R Data Visualization Course

Contents

Intro to Plotting

Variable Types

Categorial variables are usually strings and contain a few categories with many elements. *Ordinal* variables are a subtype of categoricals that are can be meaningfully ordered (e.g. spiciness: mild, medium hot). **Numeric** variables are numbers, such as a counts, volumes, or areas. They can be *discrete* integers, such as a population count, or *continuous*, taking any value with enough precision, such as a height.

```
data(heights)
prop.table(table(heights$sex))
```

```
##
##    Female      Male
## 0.2266667 0.7733333
```

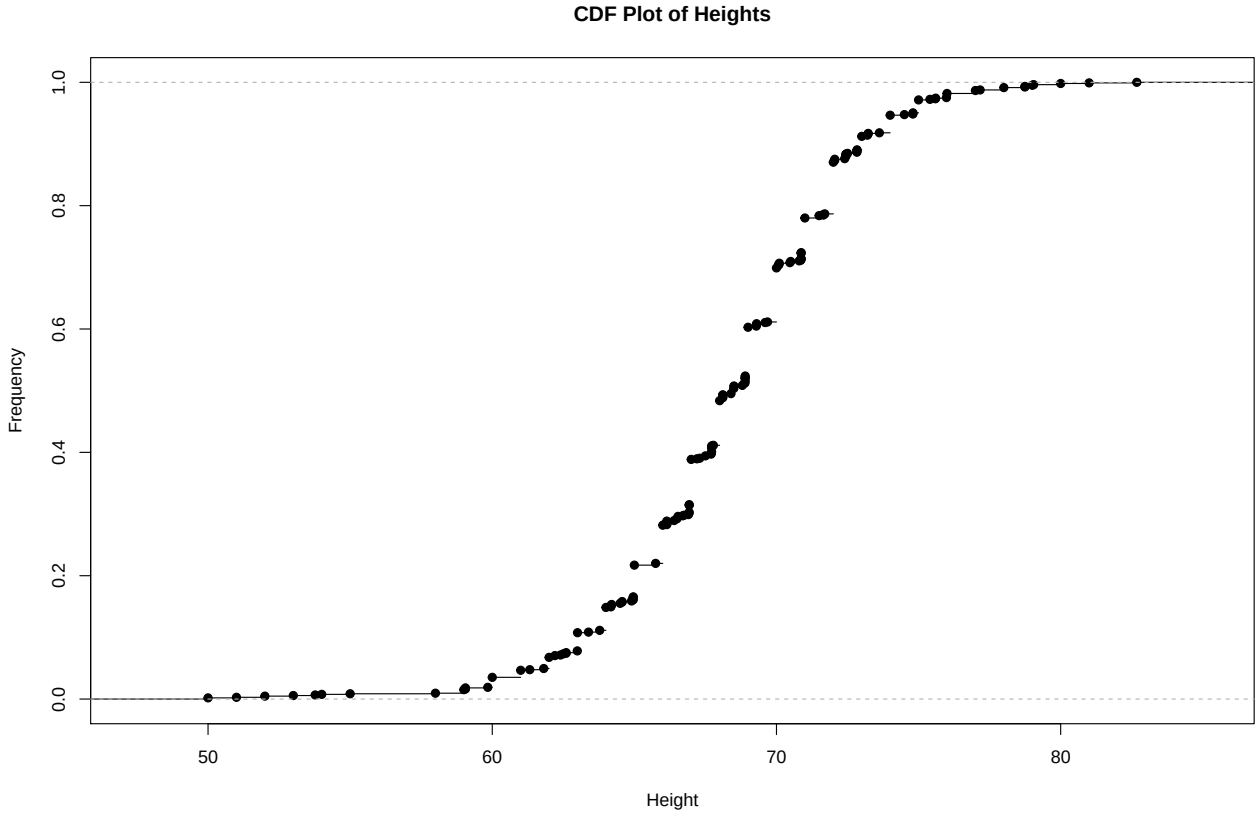
Defining the Distribution

Rather than report the frequency of each numerical element (many of which occur only once), we can report a distribution. We define a **cumulative distribution function** (CDF) that gives the proportion of the data below a value a for all values of a.

$$F(a) = Pr(x \leq a)$$

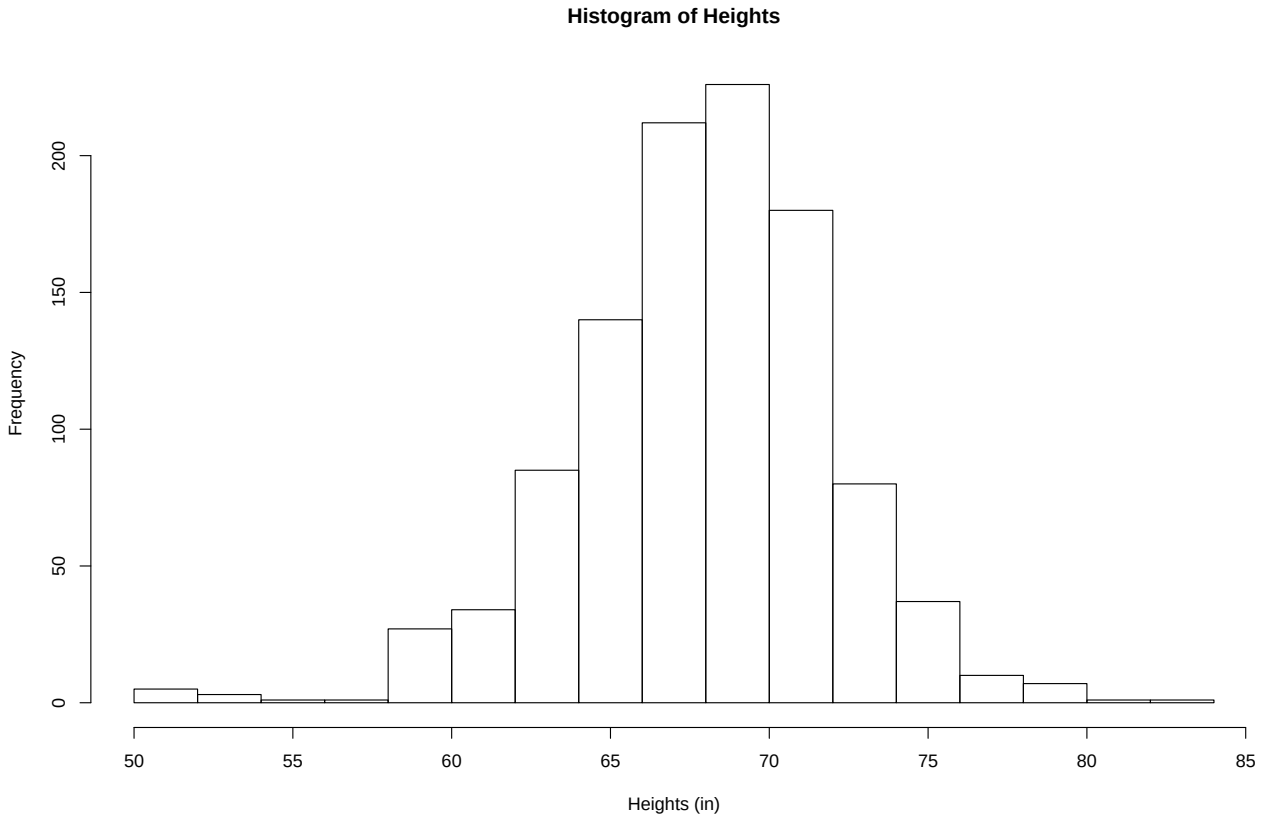
We can create a plot to demonstrate what this looks like for heights.

```
plot(ecdf(heights$height), xlab = "Height",
     ylab = "Frequency",
     main = "CDF Plot of Heights")
```



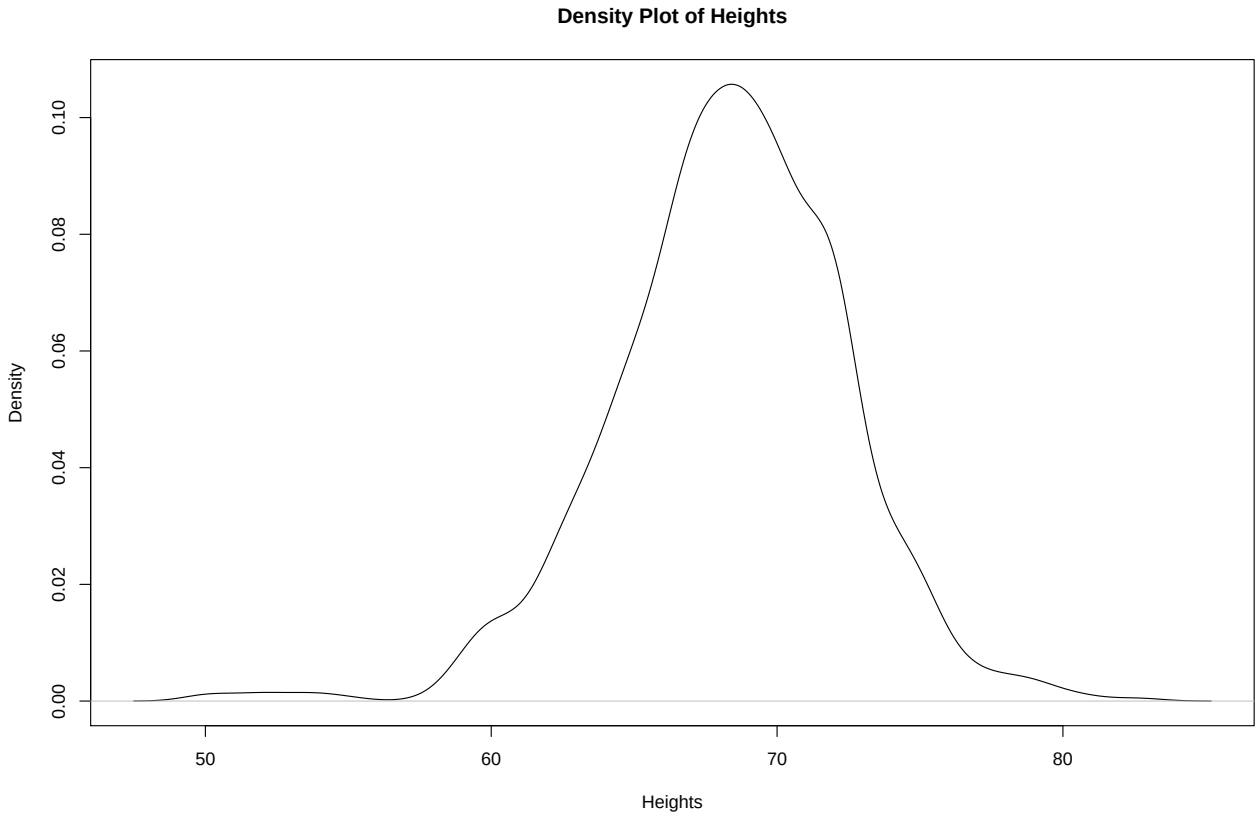
To better see whether the data are symmetric and where they are centered, we can use a histogram. Note that bin widths can be adjusted.

```
hist(heights$height,
     main = "Histogram of Heights",
     xlab = "Heights (in)")
```



It is also possible to create a smooth density plot from the histogram that makes the distribution simpler and easier to compare.

```
plot(density(heights$height),
     main = "Density Plot of Heights",
     xlab = "Heights")
```



The Normal Distribution

The normal distribution (aka Gaussian distribution) is defined by the following equation:

$$Pr(a < x < b) = \int_a^b \frac{1}{\sqrt{2\pi}s} e^{-\frac{1}{2}(\frac{x-m}{s})^2} dx$$

Here's the LateX code:

```
Pr (a < x < b) = \int_a^b \frac{1}{\sqrt{2\pi} s}                                    e^{-\frac{1}{2}(\fra
```

We define the **standard unit** as:

$$z = (x - average)/SD$$

This tells us how many standard deviations away a given value is from the mean.

To get standard units in R use: 'z <- scale (x)'

```
z <- scale(heights$height[heights$sex == "Male"])
mean(abs(z < 2))
```

```
## [1] 0.9802956
```

In a normal distribution, we would expect 95% of the values to be within two standard deviations of the mean. Here we see that the *heights* data roughly approximates that pattern.

If the data are normal, we can estimate the proportion of the data between two values with the function *pnorm()*. Here, pnorm estimates the proportion of the data with a value less the value given based on the

average and standard deviation applied to a normal curve. To find the proportion of the data expressed between the two values (e.g. 69 and 72 in), simply subtract one from the other.

```
heights_Male <- heights$height[heights$sex == "Male"]
mh_norm <- pnorm(c(69, 72), mean(heights_Male), sd(heights_Male))
mh_norm
```

```
## [1] 0.4652702 0.7714481
```

```
mh_norm[2] - mh_norm[1]
```

```
## [1] 0.3061779
```

Here's how we can estimate the percentage of men who are Lebron James' height (6ft 8in) who are in the NBA:

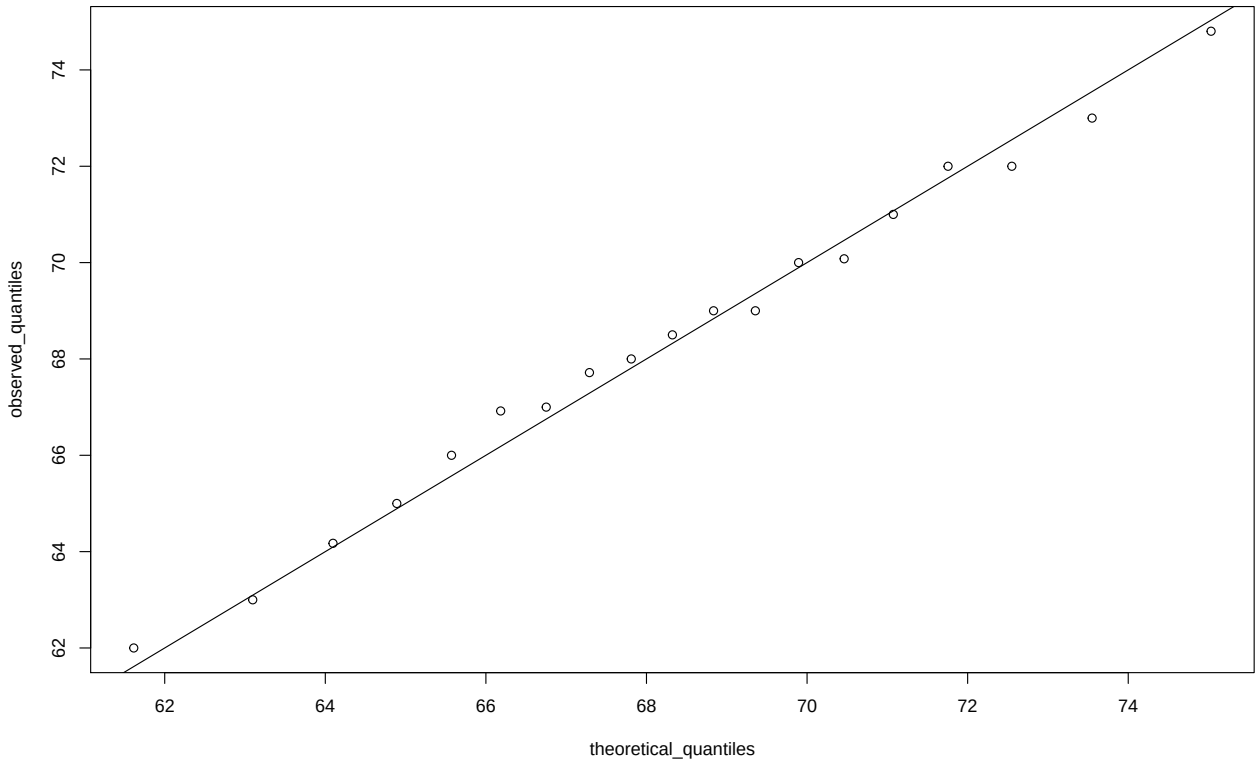
```
#estimate the proportion of men who are taller than 80 in using mean of 69 and sd of 3
p <- 1 - pnorm(80, 69, 3)
#estimate the total number of men over 6'8" to the nearest integer
N <- round(p * 7.4 * 10^9)
#divide the number of men in the NBA over 6'8" by the total
150/N * 100
```

```
## [1] 0.01649782
```

Quantile-Quantile Plots

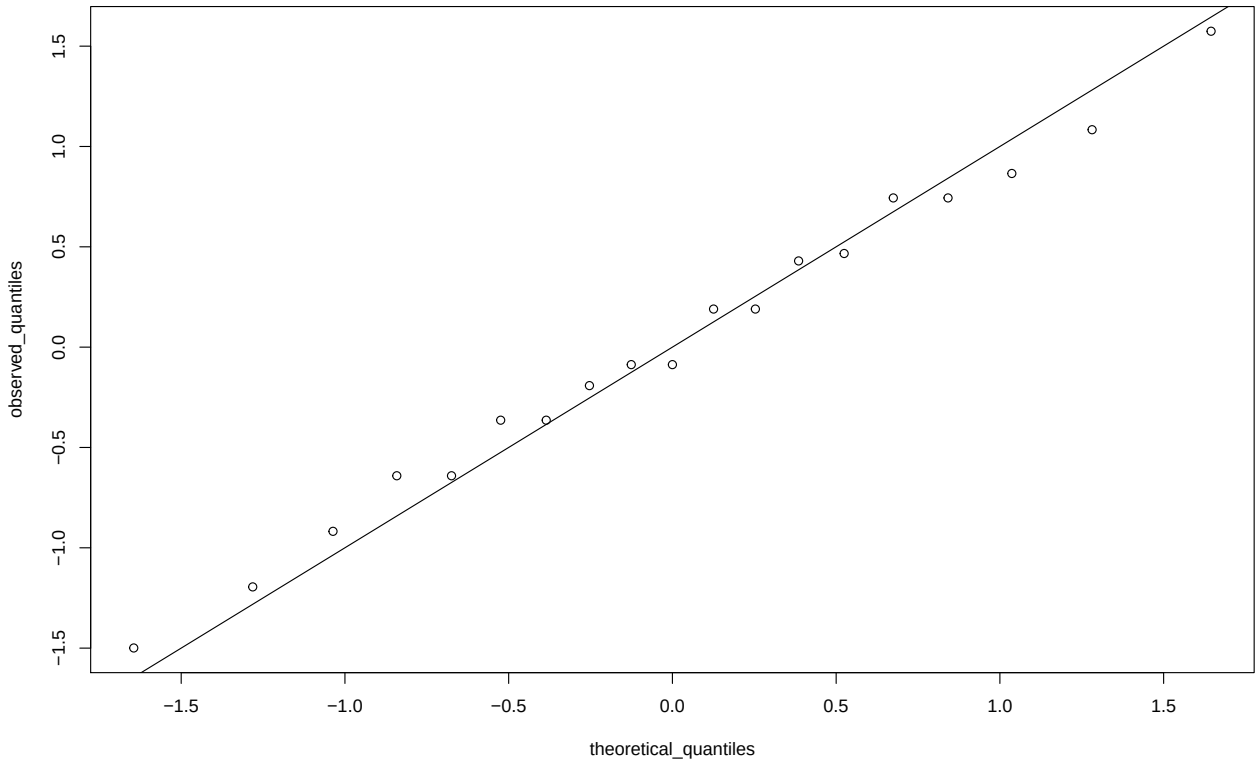
We can use *quantile-quantile plots* to determine whether our data are normal. In this type of plot, we check to see what at what value q a series of proportions p of the data are less than that value q.

```
#As an example, the mean value here represents the proportion p and 69.5 is q
x <- heights$height
mean_x <- mean(x <= 69.5)
#a series of proportions
p <- seq(0.05, 0.95, 0.05)
#create a matrix of proportions and their corresponding q values from the data
observed_quantiles <- quantile(x, p)
#create a matrix of proportions and their corresponding q values from the normal curve
theoretical_quantiles <- qnorm(p, mean = mean(x), sd = sd(x))
#plot the two quantiles against each other
plot(theoretical_quantiles, observed_quantiles)
#plot an identity line to see how closely they match
abline(0,1)
```



Alternatively, one can simplify the code by using standard units.

```
#recall that z represents the normalized values from height (standard units)
observed_quantiles <- quantile(z, p)
#create quantiles from the normal curve with qnorm()
theoretical_quantiles <- qnorm(p)
plot(theoretical_quantiles, observed_quantiles)
abline(0,1)
```



Various -iles

Percentiles are the quantiles you obtain when you set p = 0.01, 0.02, 0.03... The value at which say 21% (p = 0.21) of the data is below is the 21st percentile.

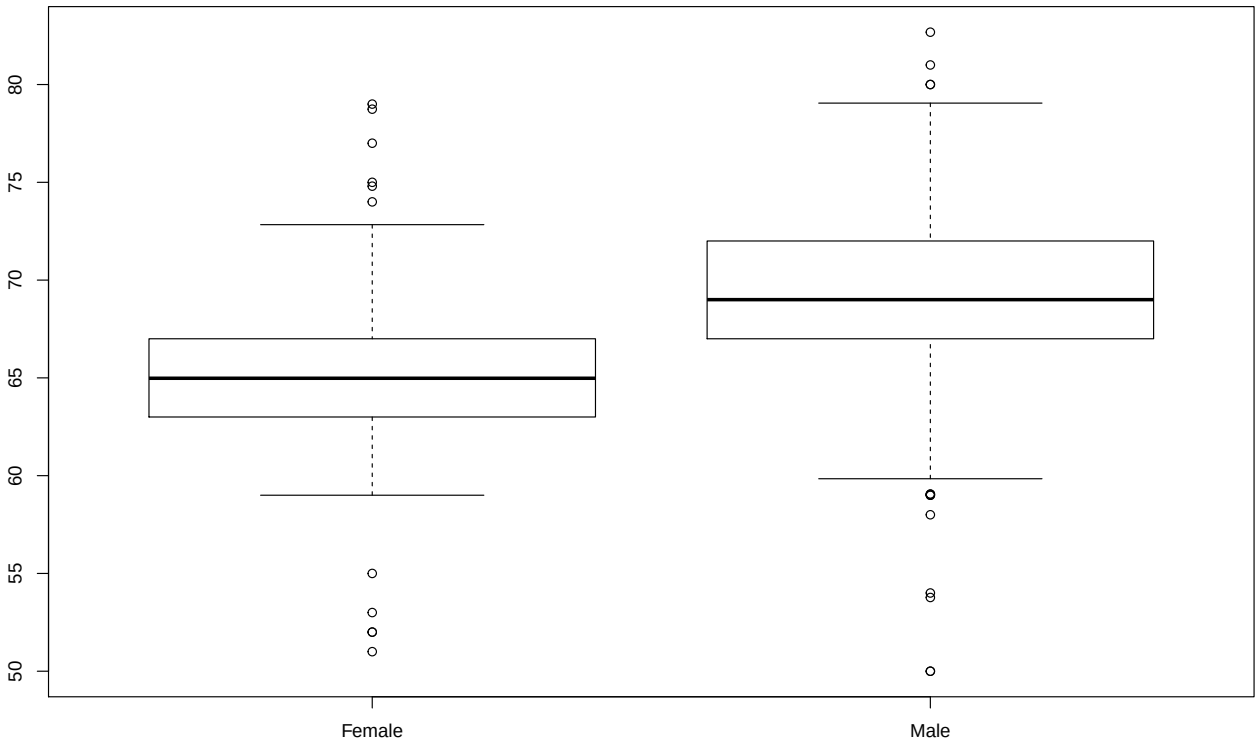
Quartiles are the 25th, 50th, 75th percentiles.

The **median** is the 50th percentile.

Boxplots

In a boxplot, the bottom of the box is located at the **25th percentile**, the top of the box is at the **75th percentile**, and a horizontal line within the box represents the **median**. The range between the **25th** and **75th percentiles** is called the **interquartile range**. **Outliers** are plotted as individual values.

```
boxplot(heights$height~heights$sex)
```



Robust Summaries of Outliers

Means and standard deviations are sensitive to outliers, while medians and median absolute deviations or resistant, or robust, to them.

An easy way to see how a single large error in a data set can effect a mean is to create a function that replaces the first element in a vector with a large value of your choice:

```
x <- heights$height
error_avg <- function(k){
  x[1] <- k
  mean(x)
}
mean(x)
```

```
## [1] 68.32301
```

```
error_avg(10000)
```

```
## [1] 77.77539
```

```
error_avg(-10000)
```

```
## [1] 58.72778
```

Medians and MADs, however, are largely unperturbed:

```
error_med <- function(k){
  x[1] <- k
  median(x)
}
median(x)
```

```
## [1] 68.5
```

```
error_med(10000)
```

```
## [1] 68.5
```

```
error_med(-10000)
```

```
## [1] 68.5
```

The Tidyverse

There are many ways to create plots in R, but the **tidyverse** and its accompanying **ggplot** functions are especially popular because they break down plots into easy to understand components.

```
require(ggplot2)
library(dslabs)
data(murders)
```

ggplot

The *grammer of graphics* is broken down into 4 components: 1. Data - source dataset 2. Geometry - type of plot 3. Aesthetic Mapping - colors to denote categories, text to identify points, etc. 4. Scale - the scaling of the data

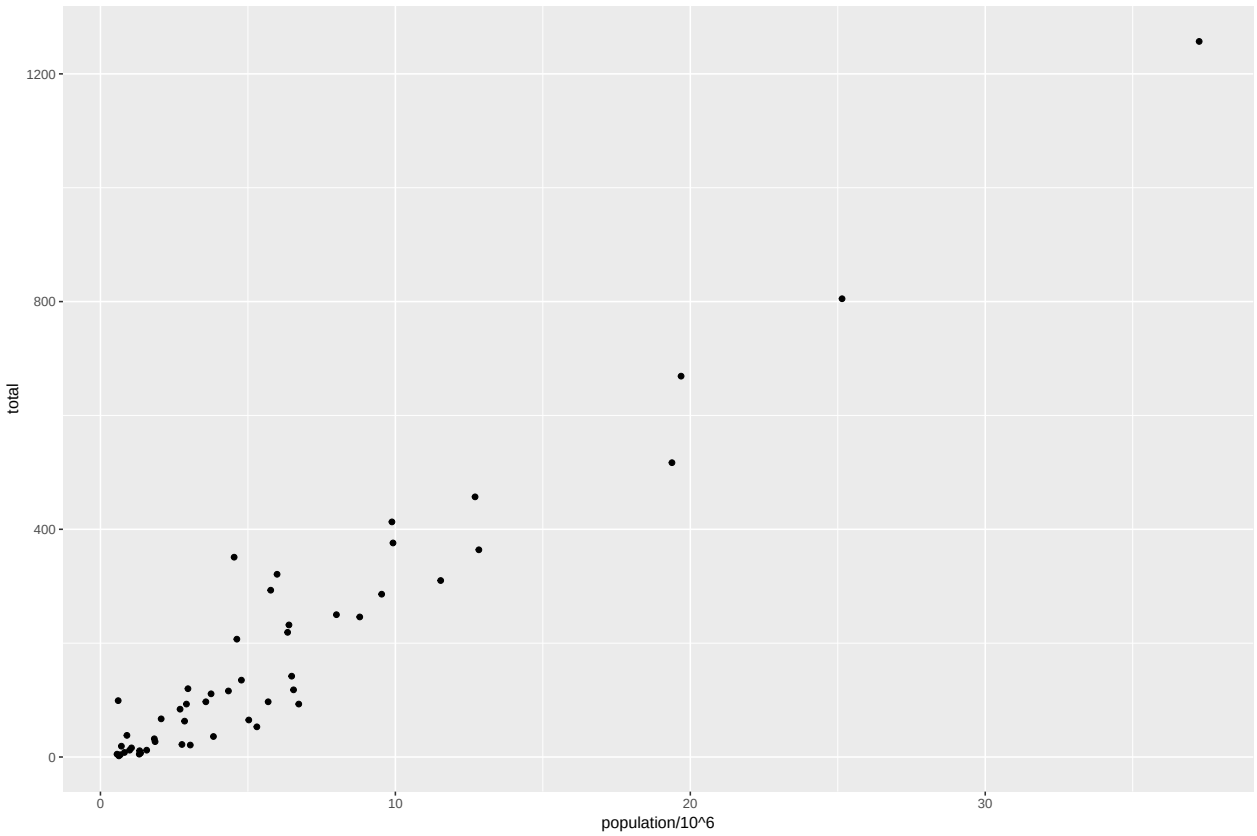
In the pipe: `ggplot(data = murders)` is equivalent to `murders %>% ggplot()`

A General form for ggplot:

```
DATA %>% ggplot() + Layer_1 + Layer_2 + Layer_3 + ... + Layer_N
```

Usually, *Layer_1* defines the geometry. In this case, we use geom_point() to create a scatterplot comparing the population to the total number of murders in each state.

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



We could have dropped the x and y from the above because *aes* expects them as its first two arguments.

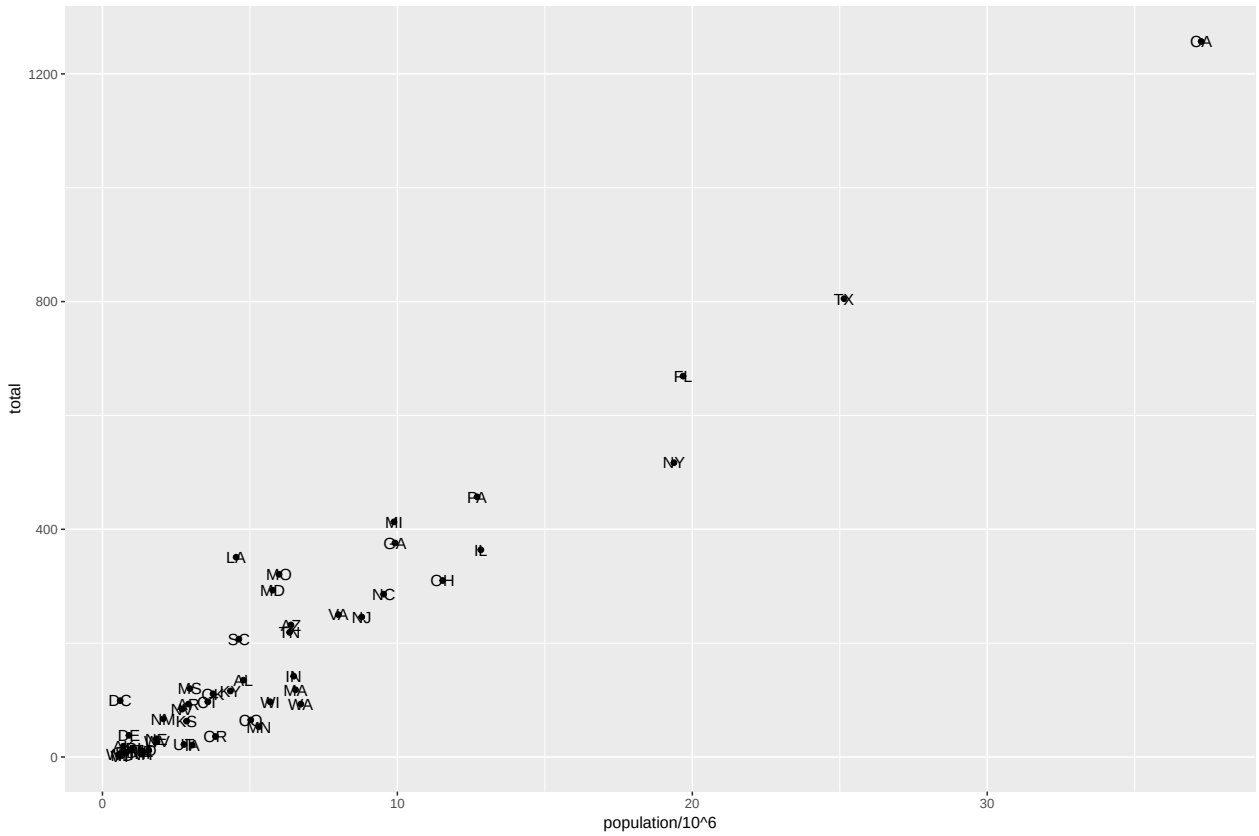
Note also that *aes* knows to look in *murders* for the variables. Generally, functions do not behave this way!

To make things quicker, let's define murders piped into ggplot() as an object:

```
library(magrittr)
m_Plot <- murders %>% ggplot()
```

Next, let's add labels to the points with geom_label().

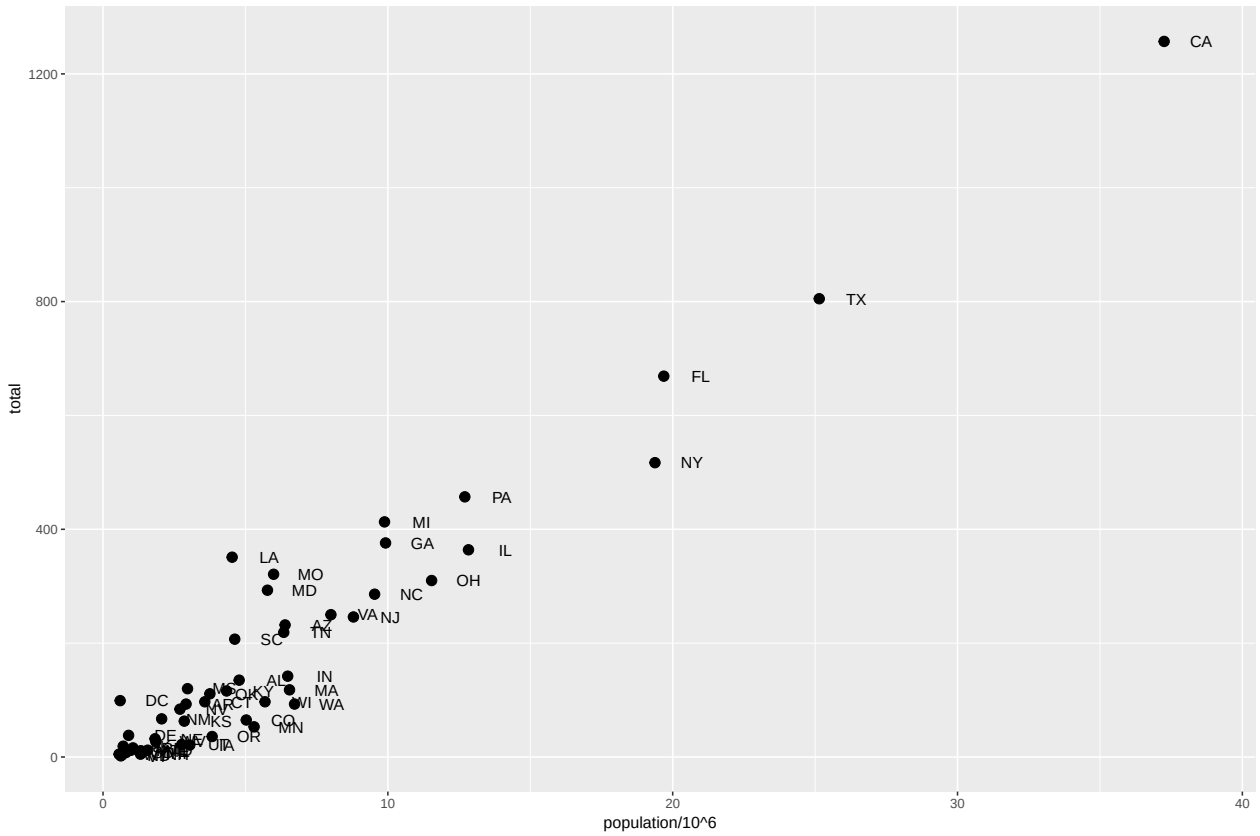
```
m_Plot +
  geom_point(aes(x = population/10^6, y = total)) +
  geom_text(aes(x = population/10^6, total, label = abb))
```



###Tinkering

Using the additional arguments in *geom_point()* and *geom_text()*, we can make this look a lot better.

```
m_Plot +
  geom_point(aes(x = population/10^6, y = total), size = 3) +
  geom_text(aes(x = population/10^6, total, label = abb), nudge_x = 1.3)
```



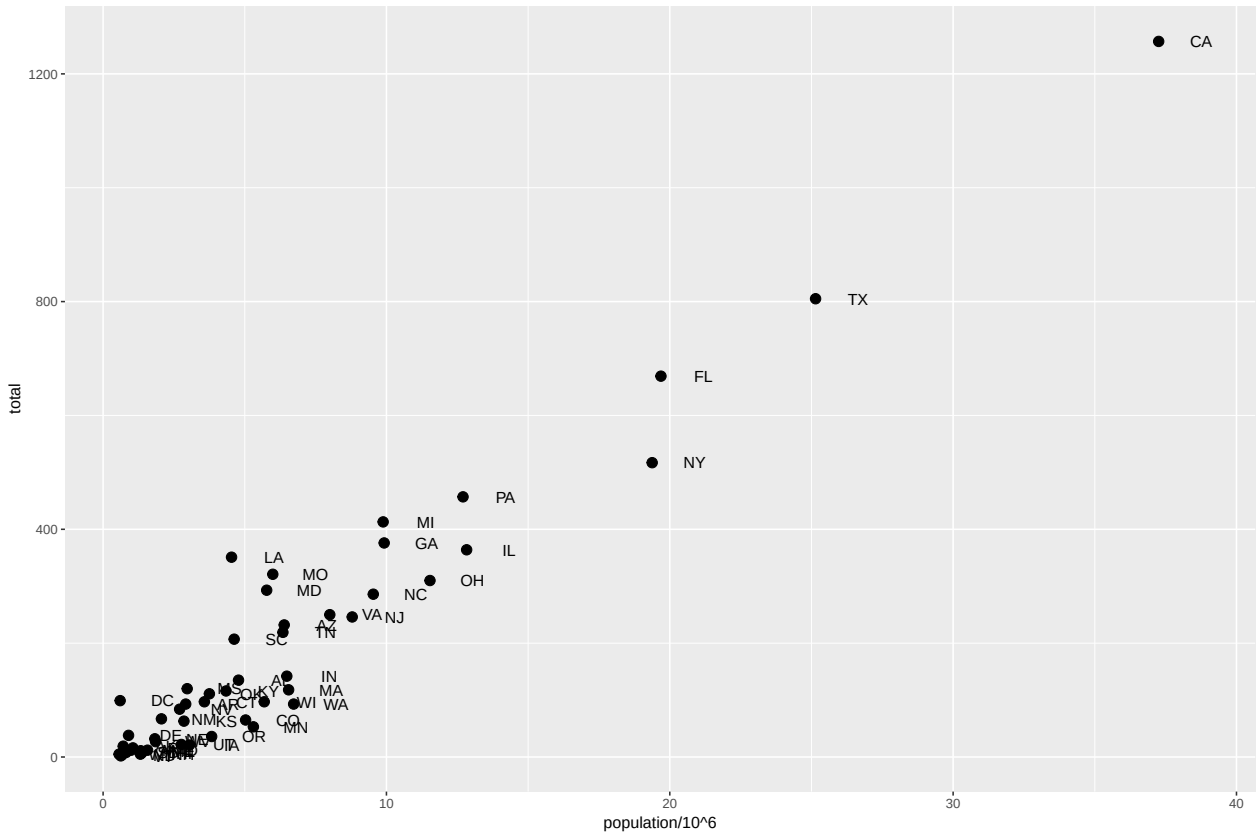
With the *size* argument in *geom_point()* and the *nudge_x* argument in *geom_text()*, we can make the points larger and move the text out of the way so that it doesn't overlap with the points.

We can also make the code more efficient by creating a *global aes* inside the ggplot function:

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

Now, when we add layers, we need only add additional information beyond the global aes:

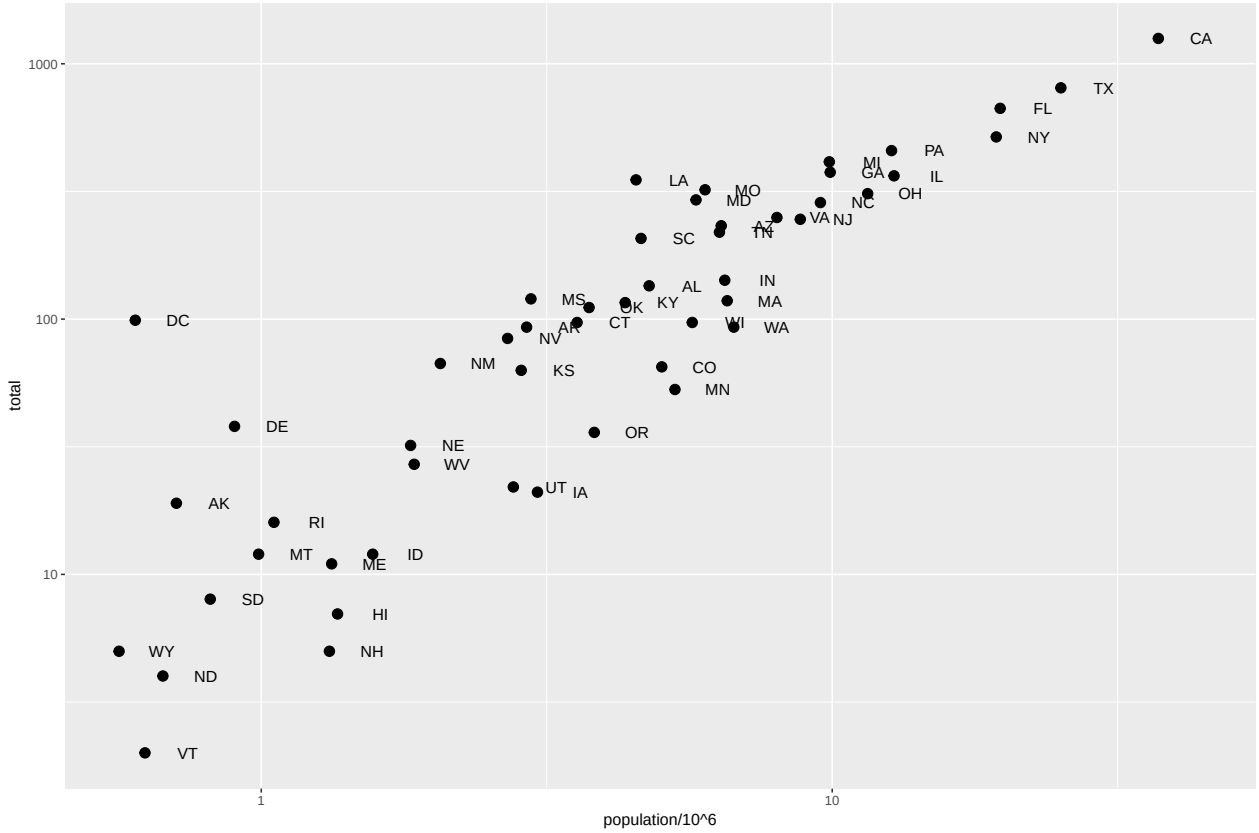
```
p + geom_point(size = 3) + #note that size is specific to geom_point
  geom_text(nudge_x = 1.5) #same for nudge_x in geom_text
```



Scales

To change the scales on the plot, simply use the scale component:

```
p + geom_point(size = 3) +
  geom_text(nudge_x = 0.075) +
  scale_x_continuous(trans = "log10") +
  scale_y_continuous(trans = "log10")
```

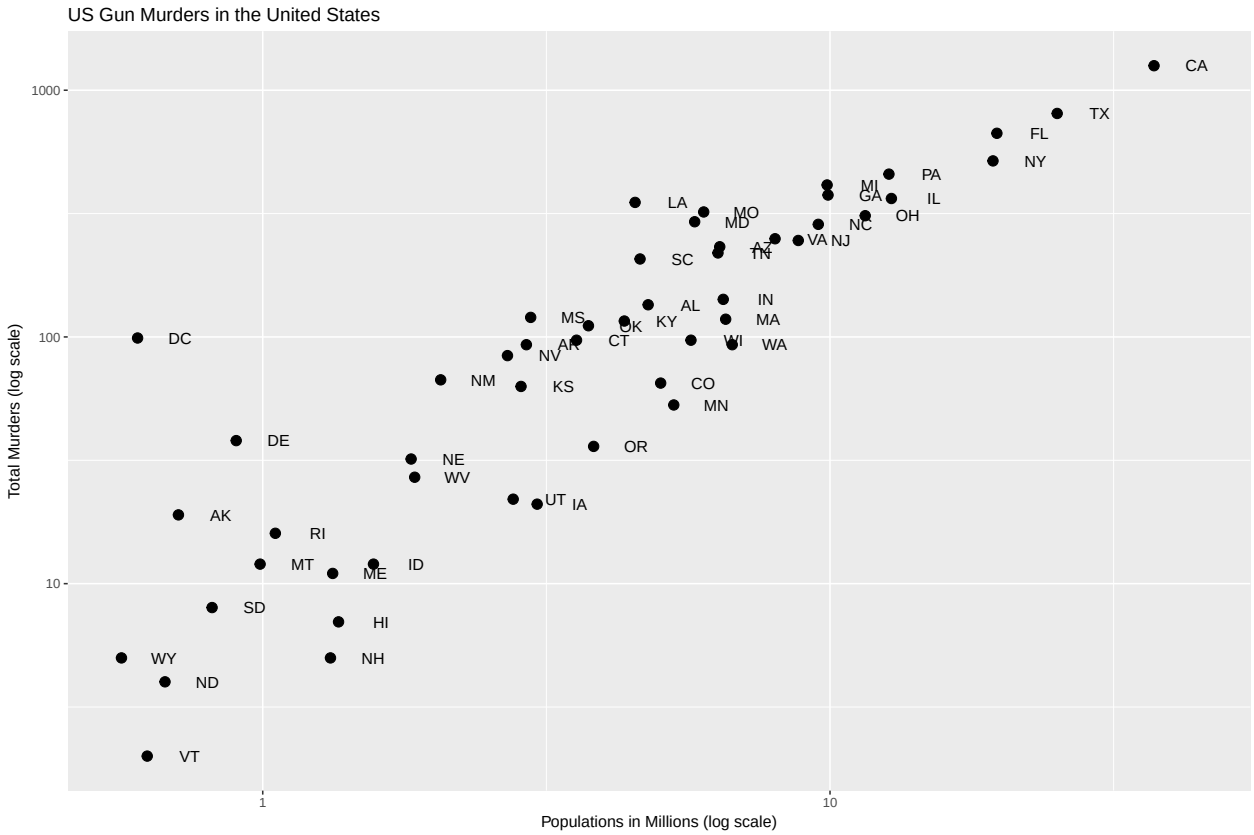


Note that *nudge_x* needed to be reduced in light of the log transformation.

Labels and Titles

Simply add the corresponding layers to the previous code:

```
p + geom_point(size = 3) +
  geom_text(nudge_x = 0.075) +
  scale_x_continuous(trans = "log10") +
  scale_y_continuous(trans = "log10") +
  xlab("Populations in Millions (log scale)") +
  ylab("Total Murders (log scale)") +
  ggtitle("US Gun Murders in the United States")
```



Let's redefine p to be this graph:

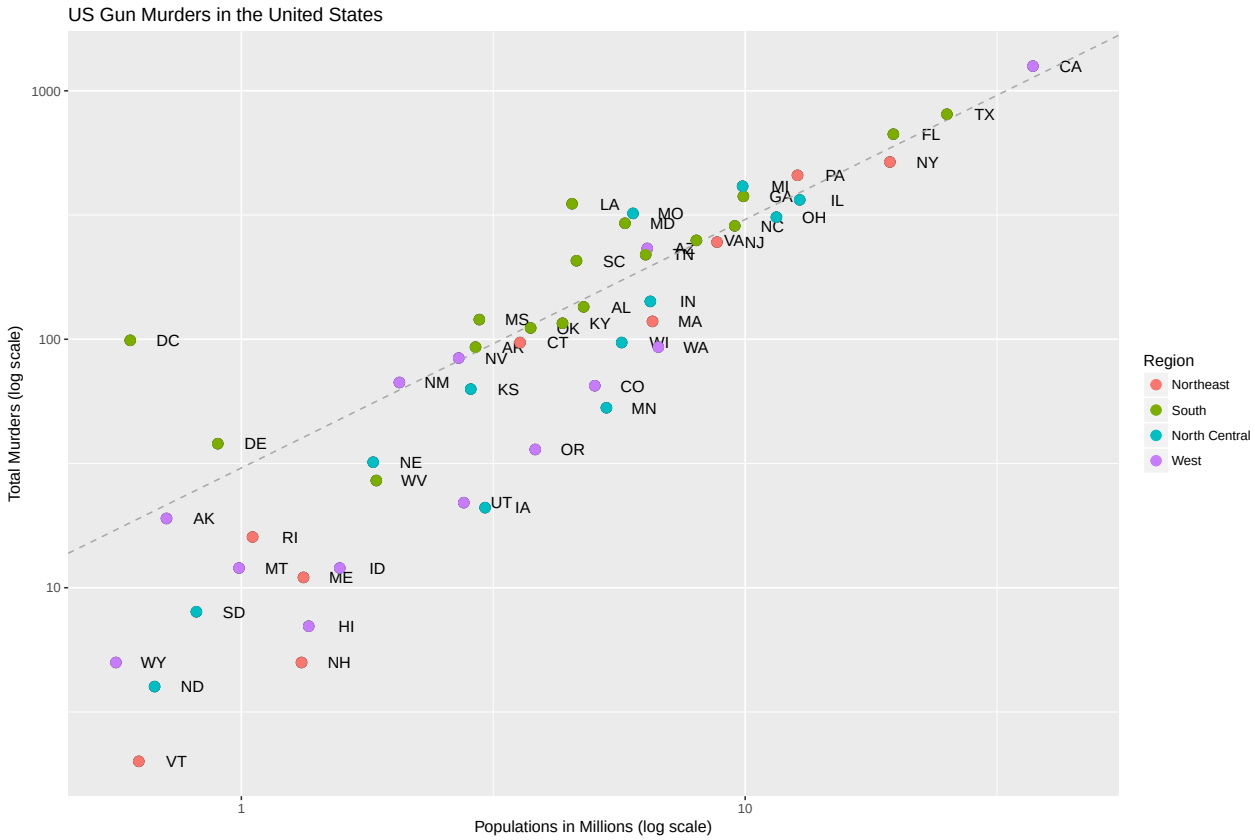
```
p <- murders %>% ggplot(aes(population/10^6, total, label = abb)) +
  geom_point(size = 3) +
  geom_text(nudge_x = 0.075) +
  scale_x_continuous(trans = "log10") +
  scale_y_continuous(trans = "log10") +
  xlab("Populations in Millions (log scale)") +
  ylab("Total Murders (log scale)") +
  ggtitle("US Gun Murders in the United States")
```

Now we can use geom_point to add a different color based on region. Note that we must use aes because this is a mapping that makes reference to the data.

Let's also add a line to show the average murder rate. Summarize() takes the mean of the vector created by um(), and the last pipe coerces the object to become numeric rather than a dataframe.

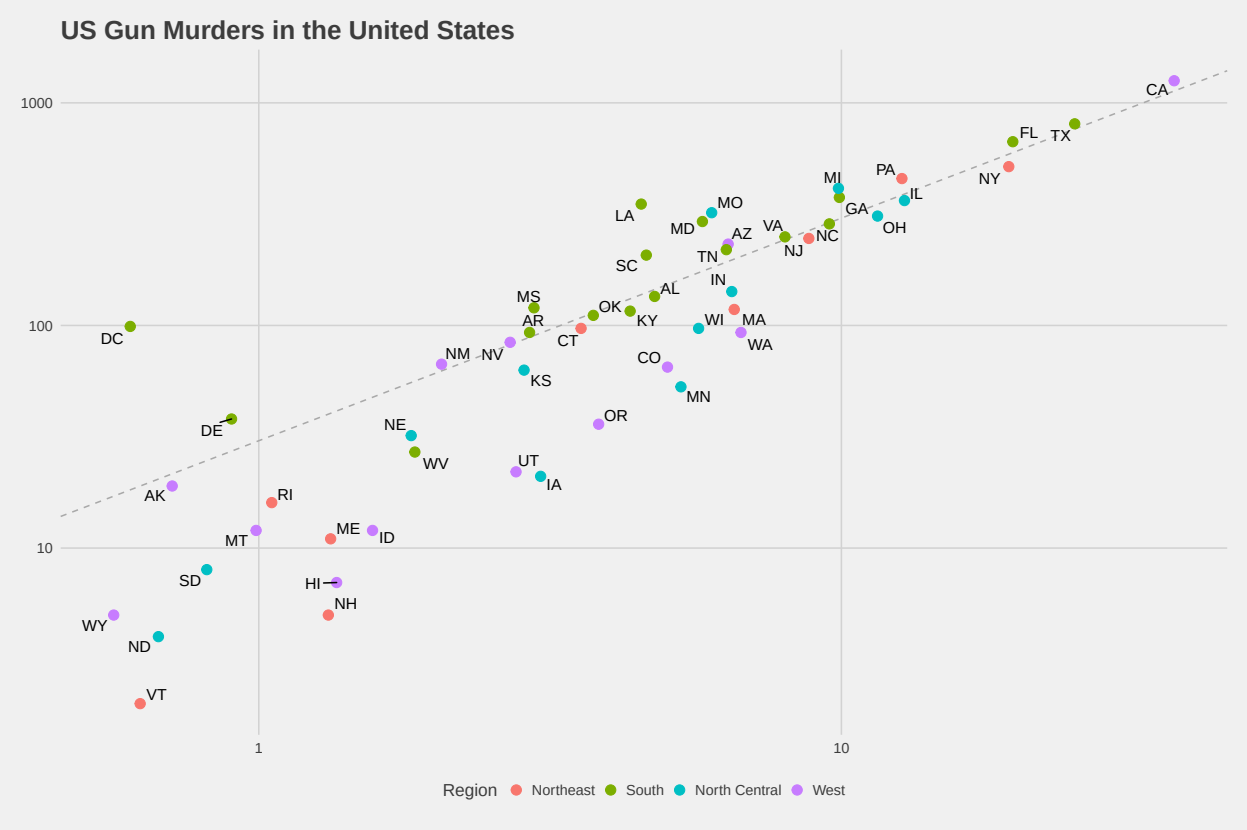
With the average rate, we can create a line with abline().

```
require(dplyr)
r <- murders %>% summarize(rate = sum(total)/ sum(population) * 10^6) %>%
  .$rate
p + geom_abline(intercept = log10(r), lty = 2, color = "darkgrey") +
  geom_point(aes(col = region), size = 3) +
  scale_color_discrete(name = "Region")
```



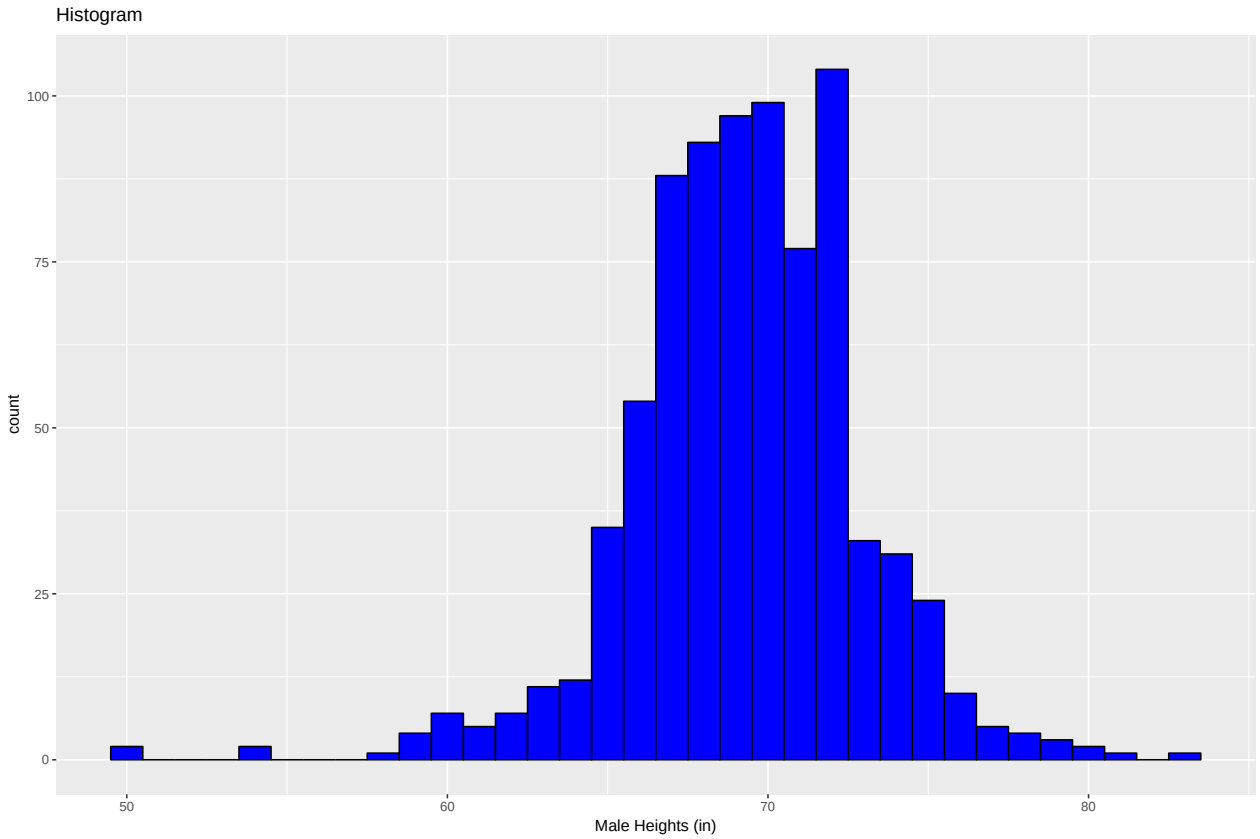
To change the theme, we're going to need to use some add-on packages.

```
require(ggthemes)
require(ggrepel)
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_continuous(trans = "log10") +
  scale_y_continuous(trans = "log10") +
  xlab("Populations in Millions (log scale)") +
  ylab("Total Murders (log scale)") +
  ggtitle("US Gun Murders in the United States") +
  scale_color_discrete(name = "Region") +
  theme_fivethirtyeight()
```



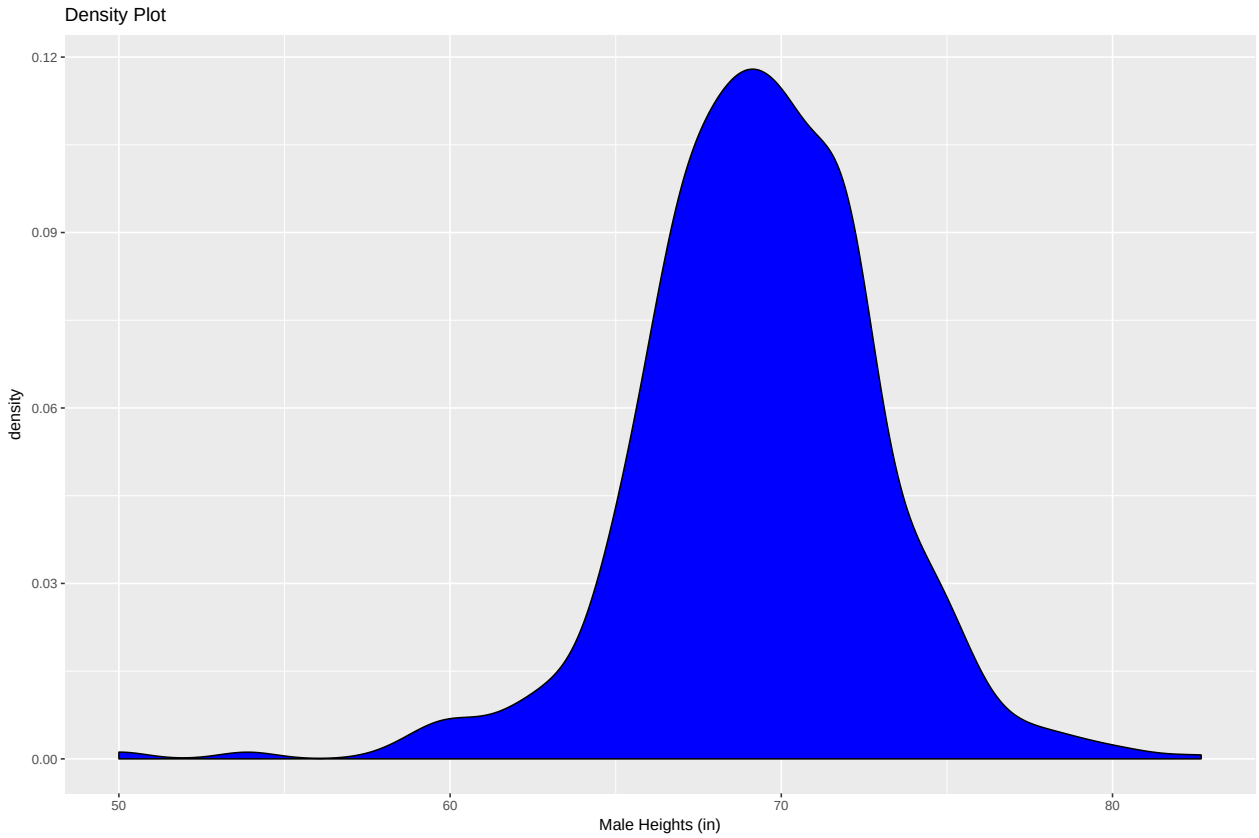
Other Examples

```
require(dslabs)
data("heights")
q <- heights %>% filter(sex=="Male") %>%
  ggplot(aes(x = height))
q + geom_histogram(binwidth = 1, fill = "blue", col = "black") +
  xlab("Male Heights (in)") +
  ggtitle("Histogram")
```



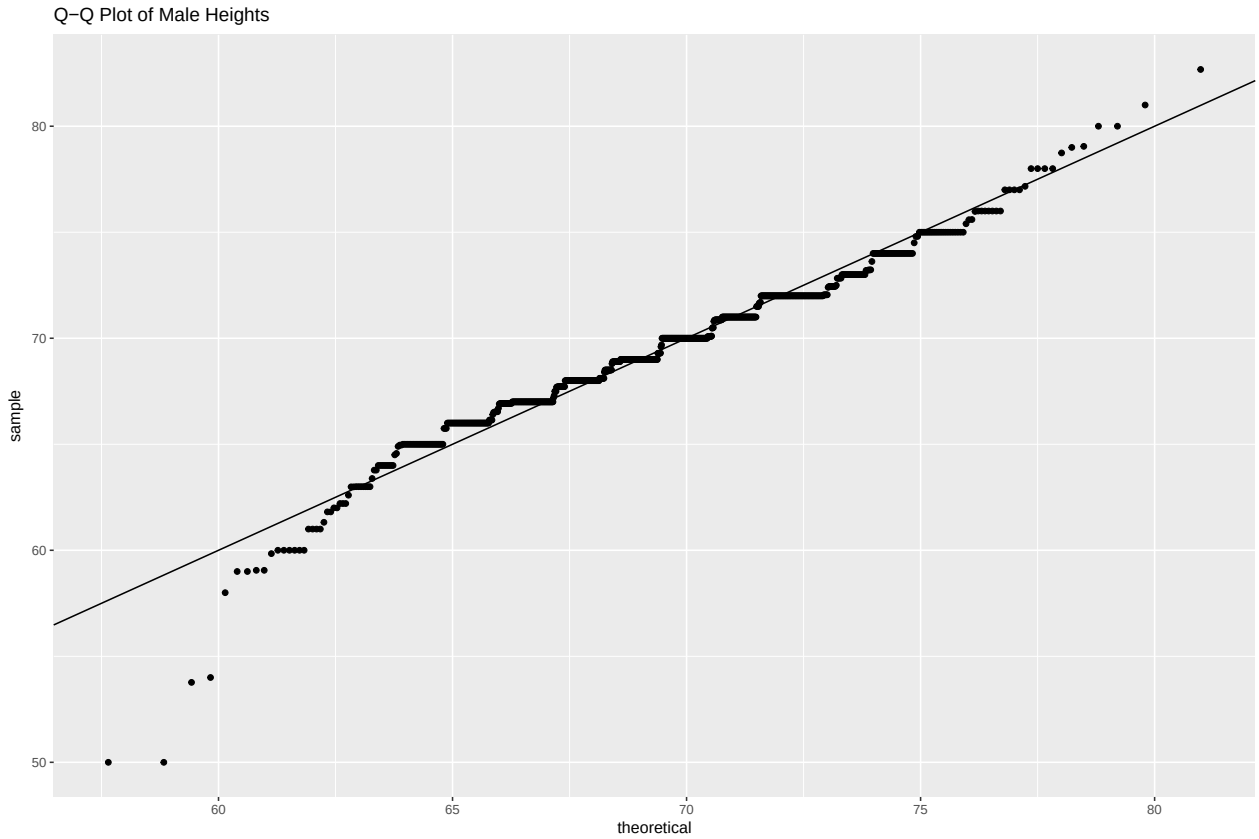
We can make a density plot instead with geom_density().

```
q + geom_density(fill = "blue") +
  xlab("Male Heights (in)") +
  ggtitle("Density Plot")
```



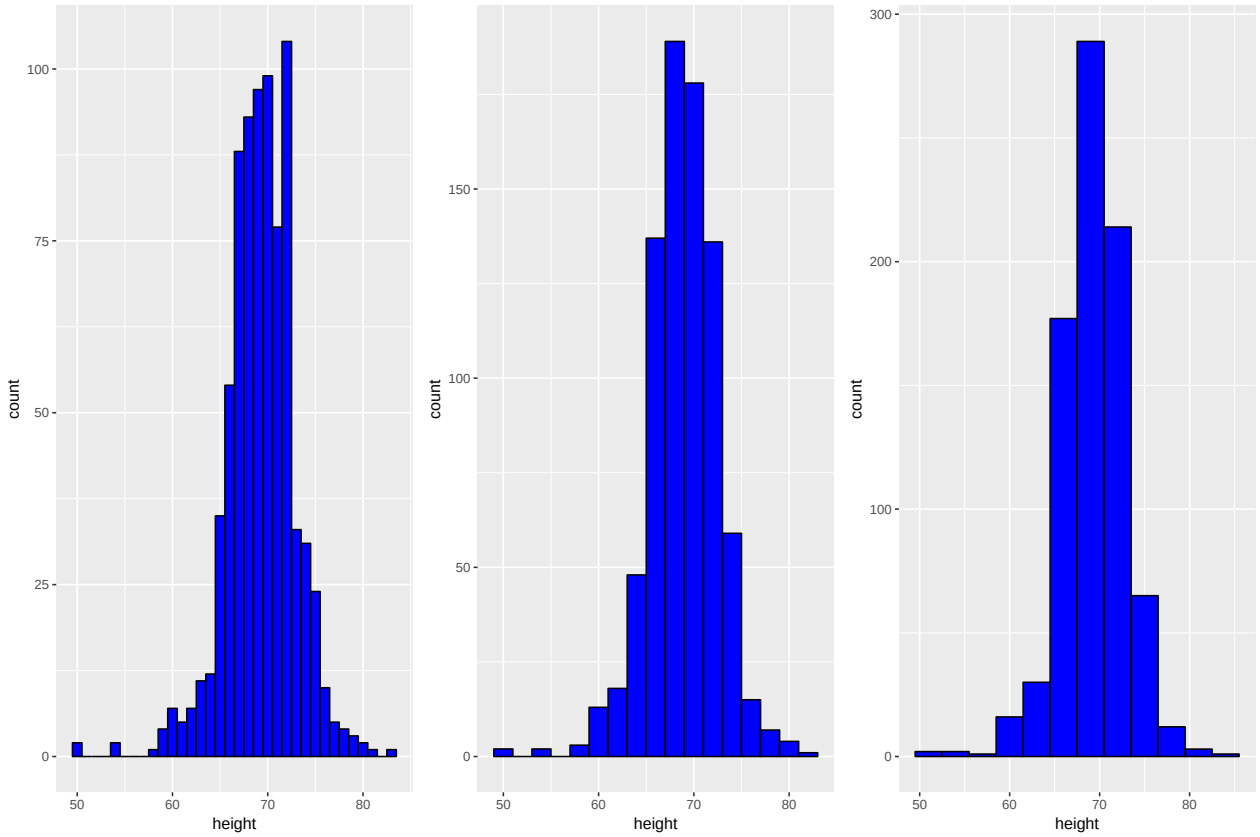
To make a qq plot, we must change the definition of q so that qq can find *samples*. By default the qq plot is made by comparing to a theoretical normal distribution with an average of **0** and standard deviation of **1**. To compare to normal with the same parameter, we must define the *dparams* argument of *geom_qq()*. Alternatively, one could *scale()* the heights in the *aes* argument of *ggplot* to put the data in standard units. This is a bit cleaner.

```
params <- heights %>% filter(sex == "Male") %>%
  summarize(mean = mean(height), sd = sd(height))
qq <- heights %>% filter(sex == "Male") %>%
  ggplot(aes(sample = height))
qqplot <- qq + geom_qq(dparams = params) +
  ggtitle("Q-Q Plot of Male Heights") + geom_abline()
#alternative, cleaner method
qq_2 <- heights %>% filter(sex == "Male") %>%
  ggplot(aes(sample = scale(height)))
qq_scale <- qq_2 + geom_qq() + geom_abline() +
  ggtitle("Q-Q Plot of Male Heights")
qqplot
```



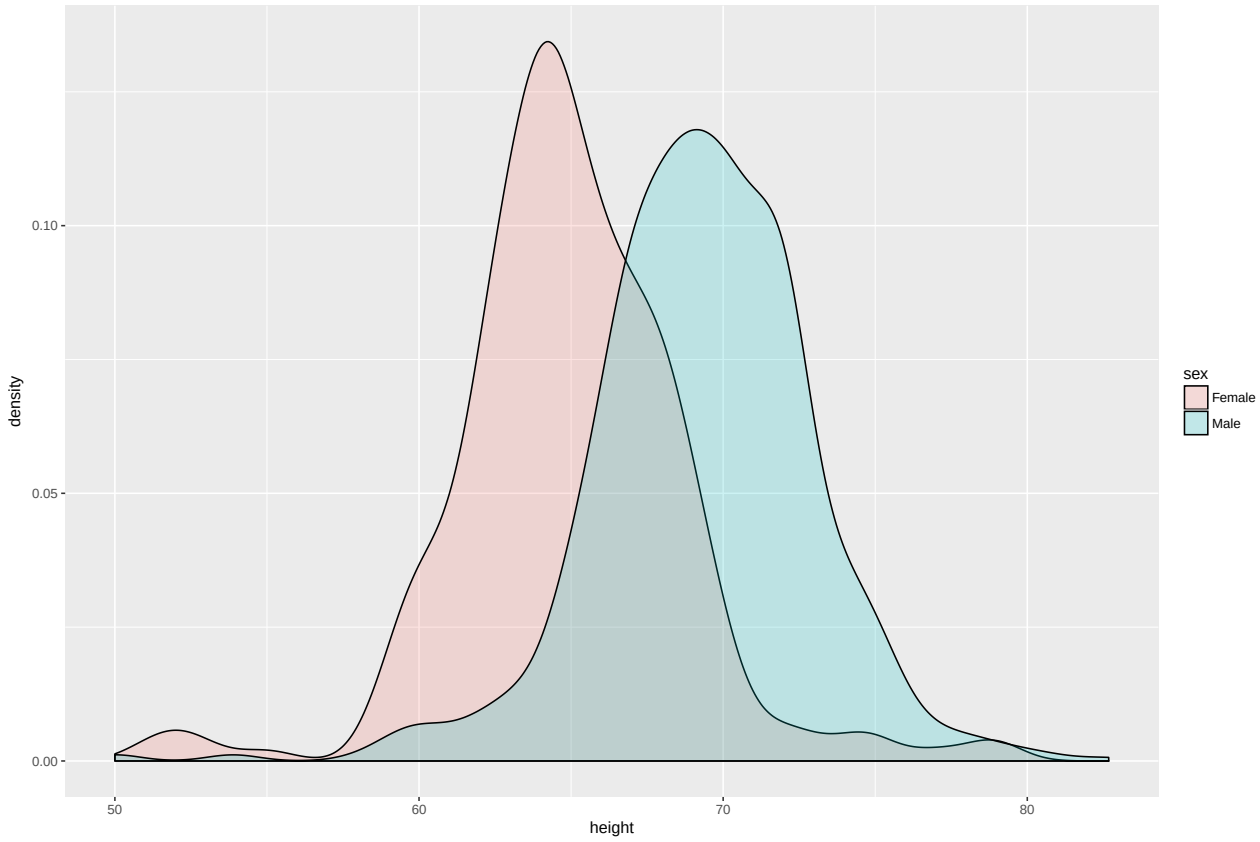
To arrange plots next to each other, use the gridExtra package.

```
q <- heights %>% filter(sex=="Male") %>%
  ggplot(aes(x = height))
p1 <- q + geom_histogram(binwidth = 1, fill = "blue", col = "black")
p2 <- q + geom_histogram(binwidth = 2, fill = "blue", col = "black")
p3 <- q + geom_histogram(binwidth = 3, fill = "blue", col = "black")
library(gridExtra)
grid.arrange(p1,p2,p3, ncol = 3)
```



When comparing two densities with different fill colors, we can change the alpha blending so that the both of the density curves are visible where they overlap.

```
heights %>% ggplot(aes(x = height, fill = sex)) +
  geom_density(alpha = 0.2)
```



Dplyr

The package **Dply** is very useful for wrangling data, be that by filtering, subsetting, or generating a variety of summary statistics.

Summarize

We'll use the *heights* and *murders* datasets from the *dslabs* package for the following examples.

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

Let's compute the average and the standard deviation for the males in the heights dataset. Note that the output of `summarize()` is a dataframe.

```
s<- heights %>% filter(sex == "Male") %>%
  summarize(average = mean(height),
            standard_deviation = sd(height))
s
```

```
##    average standard_deviation
## 1 69.31475           3.611024
```

Note that since *s* is a data frame, we can access its components with the $ accessor.

```
s$average
```

```
## [1] 69.31475
```

We can also use `summarize()` to generate any other summary statistic that returns a single value from on operation on a vector.

```
heights %>% filter(sex == "Male") %>%
  summarize(median = median(height),
            minimum = min(height),
            maximum = max(height))
```

```
##   median minimum  maximum
## 1     69      50 82.67717
```

Note that we could also generate these same values using `quantile(c(0,0,5,1))` in base R. However, if we tried to use `quantile()` inside of `summarize()`, we would get an error, since `summarize()` can only handle functions that return a single value.

The Dot Placeholder

Let's say we want to calculate the average US murder rate per 100,000 people. We can do this with `summarize()`.

```
us_murder_rate <- murders %>%
  summarize(rate = sum(total)/sum(population)*100000)
us_murder_rate
```

```
##       rate
## 1 3.034555
```

The object `us_murder_rate` has class `class(us_murder_rate)`. This can cause problems down the road if we want to use enter `us_murder_rate` into a function that requires a *numeric* data type.

The **dot placeholder** is a useful way to define a numeric object at the end of a pipe.

```
us_murder_rate <- murders %>%
  summarize(rate = sum(total) / sum(population) * 100000) %>% .$rate
us_murder_rate
```

```
## [1] 3.034555
```

Now, `us_murder_rate` is of class `class(us_murder_rate)`. Later, we will see other instances where the dot is useful, but for now, we'll only use it to produce numeric vectors from pipelines constructed with `dplyr`.

Group By

Another useful **dplyr** function is `group_by`, which groups data by a certain category, allowing us to quickly parse the data based on those groups.

```
head(heights %>% group_by(sex))
```

```
## # A tibble: 6 x 2
## # Groups:   sex [2]
##   sex    height
##   <fct>   <dbl>
## 1 Male       75
## 2 Male       70
## 3 Male       68
## 4 Male       74
```

```
## 5 Male       61
## 6 Female     65
```

This creates a special data frame known as a *group data frame*. Dplyr functions such as `summarize()` will consider the group when computing summary statistics for these objects.

```
heights %>%
  group_by(sex) %>%
  summarize(average = mean(height), standard_deviation = sd(height))
```

```
## # A tibble: 2 x 3
##   sex    average standard_deviation
##   <fct>    <dbl>              <dbl>
## 1 Female    64.9               3.76
## 2 Male      69.3               3.61
```

The resulting table computes the average and standard deviation for females and males separately.

Sorting Data Tables

While `order()` and `sort()` are useful functions for reordering vectors, `arrange()` makes sorting entire data tables easy.

To sort the murders dataset by population, all we have to do is pipe it into `arrange()`.

```
murders %>% arrange(population) %>% head()
```

```
##                  state abb        region population total
## 1              Wyoming  WY          West     563626     5
## 2 District of Columbia  DC         South     601723    99
## 3              Vermont  VT     Northeast     625741     2
## 4         North Dakota  ND North Central     672591     4
## 5               Alaska  AK          West     710231    19
## 6         South Dakota  SD North Central     814180     8
```

If we wanted to arrange by the murder rate, let's first add a column to **murders**, and then arrange by rate.

```
murders <- murders %>%
  mutate(rate = total/population * 100000)
murders %>% arrange(rate) %>% head()
```

```
##           state abb        region population total      rate
## 1       Vermont  VT     Northeast     625741     2 0.3196211
## 2 New Hampshire  NH     Northeast    1316470     5 0.3798036
## 3        Hawaii  HI          West    1360301     7 0.5145920
## 4  North Dakota  ND North Central     672591     4 0.5947151
## 5          Iowa  IA North Central    3046355    21 0.6893484
## 6         Idaho  ID          West    1567582    12 0.7655102
```

Using the **dplr** function `desc()`, we can sort in descending order.

```
murders %>% arrange(desc(rate)) %>% head()
```

```
##                  state abb        region population total      rate
## 1 District of Columbia  DC         South     601723    99 16.452753
## 2            Louisiana  LA         South    4533372   351  7.742581
## 3             Missouri  MO North Central    5988927   321  5.359892
## 4             Maryland  MD         South    5773552   293  5.074866
## 5       South Carolina  SC         South    4625364   207  4.475323
```

```
## 6             Delaware  DE         South     897934    38  4.231937
```

To do *nested sorting*, we can use a second argument in `arrange()`.

```
murders %>% arrange(region,desc(rate)) %>% head(20)
```

```
##                  state abb    region population total       rate
## 1         Pennsylvania  PA Northeast   12702379   457  3.5977513
## 2           New Jersey  NJ Northeast    8791894   246  2.7980319
## 3          Connecticut  CT Northeast    3574097    97  2.7139722
## 4             New York  NY Northeast   19378102   517  2.6679599
## 5        Massachusetts  MA Northeast    6547629   118  1.8021791
## 6         Rhode Island  RI Northeast    1052567    16  1.5200933
## 7                Maine  ME Northeast    1328361    11  0.8280881
## 8        New Hampshire  NH Northeast    1316470     5  0.3798036
## 9              Vermont  VT Northeast     625741     2  0.3196211
## 10 District of Columbia  DC     South     601723    99 16.4527532
## 11            Louisiana  LA     South    4533372   351  7.7425810
## 12             Maryland  MD     South    5773552   293  5.0748655
## 13       South Carolina  SC     South    4625364   207  4.4753235
## 14             Delaware  DE     South     897934    38  4.2319369
## 15          Mississippi  MS     South    2967297   120  4.0440846
## 16              Georgia  GA     South    9920000   376  3.7903226
## 17            Tennessee  TN     South    6346105   219  3.4509357
## 18              Florida  FL     South   19687653   669  3.3980688
## 19                Texas  TX     South   25145561   805  3.2013603
## 20             Arkansas  AR     South    2915918    93  3.1893901
```

We've been using `head()` to show the first 6 rows of the tables above, but we can also use another dplyr function, `top_n`, to show the first `n` rows. To see the top ten highest murder rates, we first arrange by murder rate, and then show the top 10 rows.

```
murders %>% arrange(desc(rate)) %>% top_n(10)
```

```
##                  state abb        region population total      rate
## 1  District of Columbia  DC         South     601723    99 16.452753
## 2             Louisiana  LA         South    4533372   351  7.742581
## 3              Missouri  MO North Central    5988927   321  5.359892
## 4              Maryland  MD         South    5773552   293  5.074866
## 5        South Carolina  SC         South    4625364   207  4.475323
## 6              Delaware  DE         South     897934    38  4.231937
## 7              Michigan  MI North Central    9883640   413  4.178622
## 8           Mississippi  MS         South    2967297   120  4.044085
## 9               Georgia  GA         South    9920000   376  3.790323
## 10              Arizona  AZ          West    6392017   232  3.629527
```

Gapminder Dataset

The `Gapminder` dataset is composed of spreadsheets from Hans Rosling's foundation, and contains various statistics countries around the world.

Let's say we want to know which of two countries, say Sri Lanka and Turkey, has the greater infant mortality rate. Here's some `dply` code to find out.

```
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
```

Guess which of the following pairs of countries have the higher infant mortality rate.

Malaysia vs. South Korea Pakistan vs. Russia Poland vs. South Africa Sri Lanka vs. Turkey Thailand vs. Vietnam

The results might surprise you.

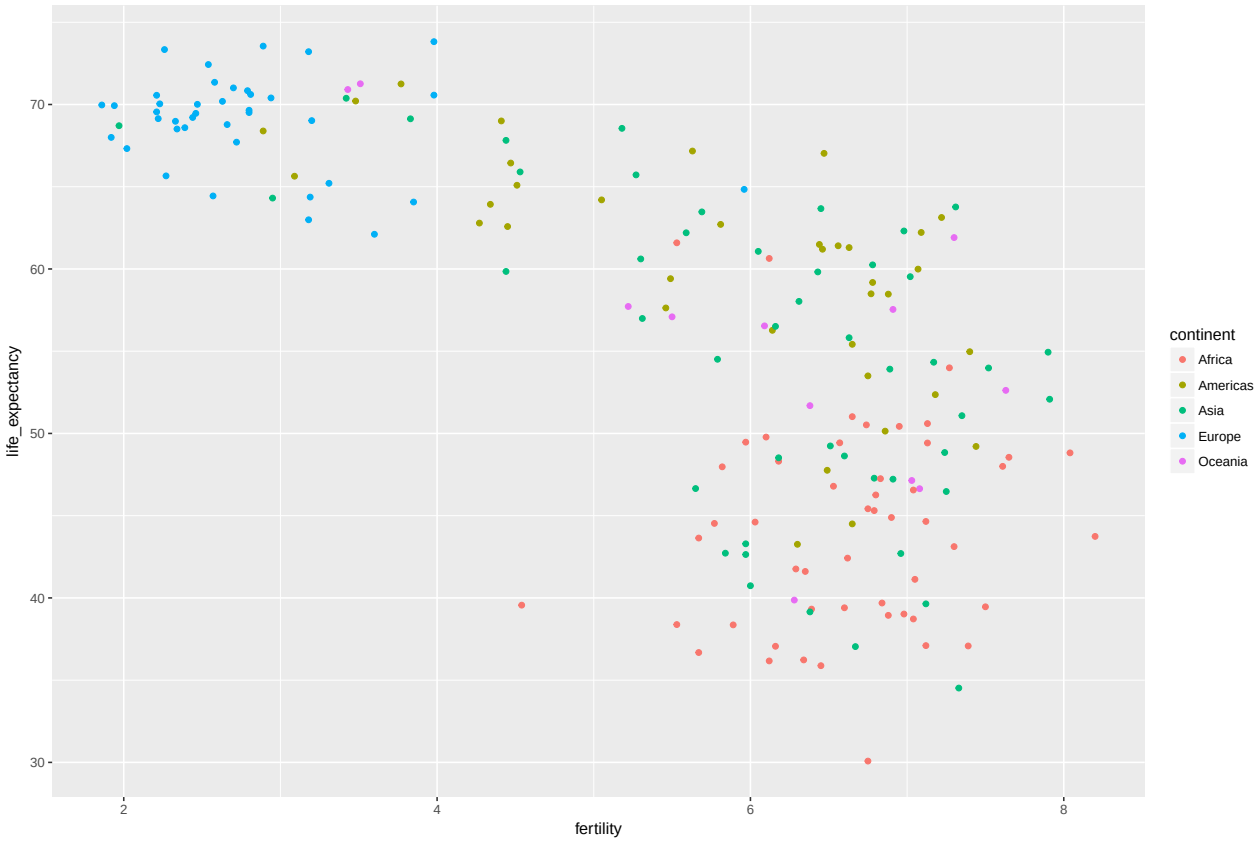
```
col_a <- gapminder %>%
  filter(year == 2015 & country %in% c("Sri Lanka", "Poland", "Malaysia", "Pakistan", "Thailand")) %>%
  select(country, infant_mortality)
col_b <- gapminder %>%
  filter(year == 2015 & country %in%
      c("Turkey", "South Korea", "Russia", "Vietnam", "South Africa")) %>%
  select(country, infant_mortality)
table_1 <- data_frame(Country_A = col_a$country, Infant_Mortality_A = col_a$infant_mortality, Country_B
table_1
```

```
## # A tibble: 5 x 4
##   Country_A Infant_Mortality_A Country_B    Infant_Mortality_B
##   <fct>                  <dbl> <fct>                     <dbl>
## 1 Malaysia                 6   South Korea                 2.9
## 2 Pakistan                65.8 Russia                      8.2
## 3 Poland                   4.5 South Africa               33.6
## 4 Sri Lanka                8.4 Turkey                     11.6
## 5 Thailand                10.5 Vietnam                    17.3
```

We often here the world divided into the West (North America and Western Europe) and the rest (Asia, Africa, Latin America). Western countries have small families and long life spans, while people living elsewhere have large families and shorter lifespans. But is this the case today?

Let's compare fertility and lifespan ~50 years ago to see where this intuition comes from.

```
filter(gapminder, year == 1962) %>%
  ggplot(aes(fertility, life_expectancy, color = continent)) +
  geom_point()
```

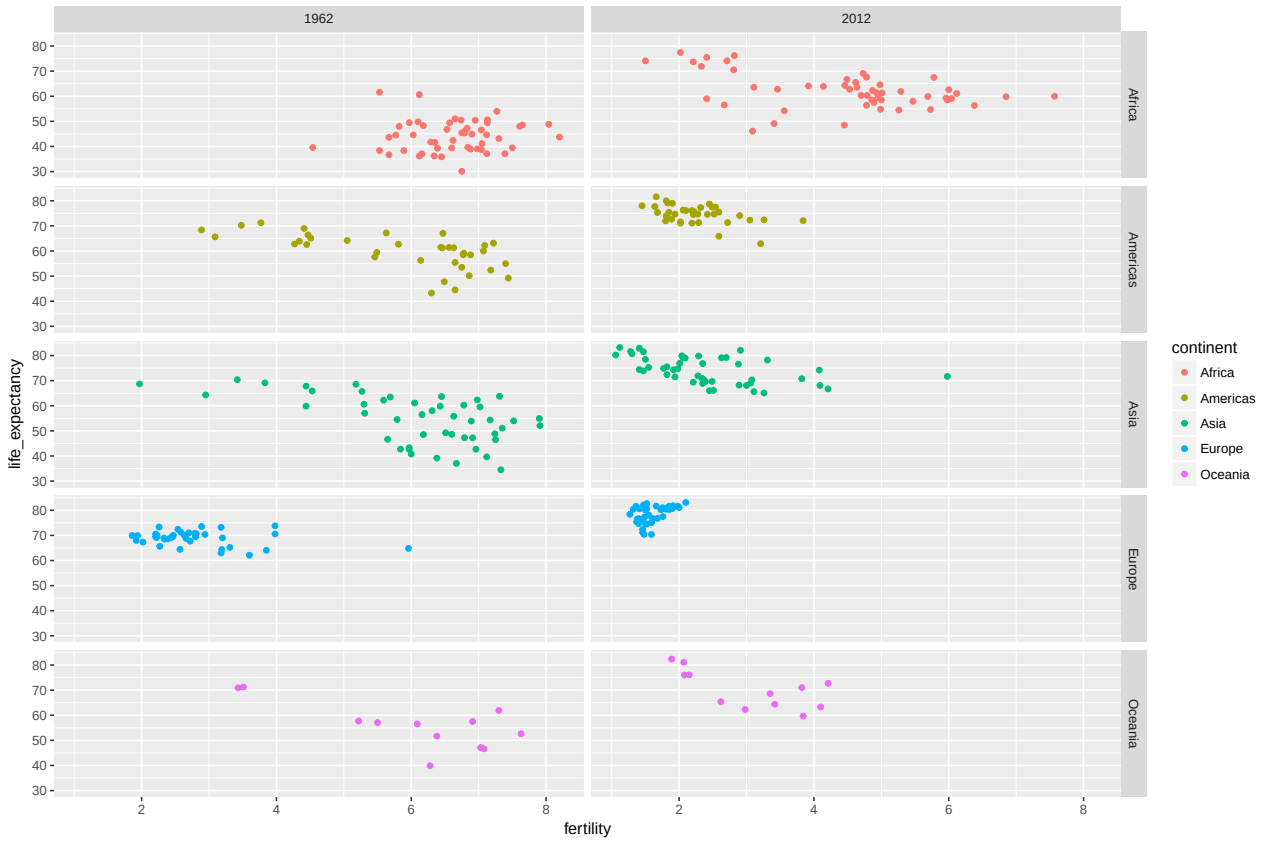


Note how there does seem to be two major groups. One in the top left corner composed of countries with a high life expectancy, and the other in the bottom right with shorter lifespans and more children. The first group is largely made up of European countries while the latter is primarily African and Asian.

To see how this has changed over the last ~50 years, we can use the `facet_grid()` function to juxtapose scatterplots for each of the intervening years.

`facet_grid` can take up to 2 arguments which will become the columns and rows of the grid.

```
filter(gapminder, year %in% c(1962, 2012)) %>%
  ggplot(aes(fertility, life_expectancy, color = continent)) +
  geom_point() +
  facet_grid(continent~year)
```

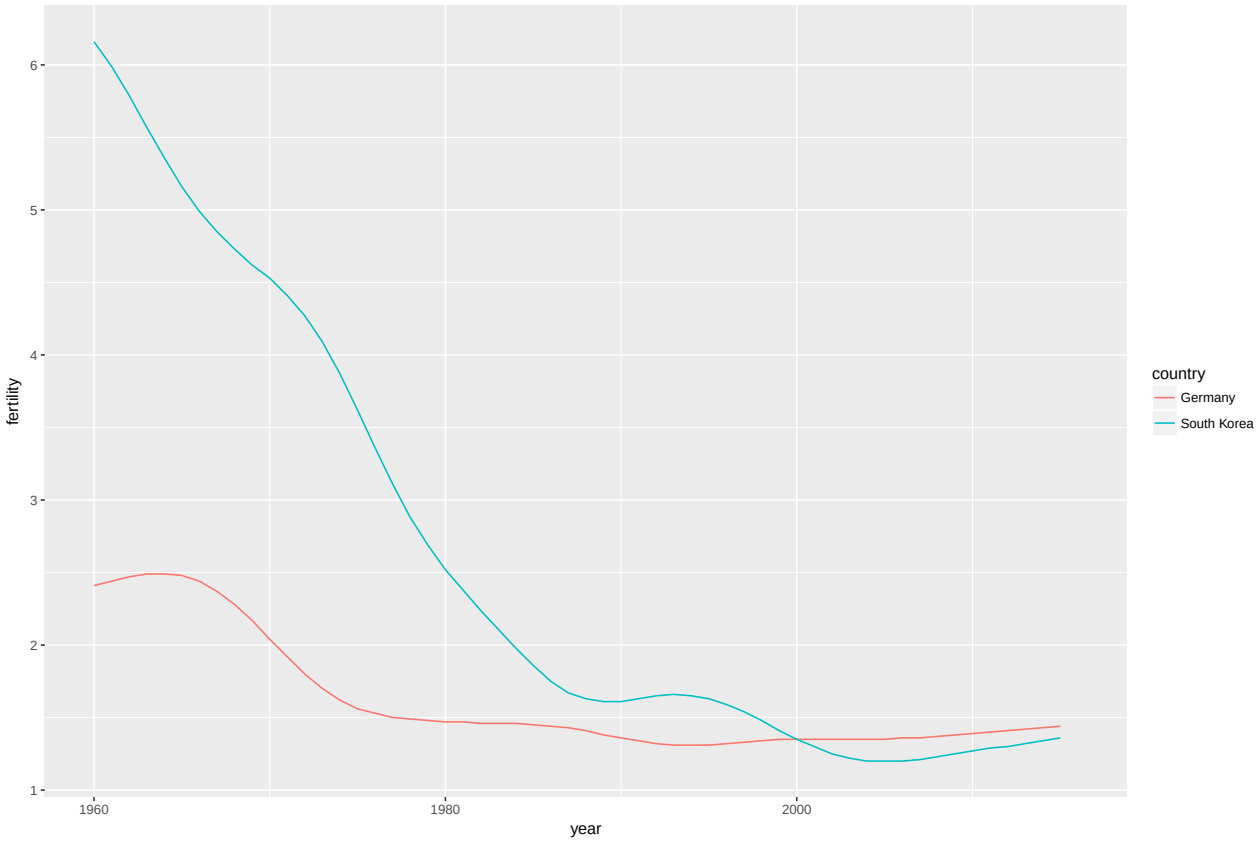


To use `facet_grid()` with only one variable (generating one column), use a `.` a placeholder followed by a `~`.

Time Series Plots

With time series plots, we can plot different statistics over time, and get a sense of how individual countries have improved.

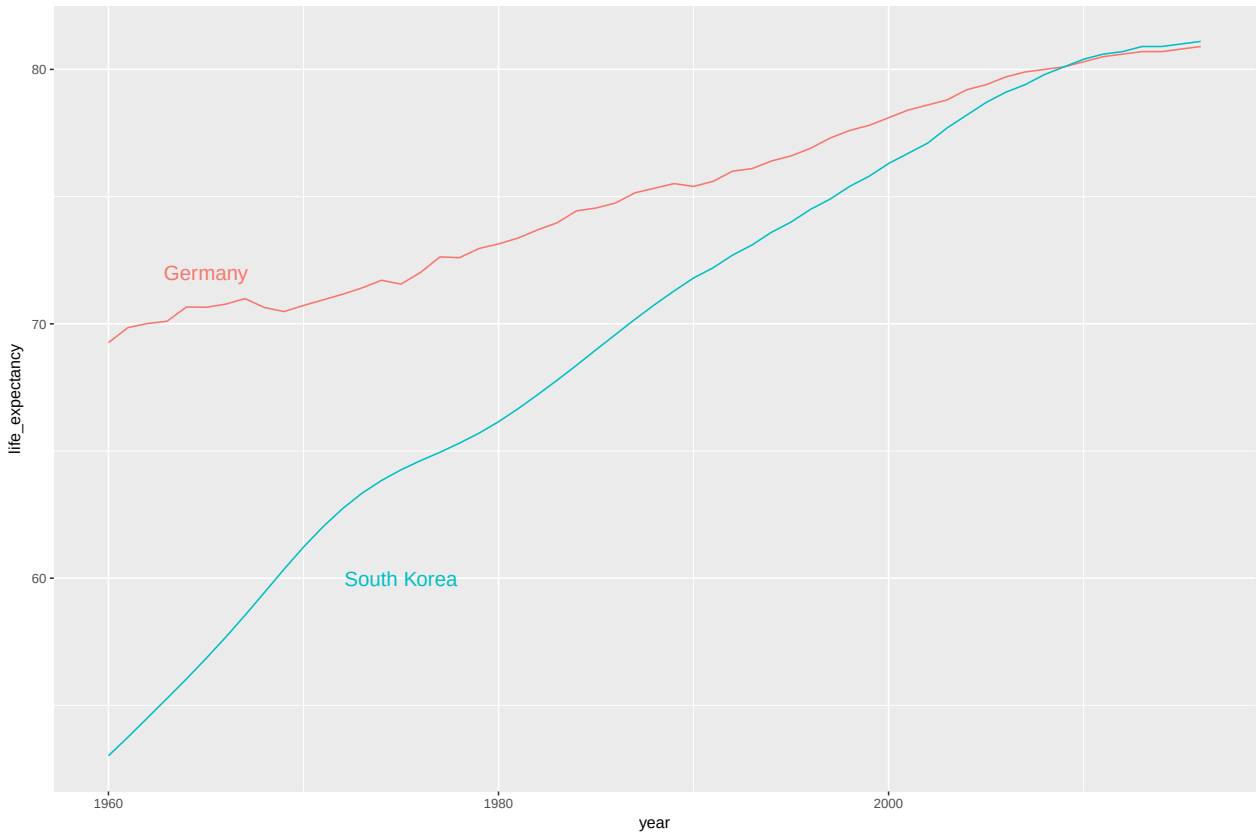
```
countries <- c("South Korea", "Germany")
gapminder %>% filter(country %in% countries) %>%
  ggplot(aes(year, fertility, col = country)) +
  geom_line()
```



Note that one could create the same plot without color using `group` within aes. Without `group` or `color`, `geom_line` will draw a lign between all of the points.

To make the same plot without a legend, we can add text to the plot itself using the `geom_text` function.

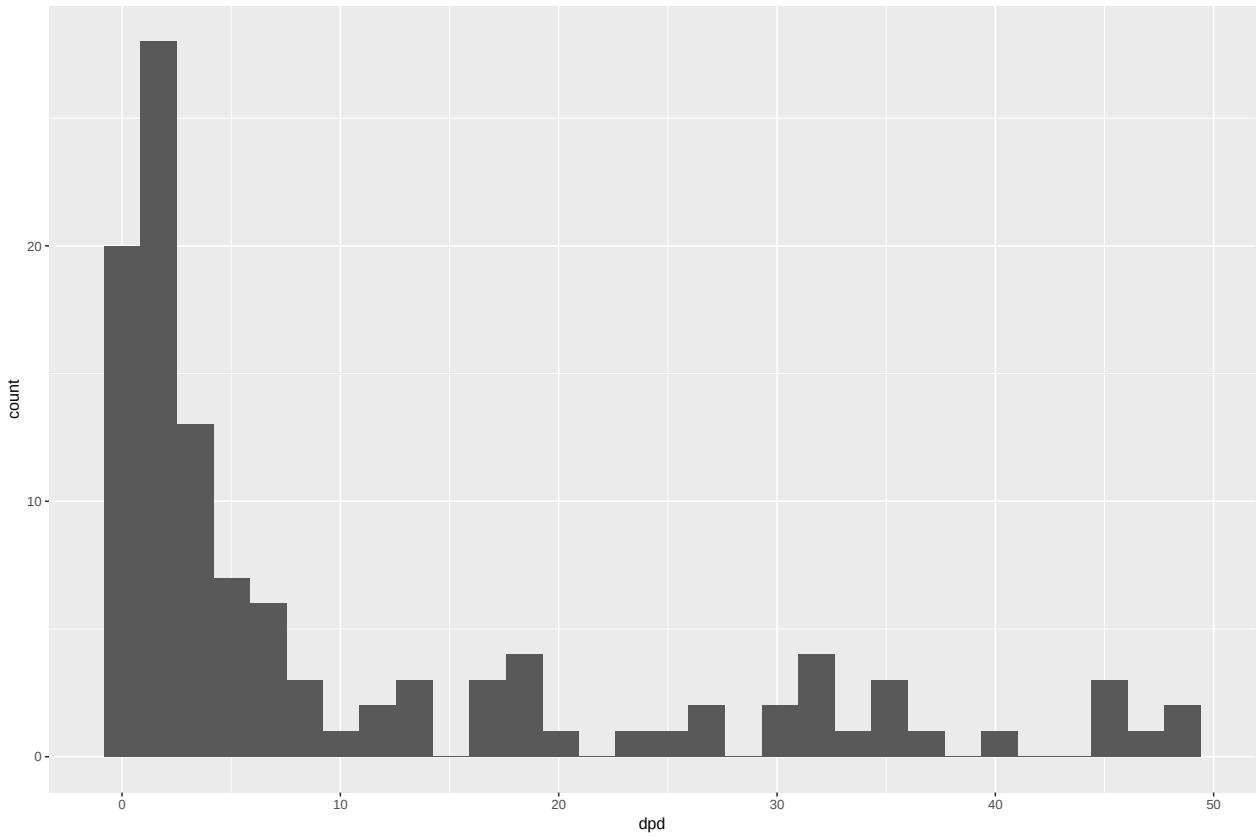
```
labels <- data.frame(country = countries, x = c(1975, 1965), y = c(60, 72))
gapminder %>% filter(country %in% countries) %>%
  ggplot(aes(year, life_expectancy, col = country)) +
  geom_line() +
  geom_text(data = labels, aes(x, y, label = country), size = 5) +
  theme(legend.position = "none")
```



Transformations

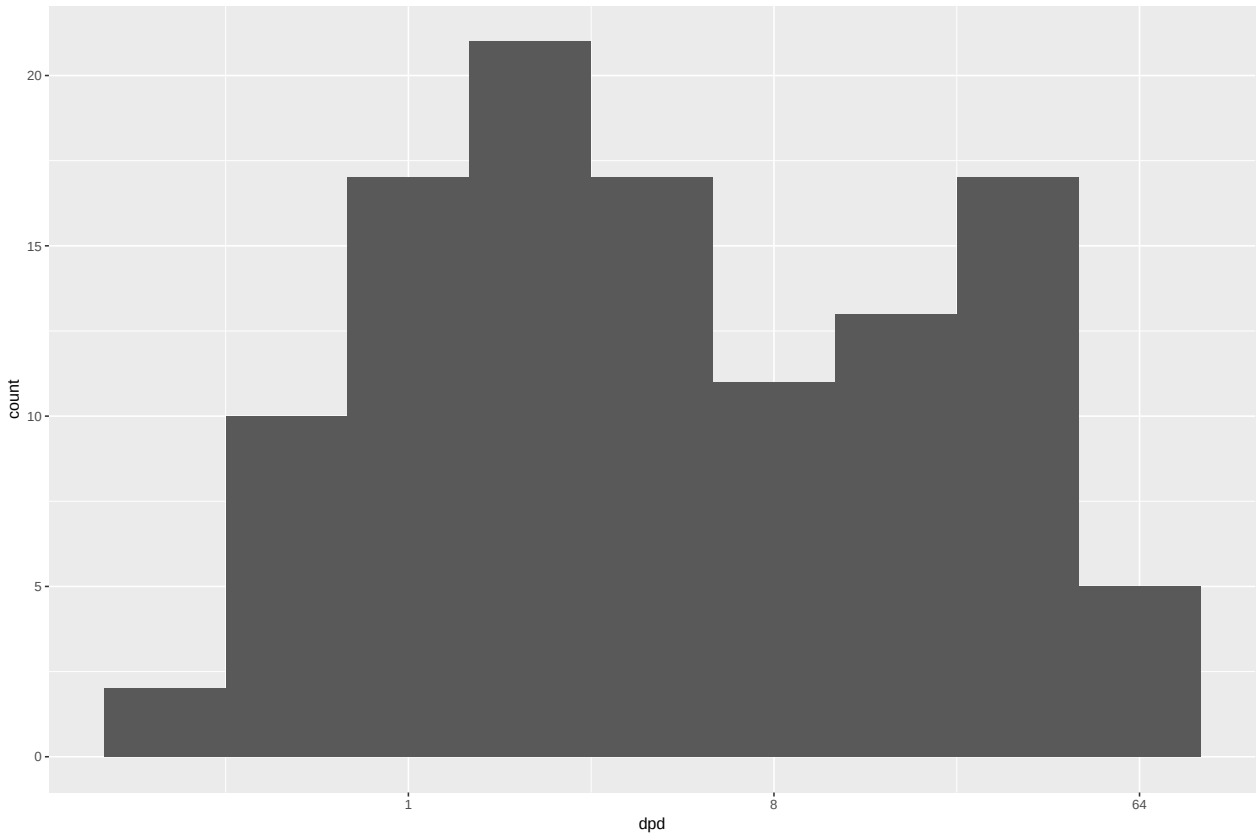
It can be useful to look at data with a transformation (e.g. log2 or log10). Let's use histograms of GDP per capita per day as an example. Here's a histogram of "dollars per day"" in 1970.

```
gapminder <- gapminder %>% mutate(dpd = gdp/population/365)
gapminder %>%
  filter(year == 1970 & !is.na(gdp)) %>%
  ggplot(aes(dpd)) +
  geom_histogram()
```



A quick way to get a sense of the data is to do a log2 transformation, so that doublings become additive.

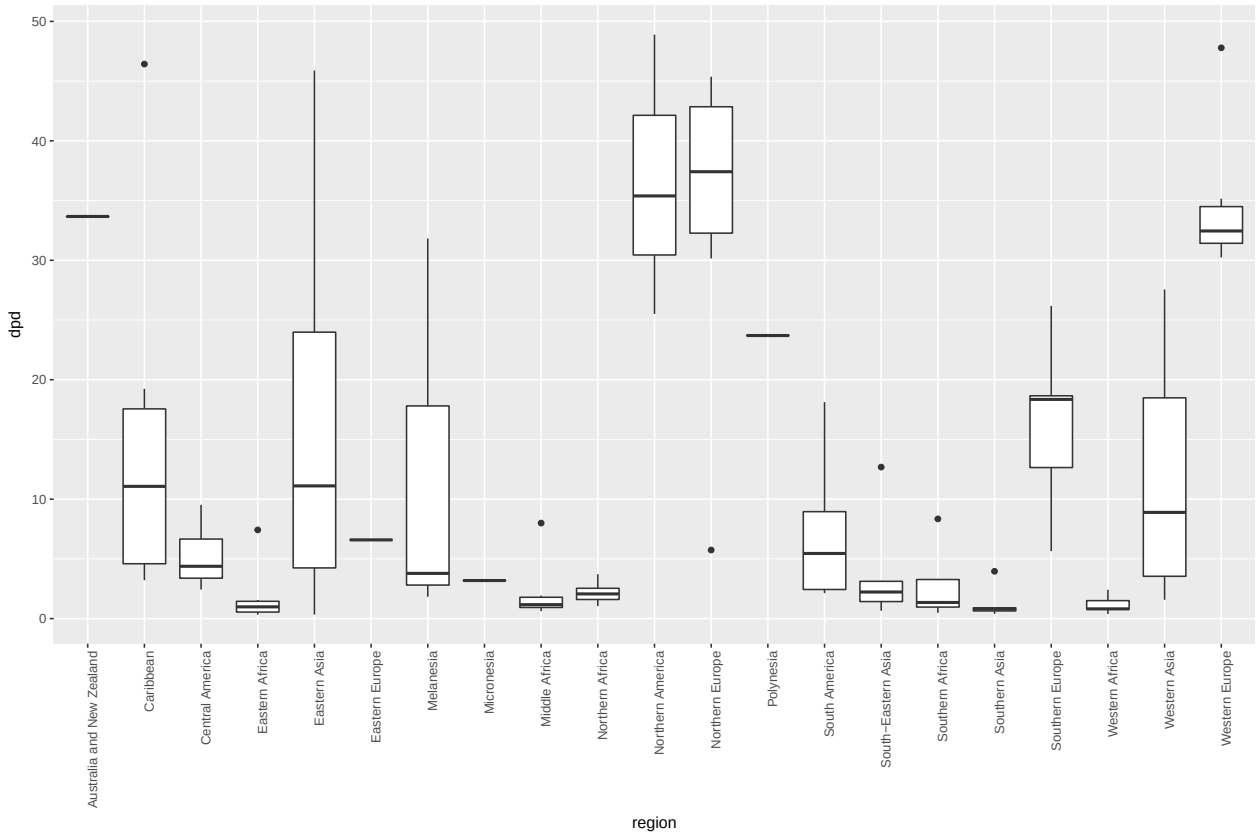
```
gapminder %>%
  filter(year == 1970 & !is.na(gdp)) %>%
  ggplot(aes(dpd)) +
  geom_histogram(binwidth = 1) +
  scale_x_continuous(trans = "log2")
```



Stratify and Boxplot

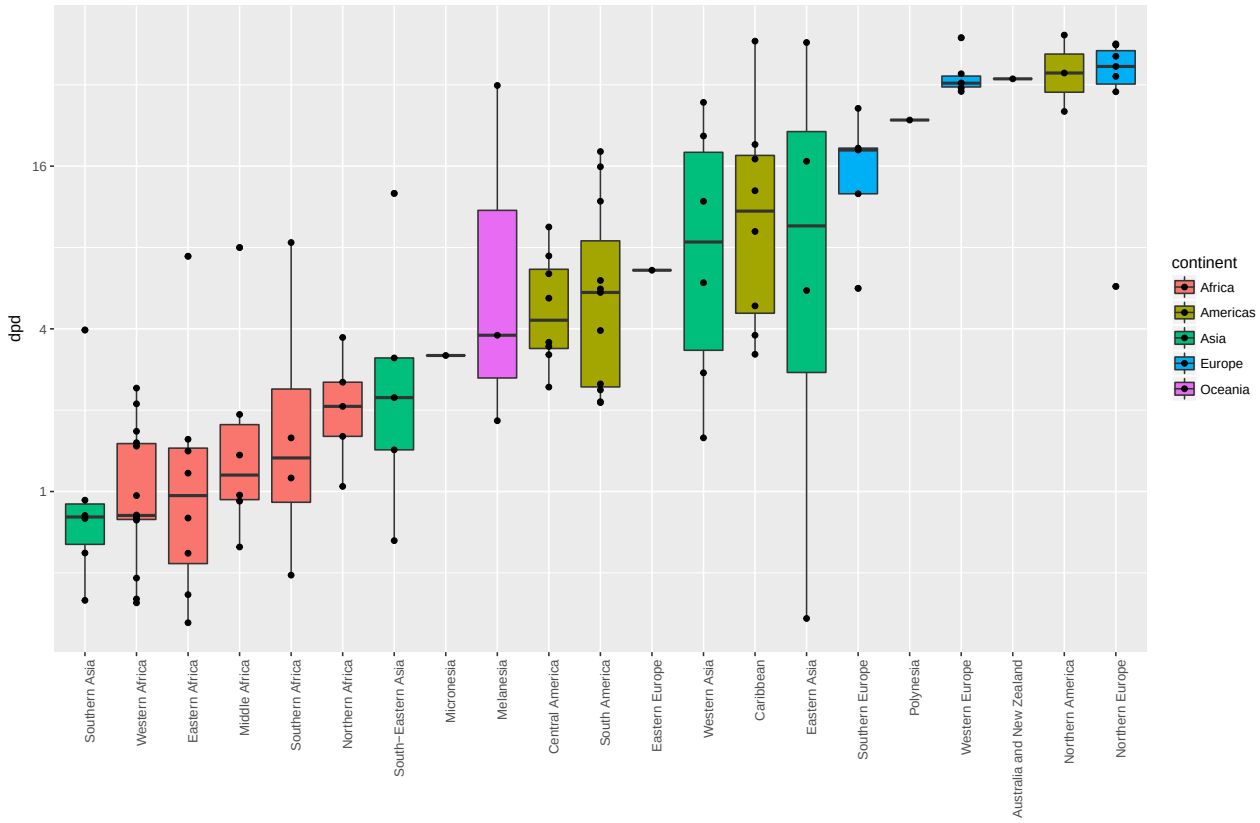
If we wanted to know whether the two modes in the histogram above are the west and the developing world, we'd need to look at the distribution of each region. The `Gapminder` dataset has 22 regions, which is unwieldy for histogram, so let's use a boxplot.

```
gapminder %>%
  filter(year == 1970 & !is.na(gdp)) %>%
  ggplot(aes(region, dpd)) +
  geom_boxplot() +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))
```

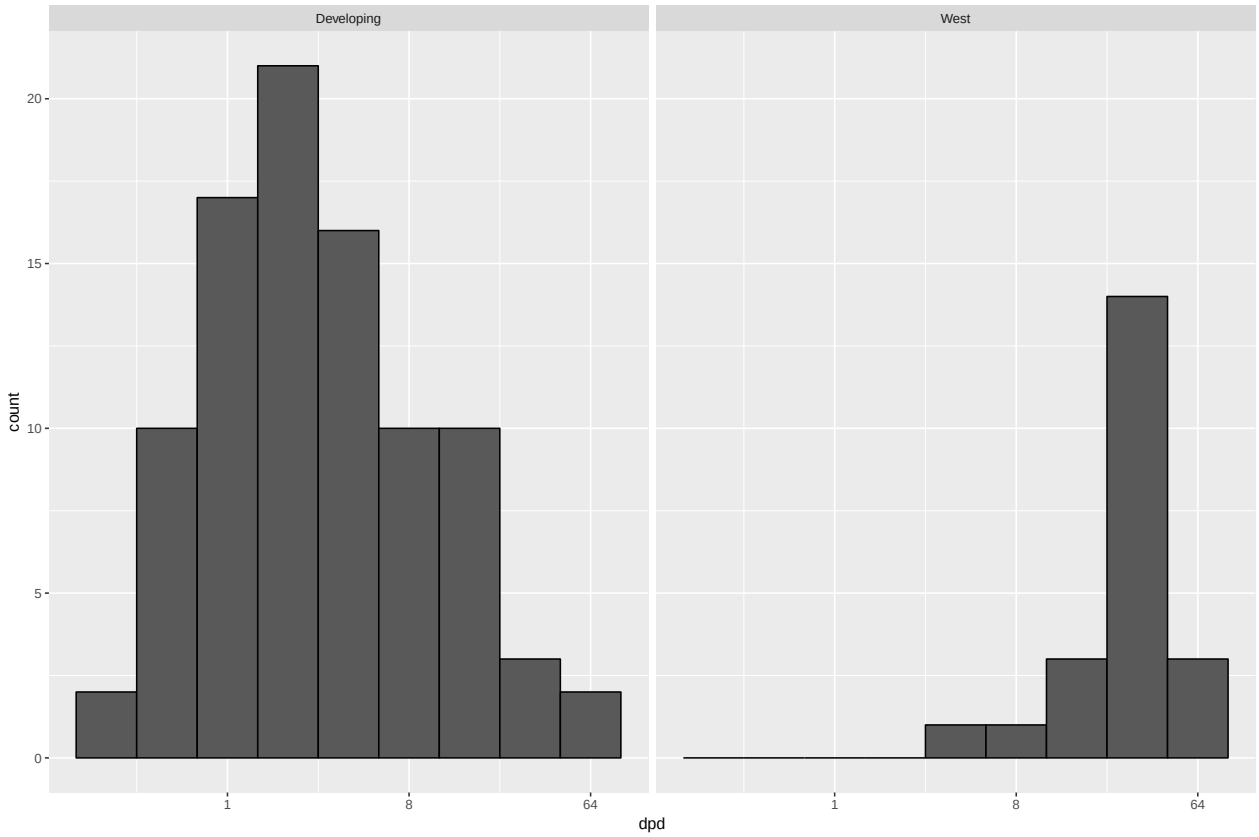


Let's put the blots in a more sensible order, and transform to a log2 scale on the y axis so that the distribution of continents is more clear.

```
gapminder %>%
  filter(year == 1970 & !is.na(gdp)) %>%
  mutate(region = reorder(region, dpd, FUN = median)) %>%
  ggplot(aes(region, dpd, fill = continent)) +
  geom_boxplot() +
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  xlab("") +
  scale_y_continuous(trans = "log2") +
  geom_point()
```



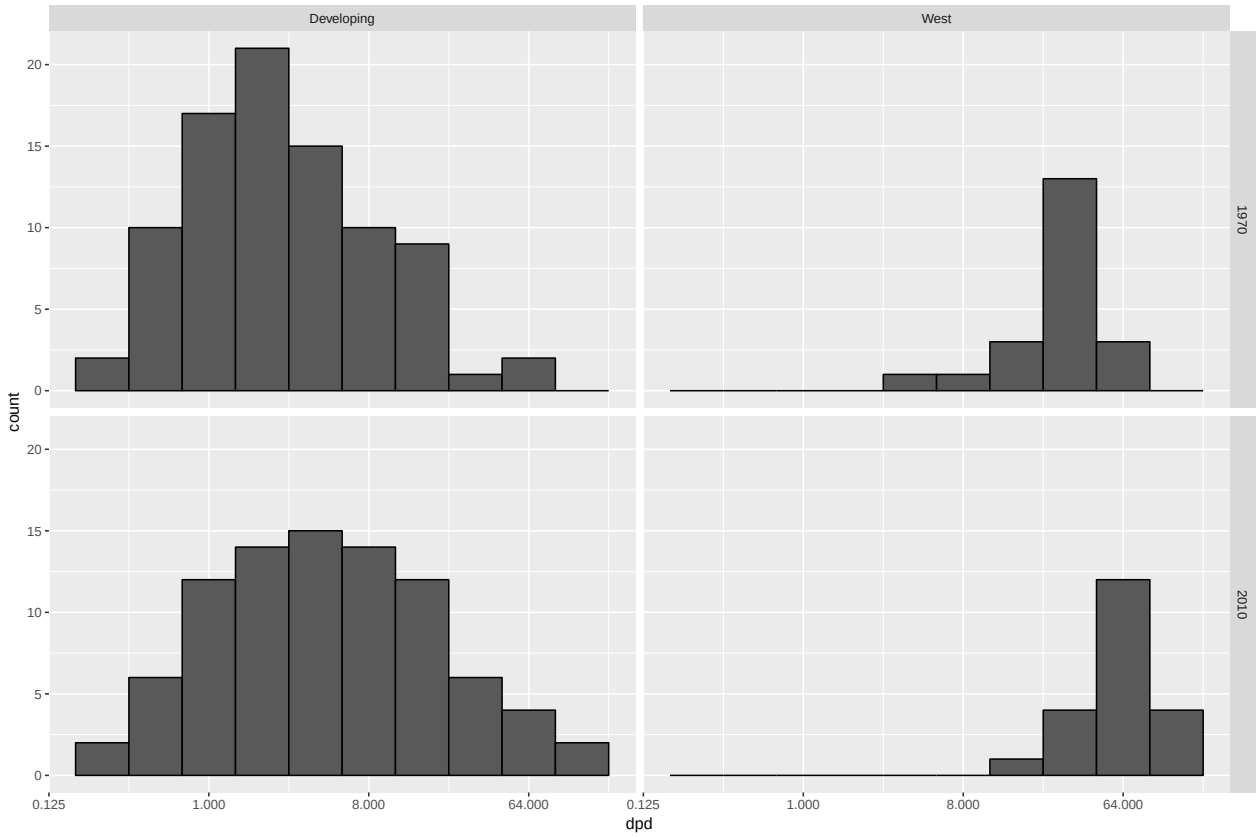

```
west <- c("Western Europe", "Northern Europe", "Southern Europe", "Northern America", "Australia and Ne
gapminder %>%
  filter(year == 1970 & !is.na(gdp)) %>%
  mutate(group = ifelse(region %in% west, "West", "Developing")) %>%
  ggplot(aes(dpd)) +
  geom_histogram(binwidth = 1, color = "black") +
  scale_x_continuous(trans = "log2") +
  facet_grid(. ~ group)
```



Let's compare a few different time points:

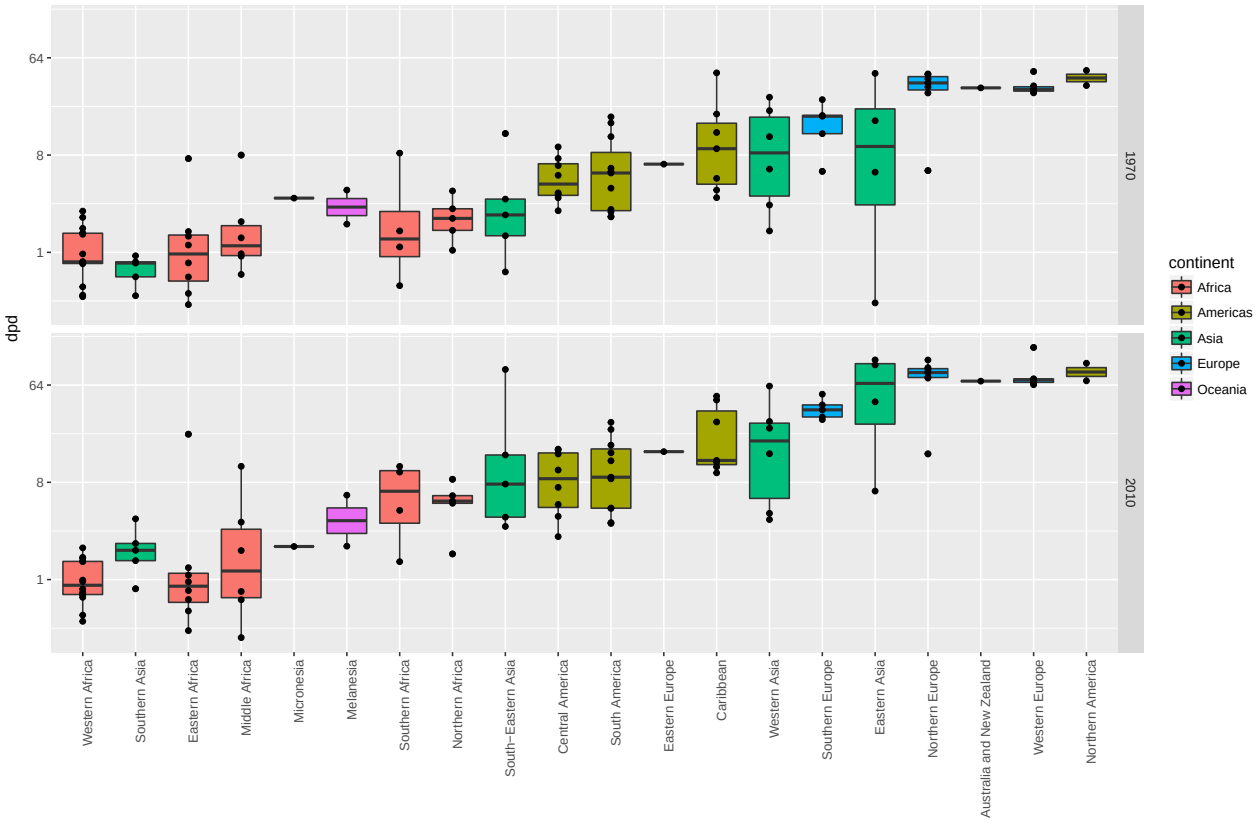
```
past_year <- 1970
country_list_1 <- gapminder %>% filter(year == past_year & !is.na(dpd)) %>% .$country
country_list_2 <- gapminder %>% filter(year == 2010 & !is.na(dpd)) %>% .$country
country_list <- intersect(country_list_1, country_list_2)
```

```
gapminder %>%
  filter(year %in% c(past_year, 2010) & country %in% country_list) %>%
  mutate(group = ifelse(region %in% west, "West", "Developing")) %>%
  ggplot(aes(dpd)) +
  geom_histogram(binwidth = 1, color = "black") +
  scale_x_continuous(trans = "log2") +
  facet_grid(year ~ group)
```



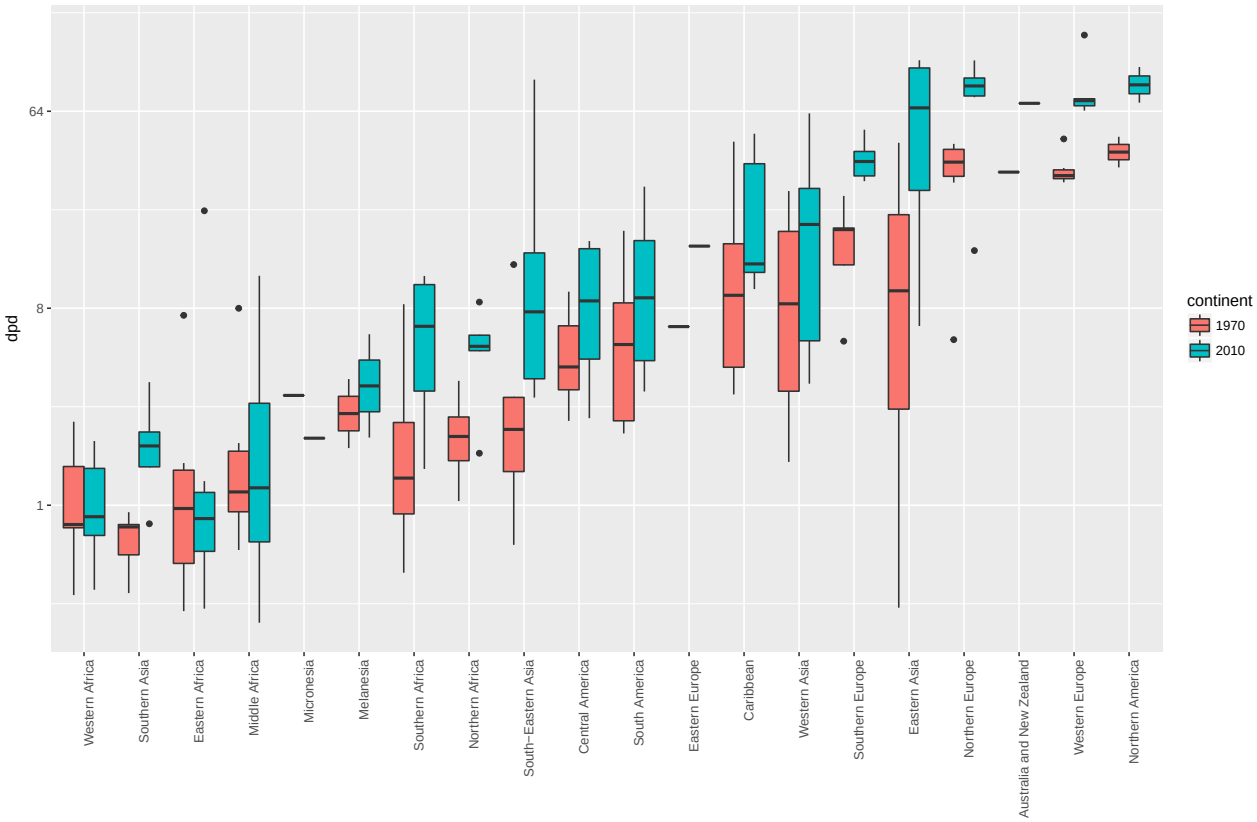
Let's compare by region between the two time points using `facet_grid` on our previous code.

```
gapminder %>%
  filter(year %in% c(past_year, 2010) & country %in% country_list) %>%
  mutate(region = reorder(region, dpd, FUN = median)) %>%
  ggplot(aes(region, dpd, fill = continent)) +
  geom_boxplot() +
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  xlab("") +
  scale_y_continuous(trans = "log2") +
  geom_point() +
  facet_grid(year ~ .)
```



Alternatively, we can use the color argument to place `dpd` plots from the same country next to each other so that the by year comparison is easier. Conveniently, the color argument will do this automatically when provided with a factor.

```
gapminder %>%
  filter(year %in% c(past_year, 2010) & country %in% country_list) %>%
  mutate(region = reorder(region, dpd, FUN = median)) %>%
  ggplot(aes(region, dpd, fill = continent)) +
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  xlab("") +
  scale_y_continuous(trans = "log2") +
  geom_boxplot(aes(region, dpd, fill = factor(year)))
```



Density Plots

Before we create density plots comparing `dpd` between the West and the developing world, it's important to consider that there are many more countries in the developing category.

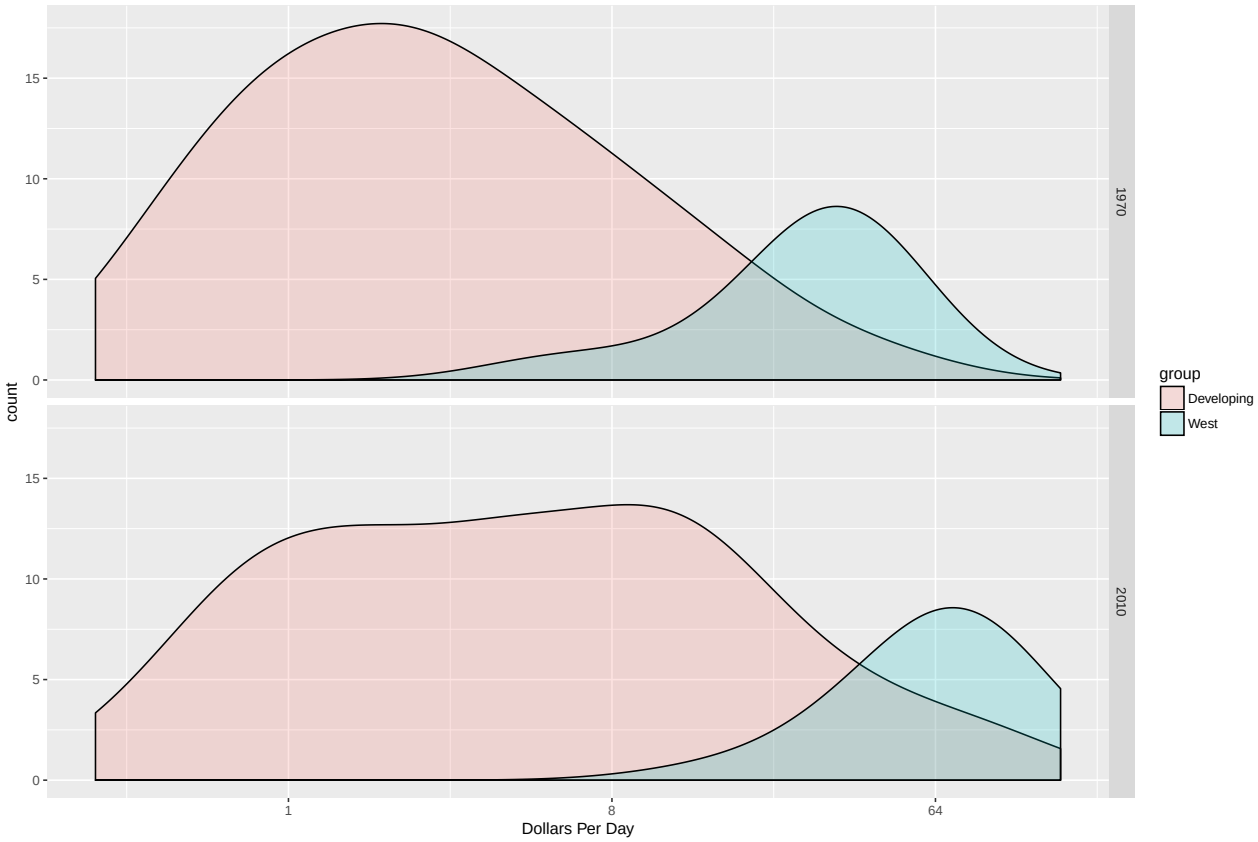
```
gapminder %>%
  filter(year == past_year & country %in% country_list) %>%
  mutate(group = ifelse(region %in% west, "West", "Developing")) %>% group_by(group) %>% summarize(n=n()
```

group	n
Developing	87
West	21

```
require(knitr)
library(knitr)
```

We can account for this by using the computed variable `count`, which is embedded in `geom_density` and accessed by add `..` to either side.

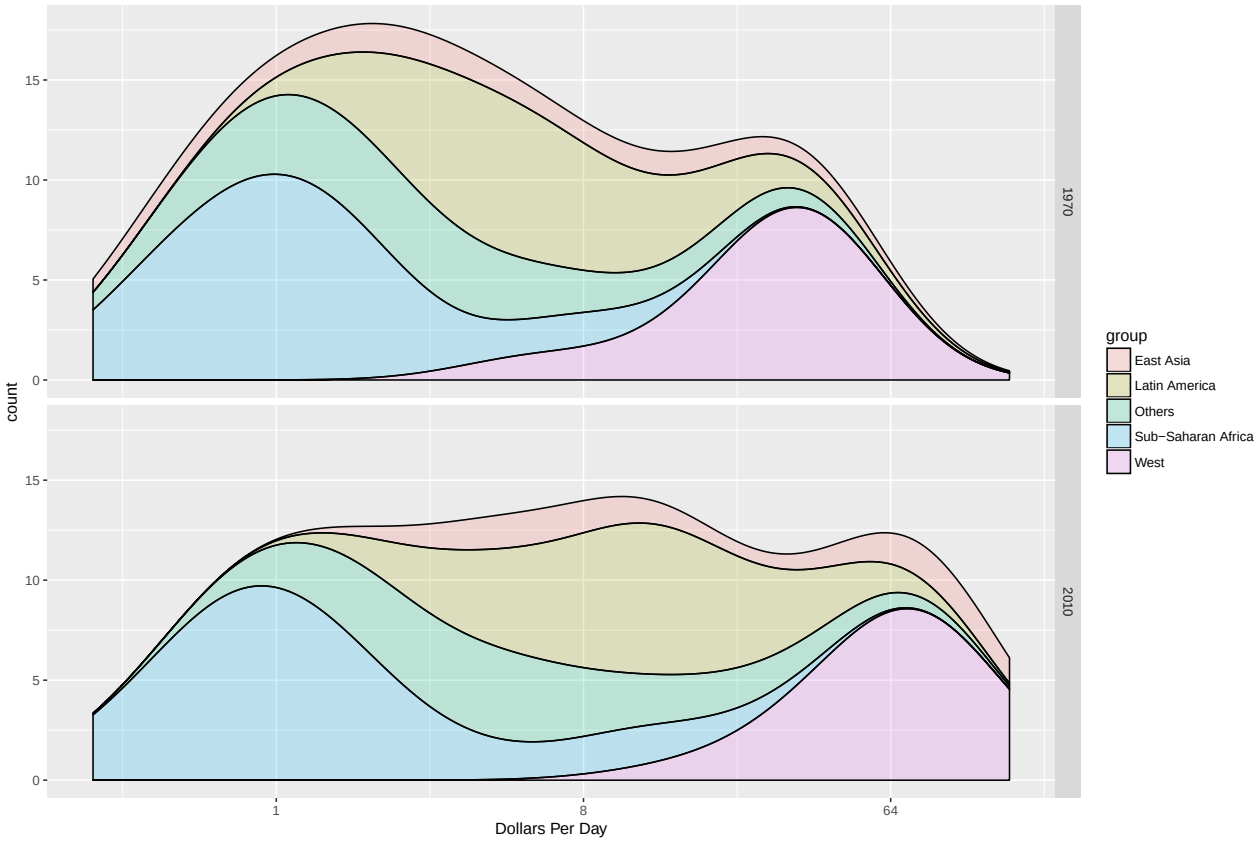
```
gapminder %>%
  filter(year %in% c(past_year, 2010) & country %in% country_list) %>%
  mutate(group = ifelse(region %in% west, "West", "Developing")) %>%
  ggplot(aes(dpd, ..count.., fill = group)) +
  scale_x_continuous(trans = "log2") +
  xlab("Dollars Per Day") +
  geom_density(alpha = 0.2, bw = 0.75) + facet_grid(year ~ .)
```



Note that we adjusted the `bw` argument to make smooth the plot, `alpha` to make the two density plots translucent, and `facet_grid` to make the 1970 and 2010 plots appear.

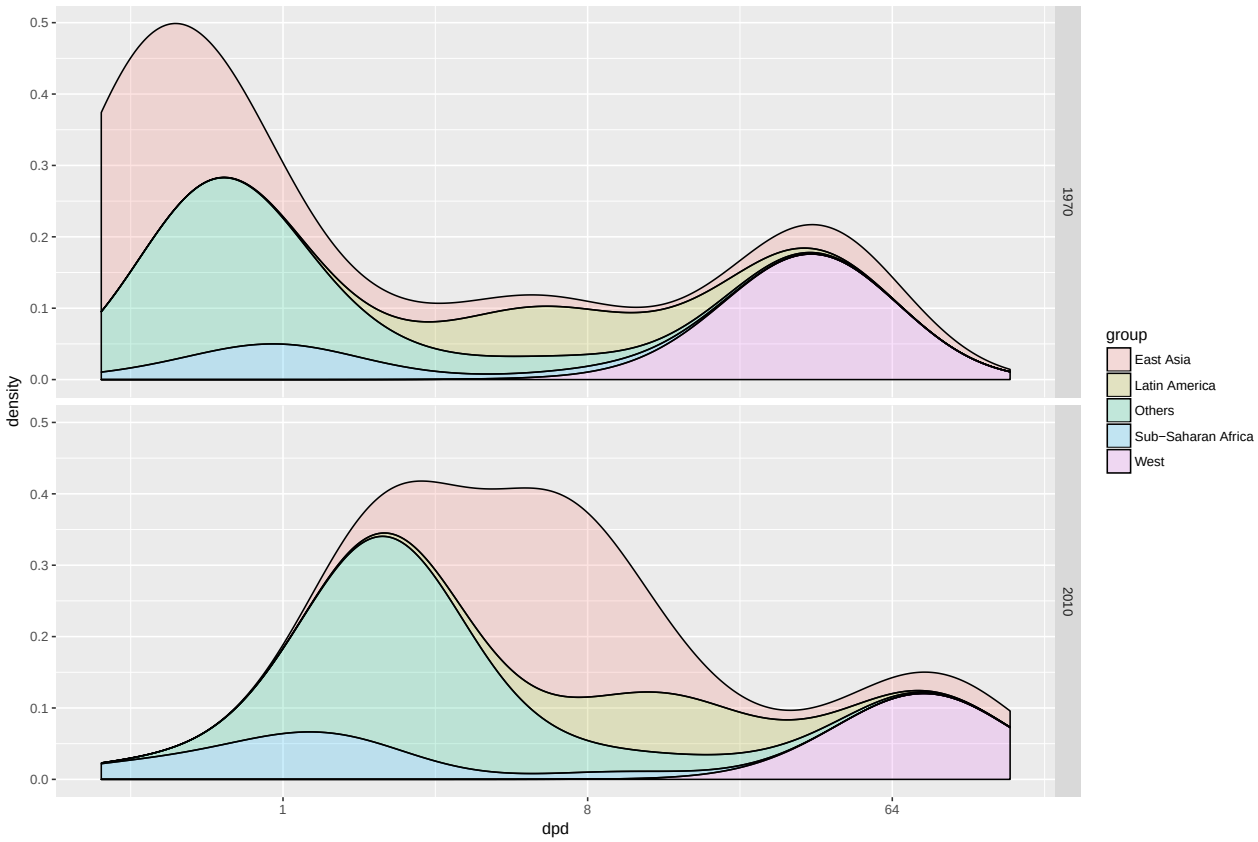
We noticed in the exploratory data anaylsis that many of the countries that saw the most dramatic change are in Asia. Let's use a new function, `case_when`, to show each region separately.

```
gapminder %>%
  filter(year %in% c(past_year, 2010) & country %in% country_list) %>%
  mutate(group = as.factor(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"
  ))) %>%
  ggplot(aes(dpd, ..count.., fill = group)) +
  scale_x_continuous(trans = "log2") +
  xlab("Dollars Per Day") +
  geom_density(alpha = 0.2, bw = 0.75, position = "stack") + facet_grid(year ~ .)
```



Let's weight the plot by population to get a more accurate picture of how per capita GDP has changed.

```
gapminder %>%
  filter(year %in% c(past_year, 2010) & country %in% country_list) %>%
  group_by(year) %>%
  mutate(weight = population/sum(population)*2) %>%
  ungroup() %>%
  mutate(group = as.factor(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"
  ))) %>%
  ggplot(aes(dpd, fill = group, weight = weight)) +
  scale_x_continuous(trans = "log2") +
  geom_density(alpha = 0.2, bw = 0.75, position = "stack") + facet_grid(year ~ .)
```



The Ecological Fallacy

It's important to remember to examine within group variability as well as differences between groups.

Let's add a few more groups to our `group` variable within gapminder to explore this idea.

```
gapminder <- gapminder %>%
  mutate(group = case_when(
    .$region %in% west ~ "West",
    .$region %in% c("Eastern Asia", "South-Eastern Asia") ~ "East Asia",
    .$region == "Southern Asia" ~ "Southern Asia",
    .$region %in% c("Caribbean", "Central America", "South America") ~ "Latin America",
    .$continent == "Africa" & .$region != "Northern Africa" ~ "Sub-Saharan Africa",
    .$region %in% c("Melanasia", "Micronesia", "Polynesia") ~ "Pacific Islands",
    TRUE ~ "Others"))
surv_income <- gapminder %>%
  filter(year == 2010 & !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)
```

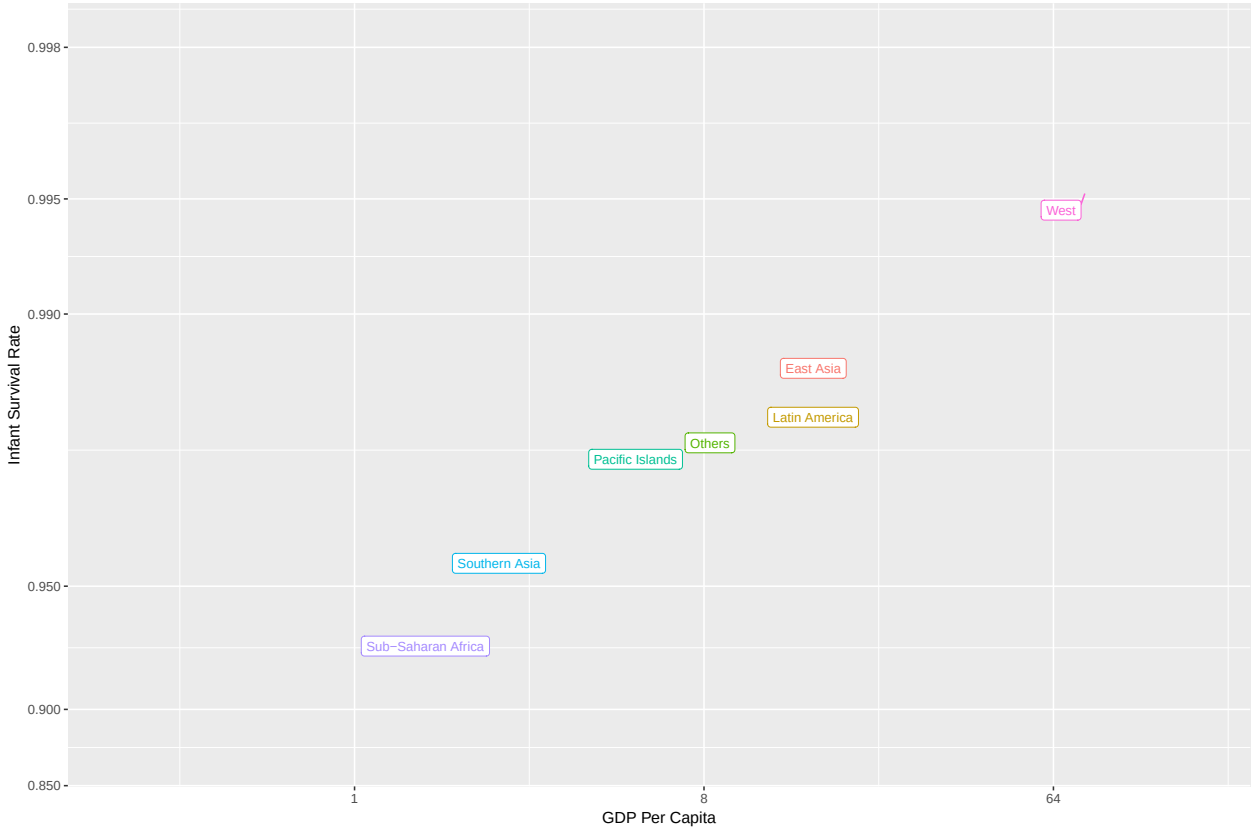
```
## # A tibble: 7 x 3
##   group              income infant_survival_rate
##   <chr>               <dbl>                <dbl>
## 1 Sub-Saharan Africa   1.76                0.936
## 2 Southern Asia        2.07                0.952
```

```
## 3 Pacific Islands      4.71                0.974
## 4 Others               9.52                0.980
## 5 Latin America       13.2                 0.983
## 6 East Asia           13.4                 0.985
## 7 West                77.1                 0.995
```

As we can see, Sub-Saharan Africa has a substantially lower income and substantially lower infant survival rate compared to the West.

As it happens, this relationship is 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, 0.9981),
                     breaks = c(0.85, 0.90, 0.95, 0.99, 0.995, 0.998)) +
  geom_label_repel(size = 3, show.legend = FALSE) +
  xlab("GDP Per Capita") +
  ylab("Infant Survival Rate")
```



We can set the range on the axes for this plot with the `limit` argument inside `scale_[axis]_continuous`, and specify which quantities are marked with `breaks`. Note here that we used a log2 transformation on the income scale, and the logistic (logit) transformation on the infant survival rate.

The logit transformation transforms the percent survival into a log odds ratio:

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

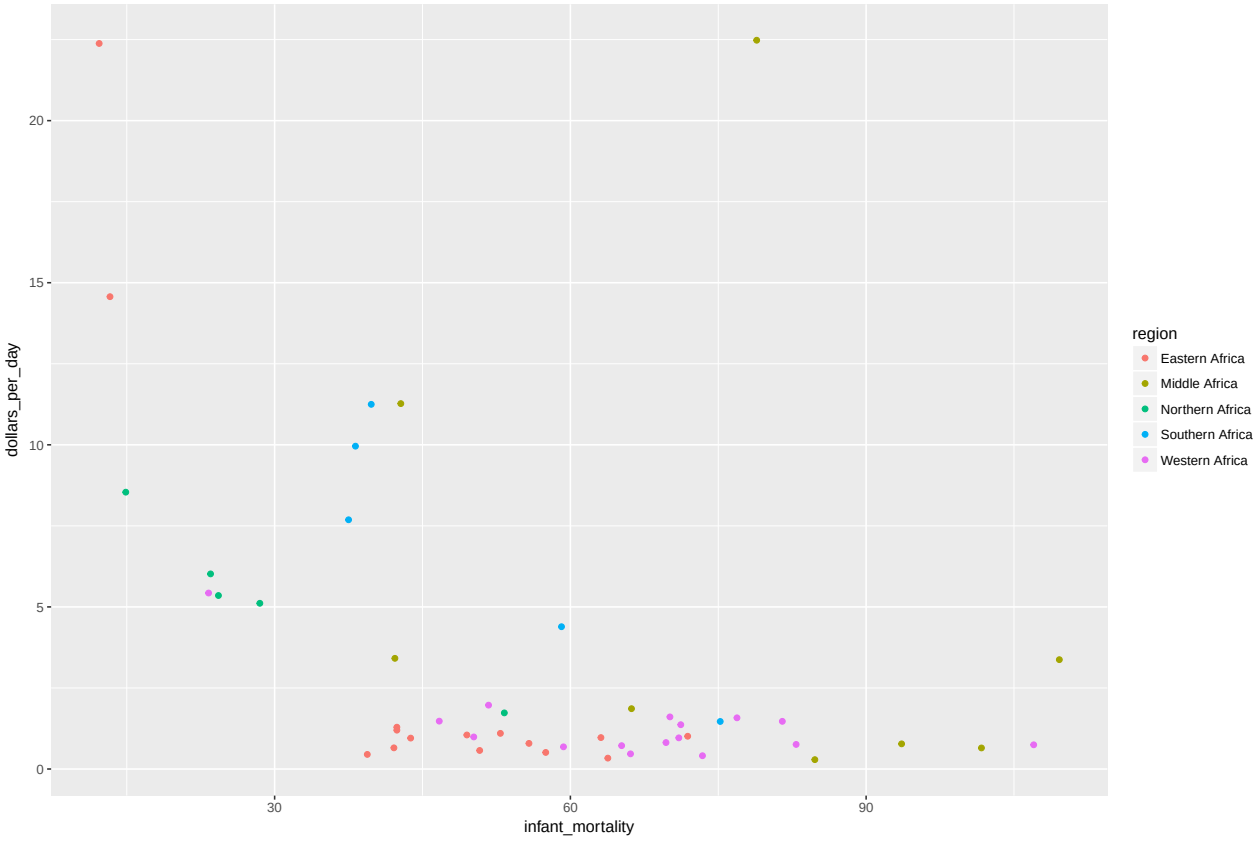
Here `p` is a proportion, and $\frac{p}{1-p}$ is the odds, which tells us how more children are expected to live than die. The log transformation makes this value symmetric, where fold increases and decreases become positive or

negative increments, respectively. This scale is useful when we want to highlight differences that are near 1 or 0.

Can we conclude from the above plot that any country with a low income must necessarily have a low infant survival rate? Can we say that every country in Sub-Saharan Africa has a lower survival rate than in southern Asia? No. To make such assumptions would be to commit the `Ecological Fallacy`.

Let's next look at country-to-country variability within regions to get a sense of why this is not true.

```
gapminder_Africa_2010 <- gapminder %>% mutate(dollars_per_day = gdp/population/365) %>% filter(year == 2
gapminder_Africa_2010 %>% ggplot(aes(infant_mortality,dollars_per_day, col = region)) + geom_point()
```



Data Visualization Principles, Part 1

Encoding Data Using Visual Cues

We have several modes at our disposal to present data, including `position`, `aligned lengths`, `angles`, `area`, `brightness`, and `color hue`.

Know When to Include Zero

If your are making a barplot, you should include zero, so that the lengths are in fact proportional, as anyone looking at the plot assumes. (Don't be like Fox News – dishonest bastards.)

If you are trying to show variability within and between groups (e.g. a single-dimension scatterplot of life expectancies by continent) and has a range that starts well above zero, the resulting blank space on the graph doesn't help you interpret the plot. Limit the range so that the variability is easier to see and interpret.

Do Not Distort Quantities

Be sure to use a visual that has a linear relationship with the raw data. For example, if circle size represents a continuous variable, be sure to make them scale by area rather than by radius (which makes the apparent proportions the square of the true proportions).

Order By a Meaningful Value

Many of the plotting functions default to ordering by alphabetical order. This is arbitrary, and usually not what we want. Remember that you can use the `reorder` function to order variables or factors so that they arranged in a meaningful, such as order countries by GDP when making a barplot. Another example would be ordering by the median value of each group when making a series of boxplots.

Data Visualization Principles, Part 2

Show the Data

Often, it's much more informative to show the distribution of the data rather than a summary (such as a mean and a SD or SE). One good when comparing a continuous variable between categoricals is to use `geom_point` with `jitter` and `alpha_blending` so that fewer points are plotted on top of each other, and those that do appear darker than single points. Histograms and smooth density plots are also good options, particularly when there are many points.

Ease Comparisons

Be sure to use common axes when showing multiple plots so that it is easy to compare between them (e.g. use the same range on the x-axis of multiple histograms).

Align plots vertically to see horizontal changes and horizontally to see vertical changes.

Generally, barplots are useful for showing a single number, but not for distributions.

Consider Transformations

It's important to use transformations where appropriate to avoid implying incorrect conclusions about data. For example, a barplot of average population size between continents would lead one to the incorrect conclusion that Asian countries have much larger populations than the others, when in fact this is due to the anomalously large populations of India and China. A much better plot is a boxplot showing the individual points with a log transformation of the y-axis, which reveals that in fact Africa has a larger median population that Asia.

Also useful are the logistic transformation (see above), best used for comparing odds (e.g. percent survival), and the square root transformation, useful for count data.

Make Visual Cues Adjacent

It's much easier to compare two parts of a plot (e.g. boxplots of GDP in 1970 vs. 2010 for each continent) when they are adjacent. We can also ease comparison by making the boxplots for each year different colors, and placing them closer to each other.

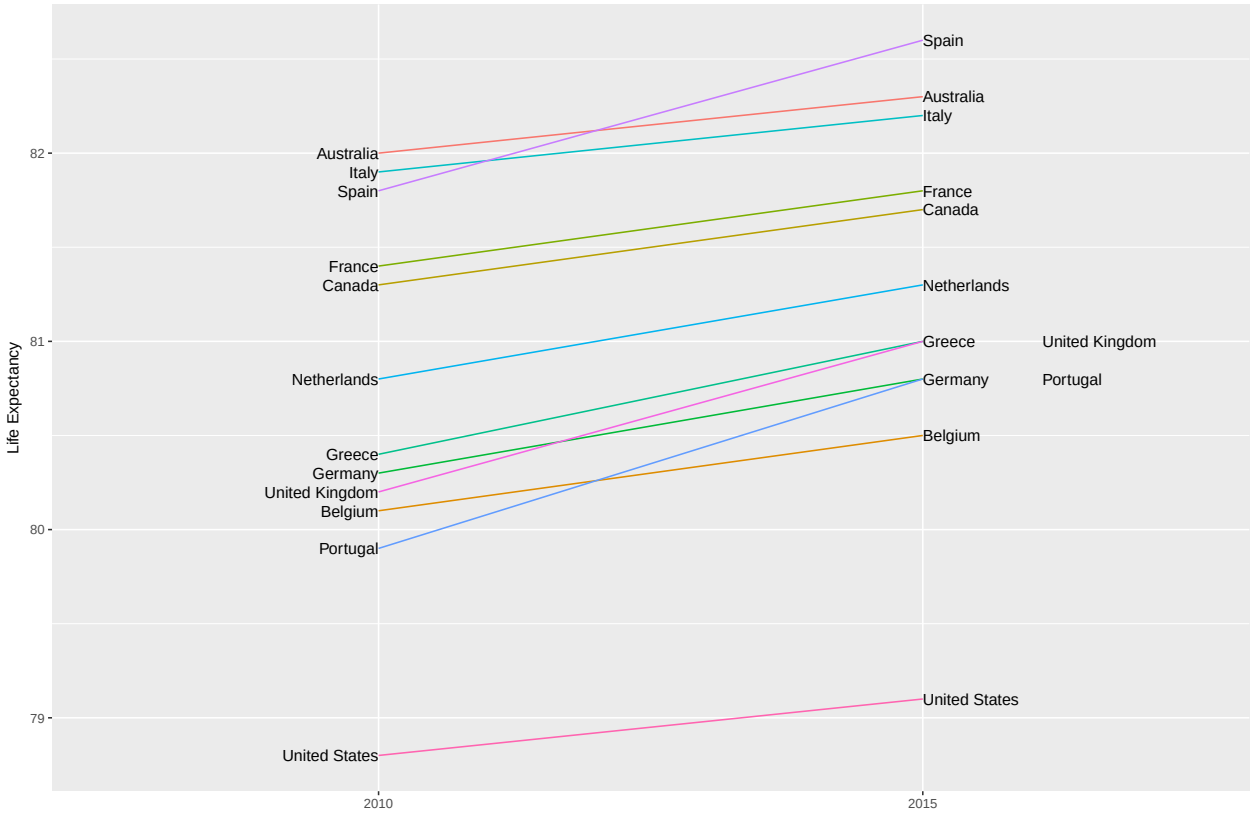
Note that the default colors for ggplot are not ideal for the 10% of the population that is colorblind. The function `scale_color_manual(values=color_blind_friendly_cols)` will add a colorblind-friendly palette to your plot.

Data Visualization Principles, Part 3

Slope Charts

We generally recommend scatterplots for comparing two continuous variables. However, if you are comparing variables of the same type at different time points with relatively few comparison, a `slope chart` can be a better way to present the data. There is no `ggplot2` geometry to make a slope chart, but with a little tinkering it can be done.

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



Also useful, is the Bland-Altman plot (aka Two Key Mean Different plot aka MA plot). This plot compares the average of two time points on one axis with the difference between the time points on the other. This

gives us a sense of both how the object in question has compared to others over time and how much it has changed.

Encoding a Third Variable

A useful to encode variables after the first two with different features of points. Categorical variabiles are best encoded with color hue and shape, which can be controlled with the `shape` argument. For continuous variables, we can color, intensity, or size.

Case Study: Vaccines

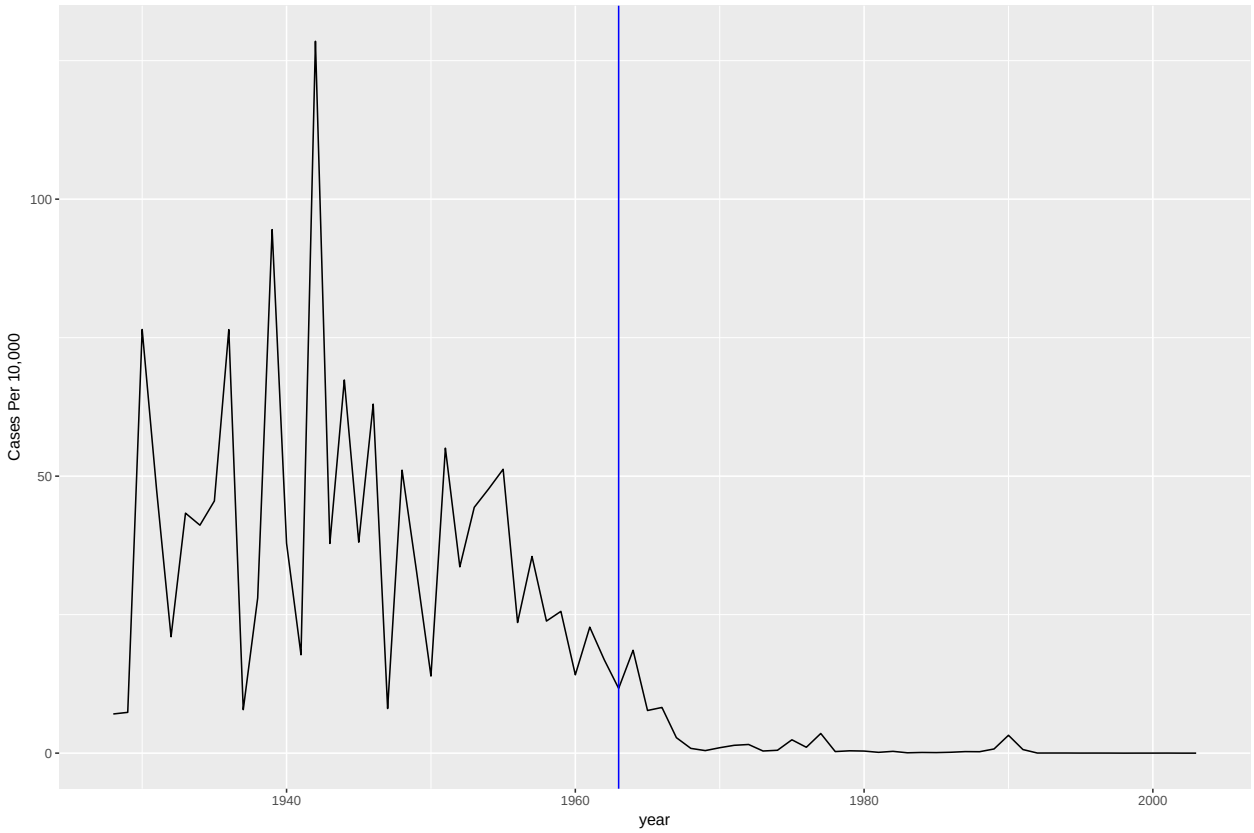
Let's look at vaccine data in the US in the last hundred years.

```
data(us_contagious_diseases)
str(us_contagious_diseases)
```

```
## 'data.frame':    18870 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: int  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 ...
```

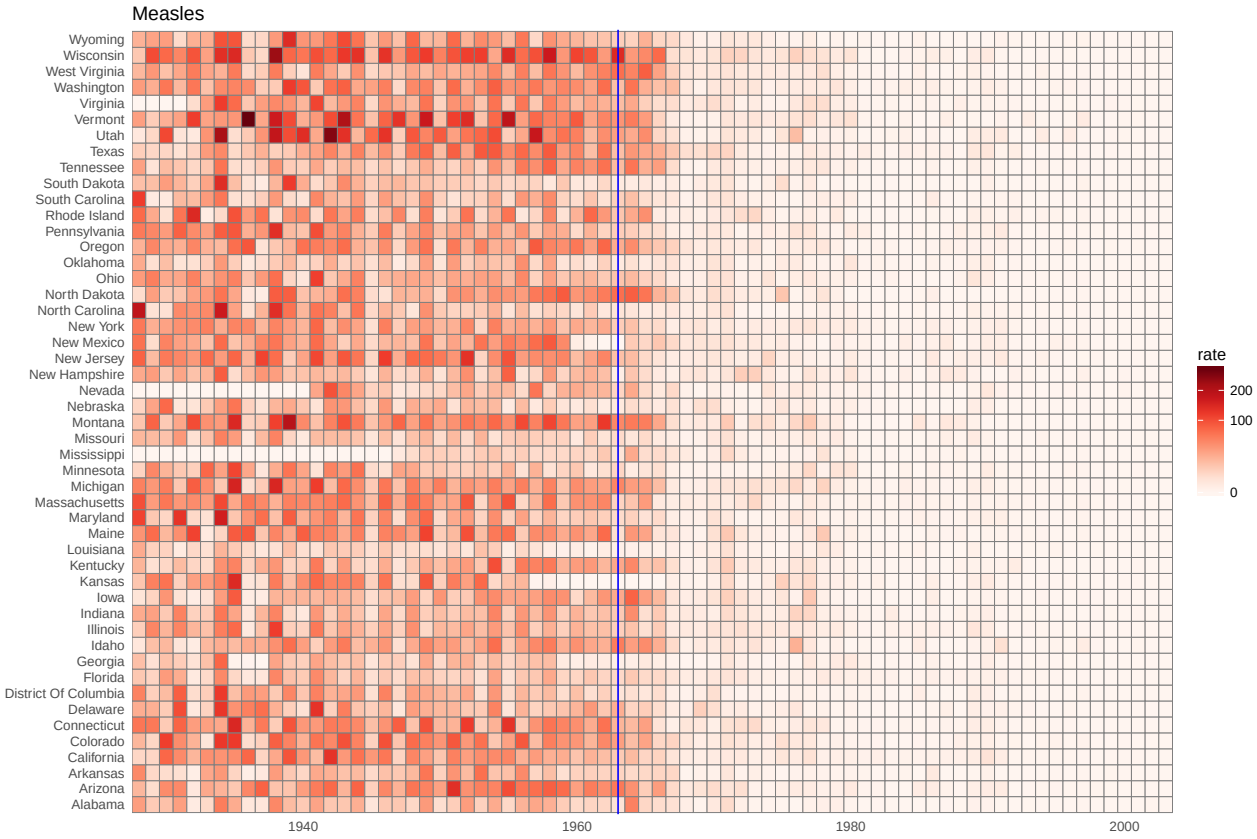
Let's look at measles rates overall in the US over the last century:

```
the_disease <- "Measles"
dat <- us_contagious_diseases %>%
  filter(!state %in% c("Hawaii", "Alaska") & disease == the_disease) %>%
  mutate(rate = count/population*10000) %>%
  mutate(state <- reorder(state, rate))
dat %>% filter(state == "California") %>%
  ggplot(aes(year, rate)) +
  geom_line() + ylab("Cases Per 10,000") +
  geom_vline(xintercept = 1963, col ="blue")
```



We can create a tile plot showing measles rates by states. Rates decline substantially when vaccines are introduced:

```
library(RColorBrewer)
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("")
```



Avoid 3D Plots

3D plots suck. Don't use them.

Avoid Too Many Significant Digits

Too many significant digits cause clutter and prevent you from understanding the data easily. Use `signif` and `round` to make tables easier to understand. Remember also to place values being compared in columns rather than rows.