R Notebook

Code

Chapter 7 ggplot2 Exploratory data visualization is perhaps the greatest strength of R. One can quickly go from idea to data to plot with a unique balance of flexibility and ease. For example, Excel may be easier than R for some plots, but it is nowhere near as flexible. D3.js may be more flexible and powerful than R, but it takes much longer to generate a plot.

Throughout the book, we will be creating plots using the ggplot225 package.

Hide

```
library(dplyr)
```

```
다음의 패키지를 부착합니다: 'dplyr'

The following objects are masked from 'package:stats':

    filter, lag

The following objects are masked from 'package:base':

    intersect, setdiff, setequal, union
```

Hide

```
library(ggplot2)
```

Many other approaches are available for creating plots in R. In fact, the plotting capabilities that come with a basic installation of R are already quite powerful. There are also other packages for creating graphics such as grid and lattice. We chose to use ggplot2 in this book because it breaks plots into components in a way that permits beginners to create relatively complex and aesthetically pleasing plots using syntax that is intuitive and comparatively easy to remember.

One reason ggplot2 is generally more intuitive for beginners is that it uses a grammar of graphics26, 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 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 is easy for beginners is that its default behavior is carefully chosen to satisfy the great majority of cases and is visually pleasing. As a result, it is possible to create informative and elegant graphs with relatively simple and readable code.

One limitation is that ggplot2 is designed to work exclusively with data tables in tidy format (where rows are observations and columns are variables). However, a substantial percentage of datasets that beginners work with are in, or can be converted into, this format. An advantage of this approach is that, assuming that our data is tidy, ggplot2 simplifies plotting code and the learning of grammar for a variety of plots.

To use ggplot2 you will have to learn several functions and arguments. These are hard to memorize, so we highly recommend you have the ggplot2 cheat sheet handy. You can get a copy here: https://www.rstudio.com/wp-content/uploads/2015/03/ggplot2-cheatsheet.pdf (https://www.rstudio.com/wp-content/uploads/2015/03/ggplot2-cheatsheet.pdf) or simply perform an internet search for “ggplot2 cheat sheet.”

7.1 The components of a graph We will construct a graph that summarizes the US murders dataset that looks like this:

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, which 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 the plot above and introduce some of the ggplot2 terminology. The main three components to note are:

Data: The US murders data table is being summarized. We refer to this as the data component. Geometry: The plot above is a scatterplot. This is referred to as the geometry component. Other possible geometries are barplot, histogram, smooth densities, qqplot, and boxplot. We will learn more about these in the Data Visualization part of the book. Aesthetic mapping: The plot uses several visual cues to represent the information provided by the dataset. The two most important cues in this plot are the point positions on the x-axis and y-axis, which represent population size and the total number of murders, respectively. Each point represents a different observation, and we map data about these observations to visual cues like x- and y-scale. Color is another visual cue that we map to region. We refer to this as the aesthetic mapping component. How we define the mapping depends on what geometry we are using. We also note that:

The points are labeled with the state abbreviations. 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. There are labels, a title, a legend, and we use the style of The Economist magazine. We will now construct the plot piece by piece.

We start by loading the dataset:

Hide

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

7.2 ggplot objects 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 what data is associated with this object:

Hide

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


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

Hide

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


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 we can assign our plot to an object, for example like this:

Hide

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

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

Hide

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

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

Hide

```
print(p)
```

Hide

```
p
```


7.3 Geometries In ggplot2 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 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 scatterplot. What geometry do we use?

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

(Image courtesy of RStudio27. CC-BY-4.0 license28.)

Geometry function names follow the pattern: geom_X where X is the name of the geometry. Some examples include geom_point, geom_bar, and geom_histogram.

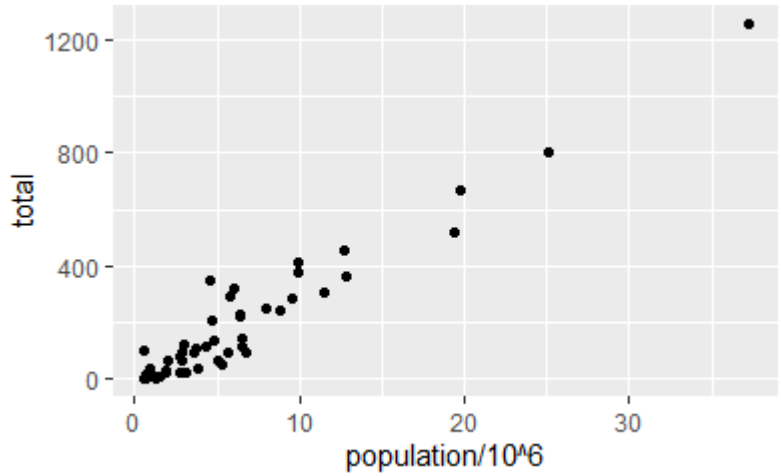
For geom_point to run properly we need to provide data and a mapping. We have already connected the object p with the murders data table, and if we add the layer geom_point it defaults to using this data. To find out what mappings are expected, we read the Aesthetics section of the help file geom_point help file:

> Aesthetics
>
> geom_point understands the following aesthetics (required aesthetics are in bold):
>
> x
>
> y
>
> alpha
>
> colour and, as expected, we see that at least two arguments are required x and y.

7.4 Aesthetic mappings Aesthetic mappings describe how properties of the data connect with features of the graph, such as distance along an axis, size, or color. The aes function connects data with what we see on the graph by defining aesthetic mappings and will be one of the functions you use most often when plotting. The outcome of the aes function is often used as the argument of a geometry function. This example produces a scatterplot of total murders versus population in millions:

Hide

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

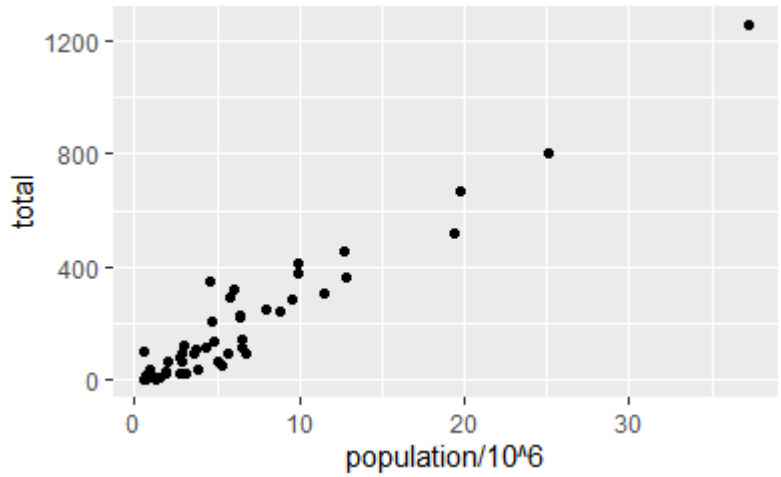


We can drop the x = and y = if we wanted to since these are the first and second expected arguments, as seen in the help page.

Instead of defining our plot from scratch, we can also add a layer to the p object that was defined above as p <- ggplot(data = murders):

Hide

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



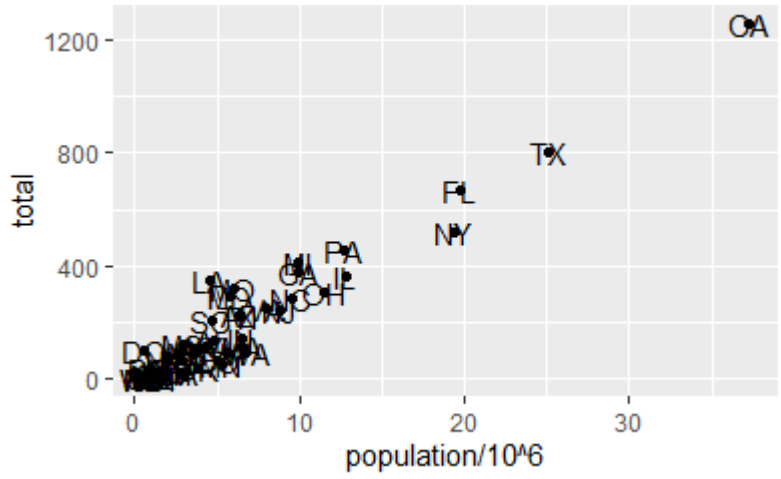
The scale and labels are defined by default when adding this layer. Like dplyr functions, aes also uses the variable names from the object component: we can use population and total without having to call them as murders$population and murders$total. The behavior of recognizing the variables from the 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.

7.5 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 with and without a rectangle behind the text, respectively.

Because each point (each state in this case) has a label, we need an aesthetic mapping to make the connection between points and labels. 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:

Hide

```
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:

Hide

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

is fine, whereas this call:

Hide

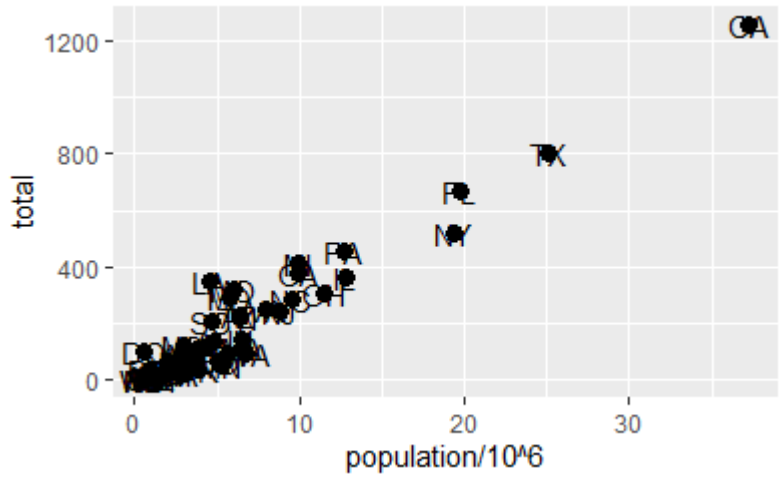
```
p_test <- p + geom_text(aes(population/10^6, total), label = abb)
```

```
Error in layer(data = data, mapping = mapping, stat = stat, geom = GeomText,  :
  object 'abb' not found
```

will give you an error since abb is not found because it is outside of the aes function. The layer geom_text does not know where to find abb since it is a column name and not a global variable. 7.5.1 Tinkering with arguments 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 size. In the help file we see that size is an aesthetic and we can change it like this:

Hide

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

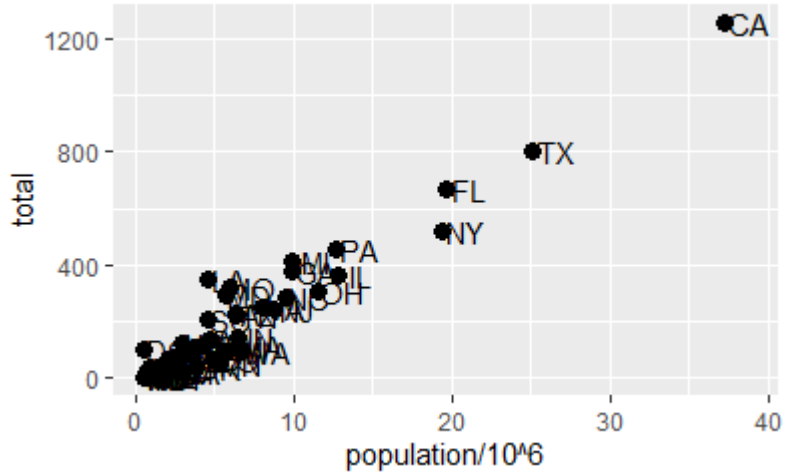


size is not a mapping: whereas mappings use data from specific observations and need to be inside aes(), operations we want to affect all the points the same way do not need to be included inside aes.

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

Hide

```
p + geom_point(aes(population/10^6, total), size = 3) +
  geom_text(aes(population/10^6, total, label = abb), nudge_x = 1.5)
```



This is preferred as it makes it easier to read the text. In Section 7.11 we learn a better way of assuring we can see the points and the labels.

7.6 Global versus local aesthetic mappings 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:

Hide

```
args(ggplot)
```

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

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

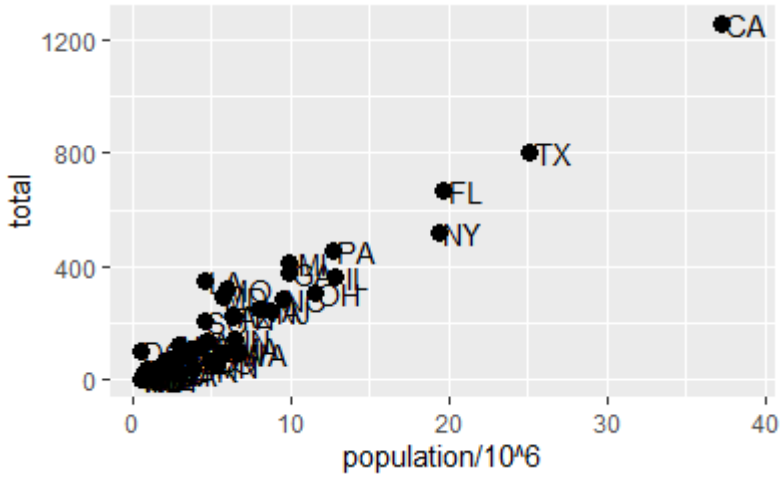
Hide

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

and then we can simply write the following code to produce the previous plot:

Hide

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

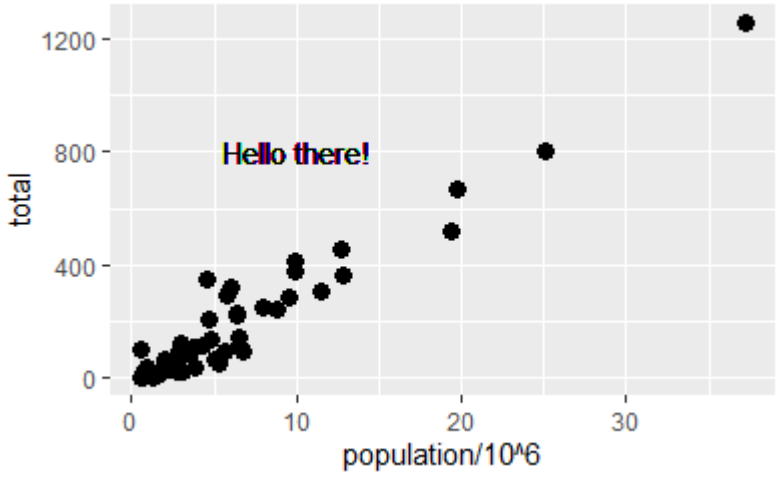


We keep the size and nudge_x arguments in geom_point and geom_text, respectively, because we want to only increase the size of points and only nudge the labels. If we put those arguments in aes then they would apply to both plots. Also note that the geom_point function does not need a label argument and therefore ignores that aesthetic.

If necessary, we can override the global mapping by defining a new mapping within each layer. These local definitions override the global. Here is an example:

Hide

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

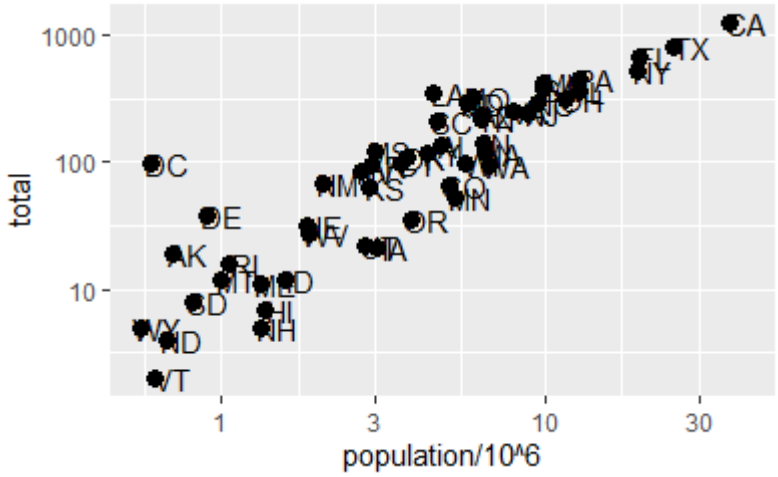


Clearly, the second call to geom_text does not use population and total.

7.7 Scales First, 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 the scale_x_continuous function lets us control the behavior of scales. We use them like this:

Hide

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

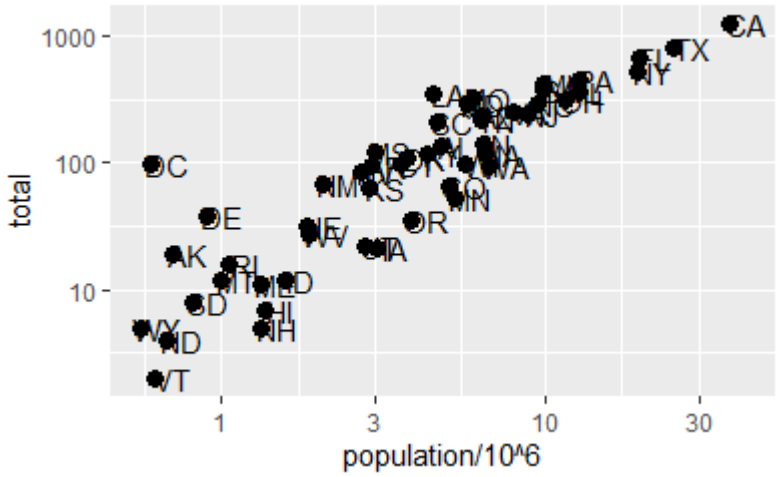


Because we are in the log-scale now, the nudge must be made smaller.

This particular transformation is so common that ggplot2 provides the specialized functions scale_x_log10 and scale_y_log10, which we can use to rewrite the code like this:

Hide

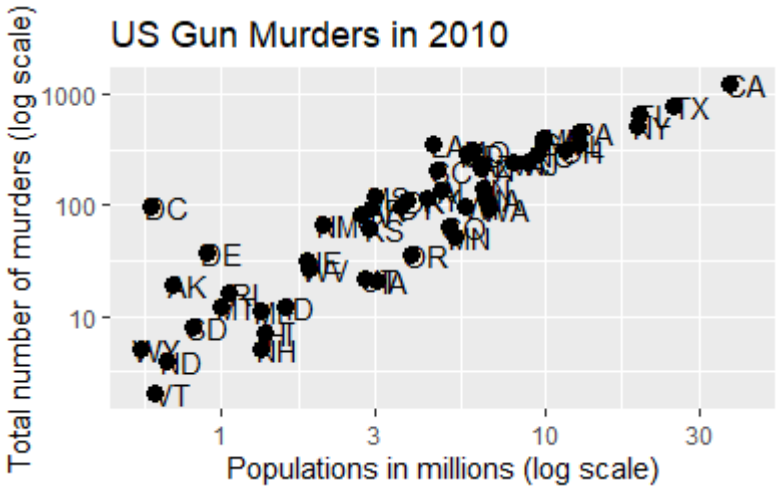
```
p + geom_point(size = 3) +
  geom_text(nudge_x = 0.05) +
  scale_x_log10() +
  scale_y_log10()
```



7.8 Labels and titles Similarly, the cheat sheet quickly reveals that to change labels and add a title, we use the following functions:

Hide

```
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")
```



We are almost there! All we have left to do is add color, a legend, and optional changes to the style. 7.9 Categories as colors We can change the color of the points using the col argument in the geom_point function. To facilitate demonstration of new features, we will redefine p to be everything except the points layer:

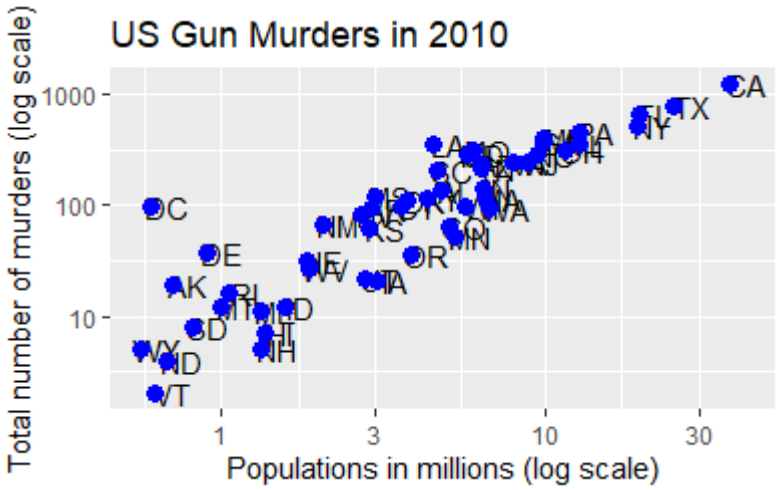
Hide

```
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:

Hide

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

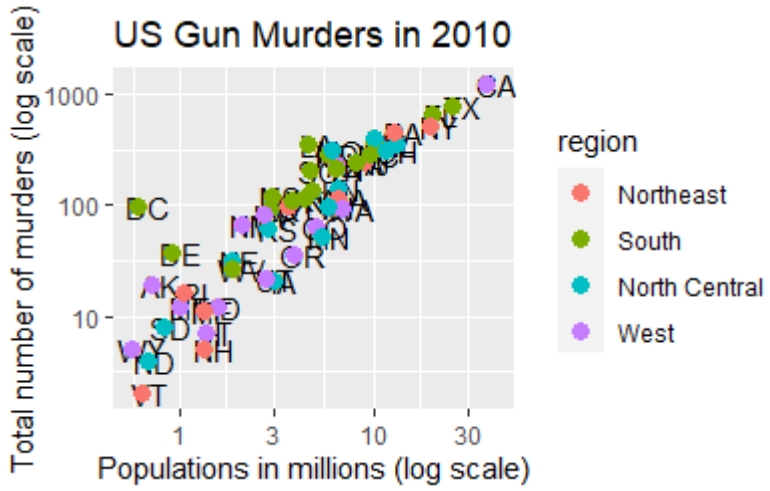


This, of course, is not what we want. We want to assign color depending on the geographical region. A nice default behavior of ggplot2 is that if we assign a categorical variable to color, it automatically assigns a different color to each category and also adds a legend.

Since the choice of color is determined by a feature of each observation, this is an aesthetic mapping. To map each point to a color, we need to use aes. We use the following code:

Hide

```
p + geom_point(aes(col=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 mappings are expected in this function call.

Here we see yet another useful default behavior: ggplot2 automatically adds a legend that maps color to region. To avoid adding this legend we set the geom_point argument show.legend = FALSE.

7.10 Annotation, shapes, and adjustments We often want to add shapes or annotation to figures that are not derived directly from the aesthetic mapping; examples include labels, boxes, shaded areas, and lines.

Here we want to add a line that represents the average murder rate for the entire country. Once we determine the per million rate to be
r , this line is defined by the formula:
y = r x , 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:

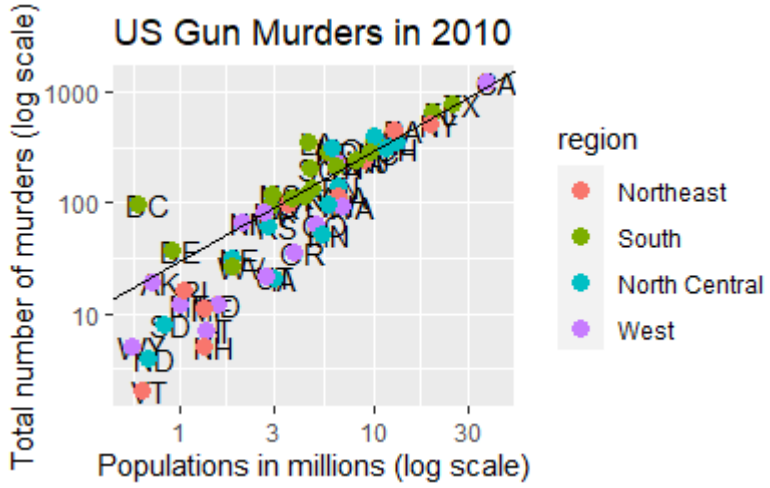
Hide

```
r <- murders %>%
  summarize(rate = sum(total) /  sum(population) * 10^6) %>%
  pull(rate)
```

To add a line we use the geom_abline function. ggplot2 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:

Hide

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



Here geom_abline does not use any information from the data object.

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

Hide

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

Note that we have redefined p and used this new p below and in the next section.

The default plots created by ggplot2 are already very useful. However, we frequently 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, we can make changes to the legend via the scale_color_discrete function. In our plot the word region is capitalized and we can change it like this:

Hide

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