Data visualization 1

STA 032: Gateway to data science Lecture 7

Jingwei Xiong

April 17, 2023

Reminders

- HW 1 is due April 17th at 12pm
- HW 2 due April 26 12pm.
- Please start the homework as soon as possible.
- Discussion will cover homework problems.

Recap

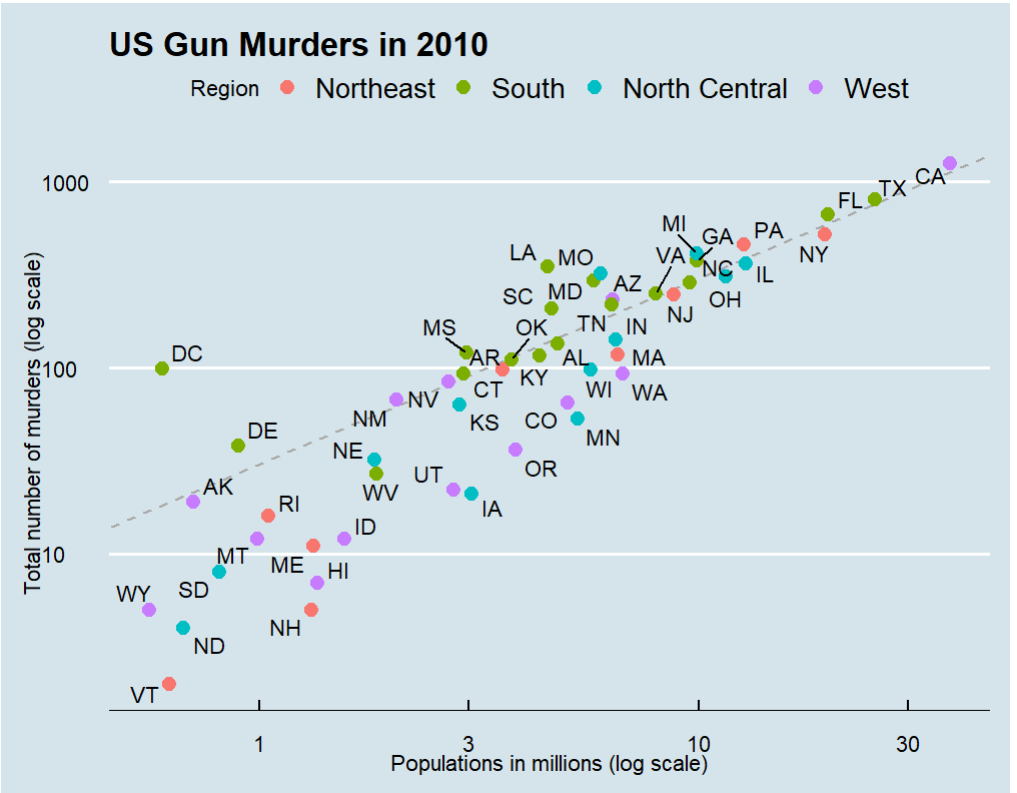
- Data manipulation tools in `tidyverse`
- Data visualization introduction examples

> Remember, before using all tidyverse functions, you need to library(tidyverse) first!
>
> Remember, before using all ggplot2 functions, you need to library(ggplot2) first!

Today

Generate this plot:

Picture | Code



Today

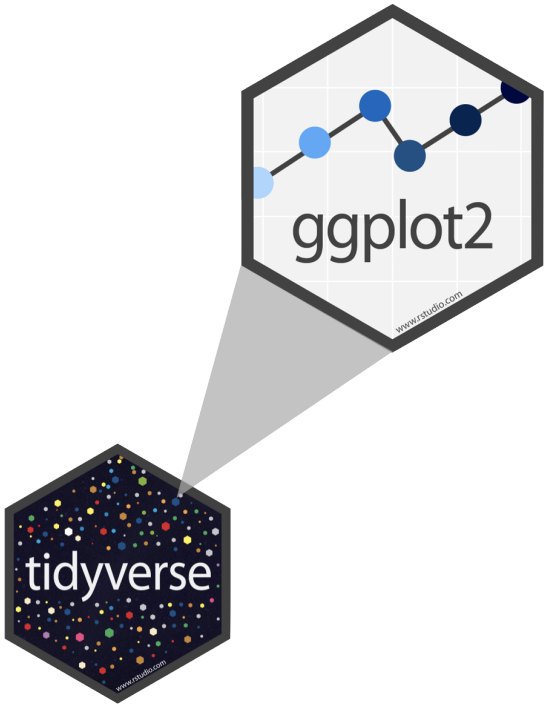
Generate this plot:

Picture | Code

```
library(tidyverse)
library(ggthemes)
library(ggrepel)
library(ggplot2)
r <- murders |>
  summarize(pop=sum(population), tot=sum(total)) |>
  mutate(rate = tot/pop*10^6) |> pull(rate)
murders |> ggplot(aes(x = population/10^6, y = total, label = abb)) +
  geom_abline(intercept = log10(r), lty=2, col="darkgrey") +
  geom_point(aes(color=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 using `ggplot2`

Slide | Words 1 | Words 2



- ggplot2 is the tidyverse's data visualization package
- gg in "ggplot2" stands for Grammar of Graphics
- Inspired by the book Grammar of Graphics by Leland Wilkinson
- We will also look at some plotting functions in base R

- ggplot2 is the tidyverse's data visualization package

Data visualization using `ggplot2`

Slide | Words 1 | Words 2

Throughout the lecture, we will be creating plots using the **ggplot2**^[https://ggplot2.tidyverse.org/] package.

Many other approaches are available for creating plots in R. We chose to use **ggplot2** 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 graphics^[http://www.springer.com/us/book/9780387245447], 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.

Data visualization using `ggplot2`

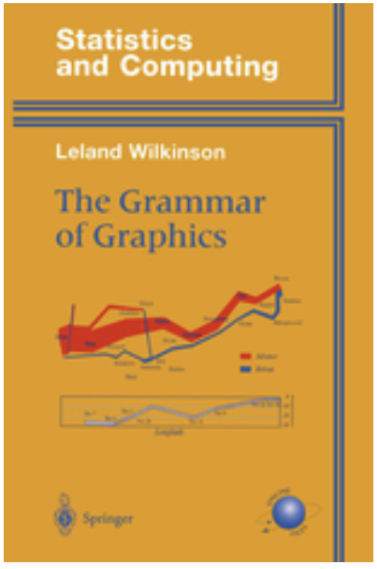
Another reason **ggplot2** is easy for beginners is that it is possible to create informative and elegant graphs with relatively simple and readable code.

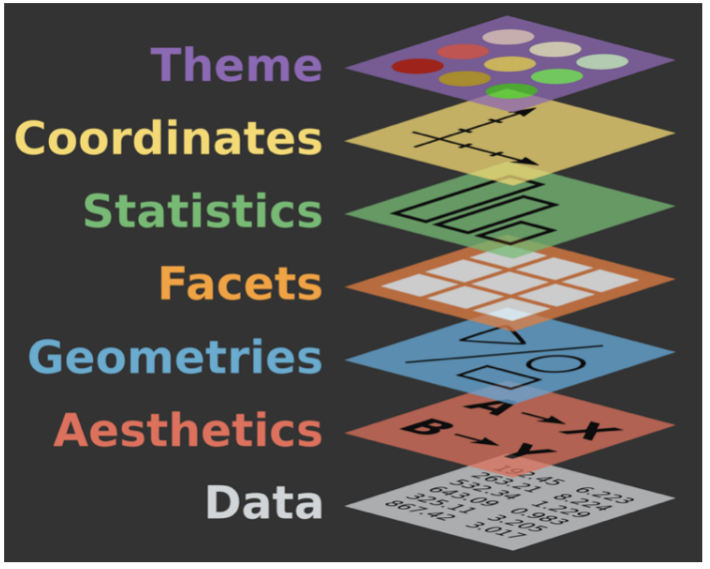
To use **ggplot2** you will have to learn several functions and arguments. These commands may be hard to memorize, but you can always return back to this tutorial and grab the code you want. Or you can simply perform an internet search for ggplot2 cheat sheet.

Grammar of Graphics

Slide | Words

A grammar of graphics is a tool that enables us to concisely describe the components of a graphic





Source: BloggoType

How these are implemented in `ggplot2`:
https://ggplot2.tidyverse.org/reference/

Grammar of Graphics

Slide | Words

Some of the key components of a graphic that can be described using the grammar of graphics include:

Data: the dataset being visualized

Aesthetics: the visual properties of the plot, such as color or size

Geometries: the visual elements that represent the data, such as points or lines

Scales: the mapping between data values and their visual representation on the plot

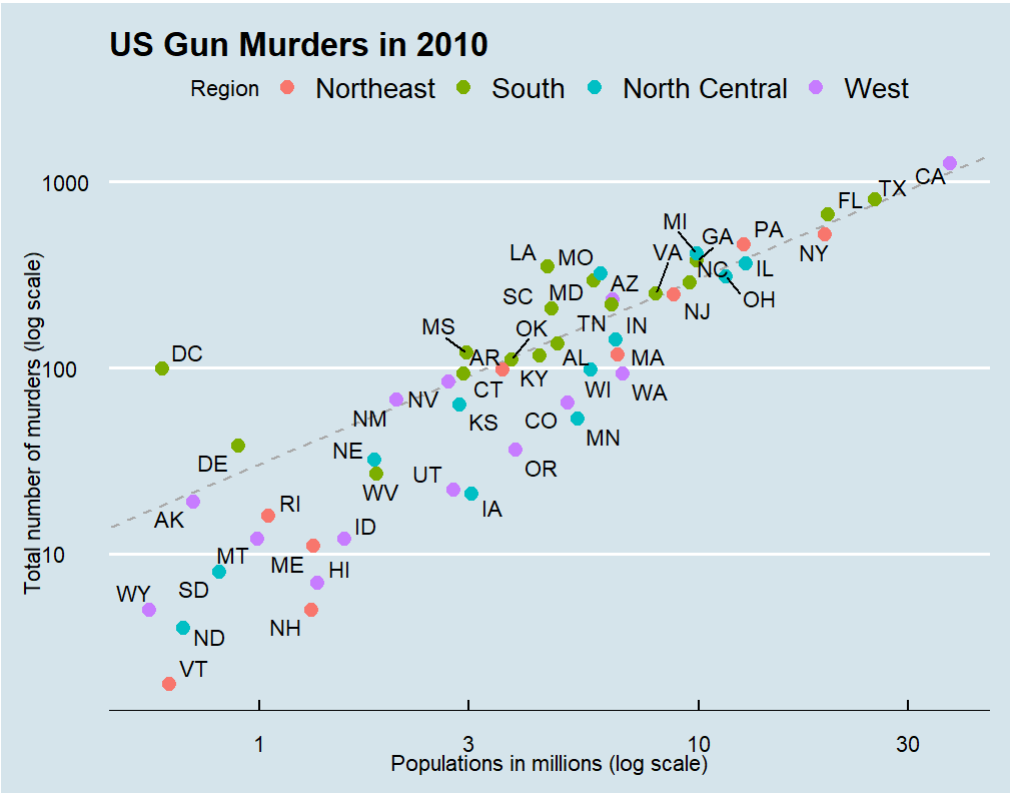
Facets: a way to split the data into smaller subsets and create multiple plots

Themes: the overall look and feel of the plot, such as font size or background color

Source: BloggoType

The components of a graph

Picture | Code | Words



The components of a graph

Picture | Code | Words

```
library(tidyverse)
library(ggthemes)
library(ggrepel)
library(ggplot2)
r <- murders |>
  summarize(pop=sum(population), tot=sum(total)) |>
  mutate(rate = tot/pop*10^6) |> pull(rate)
murders |> ggplot(aes(x = population/10^6, y = total, label = abb)) +
  geom_abline(intercept = log10(r), lty=2, col="darkgrey") +
  geom_point(aes(color=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()
```

The components of a graph

Picture Code Words

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 dataset 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.
- **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.

ggplot2

Overview | ggplot() | continue

- `ggplot()` is the main function in the `ggplot2` package
- Plots are constructed in layers concat by `+`
- Structure of the code for plots can be summarized as

```
ggplot(data = [dataset],
       mapping = aes(x = [x-variable], y = [y-variable])) +
   geom_xxx() +
   other options
```

ggplot2

Overview | ggplot() | continue

First step: define a `ggplot` object

```
# load the data:
library(dslabs); dataset(murders)
ggplot(data = murders)
```

We can also pipe the data in as the first argument:

```
murders |> ggplot()
```

ggplot2

It renders a plot, but it's blank since no geometry (plot) has been defined.

What has happened above is that the object was created and, because it was not assigned, it was automatically evaluated.

(just like 1 + 1) But 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
```

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`.

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`.

Aesthetic mappings

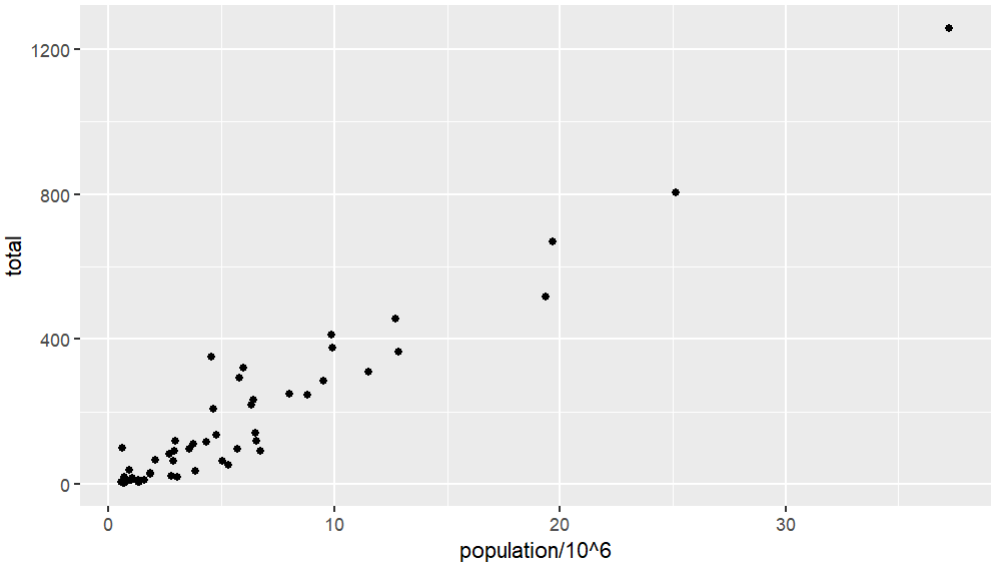
Code | Result | Words

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

- X axis: (state) population/10^6
- Y axis: total (murders)

Aesthetic mappings

Code | Result | Words



Aesthetic mappings

Code Result Words

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.

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.

> 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`.

Add text Layers

Code | Result | Word

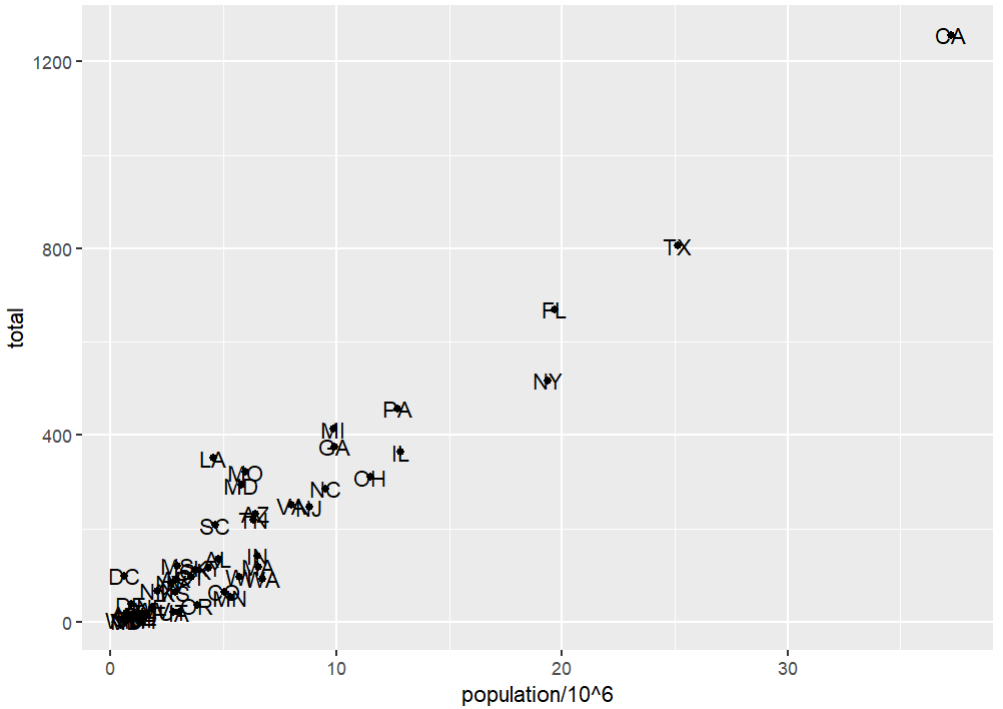
```
p = murders |> ggplot() +
  geom_point(aes(x = population/10^6, y = total)) +
  geom_text(aes(x = population/10^6, y = total, label = abb))

p
```

- mapping between point and label through the `label` argument of `aes`: `label = abb`

Add text Layers

Code | **Result** | Word



- We have successfully added a second layer to the plot. (Though the labels are quite messy)

Add text Layers

Code Result Word

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:

We have successfully added a second layer to the plot. (Though the labels are quite messy)

Tinkering with arguments

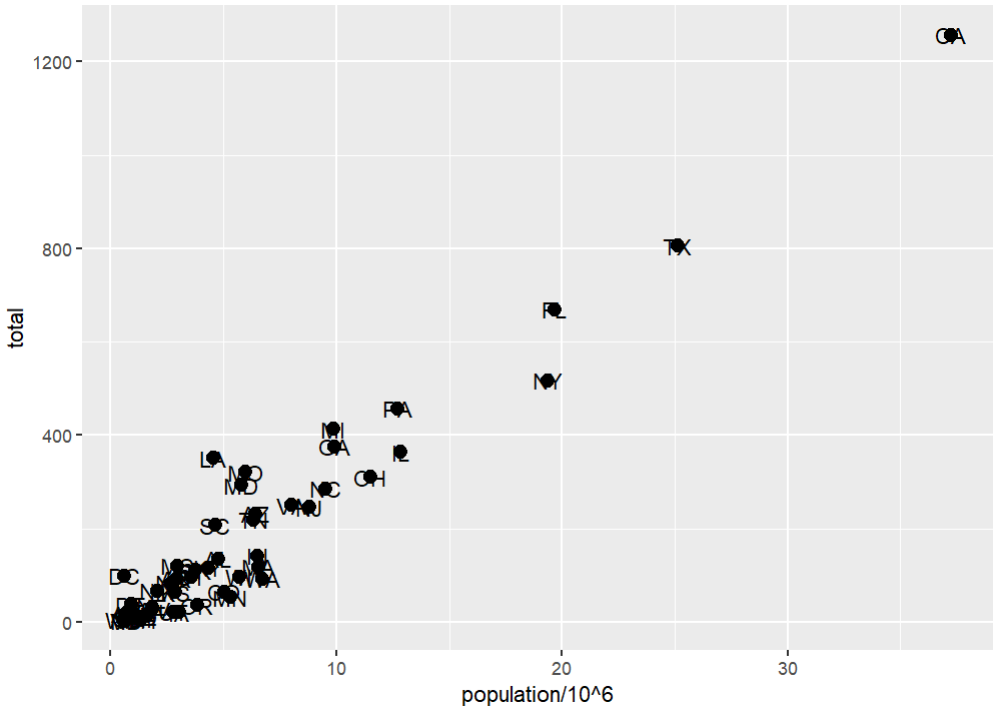
Code | Result | Code2 | Result2 | Word

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

- Change `size=3` for another point size.
- It is **not** inside `aes()` so it applies to **ALL** points!

Tinkering with arguments

Code | **Result** | Code2 | Result2 | Word



- It is **not** inside `aes()` so it applies to **ALL** points!

Tinkering with arguments

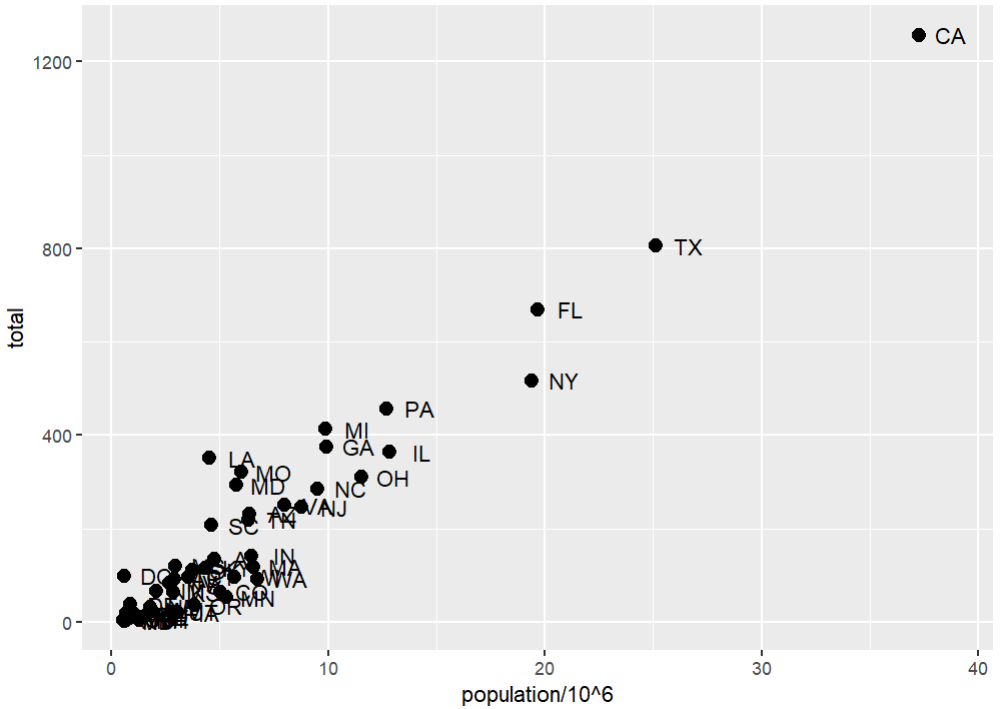
Code | Result | Code2 | Result2 | Word

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

- `nudge_x` argument moves the text slightly to the right or to the left.
- It is **not** inside `aes()` so it applies to **ALL** points!

Tinkering with arguments

Code | Result | Code2 | **Result2** | Word



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

Tinkering with arguments

Code | Result | Code2 | Result2 | Word

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:

In this case `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.

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

Global versus local aesthetic mappings

Global mapping | part 2 | Override | Plot

In the previous line of code, we define the mapping `aes(population/10^6, total)` twice, once in each geometry. We can do this when we define the blank slate `ggplot` object.

By using a **global** aesthetic mapping:

```
p = murders |>
  ggplot(aes(population/10^6, total, label = abb))
# the aes(population/10^6, total, label = abb)
# will be used for every layer
```

- The `aes(population/10^6, total, label = abb)` will be used for every layer
- For the `aes()`, the default first and second arguments are `x` and `y`.

Global versus local aesthetic mappings

Global mapping | part 2 | Override | Plot

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

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

Compared to before:

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

- The `geom_point` function does not need a `label` argument and therefore **ignores** that aesthetic.

Global versus local aesthetic mappings

Global mapping | part 2 | Override | Plot

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:

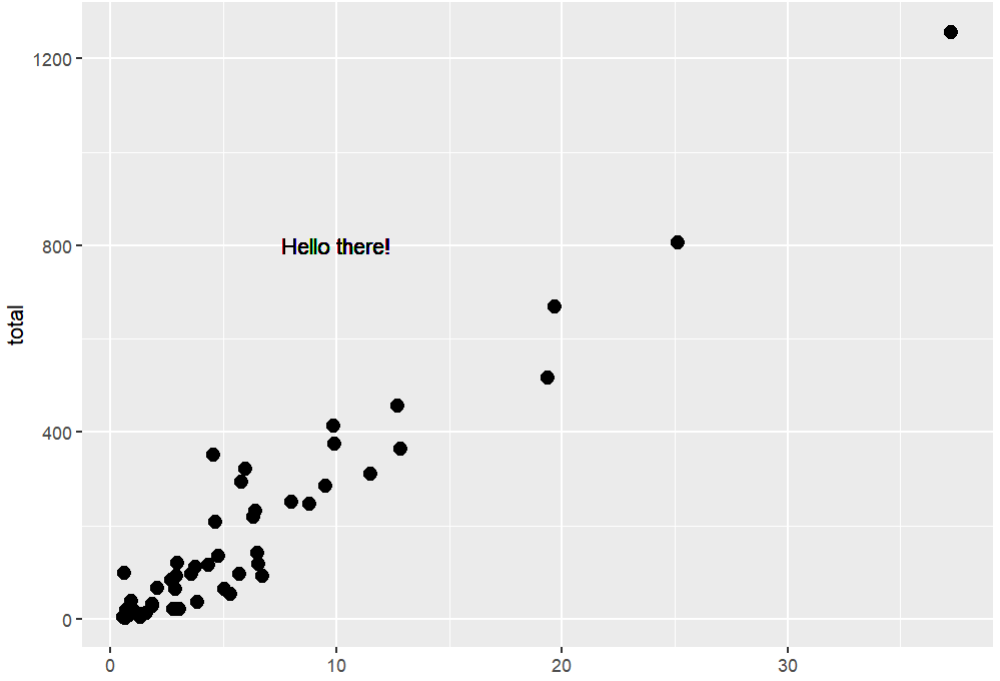
```
murders |>
  ggplot(aes(population/10^6, total, label = abb)) +
  geom_point(size = 3) +
  geom_text(aes(x = 10, y = 800, label = "Hello there!"))
```

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

Global versus local aesthetic mappings

Global mapping | part 2 | Override | Plot

```
murders |>
  ggplot(aes(population/10^6, total, label = abb)) +
  geom_point(size = 3) +
  geom_text(aes(x = 10, y = 800, label = "Hello there!"))
```



Scales

Scales | Plot | Alias

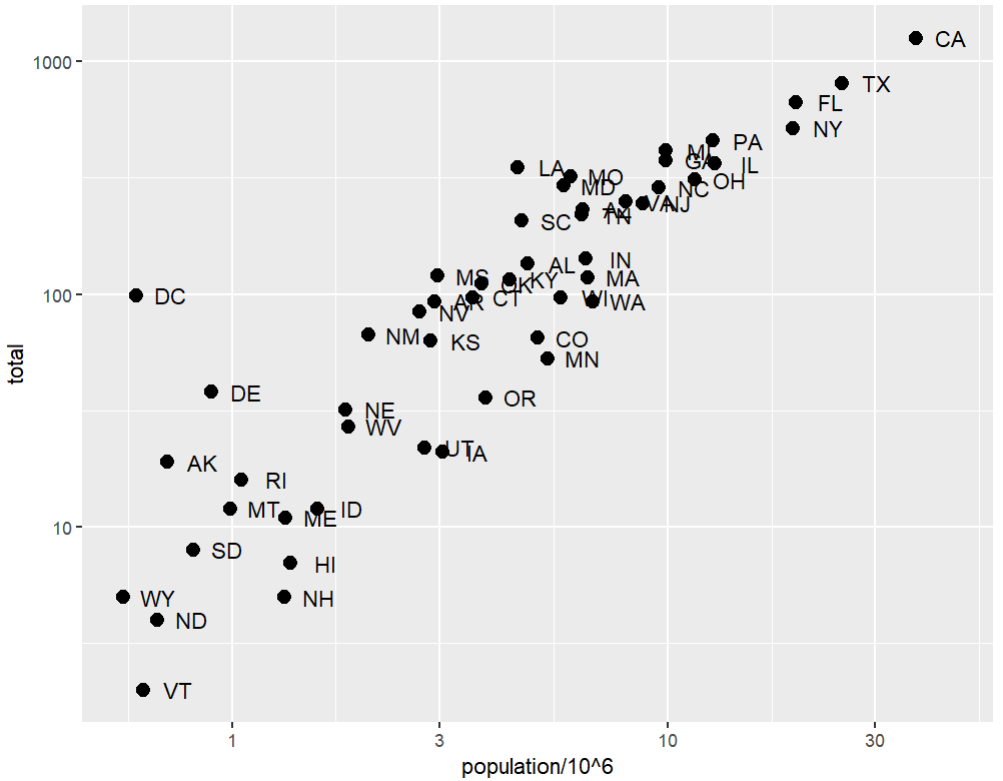
- The points are too crowded in the bottom left.
- Our desired scales are in log-scale
- use the `scale_x_continuous` function.

```
p = murders |>
  ggplot(aes(population/10^6, total, label = abb)) +
  geom_point(size = 3) +
  geom_text(nudge_x = 0.08) +
  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

Scales | Plot | Alias



Scales

Scales | Plot | Alias

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:

```
murders |>
  ggplot(aes(population/10^6, total, label = abb)) +
  geom_point(size = 3) +
  geom_text(nudge_x = 0.08) +
  scale_x_log10() +
  scale_y_log10()
```

Labels and titles

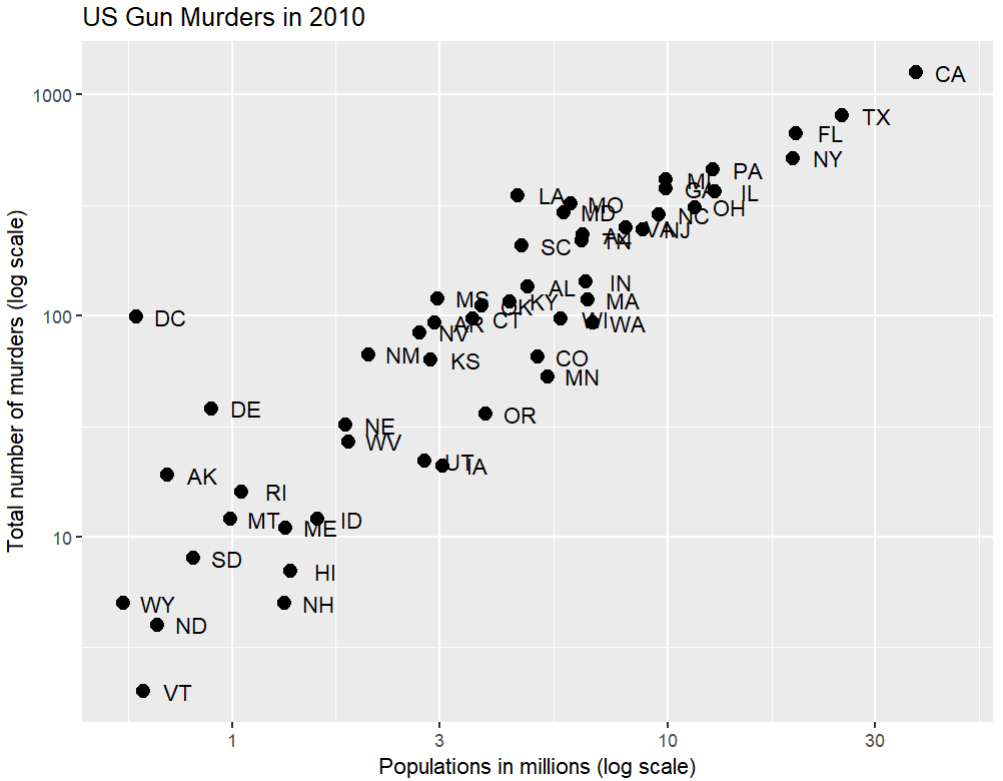
Code | Result

- To change labels and title we can:

```
p = murders |>
  ggplot(aes(population/10^6, total, label = abb)) +
  geom_point(size = 3) +
  geom_text(nudge_x = 0.08) +
  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")
```

Labels and titles

Code | Result



Categories as colors

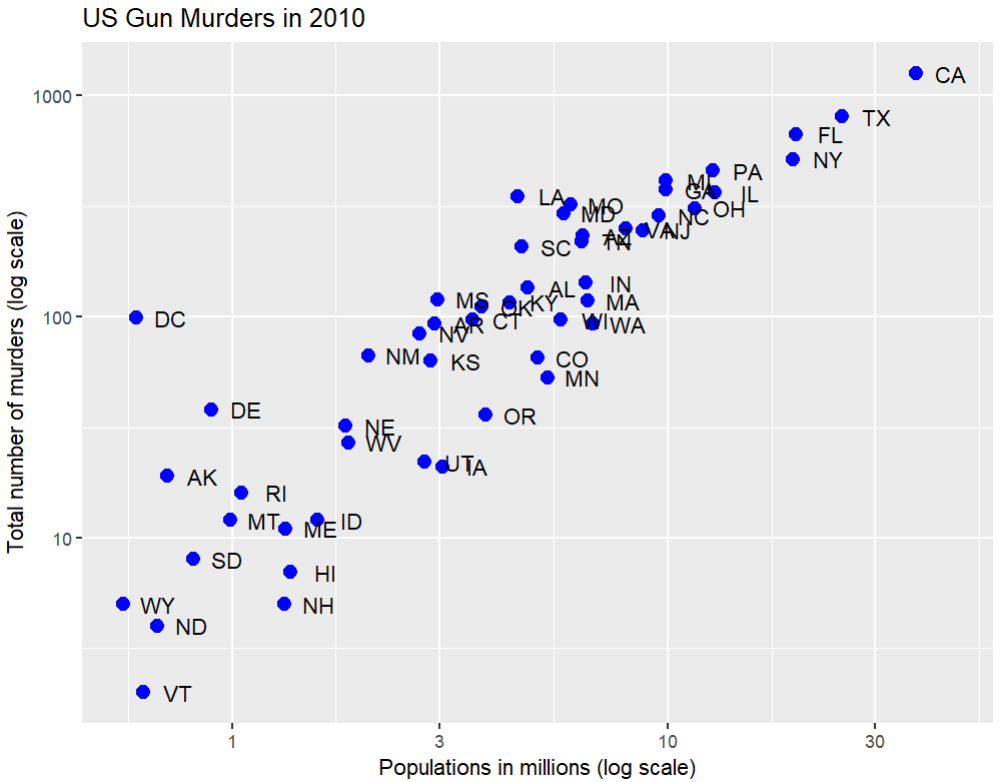
Code 1 | Result 1 | Code 2 | Result 2 | Suppress legend | Result 3 | Change legend name | Result 4

- We can change the color of the points using the `col` argument in the `geom_point` function.

```
p <-  murders |> ggplot(aes(population/10^6, total, label = abb)) +
  geom_point(size = 3, color ="blue") +
  geom_text(nudge_x = 0.08) +
  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

Code 1 | **Result 1** | Code 2 | Result 2 | Suppress legend | Result 3 | Change legend name | Result 4



Categories as colors

Code 1 | Result 1 | **Code 2** | Result 2 | Suppress legend | Result 3 | Change legend name | Result 4

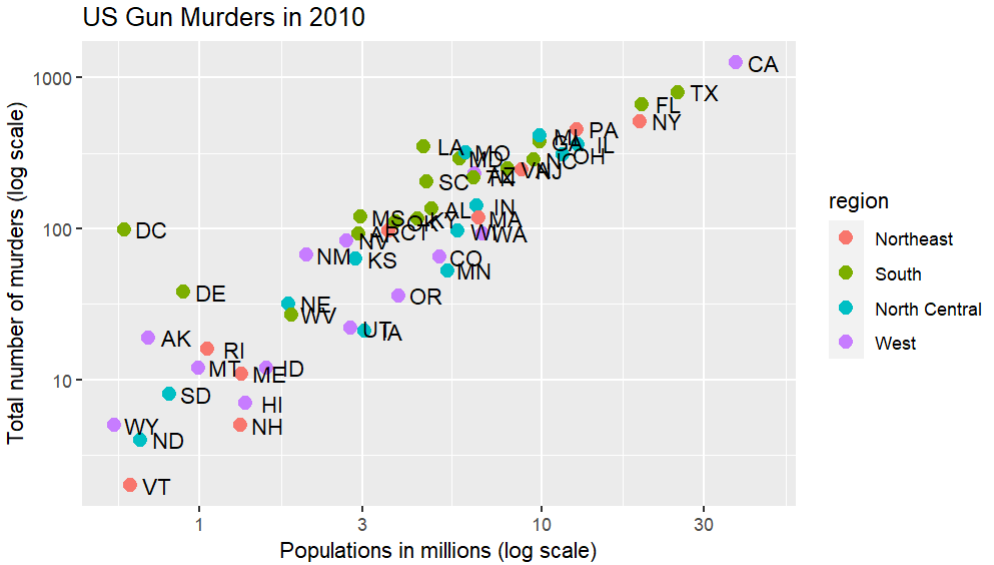
- We want to assign color depending on the **geographical region**.
- in **ggplot2** if we assign a categorical variable to color in **aes mapping**, it automatically assigns a different color to each category and also adds a legend.

```
p <-  murders |> ggplot(aes(population/10^6, total, label = abb)) +
  geom_point(aes(color = region), size = 3) +
  geom_text(nudge_x = 0.08) +
  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

Code 1 | Result 1 | Code 2 | **Result 2** | Suppress legend | Result 3 | Change legend name | Result 4

- Here we see yet another useful default behavior: **ggplot2 automatically adds a legend that maps color to region**.



Categories as colors

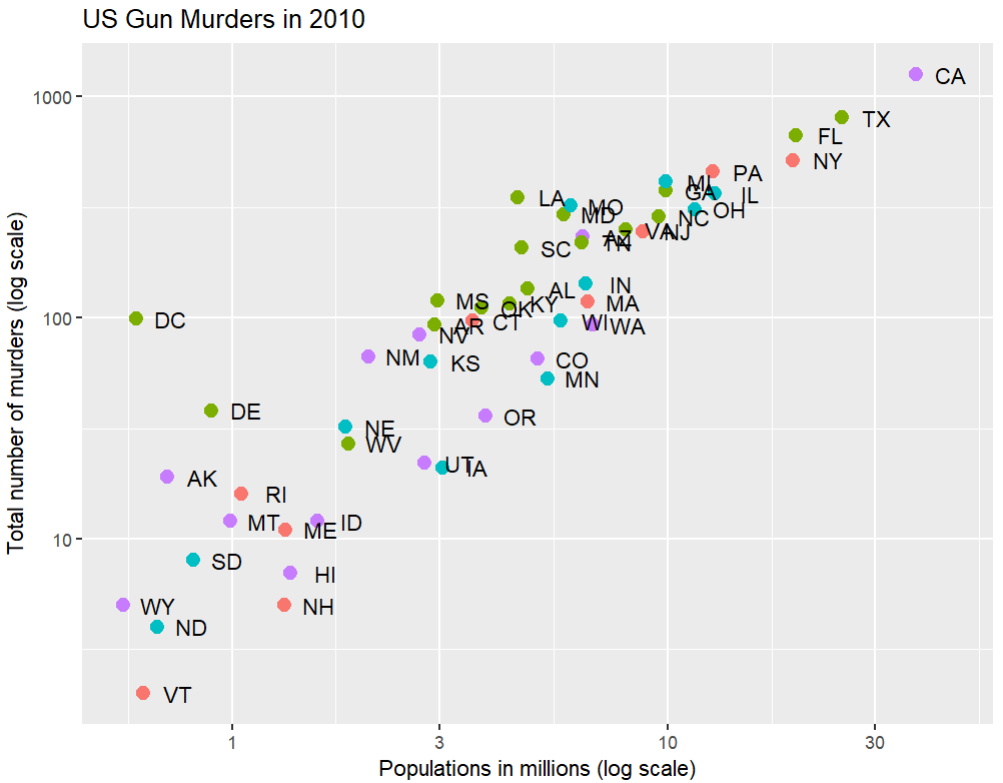
Code 1 | Result 1 | Code 2 | Result 2 | **Suppress legend** | Result 3 | Change legend name | Result 4

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

```
p <-  murders |> ggplot(aes(population/10^6, total, label = abb)) +
  geom_point(aes(color = region), size = 3,
             show.legend = FALSE) +
  geom_text(nudge_x = 0.08) +
  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

Code 1 | Result 1 | Code 2 | Result 2 | Suppress legend | **Result 3** | Change legend name | Result 4



Categories as colors

Code 1 | Result 1 | Code 2 | Result 2 | Suppress legend | Result 3

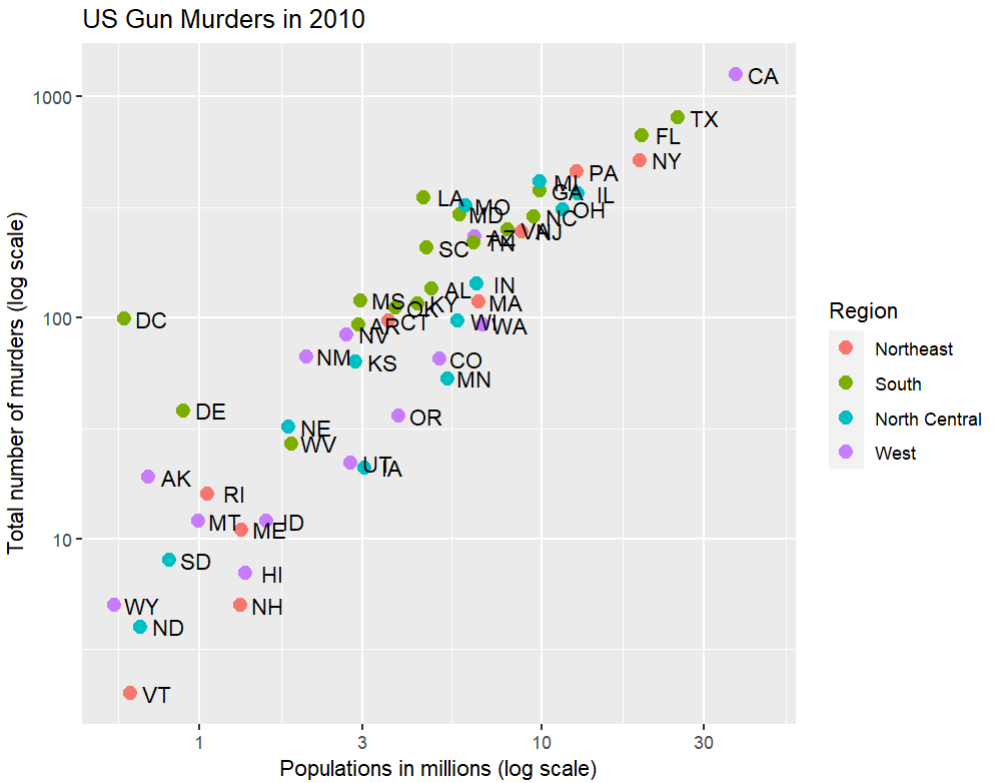
Change legend name | Result 4

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 <-  murders |> ggplot(aes(population/10^6, total, label = abb)) +
  geom_point(aes(color = region), size = 3) +
  geom_text(nudge_x = 0.08) +
  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")
p
```

Categories as colors

Code 1 | Result 1 | Code 2 | Result 2 | Suppress legend | Result 3 | Change legend name | Result 4



Add a line using geom_abline (Extra reading)

Calculate the avg rate | Add the line | Change line type | Result

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: $y = rx$, 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(rate = sum(total) /  sum(population) * 10^6) |>
  pull(rate)
```

Add a line using geom_abline (Extra reading)

Calculate the avg rate | **Add the line** | Change line type | Result

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`).

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

```
murders |> ggplot(aes(population/10^6, total, label = abb)) +
  geom_point(aes(color = region), size = 3,
             show.legend = FALSE) +
  geom_text(nudge_x = 0.08) +
  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") +
  geom_abline(slope = 1, intercept = log10(r))
```

Add a line using geom_abline (Extra reading)

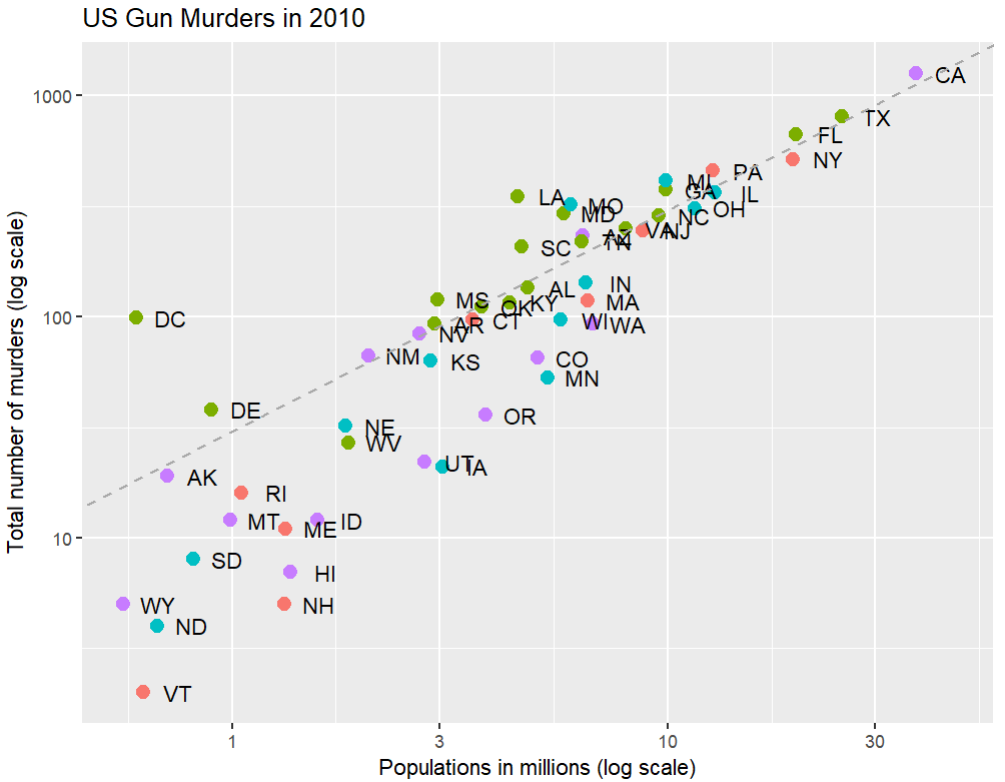
Calculate the avg rate | Add the line | **Change line type** | Result

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.

```
murders |> ggplot(aes(population/10^6, total, label = abb)) +
  geom_point(aes(color = region), size = 3,
             show.legend = FALSE) +
  geom_text(nudge_x = 0.08) +
  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") +
  geom_abline(intercept = log10(r), lty = 2, color = "darkgrey")
```

Add a line using geom_abline (Extra reading)

Calculate the avg rate | Add the line | Change line type | **Result**



Add-on packages

Intro | ggthemes | another theme | ggrepel

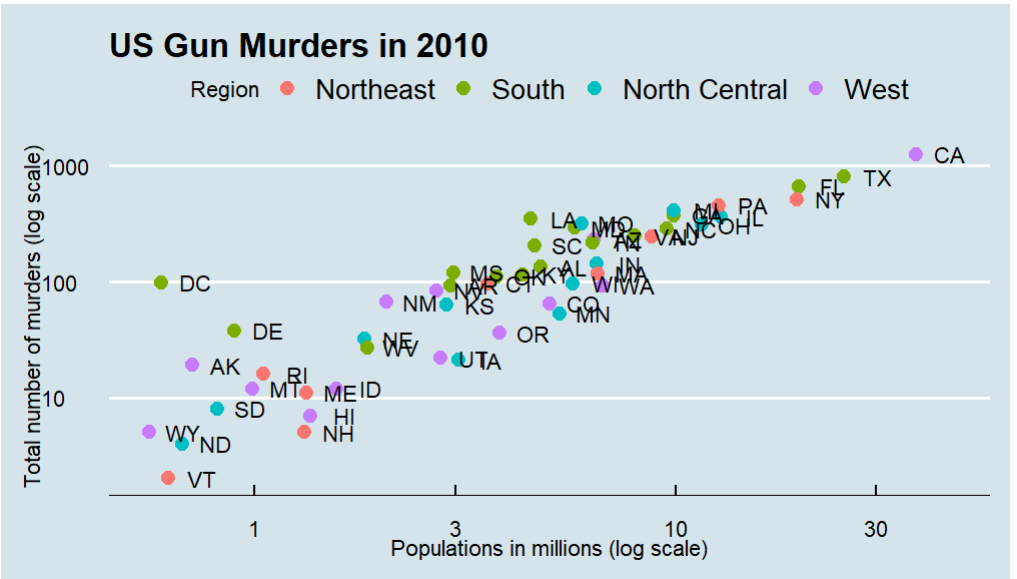
- The power of **ggplot2** is augmented further due to the availability of add-on packages.
- These packages provide additional geoms, scales, and themes that can be used to create more complex and sophisticated visualizations.
- Some popular ggplot2 add-on packages include:
  - **ggthemes**: provides additional themes for ggplot2 plots, including options for customizing plot backgrounds, fonts, and colors.
  - **ggmap**: allows for the integration of maps and geographic data into ggplot2 plots.
  - **ggrepel**: adds options for preventing text labels from overlapping on ggplot2 plots.
  - **plotly**: allows for the creation of interactive ggplot2 plots that can be explored and manipulated in a web browser.
  - **ggpubr**: provides several functions to customize the appearance of plots
  - **ggridge**: provides functions for creating ridge plots

Add-on packages

Intro | ggthemes | another theme | ggrepel

- After installing the `ggthemes` package, you can change the style by adding a layer like this:

```
library(ggthemes)
# We have already define p in previous part
p + theme_economist()
```

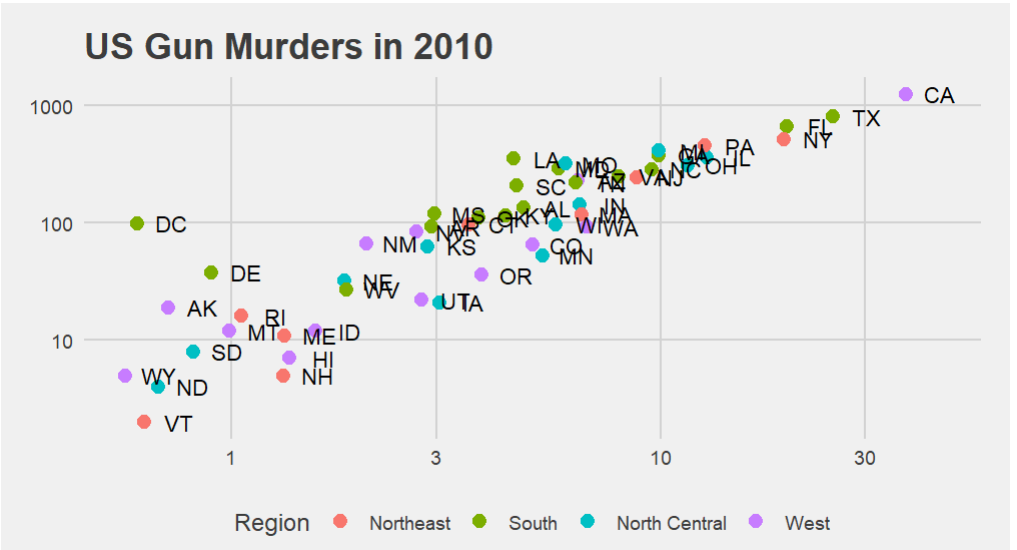


Add-on packages

Intro | ggthemes | another theme | ggrepel

- You can see how some of the other themes look by simply changing the function. For instance, you might try the `theme_fivethirtyeight()` theme instead.
- link of possible themes of ggthemes

```
p + theme_fivethirtyeight()
```



Add-on packages

- The add-on package **ggrepel** includes a geometry that adds labels while ensuring that they don't fall on top of each other.
- We simply change `geom_text` with `geom_text_repel`. Don't forget to `install.packages(ggrepel)` and `library(ggrepel)`!

```
#install.packages(ggrepel)
library(ggrepel)
# Picture and code in the next slide
```

Putting it all together

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

Code | Plot

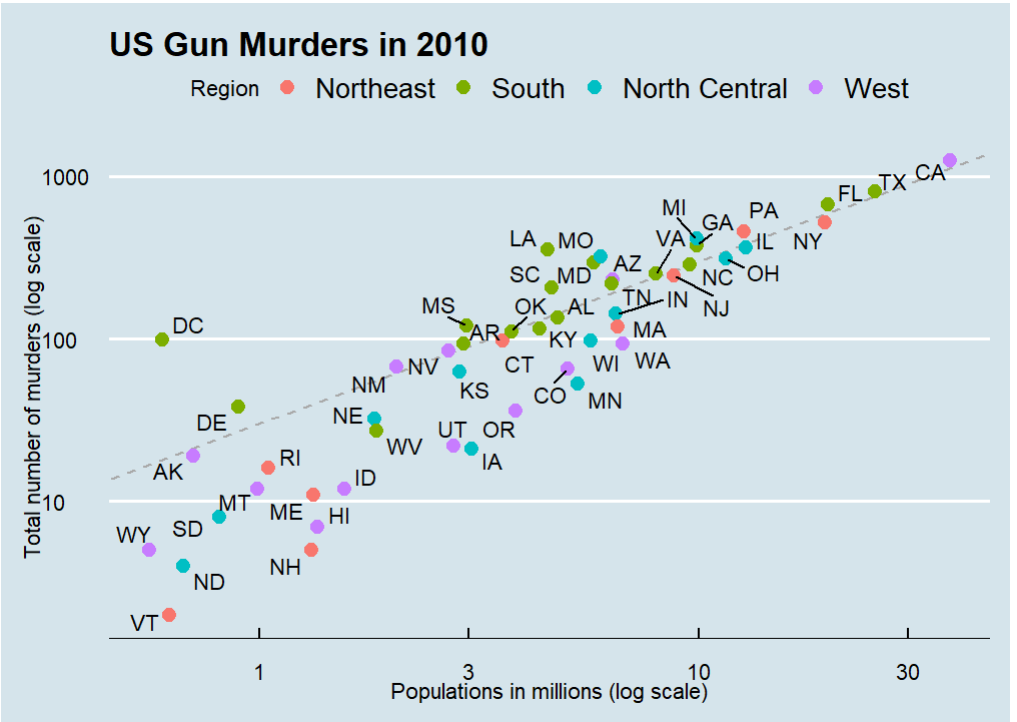
```
library(ggthemes)
library(ggrepel)
r <- murders |>
  summarize(rate = sum(total) /  sum(population) * 10^6) |>
  pull(rate)

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()
```

Putting it all together

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

Code | Plot



A note on piping and layering

- Pipe `%>%` used mainly in `dplyr` pipelines
    - Pipe the output of the previous line of code as the first input of the next line of code
- `+` used in `ggplot2` plots is used for "layering"
    - Create the plot in layers, separated by `+`

dplyr

❌

```
hotels +
  select(hotel, lead_time)
```

```
Error in select(hotel, lead_time): object 'hotel' not found
```

✅

```
hotels %>%
  select(hotel, lead_time)
```

```
# A tibble: 119,390 × 2
   hotel        lead_time
   <chr>            <dbl>
 1 Resort Hotel       342
 2 Resort Hotel       737
 3 Resort Hotel         7
 4 Resort Hotel        13
 5 Resort Hotel        14
 6 Resort Hotel        14
 7 Resort Hotel         0
 8 Resort Hotel         9
 9 Resort Hotel        85
10 Resort Hotel        75
# … with 119,380 more rows
```

ggplot2

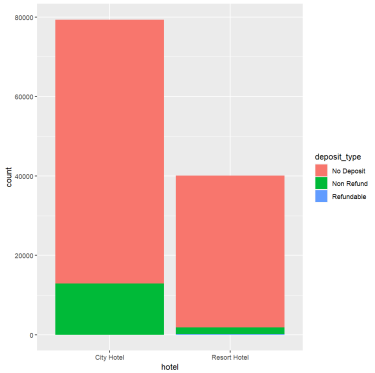
❌

```
ggplot(hotels, aes(x = hotel, fill = deposit_type)) %>%
  geom_bar()
```

```
Error in geom_bar():
! mapping must be created by aes()
ℹ Did you use %>% or |> instead of +?
```

✅

```
ggplot(hotels, aes(x = hotel, fill = deposit_type)) +
  geom_bar()
```



Code styling

Many of the styling principles are consistent across %>% and +:

- always a space before
- always a line break after (for pipelines with more than 2 lines)

❌

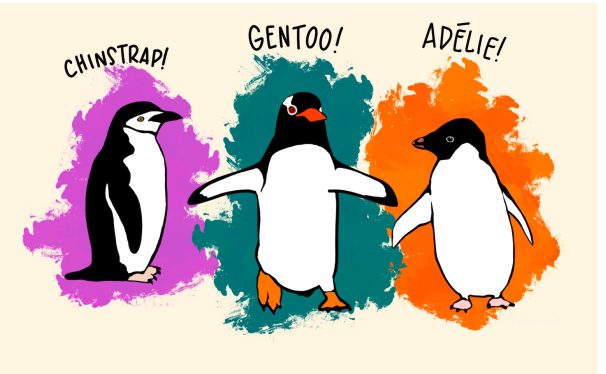
```
ggplot(hotels,aes(x=hotel,y=deposit_type))+geom_bar()
```

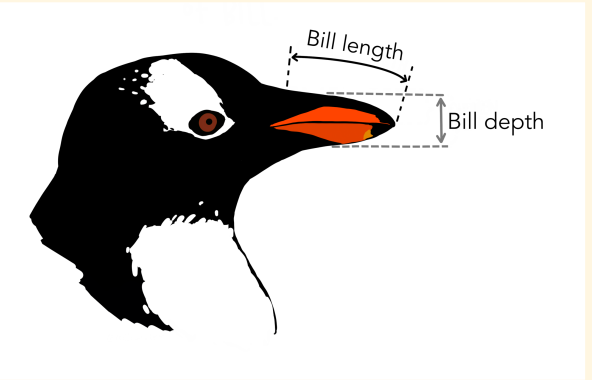
✅

```
ggplot(hotels, aes(x = hotel, y = deposit_type)) +
  geom_bar()
```

Another example: Palmer Penguins

Data contains information on 344 penguins, including: penguin species, island in Palmer Archipelago, size (flipper length, body mass, bill dimensions), and sex.

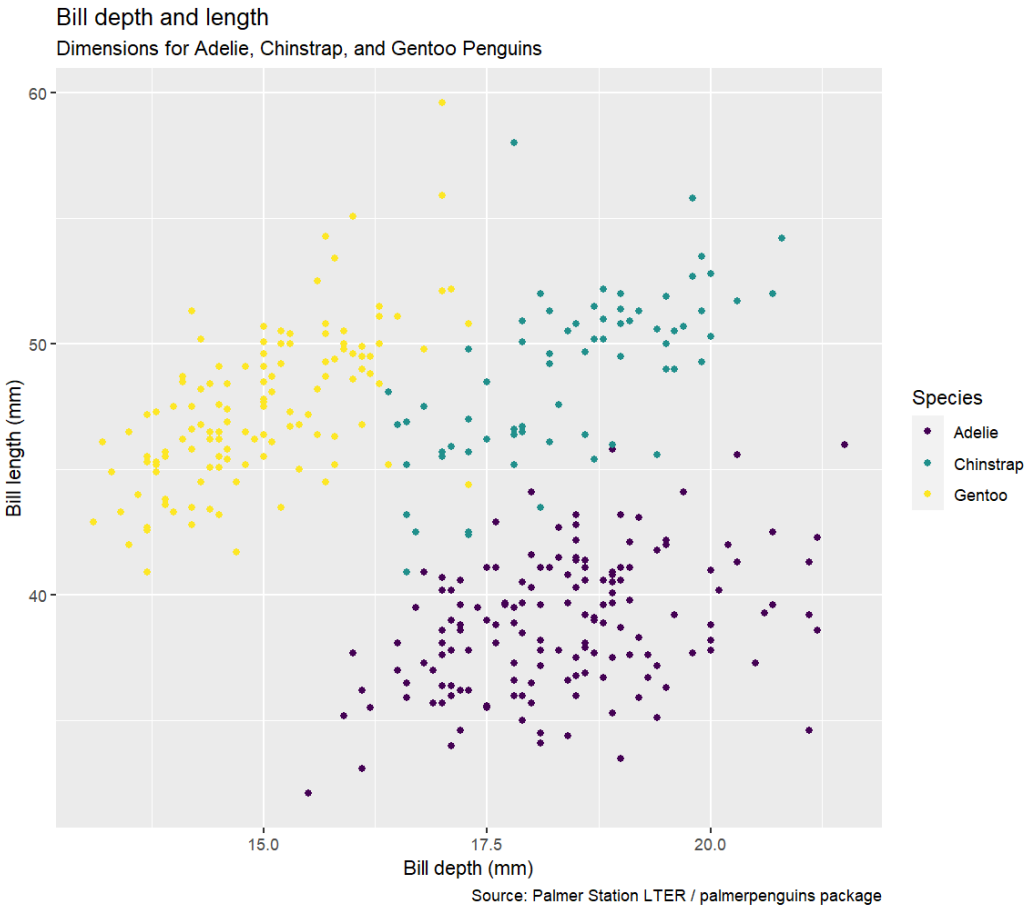




```r
library(palmerpenguins)
dplyr::glimpse(penguins)
```

```
Rows: 344
Columns: 8
$ species           <fct> Adelie, Adelie, Adelie, Adelie, Adelie, Adelie, Ade
$ island            <fct> Torgersen, Torgersen, Torgersen, Torgersen, Torgers
$ bill_length_mm    <dbl> 39.1, 39.5, 40.3, NA, 36.7, 39.3, 38.9, 39.2, 34.1,
$ bill_depth_mm     <dbl> 18.7, 17.4, 18.0, NA, 19.3, 20.6, 17.8, 19.6, 18.1,
$ flipper_length_mm <int> 181, 186, 195, NA, 193, 190, 181, 195, 193, 190, 18
$ body_mass_g       <int> 3750, 3800, 3250, NA, 3450, 3650, 3625, 4675, 3475,
$ sex               <fct> male, female, female, NA, female, male, female, mal
$ year              <int> 2007, 2007, 2007, 2007, 2007, 2007, 2007, 2007, 200
```

Plot Code


Bill depth and length
Dimensions for Adelie, Chinstrap, and Gentoo Penguins
60
50
40
Bill length (mm)
15.0
17.5
20.0
Bill depth (mm)
Source: Palmer Station LTER / palmerpenguins package
Species
Adelie
Chinstrap
Gentoo

Plot Code

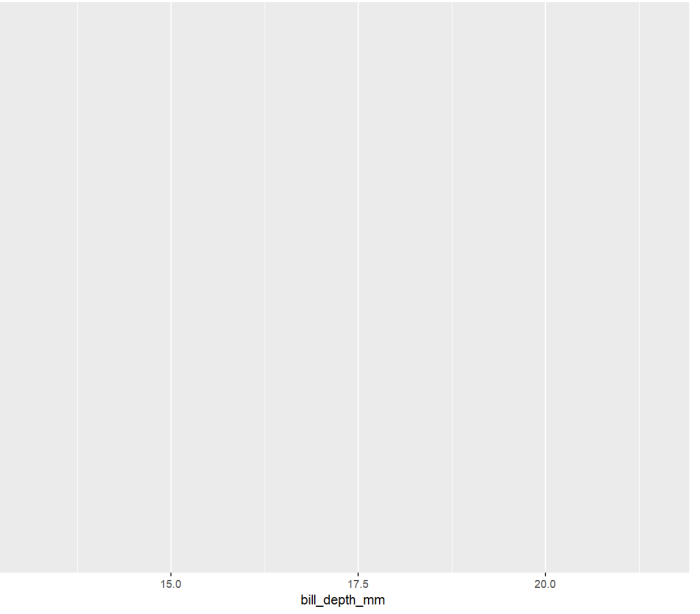
```
ggplot(data = penguins,
       mapping = aes(x = bill_depth_mm,
                     y = bill_length_mm,
                     color = species)) +
  geom_point() +
  labs(title = "Bill depth and length",
       subtitle = "Dimensions for Adelie, Chinstrap, and Gentoo Penguins",
       x = "Bill depth (mm)",
       y = "Bill length (mm)",
       color = "Species",
       caption = "Source: Palmer Station LTER / palmerpenguins package") +
  scale_color_viridis_d()
```

Start with the `penguins` data frame

```
ggplot(data = penguins)
```

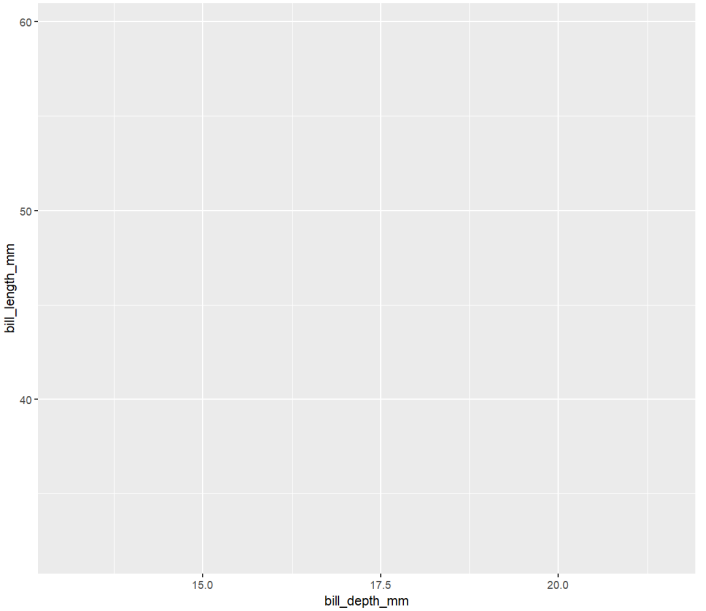
> Start with the `penguins` data frame, **map bill depth to the x-axis**

```
ggplot(data = penguins,
       mapping = aes(x = bill_depth_mm))
```



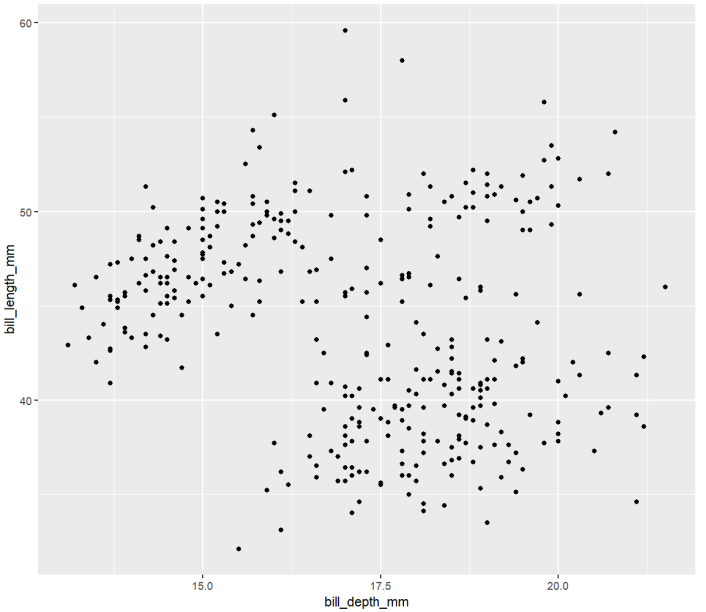
> Start with the `penguins` data frame, map bill depth to the x-axis **and map bill length to the y-axis.**

```
ggplot(data = penguins,
       mapping = aes(x = bill_depth_mm,
                     y = bill_length_mm))
```



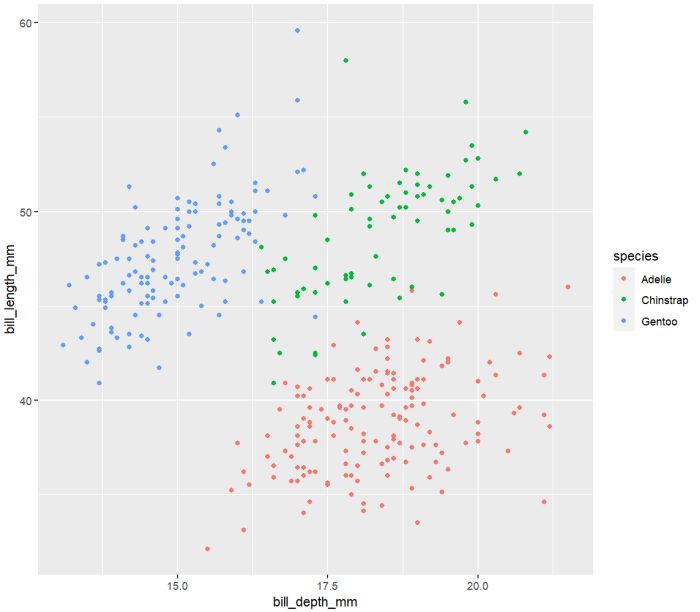
> Start with the `penguins` data frame, map bill depth to the x-axis and map bill length to the y-axis. **Represent each observation with a point**

```
ggplot(data = penguins,
       mapping = aes(x = bill_depth_mm,
                     y = bill_length_mm)) +
  geom_point()
```



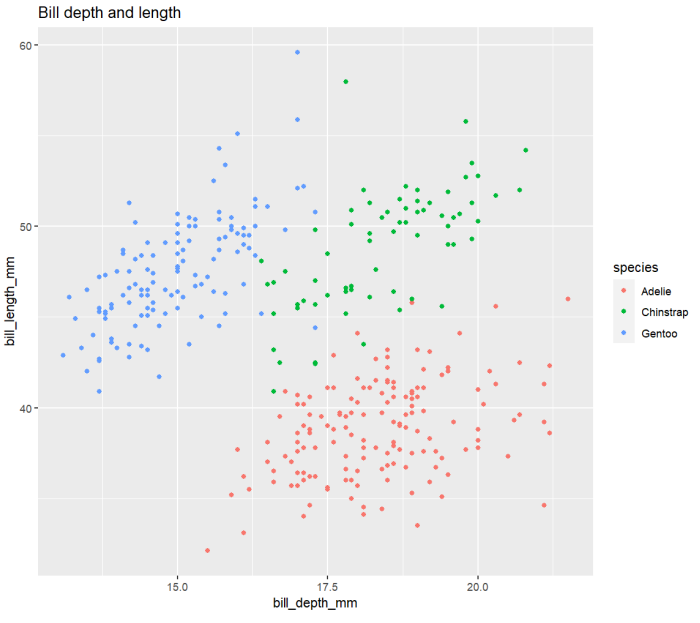
> Start with the `penguins` data frame, map bill depth to the x-axis and map bill length to the y-axis. Represent each observation with a point **and map species to the color of each point.**

```
ggplot(data = penguins,
       mapping = aes(x = bill_depth_mm,
                     y = bill_length_mm,
                     color = species)) +
  geom_point()
```



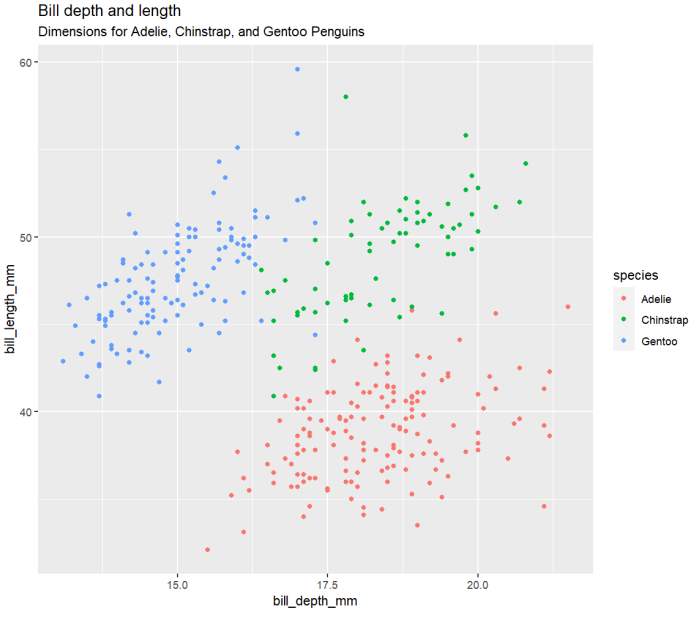
> Start with the `penguins` data frame, map bill depth to the x-axis and map bill length to the y-axis. Represent each observation with a point and map species to the color of each point. **Title the plot "Bill depth and length"**

```
ggplot(data = penguins,
       mapping = aes(x = bill_depth_mm,
                     y = bill_length_mm,
                     color = species)) +
  geom_point() +
  labs(title = "Bill depth and length")
```



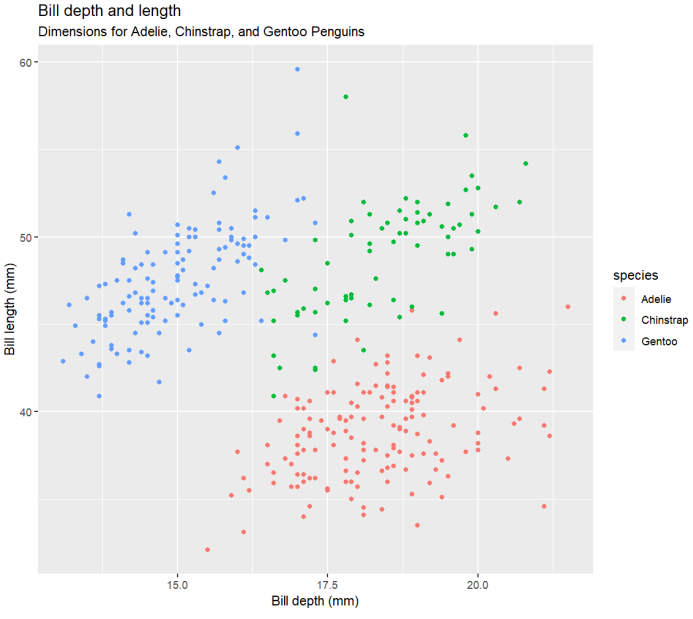
> Start with the `penguins` data frame, map bill depth to the x-axis and map bill length to the y-axis. Represent each observation with a point and map species to the color of each point. Title the plot "Bill depth and length", **add the subtitle "Dimensions for Adelie, Chinstrap, and Gentoo Penguins"**

```
ggplot(data = penguins,
       mapping = aes(x = bill_depth_mm,
                     y = bill_length_mm,
                     color = species)) +
  geom_point() +
  labs(title = "Bill depth and length",
       subtitle = "Dimensions for Adelie, Chinst
```



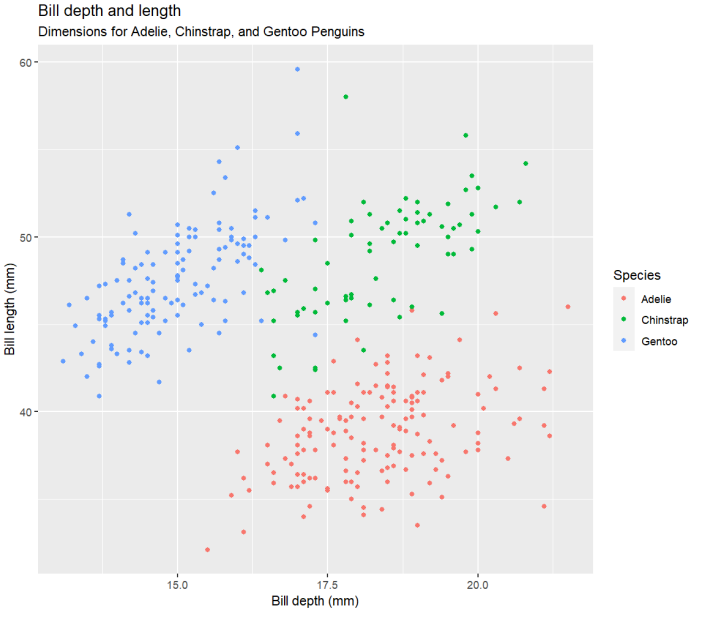
> Start with the `penguins` data frame, map bill depth to the x-axis and map bill length to the y-axis. Represent each observation with a point and map species to the color of each point. Title the plot "Bill depth and length", add the subtitle "Dimensions for Adelie, Chinstrap, and Gentoo Penguins", **label the x and y axes as "Bill depth (mm)" and "Bill length (mm)", respectively**

```
ggplot(data = penguins,
       mapping = aes(x = bill_depth_mm,
                     y = bill_length_mm,
                     color = species)) +
  geom_point() +
  labs(title = "Bill depth and length",
       subtitle = "Dimensions for Adelie, Chinst
       x = "Bill depth (mm)",
       y = "Bill length (mm)")
```



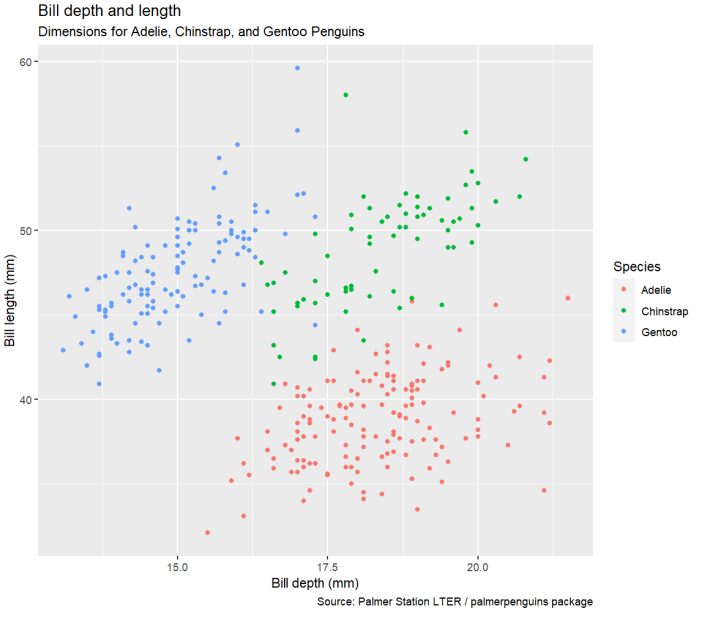
> Start with the `penguins` data frame, map bill depth to the x-axis and map bill length to the y-axis. Represent each observation with a point and map species to the color of each point. Title the plot "Bill depth and length", add the subtitle "Dimensions for Adelie, Chinstrap, and Gentoo Penguins", label the x and y axes as "Bill depth (mm)" and "Bill length (mm)", respectively, **label the legend "Species"**

```
ggplot(data = penguins,
       mapping = aes(x = bill_depth_mm,
                     y = bill_length_mm,
                     color = species)) +
  geom_point() +
  labs(title = "Bill depth and length",
       subtitle = "Dimensions for Adelie, Chinst
       x = "Bill depth (mm)",
       y = "Bill length (mm)",
       color = "Species")
```



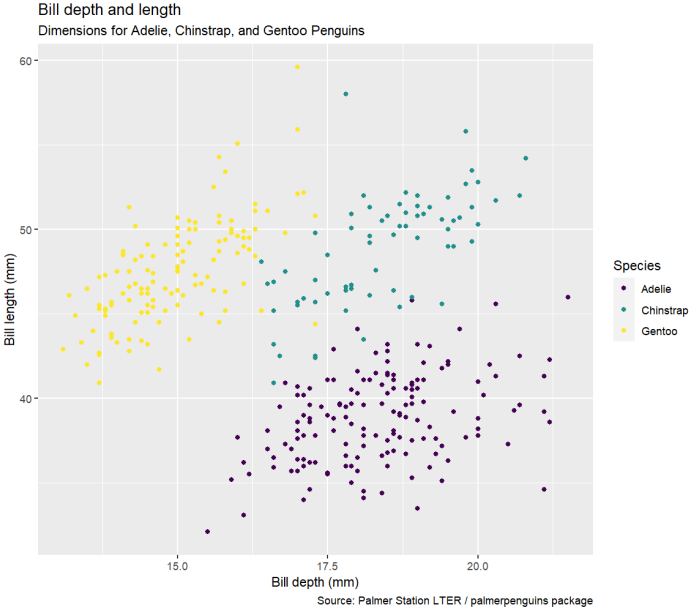
> Start with the `penguins` data frame, map bill depth to the x-axis and map bill length to the y-axis. Represent each observation with a point and map species to the color of each point. Title the plot "Bill depth and length", add the subtitle "Dimensions for Adelie, Chinstrap, and Gentoo Penguins", label the x and y axes as "Bill depth (mm)" and "Bill length (mm)", respectively, label the legend "Species", **and add a caption for the data source.**

```
ggplot(data = penguins,
       mapping = aes(x = bill_depth_mm,
                     y = bill_length_mm,
                     color = species)) +
  geom_point() +
  labs(title = "Bill depth and length",
       subtitle = "Dimensions for Adelie, Chinst
       x = "Bill depth (mm)",
       y = "Bill length (mm)",
       color = "Species",
       caption = "Source: Palmer Station LTER /
```

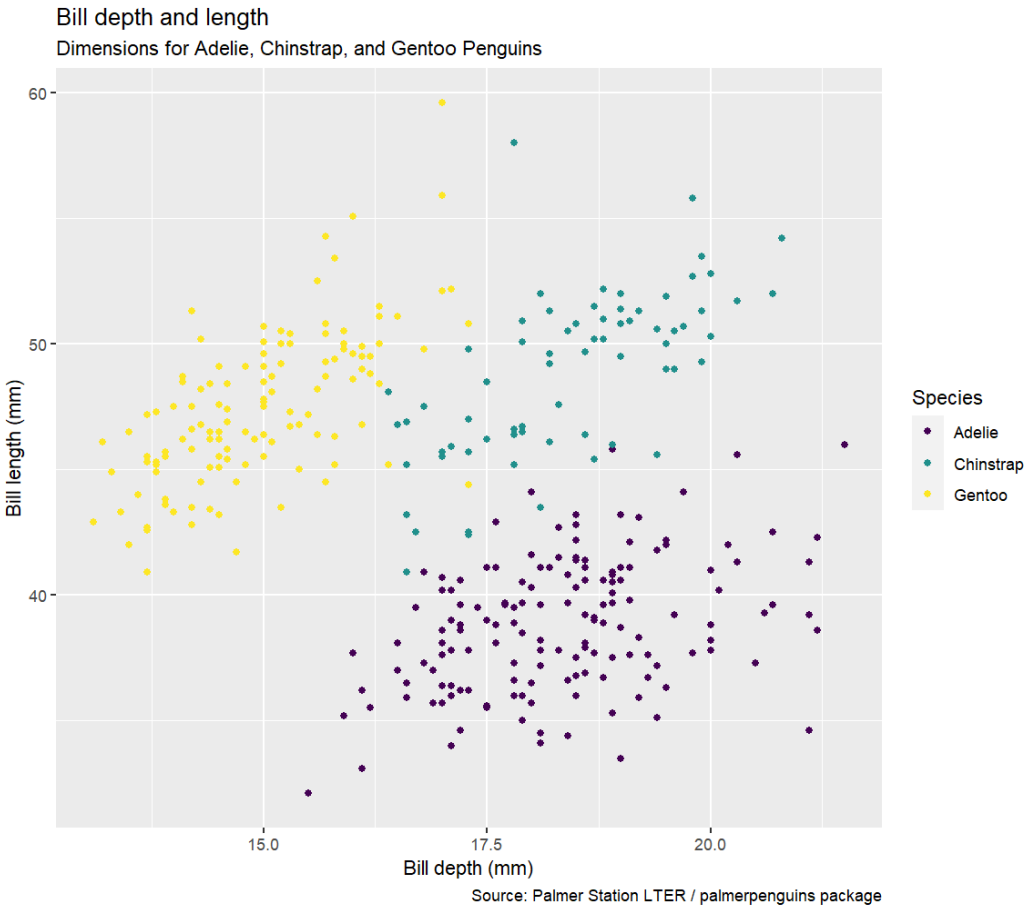


> Start with the `penguins` data frame, map bill depth to the x-axis and map bill length to the y-axis. Represent each observation with a point and map species to the color of each point. Title the plot "Bill depth and length", add the subtitle "Dimensions for Adelie, Chinstrap, and Gentoo Penguins", label the x and y axes as "Bill depth (mm)" and "Bill length (mm)", respectively, label the legend "Species", and add a caption for the data source. **Finally, use a discrete color scale that is designed to be perceived by viewers with common forms of color blindness.**

```
ggplot(data = penguins,
       mapping = aes(x = bill_depth_mm,
                     y = bill_length_mm,
                     color = species)) +
  geom_point() +
  labs(title = "Bill depth and length",
       subtitle = "Dimensions for Adelie, Chinst
       x = "Bill depth (mm)",
       y = "Bill length (mm)",
       color = "Species",
       caption = "Source: Palmer Station LTER /
  scale_color_viridis_d()
```



Plot Code Narrative



Plot | Code | Narrative

```
ggplot(data = penguins,
       mapping = aes(x = bill_depth_mm,
                     y = bill_length_mm,
                     color = species)) +
  geom_point() +
  labs(title = "Bill depth and length",
       subtitle = "Dimensions for Adelie, Chinstrap, and Gentoo Penguins",
       x = "Bill depth (mm)", y = "Bill length (mm)",
       color = "Species",
       caption = "Source: Palmer Station LTER / palmerpenguins package") +
  scale_color_viridis_d()
```

Plot Code Narrative

Start with the `penguins` data frame, map bill depth to the x-axis and map bill length to the y-axis.

Represent each observation with a point and map species to the color of each point.

Title the plot "Bill depth and length", add the subtitle "Dimensions for Adelie, Chinstrap, and Gentoo Penguins", label the x and y axes as "Bill depth (mm)" and "Bill length (mm)", respectively, label the legend "Species", and add a caption for the data source.

Finally, use a discrete color scale that is designed to be perceived by viewers with common forms of color blindness.

Readings

- Chapter 7:Introduction to data visualization
- Chapter 8:ggplot2