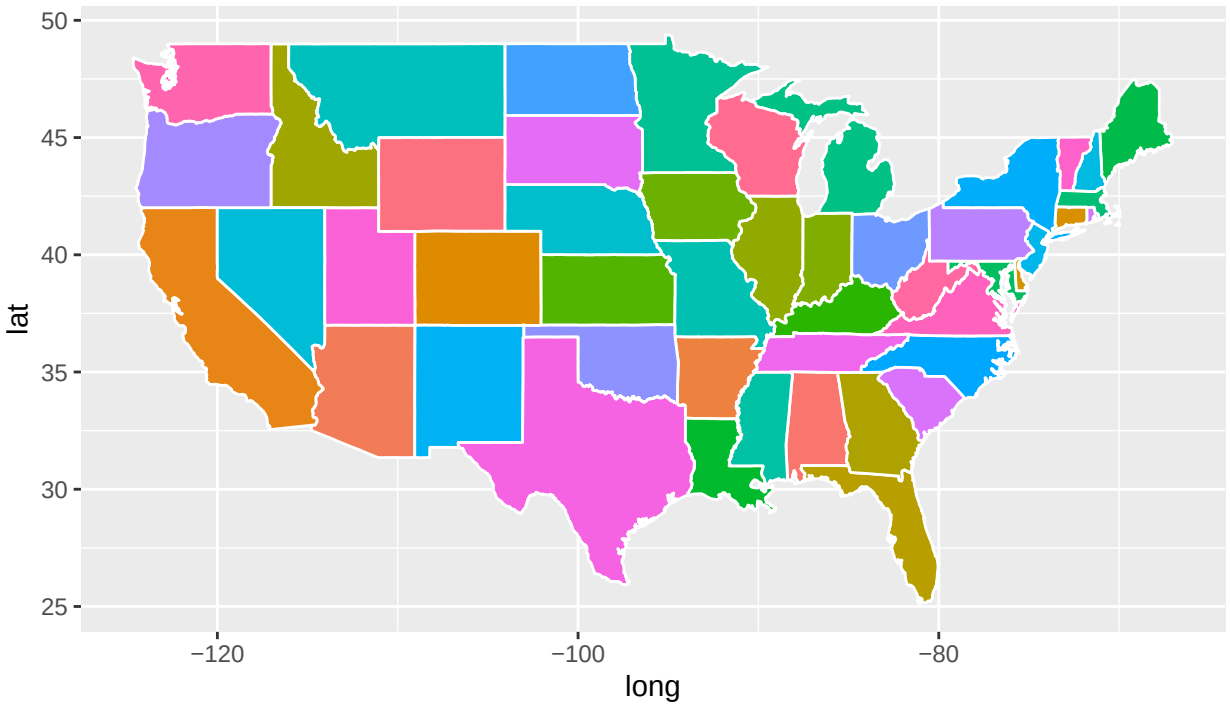
```
us_states <- map_data("state")

us_states %>% ggplot(aes(x = long, y = lat, fill = region, group = group)) +
              geom_polygon(color = "white") +
              coord_fixed(1.3)
```



To remove the legend we add the layer `guides(fill = FALSE)`.

```
us_states %>% ggplot(aes(x = long, y = lat, fill = region, group = group)) +
              geom_polygon(color = "white") +
              coord_fixed(1.3) +
              guides(fill = FALSE) # do this to leave off the color legend
```

```
## Warning: `guides(<scale> = FALSE)` is deprecated. Please use `guides(<scale> =
## "none")` instead.
```

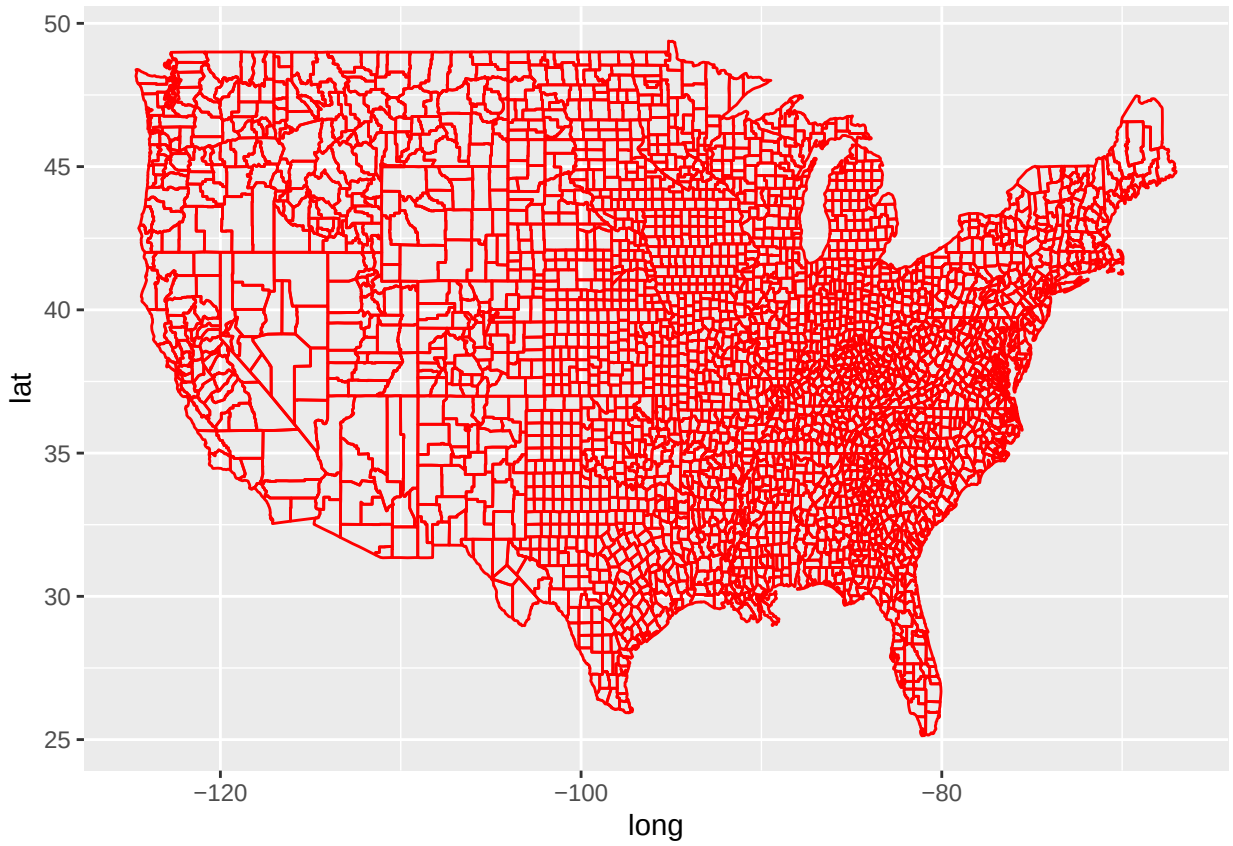


Adding state labels with the `maps` package is more complex and not worth our time here. If you need to label states, we recommend using the `usmap` package and first reading this post and then this post.

US Counties

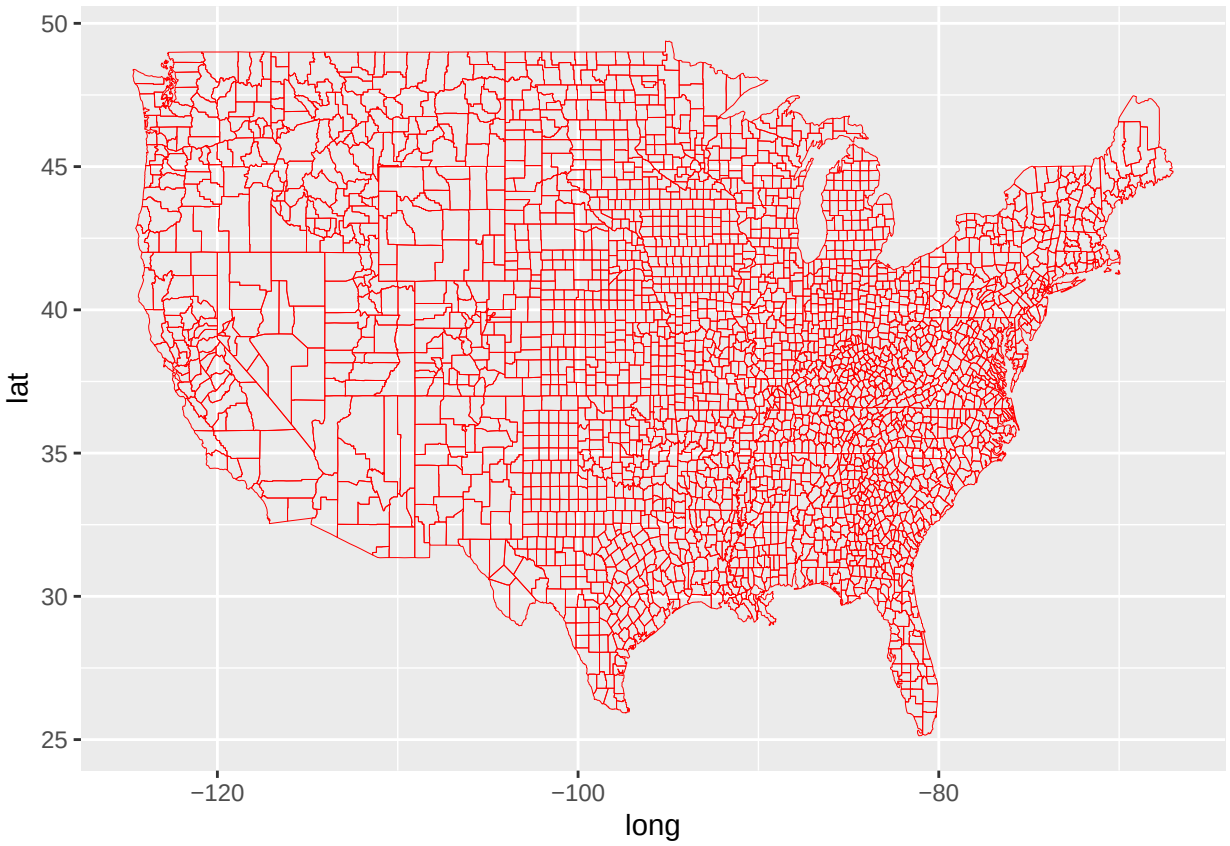
Mapping US state counties is also possible using the `maps` package by using `map_data("county")`. Here we choose a red outline and no fill color, but we notice that the county lines are a little thick and make some counties difficult to see.

```
AllCounty <- map_data("county")
AllCounty %>% ggplot(aes(x = long, y = lat, group = group)) +
              geom_polygon(color = "red", fill = NA)
```



We can change the width of the lines using the `size` argument, making individual counties a bit easier to see.

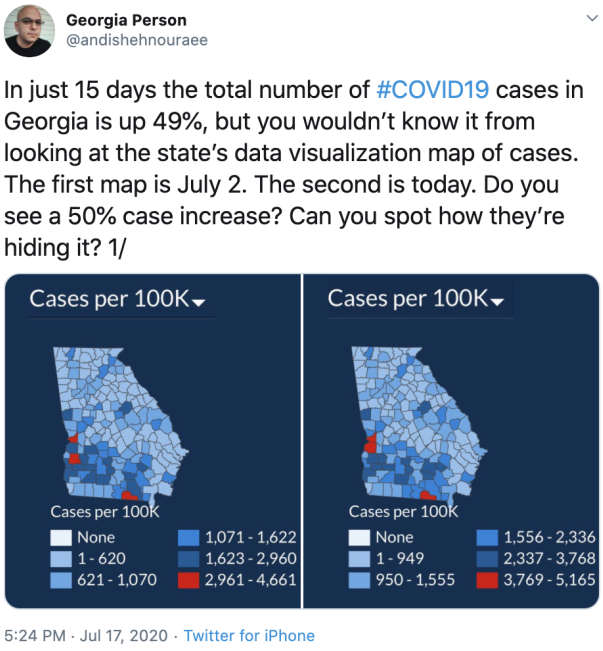
```
AllCounty %>% ggplot(aes(x = long, y = lat, group = group)) +
              geom_polygon(color = "red", fill = NA, size = .1 )
```



Case Study: Georgia Counties COVID-19 Cases

Now that we know the basic setup for plotting different kinds of maps, let's create a meaningful map. Specifically, let's create a corrected version of a misleading set of maps.

On July 17th, 2020 Andisheh Nouraee tweeted about the following screenshots on Twitter, calling out Georgia's Department of Public Health for hiding the severity of the COVID-19 outbreak in Georgia with its map visualizations.



Nouraee had been taking daily screenshots and noticed that the legend numbers kept changing. With changing legend number cutoffs for each color, it seems like the number of cases is staying steady, rather than increasing. As ethical data scientists and members of the Harvard Chan School Community, we know that plots like this spread misinformation and can lead to worse health outcomes. In order to correct this mistake and accurately show the severity of the outbreak, we will be recreating this plot with a common legend for both dates. We'll use the `maps` package and a bit of data wrangling to achieve the desired final product.

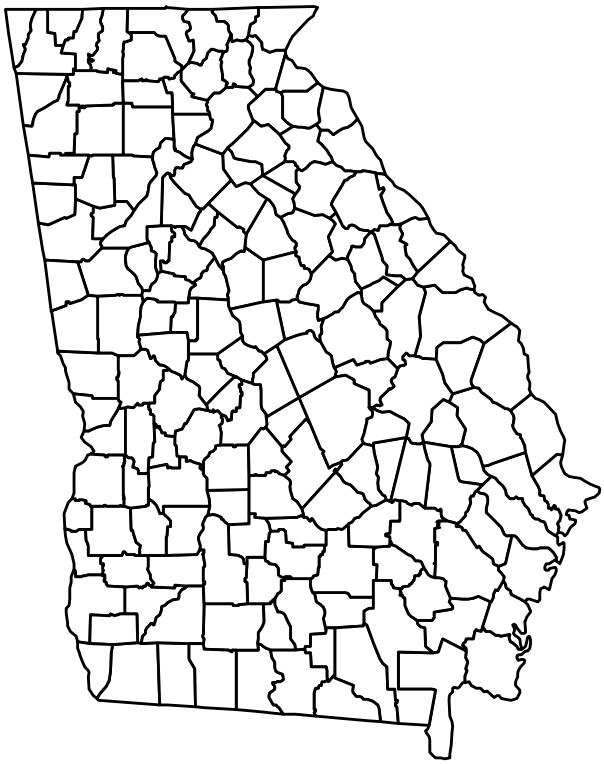
Let's first make a plot of Georgia and its counties. We specify the `county` data as before, but add `georgia` to indicate we only want the counties for the state of Georgia to be plotted.

```
GeorgiaCounty <- map_data("county", "georgia")
head(GeorgiaCounty)
```

```
##        long      lat group order  region subregion
## 1 -82.44862 31.94813     1     1 georgia   appling
## 2 -82.42570 31.94813     1     2 georgia   appling
## 3 -82.40852 31.94240     1     3 georgia   appling
## 4 -82.39706 31.94240     1     4 georgia   appling
## 5 -82.38560 31.93094     1     5 georgia   appling
## 6 -82.35122 31.91948     1     6 georgia   appling
```

Then we plot the state and counties without any axes or background color or grid lines. For now the counties are white with black outlines, we'll change this to depend on the number of cases. We'll also produce 2 plots of Georgia counties, one for July 2nd and one for July 17th.

```
GeorgiaCounty %>% ggplot(aes(x = long, y = lat, group = group)) +
                  geom_polygon(fill = "white", color = "black") +
                  theme(panel.grid.major = element_blank(),
                        panel.background = element_blank(),
                        axis.title = element_blank(),
                        axis.text = element_blank(),
                        axis.ticks = element_blank()) +
                  coord_fixed(1.3)
```



We have the data needed to make a plot of counties in Georgia, but we don't have any COVID-19 case data yet. Using the `us-counties.csv` data file posted by the New York Times on GitHub, we can read in daily new COVID-19 cases data for all counties in the US starting in January 2020.

```
url <- "https://raw.githubusercontent.com/nytimes/covid-19-data/master/us-counties.csv"
cases <- read_csv(url)
```

```
## Rows: 2502832 Columns: 6
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr  (3): county, state, fips
## dbl  (2): cases, deaths
## date (1): date
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
head(cases)
```

```
## # A tibble: 6 x 6
##   date       county    state      fips  cases deaths
##   <date>     <chr>     <chr>      <chr> <dbl>  <dbl>
## 1 2020-01-21 Snohomish Washington 53061     1      0
## 2 2020-01-22 Snohomish Washington 53061     1      0
## 3 2020-01-23 Snohomish Washington 53061     1      0
## 4 2020-01-24 Cook      Illinois   17031     1      0
## 5 2020-01-24 Snohomish Washington 53061     1      0
## 6 2020-01-25 Orange    California 06059     1      0
```

We can see we have more data than we need since we are focusing on Georgia and only two specific dates. Let's filter out rows we won't need for this plot and save the relevant rows as a new data frame.

```
dates <- c(ymd("2020-07-02", "2020-07-17"))
georgia_cases <- cases %>% filter(state == "Georgia", date %in% dates)

dim(georgia_cases)
```

```
## [1] 320   6
```

```
head(georgia_cases)
```

```
## # A tibble: 6 x 6
##   date       county   state   fips  cases deaths
##   <date>     <chr>    <chr>   <chr> <dbl>  <dbl>
## 1 2020-07-02 Appling  Georgia 13001   266     14
## 2 2020-07-02 Atkinson Georgia 13003   155      2
## 3 2020-07-02 Bacon    Georgia 13005   254      4
## 4 2020-07-02 Baker    Georgia 13007    43      3
## 5 2020-07-02 Baldwin  Georgia 13009   543     34
## 6 2020-07-02 Banks    Georgia 13011   140      1
```

Looking back at the original plots, we find that the number of cases are presented per 100,000 people. We need to know the population of each county in Georgia before we can create a case rate per 100,000 people. But we aren't provided population counts in the `georgia_cases` or `GeorgiaCounty` data frames - we need to find another dataset to pull this information from and join it with what we have.

We can get the population data we need from the Georgia Department of Public Health website. There isn't a raw version of this dataset available so we downloaded it, saved it as `county_pop.csv`, and placed it in our course data visualization folder on GitHub.

Now we can read in the data and see what we have.

```
pop_data <- read_csv("county_pop.csv")
```

```
## Rows: 161 Columns: 12
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (4): county_name, county_id, State FIPS code, County FIPS code
## dbl (8): cases, population, hospitalization, deaths, case rate, death rate, ...
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
dim(pop_data)
```

```
## [1] 161  12
```

```
head(pop_data)
```

```
## # A tibble: 6 x 12
##   county_name cases county_id `State FIPS code` `County FIPS code` population
##   <chr>       <dbl> <chr>     <chr>             <chr>                    <dbl>
## 1 Appling      1054 US-13001  13                1                        18561
## 2 Atkinson      447 US-13003  13                3                         8330
## 3 Bacon         602 US-13005  13                5                        11404
## 4 Baker          84 US-13007  13                7                         3116
## 5 Baldwin      2132 US-13009  13                9                        44428
## 6 Banks         501 US-13011  13                11                       19982
## # ... with 6 more variables: hospitalization <dbl>, deaths <dbl>,
## #   `case rate` <dbl>, `death rate` <dbl>, `14 day case rate` <dbl>,
```

```
## #   `14 day cases` <dbl>
```

The state counties are listed under `county_name` and population under `population`. These are the only 2 columns we need for joining population data to our cases data frame. First we only keep the `county_name` and `population` columns to make the join simpler and to make sure we don't create any duplicate columns when joining to the cases data frame. Next we use `left_join` to add the population for each county to the cases data frame. Note that the column names for county are different in each dataset - in the cases data frame the column name is `county` while it is `county_name` in the population data frame. We could rename `county_name` to `county`, or we can simply let the join function know that the data frames have different names for the same thing using `by = c("county" = "county_name")`.

```
pop_data <- pop_data %>% select(county_name, population)

data_full <- left_join(georgia_cases, pop_data, by = c("county" = "county_name"))
dim(data_full)
```

```
## [1] 320   7
```

```
head(data_full)
```

```
## # A tibble: 6 x 7
##   date       county   state   fips  cases deaths population
##   <date>     <chr>    <chr>   <chr> <dbl>  <dbl>      <dbl>
## 1 2020-07-02 Appling  Georgia 13001   266     14      18561
## 2 2020-07-02 Atkinson Georgia 13003   155      2       8330
## 3 2020-07-02 Bacon    Georgia 13005   254      4      11404
## 4 2020-07-02 Baker    Georgia 13007    43      3       3116
## 5 2020-07-02 Baldwin  Georgia 13009   543     34      44428
## 6 2020-07-02 Banks    Georgia 13011   140      1      19982
```

Now that the `population` column has been added to the `cases` data frame and a new combined data frame, `data_full`, has been created, we can create a cases per 100,000 population variable called `rate`.

```
data_full <- data_full %>% mutate(rate = 100000*(cases/population))
head(data_full)
```

```
## # A tibble: 6 x 8
##   date       county   state   fips  cases deaths population  rate
##   <date>     <chr>    <chr>   <chr> <dbl>  <dbl>      <dbl> <dbl>
## 1 2020-07-02 Appling  Georgia 13001   266     14      18561 1433.
## 2 2020-07-02 Atkinson Georgia 13003   155      2       8330 1861.
## 3 2020-07-02 Bacon    Georgia 13005   254      4      11404 2227.
## 4 2020-07-02 Baker    Georgia 13007    43      3       3116 1380.
## 5 2020-07-02 Baldwin  Georgia 13009   543     34      44428 1222.
## 6 2020-07-02 Banks    Georgia 13011   140      1      19982  701.
```

We have one more join to do before we can create the plot. The `data_full` data frame needs to be joined to the `GeorgiaCounty` data frame we created earlier that contains the mapping data for the counties. Before we can join, look at the `county` column in `data_full` and `subregion` column in `GeorgiaCounty`. These are both referring to the counties in Georgia, but the `county` entries are capitalized while `subregion` entries are not. If we want to join these data frames, the quickest way will be to make the `county` column entries lowercase using `str_to_lower` and then joining.

```
georgia_map <- data_full %>% mutate(county = str_to_lower(county)) %>%
  left_join(GeorgiaCounty, by = c("county" = "subregion"))

dim(georgia_map)
```

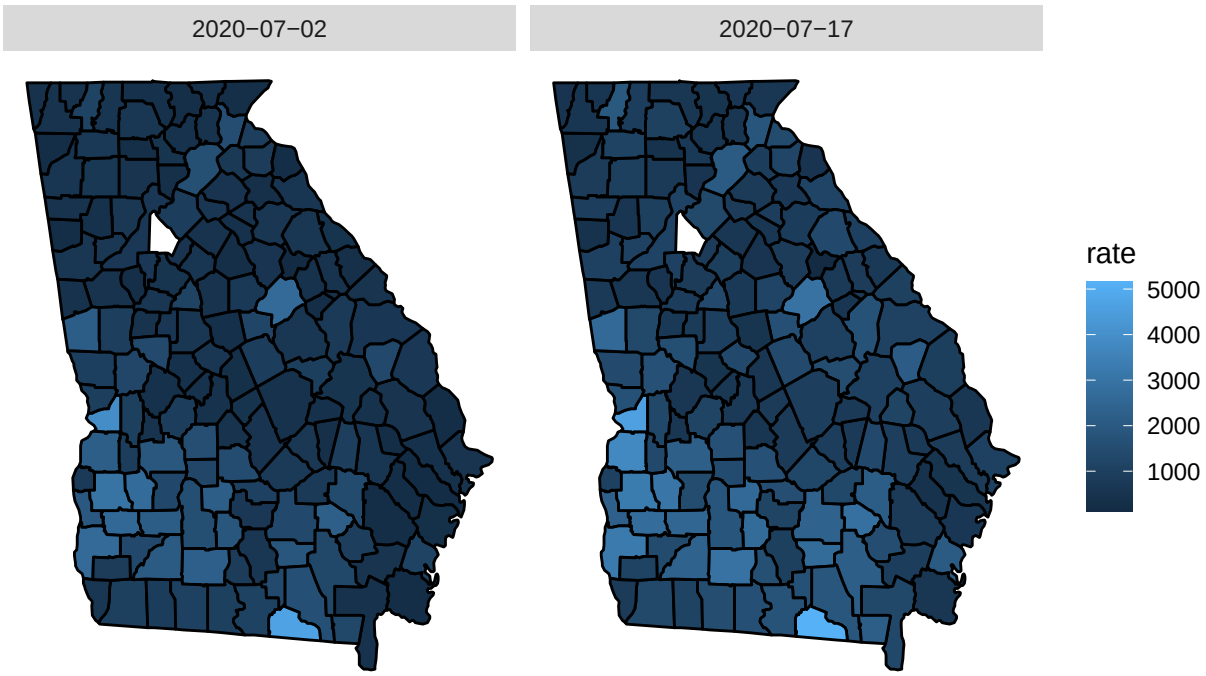
```
## [1] 10192    13
```

```
head(georgia_map)
```

```
## # A tibble: 6 x 13
##   date       county  state fips  cases deaths population  rate  long   lat group
##   <date>     <chr>   <chr> <chr> <dbl>  <dbl>      <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 2020-07-02 appling Geor~ 13001   266     14      18561 1433. -82.4  31.9     1
## 2 2020-07-02 appling Geor~ 13001   266     14      18561 1433. -82.4  31.9     1
## 3 2020-07-02 appling Geor~ 13001   266     14      18561 1433. -82.4  31.9     1
## 4 2020-07-02 appling Geor~ 13001   266     14      18561 1433. -82.4  31.9     1
## 5 2020-07-02 appling Geor~ 13001   266     14      18561 1433. -82.4  31.9     1
## 6 2020-07-02 appling Geor~ 13001   266     14      18561 1433. -82.4  31.9     1
## # ... with 2 more variables: order <int>, region <chr>
```

Now we are ready to start plotting. Before we specify the cutoffs and colors we want, let's create the default version of the plot with counties filled by case rate and a subplot for each date. Note that we need to remove the `fill = "white"` code we used earlier.

```
georgia_map %>% ggplot(aes(x = long, y = lat, group = group, fill = rate)) +
                geom_polygon(color = "black") +
                theme(panel.grid.major = element_blank(),
                      panel.background = element_blank(),
                      axis.title = element_blank(),
                      axis.text = element_blank(),
                      axis.ticks = element_blank()) +
                coord_fixed(1.3) +
                facet_grid( ~ date)
```



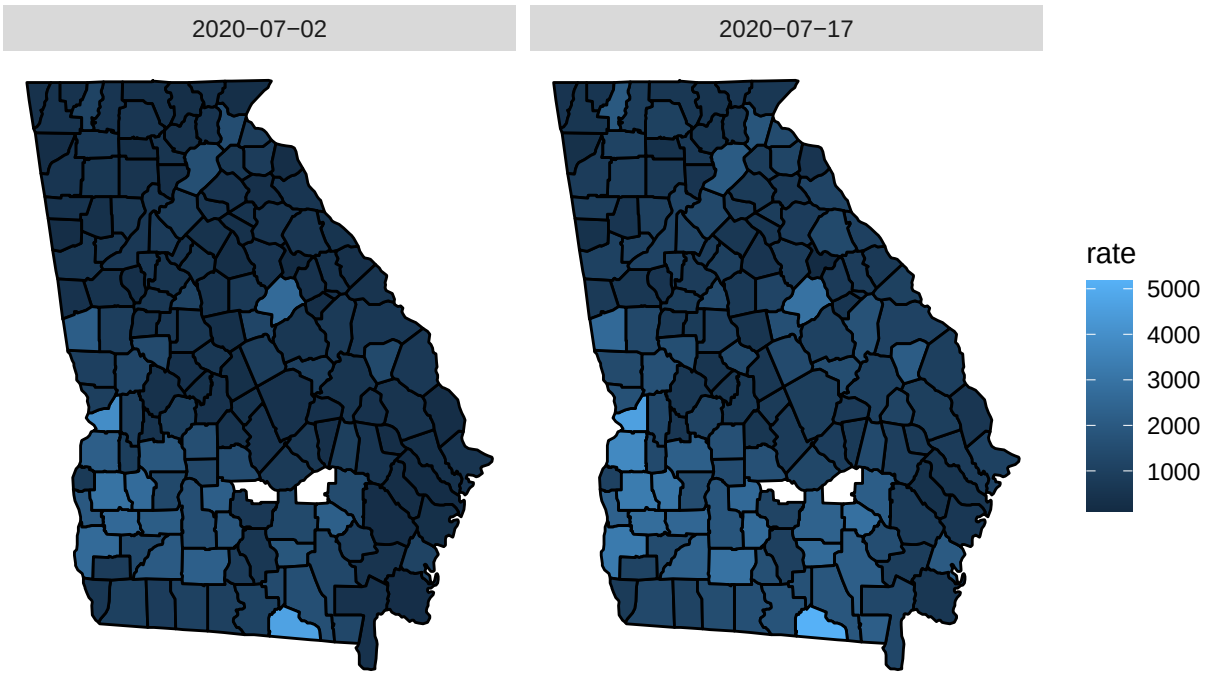
One thing we notice is that one county is white - it's missing rate data. On closer inspection of the `GeorgiaCounty` and `georgia_cases` data frames, we notice that DeKalb county is saved as "de kalb" in the `GeorgiaCounty` data frame, but as "DeKalb" in the in the `georgia_cases` data frame. That extra space means that even if we make all letters lowercase, `de kalb` and `dekalb` won't match and when we join the data frames, DeKalb county will be missing rate data. We need to remove the space in the `GeorgiaCounty`

data frame and then join the data frames again.

```
GeorgiaCounty <- GeorgiaCounty %>%
  mutate(subregion = str_replace(subregion, " ", ""))

georgia_map <- data_full %>% mutate(county = str_to_lower(county)) %>%
  left_join(GeorgiaCounty, by = c("county" = "subregion"))

head(georgia_map)
```

```
## # A tibble: 6 x 13
##   date       county  state fips  cases deaths population  rate  long   lat group
##   <date>     <chr>   <chr> <chr> <dbl>  <dbl>      <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 2020-07-02 appling Geor~ 13001   266     14      18561 1433. -82.4  31.9     1
## 2 2020-07-02 appling Geor~ 13001   266     14      18561 1433. -82.4  31.9     1
## 3 2020-07-02 appling Geor~ 13001   266     14      18561 1433. -82.4  31.9     1
## 4 2020-07-02 appling Geor~ 13001   266     14      18561 1433. -82.4  31.9     1
## 5 2020-07-02 appling Geor~ 13001   266     14      18561 1433. -82.4  31.9     1
## 6 2020-07-02 appling Geor~ 13001   266     14      18561 1433. -82.4  31.9     1
## # ... with 2 more variables: order <int>, region <chr>
```

```
georgia_map %>% ggplot(aes(x = long, y = lat, group = group, fill = rate)) +
                geom_polygon(color = "black") +
                theme(panel.grid.major = element_blank(),
                      panel.background = element_blank(),
                      axis.title = element_blank(),
                      axis.text = element_blank(),
                      axis.ticks = element_blank()) +
                coord_fixed(1.3) +
                facet_grid( ~ date)
```

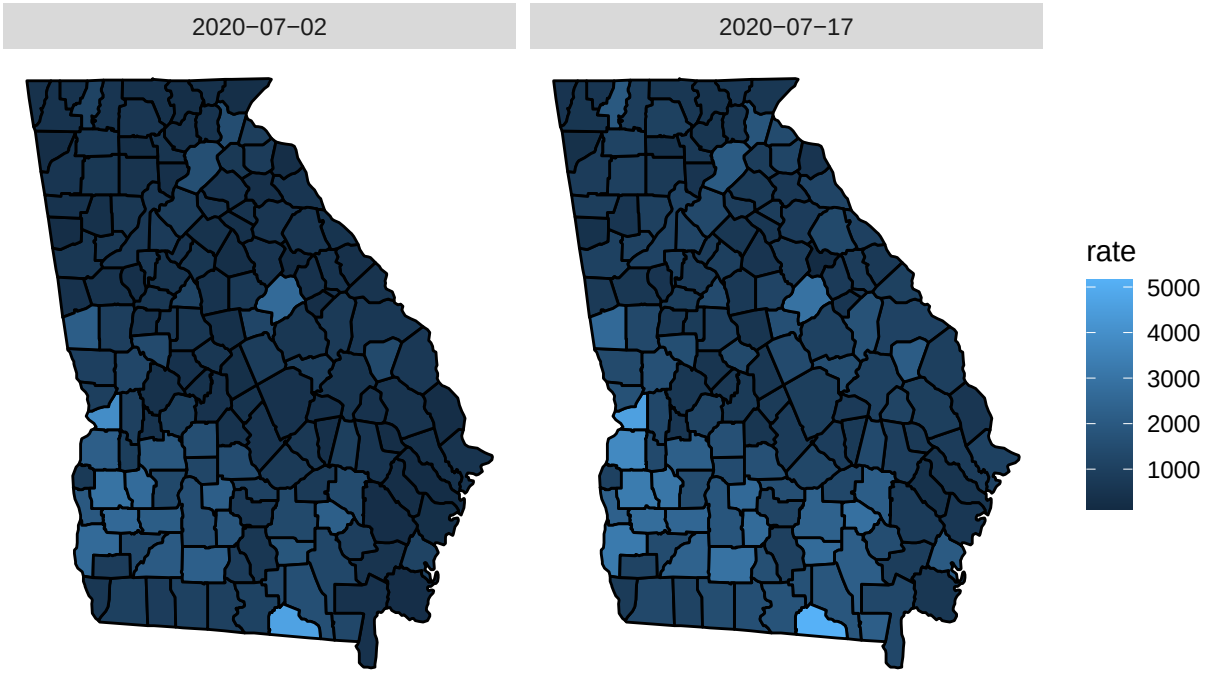


Uh oh. It's happened again but with 2 different counties. The names of the counties are not matching up again. Looking at a map of Georgia counties we can match the white counties to Ben Hill and Jeff Davis. Both of these had spaces in the `GeorgiaCounty` and `georgia_map` data (and thus matched), but we removed

the spaces and now they don't match. We have 2 options: remove all spaces in both data frames, or go back and only remove the space in DeKalb. Since we don't need the names of the counties in this map, let's remove all spaces from both data frames.

```
data_full <- data_full %>%
  mutate(county = str_replace(county, " ", ""))

georgia_map <- data_full %>% mutate(county = str_to_lower(county)) %>%
  left_join(GeorgiaCounty, by = c("county" = "subregion"))
```

```
georgia_map %>% ggplot(aes(x = long, y = lat, group = group, fill = rate)) +
                geom_polygon(color = "black") +
                theme(panel.grid.major = element_blank(),
                      panel.background = element_blank(),
                      axis.title = element_blank(),
                      axis.text = element_blank(),
                      axis.ticks = element_blank()) +
                coord_fixed(1.3) +
                facet_grid( ~ date)
```



This looks good and can see that there is a common legend for both plots - exactly what we want! Now we just need to change the colors a bit and customize the legend cutoff values, labels and title.

We'll use the base R `cut` function to specify case rate ranges and create a new column in our `georgia_map` data frame named `manual_fill` to use in our plot. The first argument of the `cut` function specifies what variable or column you want to split into ranges. The cut points are then specified with the `breaks` argument, and the labels for the resulting intervals are specified with the `labels` argument. Finally, we need to add `right = TRUE` to indicate the intervals should be closed on the right and open on the left. This means, for example, an interval for 1-10 is represented as (1, 10] where 10 is included but 1 is not.

```
georgia_map <- georgia_map %>%
  mutate(manual_fill = cut(rate, breaks = c(0, 620, 1070, 1622, 2960, Inf),
                           labels = c("1-620", "621-1,070", "1,071-1,622",
                                      "1,623-2,960", ">2,960"),
```

```
                           right = TRUE))
head(georgia_map)
```

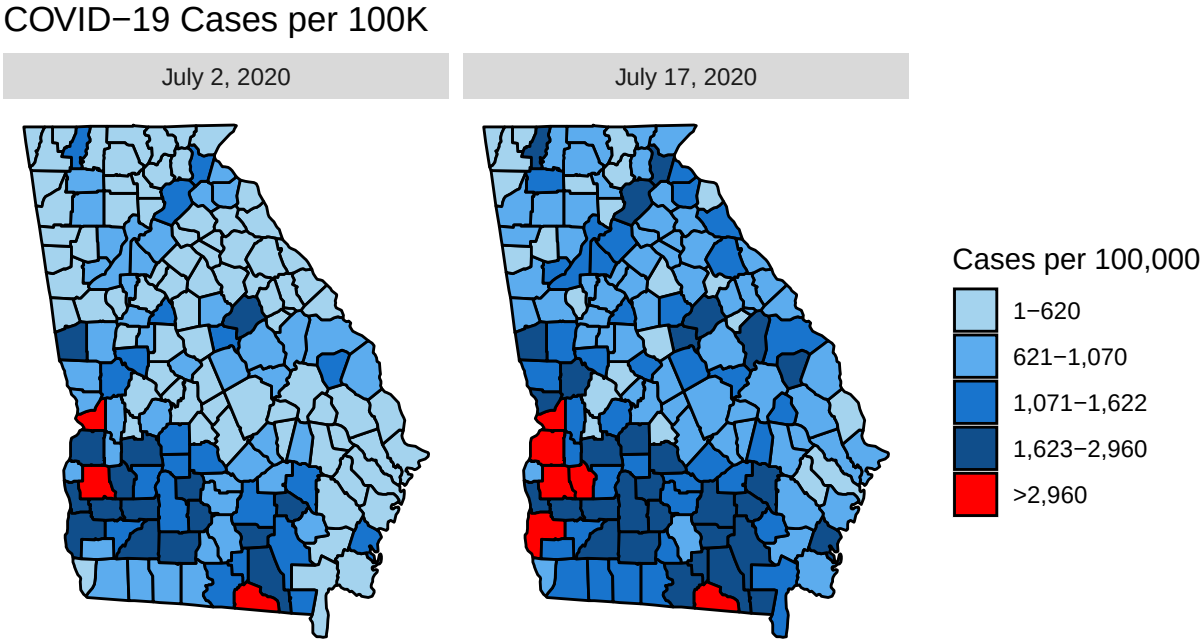
```
## # A tibble: 6 x 14
##   date       county  state fips  cases deaths population  rate  long   lat group
##   <date>     <chr>   <chr> <chr> <dbl>  <dbl>      <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 2020-07-02 appling Geor~ 13001   266     14      18561 1433. -82.4  31.9     1
## 2 2020-07-02 appling Geor~ 13001   266     14      18561 1433. -82.4  31.9     1
## 3 2020-07-02 appling Geor~ 13001   266     14      18561 1433. -82.4  31.9     1
## 4 2020-07-02 appling Geor~ 13001   266     14      18561 1433. -82.4  31.9     1
## 5 2020-07-02 appling Geor~ 13001   266     14      18561 1433. -82.4  31.9     1
## 6 2020-07-02 appling Geor~ 13001   266     14      18561 1433. -82.4  31.9     1
## # ... with 3 more variables: order <int>, region <chr>, manual_fill <fct>
```

One final step before plotting is specifying the colors we want. The colors chosen from this resource are as close as possible to the colors used in the original plots.

```
# Specify desired colors
pal <- c("lightskyblue2", "steelblue2", "dodgerblue3", "dodgerblue4", "red")
```

Using `manual_fill` to fill the counties and a `scale_fill_manual` layer specifying the title and labels of the legend, we get the desired result and can now see that the cases have actually increased between July 2nd and July 17th.

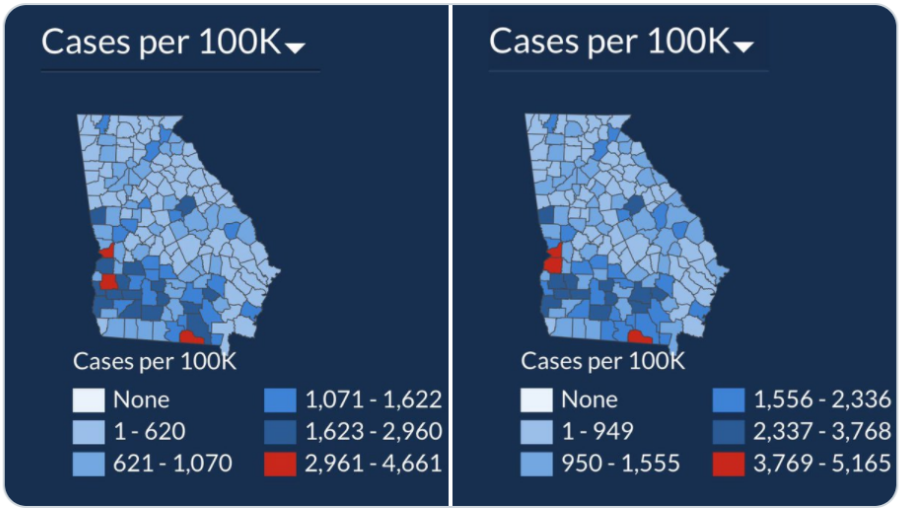
```
date.labs <- c("July 2, 2020", "July 17, 2020")
names(date.labs) <- c("2020-07-02", "2020-07-17")

georgia_map %>% ggplot(aes(x = long, y = lat, group = group)) +
  geom_polygon(aes(fill = manual_fill), color = "black")  +
  scale_fill_manual(name = "Cases per 100,000", values = pal) +
  coord_fixed(1.3) +
  theme(panel.grid.major = element_blank(),
        panel.background = element_blank(),
        axis.title = element_blank(),
        axis.text = element_blank(),
        axis.ticks = element_blank()) +
  ggtitle("COVID-19 Cases per 100K") +
  facet_grid(. ~ date, labeller = labeller(date = date.labs))
```



And we can see the difference between our corrected version and the misleading original.

Georgia Person
@andishehnouraee

In just 15 days the total number of #COVID19 cases in Georgia is up 49%, but you wouldn't know it from looking at the state's data visualization map of cases. The first map is July 2. The second is today. Do you see a 50% case increase? Can you spot how they're hiding it? 1/



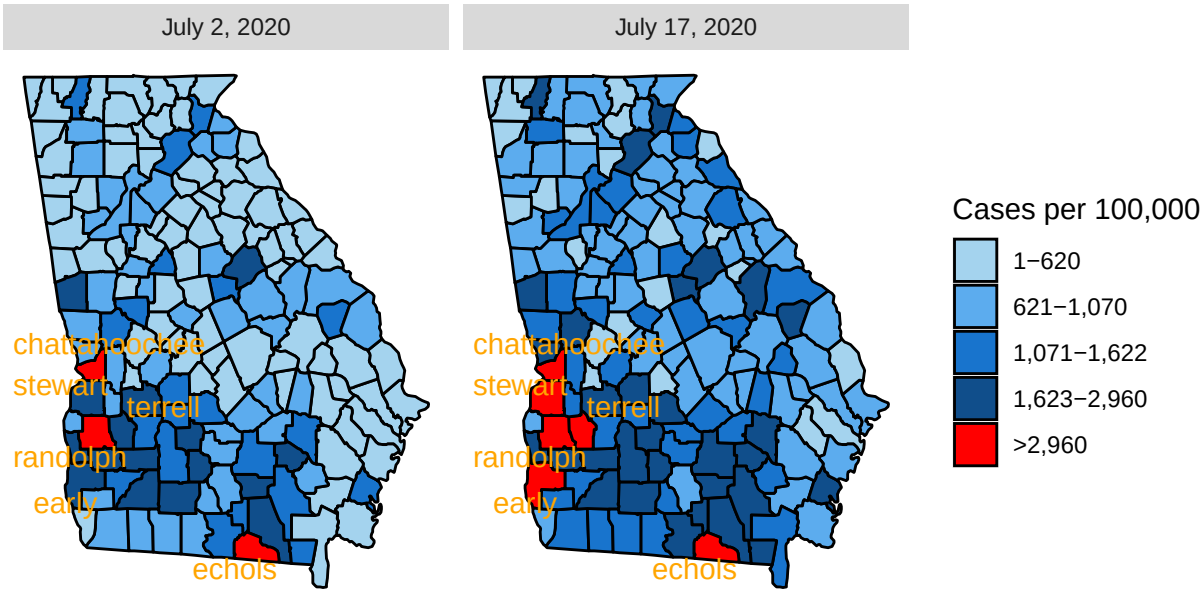
5:24 PM · Jul 17, 2020 · Twitter for iPhone

If we want to know which counties are in red on July 17th, we can add their names as labels to our graph. Note that because we are using `facet_grid` the county names will appear on both subplots.

```
georgia_map_labels <- georgia_map %>%
  filter(rate > 2960) %>%
  group_by(county) %>%
  summarize(lat = mean(lat),
            long = mean(long))
```

```
library(ggrepel)

georgia_map %>% ggplot(aes(x = long, y = lat)) +
  geom_polygon(aes(fill = manual_fill, group = group), color = "black")  +
  scale_fill_manual(name = "Cases per 100,000", values = pal) +
  coord_fixed(1.3) +
  geom_text_repel(data = georgia_map_labels, aes(long, lat, label = county), color = "orange") +
  theme(panel.grid.major = element_blank(),
        panel.background = element_blank(),
        axis.title = element_blank(),
        axis.text = element_blank(),
        axis.ticks = element_blank()) +
  ggtitle("COVID-19 Cases per 100K") +
  facet_grid(. ~ date, labeller = labeller(date = date.labs))
```

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

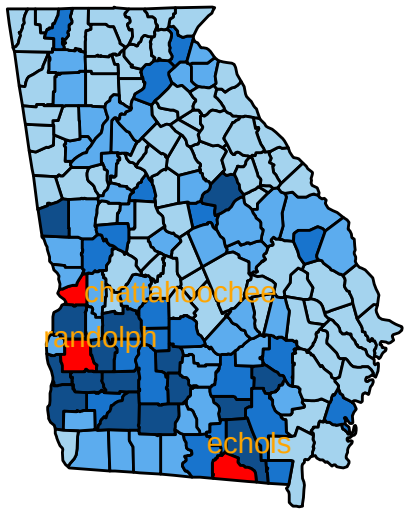


If you don't want this to happen and only want to label the red counties for each date, you would plot each date separately and then combine the plots using the `ggarrange` function from the `egg` package. The `ggarrange` function is similar to the `grid.arrange` function, but `ggarrange` has the added benefit of making the subplots the same size. If we were to use `grid.arrange`, the July 2nd plot would be much bigger than the July 17th plot because the July 2nd plot does not include a legend.

```
july_2_labels <- georgia_map %>%
  filter(date == "2020-07-02", rate > 2960) %>%
  group_by(county) %>%
```

```
  summarize(lat = mean(lat),
            long = mean(long))

july_17_labels <- georgia_map %>%
  filter(date == "2020-07-17", rate > 2960) %>%
  group_by(county) %>%
  summarize(lat = mean(lat),
            long = mean(long))


p1 <- georgia_map %>% filter(date == "2020-07-02") %>%
  ggplot(aes(x = long, y = lat)) +
  geom_polygon(aes(fill = manual_fill, group = group), color = "black")  +
  scale_fill_manual(values = pal) +
  coord_fixed(1.3) +
  geom_text_repel(data = july_2_labels, aes(long, lat, label = county), color = "orange") +
  theme(legend.position = "none",
        panel.grid.major = element_blank(),
        panel.background = element_blank(),
        axis.title = element_blank(),
        axis.text = element_blank(),
        axis.ticks = element_blank()) +
  ggtitle("July 2, 2020")

p2 <- georgia_map %>% filter(date == "2020-07-17") %>%
  ggplot(aes(x = long, y = lat)) +
  geom_polygon(aes(fill = manual_fill, group = group), color = "black")  +
  scale_fill_manual(name = "Cases per 100,000", values = pal) +
  coord_fixed(1.3) +
  geom_text_repel(data = july_17_labels, aes(long, lat, label = county), color = "orange") +
  theme(panel.grid.major = element_blank(),
        panel.background = element_blank(),
        axis.title = element_blank(),
        axis.text = element_blank(),
        axis.ticks = element_blank()) +
  ggtitle("July 17, 2020")

library(egg)
ggarrange(p1, p2, nrow = 1)
```

July 2, 2020



July 17, 2020