07 ggplot2

Soyoung Park

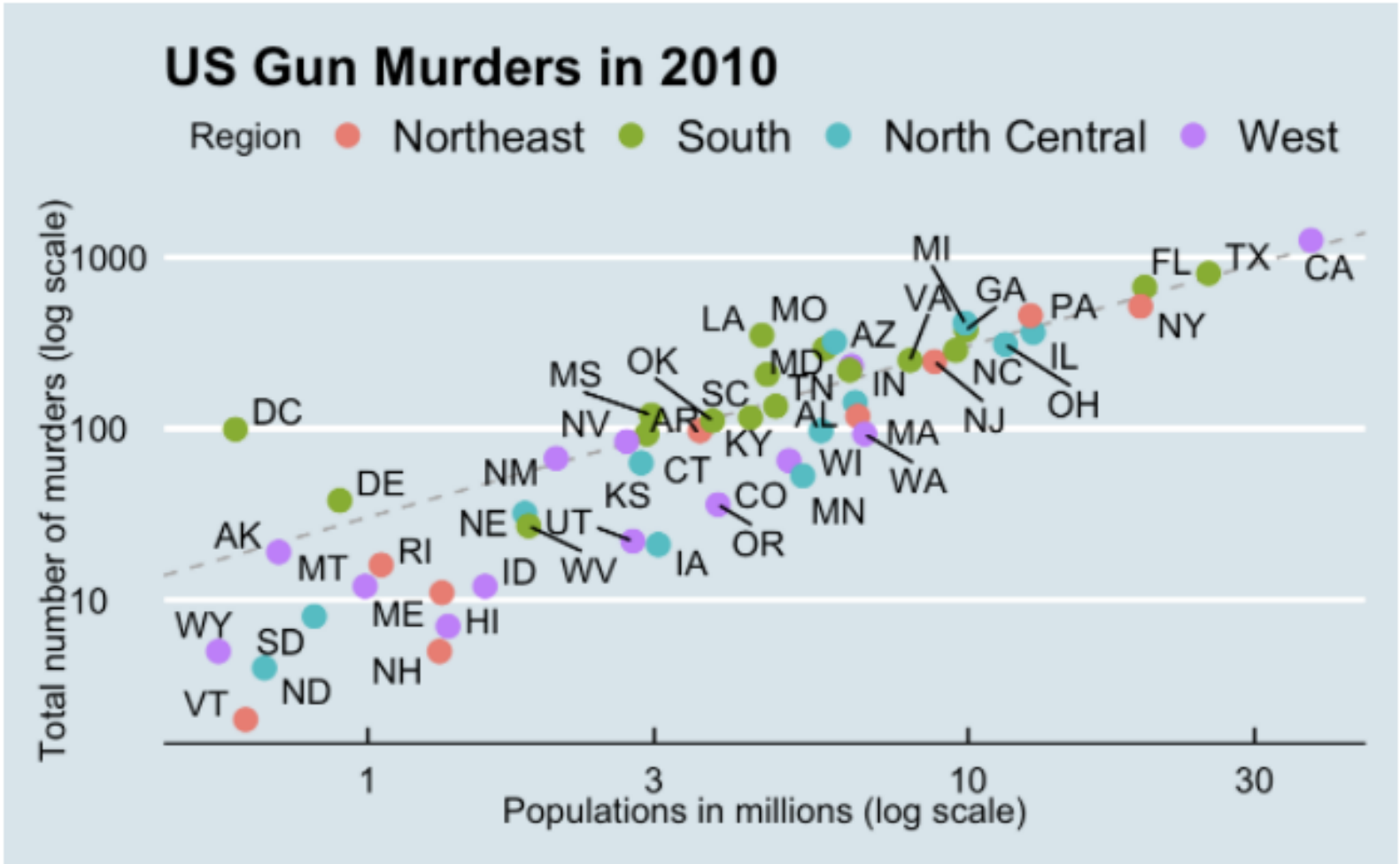
Pusan National University
Department of Statistics

Loading libraries

```
library(dplyr)
library(ggplot2)
```

The components of a graph

We will construct a graph that summarizes the US murders dataset that looks like this:



The components of a graph

This data visualization shows us pretty much all the information in the data table. The code needed to make this plot is relatively simple. The first step in learning ggplot2 is to be able to break a graph apart into components. 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.
- **Aesthetic mapping** : The plot uses several visual cues to represent the information provided by the dataset.

ggplot objects

We start by loading the dataset:

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

ggplot objects

The first step in creating a ggplot2 graph is to define a `ggplot` object.

```
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:

```
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.

ggplot objects

We can assign our plot to 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 each produce the same plot we see above:

```
print(p)
p
```

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.

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

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)`:

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

Layers

The `geom_label` and `geom_text` functions permit us to add text to the plot with and without a rectangle behind the text, respectively.

```
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.

Tinkering with arguments

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))
```

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:

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

Global versus local aesthetic

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

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

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

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

Global versus local 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:

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

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. We use them 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.

Scales

This particular transformation is so common that **ggplot2** provides the specialized functions : `scale_x_log10` and `scale_y_log10`

```
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:

```
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.

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:

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

Categories as colors

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. To map each point to a color, we need to use `aes`. We use the following code:

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

To avoid adding this legend we set the `geom_point` argument `show.legend = FALSE`.

Annnotation, 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. To compute this value, we use our dplyr skills:

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

To add a line we use the `geom_abline` function.

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

Annnotation, shapes, and adjustments

We can change the line type and color of the lines using arguments.

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

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:

```
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 needed to put the finishing touches on our plot require the **ggthemes** and **ggrepel** packages.

The style of a **ggplot2** graph can be changed using the `theme` functions. Several themes are included as part of the **ggplot2** package. we use a function in the **dslabs** package that automatically sets a default theme:

```
ds_theme_set()
```

Add-on packages

Many other themes are added by the package **ggthemes**. Among those are the theme_economist theme that we used.

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

You can see how some of the other themes look by simply changing the function.

```
p + theme_fivethirtyeight()
```

Putting it all together

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

```
library(ggthemes)
library(ggrepel)

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

Putting it all together

```
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()
```

Quick plots with qplot

We demonstrated how to generate these plots with `hist`, `plot`, and `boxplot`. However, if we want to keep consistent with the ggplot style, we can use the function `qplot`.

If we have values in two vectors, say:

```
data(murders)
x <- log10(murders$population)
y <- murders$total
```

The `qplot` function sacrifices the flexibility provided by the `ggplot` approach, but allows us to generate a plot quickly.

```
qplot(x, y)
```

Grids of plots

There are often reasons to graph plots next to each other. The **gridExtra** package permits us to do that:

```
library(gridExtra)
p1 <- qplot(x)
p2 <- qplot(x,y)
grid.arrange(p1, p2, ncol = 2)
```

Exercises

Start by loading the **dplyr** and **ggplot2** library as well as the `murders` and `heights` data.

```
library(dplyr)
library(ggplot2)
library(dslabs)
data(heights)
data(murders)
```

Exercises

1. With **ggplot2** plots can be saved as objects. For example we can associate a dataset with a plot object like this

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

What is class of the object p?

Exercises

2. Remember that to print an object you can use the command `print` or simply type the object. Print the object `p` defined in exercise one and describe what you see.

a) Nothing happens.

b) A blank slate plot.

c) A scatterplot.

d) A histogram.

Exercises

3. Using the pipe %>%, create an object `p` but this time associated with the `heights` dataset instead of the `murders` dataset.

4. What is the class of the object `p` you have just created?

Exercises

5. For the murders data we plotted total murders versus population sizes. Explore the `murders` data frame to remind yourself what are the names for these two variables and select the correct answer. **Hint**: Look at `?murders`.

a) **state** and **abb**.

b) **total_murders** and **population_size**.

c) **total** and **population**.

d) **murders** and **size**.

Exercises

6. To create the scatterplot we add a layer with `geom_point`.

```
murders %>% ggplot(aes(x = , y = )) +
  geom_point()
```

except we have to define the two variables x and y. Fill this out with the correct variable names.

Exercises

7. Note that if we don't use argument names, we can obtain the same plot by making sure we enter the variable names in the right order like this:

```
murders %>% ggplot(aes(population, total)) +
  geom_point()
```

Remake the plot but now with total in the x-axis and population in the y-axis.

Exercises

8. If instead of points we want to add text, we can use the `geom_text()` or `geom_label()` geometries. The following code

```
murders %>% ggplot(aes(population, total)) + geom_label()
```

will give us the error message. Why is this?

a) We need to map a character to each point through the label argument in aes.

b) We need to let `geom_label` know what character to use in the plot.

c) The `geom_label` geometry does not require x-axis and y-axis values.

d) `geom_label` is not a ggplot2 command.

Exercises

9. Change the color of the labels to blue. How will we do this?

a) Adding a column called `blue` to `murders`.

b) Because each label needs a different color we map the colors through `aes`.

c) Use the `color` argument in `ggplot`.

d) Because we want all colors to be blue, we do not need to map colors, just use the color argument in `geom_label`.

Exercises

10. Now suppose we want to use color to represent the different regions. In this case which of the following is most appropriate:

a) Adding a column called `color` to `murders` with the color we want to use.

b) Because each label needs a different color we map the colors through the color argument of `aes` .

c) Use the `color` argument in `ggplot`.

d) Because we want all colors to be blue, we do not need to map colors, just use the color argument in `geom_label`.

Exercises

11. Now we are going to change the x-axis to a log scale to account for the fact the distribution of population is skewed. Let's start by defining an object `p` holding the plot we have made up to now

```
p <- murders %>%
  ggplot(aes(population, total, label = abb, color = region))
  geom_label()
```

To change the y-axis to a log scale we learned about the `scale_x_log10()` function. Add this layer to the object `p` to change the scale and render the plot.

Exercises

12. Repeat the previous exercise but now change both axes to be in the log scale

13. Now edit the code above to add the title "Gun murder data" to the plot. Hint: use the `ggtitle` function.