

Data Visualization with ggplot2

W. Evan Johnson, Ph.D.
Professor, Division of Infectious Disease
Director, Center for Data Science
Rutgers University – New Jersey Medical School

'r Sys.Date()

Data Visualization with R

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.

Data Visualization with ggplot2

We will be creating plots using the `ggplot2`¹ package.

```
library(dplyr)  
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` 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.

¹<https://ggplot2.tidyverse.org/>

Data Visualization with ggplot2

One reason ggplot2 is generally more intuitive for beginners is that it uses a **grammar of graphics**², 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.

²<http://www.springer.com/us/book/9780387245447>

Data Visualization with ggplot2

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.

Data Visualization with ggplot2

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 with an internet search for “ggplot2 cheat sheet” or by clicking here:

<https://www.rstudio.com/wp-content/uploads/2015/03/ggplot2-cheatsheet.pdf>

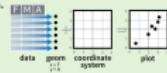
Data Visualization with ggplot2

Data Visualization with ggplot2 Cheat Sheet



Basics

ggplot2 is based on the **grammar of graphics**, the idea that you can build every graph from the same few components: a **data** set, a set of **geoms**—visual marks that represent data points, and a **coordinate system**.



To display data values, map variables in the data set to aesthetic properties of the geom like **size**, **color**, and **x** and **y** locations.



Build a graph with **qplot()** or **ggplot()**

qplot(x = cty, y = hwy, color = cyl, data = mpg, geom = "point")
Creates a complete plot with given data, geom, and mappings. Supplies many useful defaults.

ggplot(data = mpg, aes(x = cty, y = hwy))

Begins a plot that you finish by adding layers to. No defaults, but provides more control than qplot().

**ggplot(mpg, aes(hwy, cty)) +
geom_point(aes(color = cyl)) +
geom_smooth(method = "lm") +
coord_cartesian() +
scale_x_sqrt() +
theme_bw()**

Elements with +
elements with
stat = "identity"
layer specific
mappings
additional elements

Add a new layer to a plot with a **geom_*** or **stat_*** function. Each provides a geom, a set of aesthetic mappings, and a default stat and position adjustment.

last_plot()

Returns the last plot

ggsave("plot.png", width = 5, height = 5)

Saves last plot as 5" x 5" file named "plot.png" in working directory. Matches file type to file extension.

Geoms – Use a geom to represent data points, use the geom's aesthetic properties to represent variables. Each function returns a layer.

One Variable

- Continuous**
`a <- ggplot(mpg, aes(hwy))
a + geom_area(stat = "bin")
x, y, alpha, color, fill, linetype, size
b + geom_area(aes(ydensity..), stat = "bin")
x, y, alpha, color, fill, linetype, size, weight
b + geom_density(kernel = "gaussian")
x, y, alpha, color, fill, linetype, size, weight
b + geom_dotplot(..count.., ..density..)
x, y, alpha, color, fill`
- a + geom_freqpoly()**
`x, y, alpha, color, linetype, size
b + geom_histogram(binwidth = 5)
x, y, alpha, color, fill, linetype, weight
b + geom_histogram(aes(..density..))`

Discrete

- `b + geom_bar()`
`x, alpha, color, fill, linetype, size, weight`
- b + geom_text()**
`x, y, label, alpha, angle, color, family, fontface,
hjust, lineheight, size, vjust`

Graphical Primitives

- `c <- ggplot(map, aes(long, lat))`
- c + geom_polygon(aes(group = group))**
`x, y, alpha, color, fill, linetype, size`

- `d <- ggplot(economics, aes(date, unemployed))`
- d + geom_path(linewidth = "butt",
linejoin = "round", lineorder = 1)**
`x, y, alpha, color, linetype, size`
- d + geom_rect(presymmin = unemployed - 900,
ymax = unemployed + 900)**
`x, ymax, ymin, alpha, color, fill, linetype, size`

- e <- ggplot(seals, aes(x = long, y = lat))**
- e + geom_segment(aes(xend = long + delta_long,
yend = lat + delta_lat))**
`x, xend, y, yend, alpha, color, linetype, size`

- f + geom_rect(xmin = long, ymin = lat,
xmax = long + delta_long,
ymax = lat + delta_lat)**
`xmax, xmin, ymin, alpha, color, fill,
linetype, size`

Two Variables

- Continuous X, Continuous Y**
`f <- ggplot(mpg, aes(cty, hwy))`

- f + geom_blank()**
- f + geom_jitter()**
`x, y, alpha, color, fill, shape, size`
- f + geom_point()**
`x, y, alpha, color, fill, shape, size`
- f + geom_quantile()**
`x, y, alpha, color, linetype, size, weight`

- f + geom_rug(sides = "bl")**
`alpha, color, linetype, size`

- f + geom_smooth(model = lm)**
`x, y, alpha, color, fill, linetype, size, weight`

- C f + geom_text(label = "ctyl")**
`x, y, label, alpha, angle, color, family, fontface,
hjust, lineheight, size, vjust`

Discrete X, Continuous Y

- g <- ggplot(mpg, aes(class, hwy))**

- g + geom_bar(stat = "identity")**
`x, y, alpha, color, fill, linetype, size, weight`
- g + geom_boxplot()**
`lower, middle, upper, x, ymax, ymin, alpha,
color, fill, linetype, size, weight`
- g + geom_dotplot(binaxis = "y",
stackdir = "center")**
`x, y, alpha, color, fill`
- g + geom_rug(scale = "area")**
`x, y, alpha, color, fill, linetype, size, weight`

Discrete X, Discrete Y

- h <- ggplot(diamonds, aes(cut, color))**

- h + geom_jitter()**
`x, y, alpha, color, fill, shape, size`

Continuous Bivariate Distribution

- `i <- ggplot(movies, aes(year, rating))`

- i + geom_bin2d(binwidth = c(15, 0.5))**
`xmax, xmin, ymax, ymin, alpha, color, fill,
linetype, size, weight`
- i + geom_hex()**
`x, y, alpha, colour, linetype, size`

Continuous Function

- `j <- ggplot(economics, aes(date, unemploy))`

- j + geom_area()**
`x, y, alpha, color, fill, linetype, size`
- j + geom_line()**
`x, y, alpha, color, linetype, size`
- j + geom_step(direction = "hv")**
`x, y, alpha, color, linetype, size`

Visualizing error

- `df <- data.frame(grp = c("A", "B"), fit = 4:5; se = 1:2)`
- `k <- ggplot(df, aes(grp, fit, ymin = fit - se, ymax = fit + se))`

- k + geom_crossbar(fatten = 2)**
`x, y, ymax, ymin, alpha, color, fill, linetype,
size`
- k + geom_errorbar()**
`x, ymax, ymin, alpha, color, linetype, size,
width (also geom_errorbarh())`
- k + geom_linerange()**
`x, ymin, ymax, alpha, color, linetype, size`
- k + geom_pointrangle()**
`x, y, ymin, ymax, alpha, color, fill, linetype,
shape, size`

Maps

- `data <- data.frame(murder = USArrests$Murder,
state = tolower(rownames(USArrests)))`
- `map <- map_data("state")`

- (+ geom_map(aes(fill = state), map = map) +
expand_limits(+ maplong, y = maplat))**
`map_id, alpha, color, fill, linetype, size`

Three Variables

- `sealsSz <- with(seals, sqrt(delta_long * 2 + delta_lat * 2))`
- `m <- ggplot(seals, aes(long, lat))`

- m + geom_contour(aes(z = z))**
`x, y, z, alpha, colour, linetype, size, weight`

- m + geom_raster(aes(fill = z), hjust = 0.5,
vjust = 0.5, interpolate = FALSE)**
`x, y, alpha, fill`

- m + geom_tile(aes(fill = z))**
`x, y, alpha, color, fill, linetype, size`

Data Visualization with ggplot2

Stats - An alternative way to build a layer

Some plots visualize a transformation of the original data set. Use a `stat` to choose a common transformation to visualize, e.g. `a + geom_bar(stat = "bin")`



Each stat creates additional variables to map aesthetics to. These variables use a common `.name_` syntax.

stat functions and geom functions both combine a stat with a geom to make a layer, i.e. `stat_bin(geom="bar")` does the same as `geom_bar(stat="bin")`.



`a + stat_bin(binwidth = 1, origin = 10)` 1D distributions

`a + stat_bindot(binwidth = 1, bins = "x")` x,y, count, ..count, ..density, ..ndensity.

`a + stat_density(binwidth = 1, kernel = "gaussian")` x,y, ..count, ..density, ..scaled.

`f + stat_binned(bins = 30, drop = TRUE)` 2D distributions

`f + stat_spoke(angle = 20, angle2 = 20)` angle, radius, x, sent, y, yend | x, ..sent, ..y, ..yend.

`m + stat_summary_hex(binwidth = 10, fun = mean)` x,y, z, fill | ..value, ..binwidth, ..mean.

`m + stat_summary2d(binwidth = 2, bins = 30, fun = mean)` x,y, z, fill | ..value, ..binwidth.

`g + stat_bospill(level = 1.5)` Comparisons.

`x,y | lower, ..middle, ..upper, ..outliers,`

`g + stat_identity(adjust = 1, kernel = "gaussian", scale = "area")` x,y, ..density, ..scaled, ..count, ..n, ..width, ..width.

`f + stat_ecdf(n = 40)` Functions

`x,y | ..x, ..y, ..lower, ..upper,`

`f + stat_quantile(quantiles = c(0.25, 0.5, 0.75), formula = y ~ log(x), n = 100)`

`x,y | quantile, ..x, ..y,`

`f + stat_smooth(method = "auto", formula = y ~ x, se = TRUE, n = 80, fullrange = FALSE, level = 0.95)`

`x,y | ..y, ..lower, ..upper, ..ymin, ..ymax,`

`ggplot() + stat_function(fun = sin)` General Purpose

`fun = dnorm, n = 101, args = list(sd = 0.5)`

`x | ..y,`

`f + stat_identity()`

`geom = stat_qqline(sample = 1:100, distribution = qt, diagonal = TRUE)`

`sample, x,y | ..x, ..y,`

`f + stat_sum()`

`x, y, size | ..size,`

`f + stat_summary(fun.data = "mean_cl_boot")`

`f + stat_unique()`

Scales

Scales control how a plot maps data values to the visual values of an aesthetic. To change the mapping, add a custom scale.



range of values to include in mapping

range of values to use to legend

range of values to use in legends

range of values to use in legends

use to map

<

The Components of a Graph

We will construct the following graph for the US murders dataset:



The Components of a Graph

We can clearly see a 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 Components of a Graph

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.
- **Geometry:** The plot above is a scatterplot. 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. Each point represents a different observation, and we *map* data about these observations to visual cues. Color is another visual cue that we map to region.

The Components of a Graph

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.

The Components of a Graph

We start by loading the dataset:

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

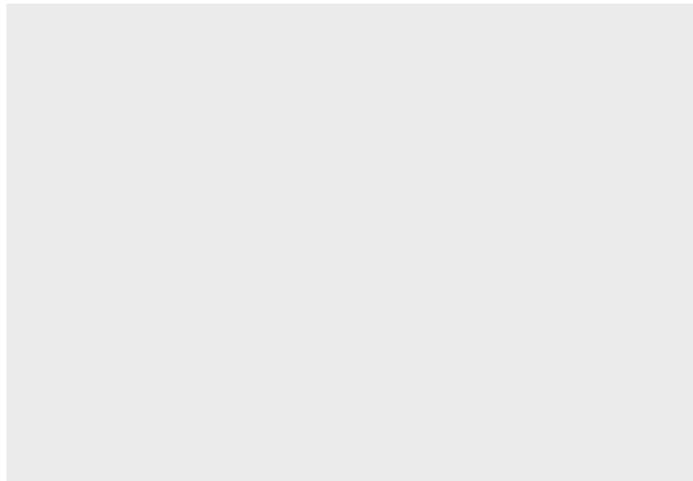
The first step in creating a `ggplot2` graph is to define a `ggplot` object. We do this with the function `ggplot`, which initializes the graph. If we read the help file for this function, we see that the first argument is used to specify what data is associated with this object:

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

ggplot objects

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

```
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

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:

```
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, like this:

```
\vskip .1in  
DATA %>% ggplot() + LAYER 1 + ... + LAYER N
```

Usually, the first added layer defines the geometry. We want to make a scatterplot. What geometry do we use?

Geometries

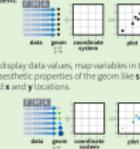
Taking a quick look at the cheat sheet, we see that the function used to create plots with this geometry is `geom_point`.

Data Visualization with ggplot2 Cheat Sheet



Basics

`ggplot2` is based on the *grammar of graphics*, the idea that you can build every graph from the same few components: a `data` set, a set of `geoms`—visual marks that represent data points, and a `coordinate system`.



To display data values, map variables in the data set to aesthetic properties of the geom like `size`, `color`, and `shape` or `locations`.



Build a graph with `qplot()` or `ggplot()`

`qplot(x, y, ..., geom = "point")`
Creates a complete plot with given data, geom, and mappings. Supplies many useful defaults.

`ggplot(data = mtcars, aes = cty) + geom_point()`

Beginns a plot that you finish by adding layers to. No plot, but provides more control than qplot().

`geom_point(mapping = aes(x = cty)) +
 geom_point(mapping = aes(x = mpg)) +
 geom_point(mapping = aes(x = disp)) +
 geom_point(mapping = aes(x = weight)) +
 geom_point(mapping = aes(x = hp)) +
 geom_point(mapping = aes(x = drat)) +
 geom_point(mapping = aes(x = carb)) +
 geom_point(mapping = aes(x = gear)) +
 geom_point(mapping = aes(x = vs)) +
 geom_point(mapping = aes(x = am)) +
 geom_point(mapping = aes(x = qsec)) +
 geom_point(mapping = aes(x = mpg))`

Add a new layer to a plot with a `geom_*` or `stat_*`. Each provides a geom, a set of aesthetic mappings, and a default stat and position adjustment.

`last_plot() +
 geom_point(mapping = aes(x = mpg, y = hp))`

Saves last plot as 5' x 5' file named "plot.png" in working directory. Matches file type to extension.

RStudio® is a trademark of RStudio, Inc. | <https://www.rstudio.com> | 344-448-1212 | rstudio.com

Geoms - use a geom to represent data points, use the geom's aesthetic properties to represent variables. Each function returns a layer.

One Variable

Continuous

`a <- ggplot(mpg, aes(hwy))`

`+ geom_area()`

`+ geom_area(stat = "bin")`

`+ geom_density(kernel = "gaussian")`

`x, y, alpha, color, fill, linetype, size, weight`

`+ geom_dotplot()`

`x, y, alpha, color, fill`

`+ geom_freqpoly()`

`x, y, alpha, color, linetype, size`

`+ geom_freqpoly(mapping = density, lwd = 1)`

`+ geom_histogram(binwidth = 5)`

`x, y, alpha, color, fill, linetype, size, weight`

`+ geom_hex()`

`x, y, alpha, color, fill, size`

`+ geom_jitter()`

`x, y, alpha, color, fill, shape, size`

`+ geom_linerider()`

`x, y, alpha, color, fill, shape, size`

`+ geom_quantile()`

`x, y, alpha, color, linetype, size, weight`

`+ geom_rug()`

`alpha, color, linetype, size`

`+ geom_smooth(model = lm)`

`x, y, alpha, color, fill, linetype, size, weight`

`+ geom_smooth(mapping = ..density..)`

`x, y, alpha, color, fill, linetype, size, weight`

`+ geom_text(mapping = ctly)`

`x, y, alpha, color, fill, fontface, fontfamily, fontsize, fontweight, size, weight`

`+ geom_bar()`

`x, y, alpha, color, fill, linetype, size, weight`

`+ geom_boxplot()`

`lower, middle, upper, x, ymin, alpha,`

`color, fill, linetype, shape, size, weight`

`+ geom_dotplot(mapping = aes(y =`

`stackdir = "center")`

`x, y, alpha, color, fill, linetype, size, weight`

`+ geom_violin(scale = "area")`

`x, y, alpha, color, fill, linetype, size, weight`

`+ geom_rect(mapping = aes(xmin =`

`long, xmax = long + delta_long,`

`ymin = long, ymax = long + delta_long,`

`color, fill, linetype, size`

`+ geom_rect(mapping = aes(xmin =`

`long, xmax = long + delta_long,`

`ymin = long, ymax = long + delta_long,`

`color, fill, linetype, size`

`+ geom_rect(mapping = aes(xmin =`

`long, xmax = long + delta_long,`

`ymin = long, ymax = long + delta_long,`

`color, fill, linetype, size`

`+ geom_rect(mapping = aes(xmin =`

`long, xmax = long + delta_long,`

`ymin = long, ymax = long + delta_long,`

`color, fill, linetype, size`

`+ geom_rect(mapping = aes(xmin =`

`long, xmax = long + delta_long,`

`ymin = long, ymax = long + delta_long,`

`color, fill, linetype, size`

`+ geom_rect(mapping = aes(xmin =`

`long, xmax = long + delta_long,`

`ymin = long, ymax = long + delta_long,`

`color, fill, linetype, size`

`+ geom_rect(mapping = aes(xmin =`

`long, xmax = long + delta_long,`

`ymin = long, ymax = long + delta_long,`

`color, fill, linetype, size`

`+ geom_rect(mapping = aes(xmin =`

`long, xmax = long + delta_long,`

`ymin = long, ymax = long + delta_long,`

`color, fill, linetype, size`

`+ geom_rect(mapping = aes(xmin =`

`long, xmax = long + delta_long,`

`ymin = long, ymax = long + delta_long,`

`color, fill, linetype, size`

`+ geom_rect(mapping = aes(xmin =`

`long, xmax = long + delta_long,`

`ymin = long, ymax = long + delta_long,`

`color, fill, linetype, size`

`+ geom_rect(mapping = aes(xmin =`

`long, xmax = long + delta_long,`

`ymin = long, ymax = long + delta_long,`

`color, fill, linetype, size`

`+ geom_rect(mapping = aes(xmin =`

`long, xmax = long + delta_long,`

`ymin = long, ymax = long + delta_long,`

`color, fill, linetype, size`

`+ geom_rect(mapping = aes(xmin =`

`long, xmax = long + delta_long,`

`ymin = long, ymax = long + delta_long,`

`color, fill, linetype, size`

`+ geom_rect(mapping = aes(xmin =`

`long, xmax = long + delta_long,`

`ymin = long, ymax = long + delta_long,`

`color, fill, linetype, size`

`+ geom_rect(mapping = aes(xmin =`

`long, xmax = long + delta_long,`

`ymin = long, ymax = long + delta_long,`

`color, fill, linetype, size`

`+ geom_rect(mapping = aes(xmin =`

`long, xmax = long + delta_long,`

`ymin = long, ymax = long + delta_long,`

`color, fill, linetype, size`

`+ geom_rect(mapping = aes(xmin =`

`long, xmax = long + delta_long,`

`ymin = long, ymax = long + delta_long,`

`color, fill, linetype, size`

`+ geom_rect(mapping = aes(xmin =`

`long, xmax = long + delta_long,`

`ymin = long, ymax = long + delta_long,`

`color, fill, linetype, size`

`+ geom_rect(mapping = aes(xmin =`

`long, xmax = long + delta_long,`

`ymin = long, ymax = long + delta_long,`

`color, fill, linetype, size`

`+ geom_rect(mapping = aes(xmin =`

`long, xmax = long + delta_long,`

`ymin = long, ymax = long + delta_long,`

`color, fill, linetype, size`

`+ geom_rect(mapping = aes(xmin =`

`long, xmax = long + delta_long,`

`ymin = long, ymax = long + delta_long,`

`color, fill, linetype, size`

`+ geom_rect(mapping = aes(xmin =`

`long, xmax = long + delta_long,`

`ymin = long, ymax = long + delta_long,`

`color, fill, linetype, size`

`+ geom_rect(mapping = aes(xmin =`

`long, xmax = long + delta_long,`

`ymin = long, ymax = long + delta_long,`

`color, fill, linetype, size`

`+ geom_rect(mapping = aes(xmin =`

`long, xmax = long + delta_long,`

`ymin = long, ymax = long + delta_long,`

`color, fill, linetype, size`

`+ geom_rect(mapping = aes(xmin =`

`long, xmax = long + delta_long,`

`ymin = long, ymax = long + delta_long,`

`color, fill, linetype, size`

`+ geom_rect(mapping = aes(xmin =`

`long, xmax = long + delta_long,`

`ymin = long, ymax = long + delta_long,`

`color, fill, linetype, size`

`+ geom_rect(mapping = aes(xmin =`

`long, xmax = long + delta_long,`

`ymin = long, ymax = long + delta_long,`

`color, fill, linetype, size`

`+ geom_rect(mapping = aes(xmin =`

`long, xmax = long + delta_long,`

`ymin = long, ymax = long + delta_long,`

`color, fill, linetype, size`

`+ geom_rect(mapping = aes(xmin =`

`long, xmax = long + delta_long,`

`ymin = long, ymax = long + delta_long,`

`color, fill, linetype, size`

`+ geom_rect(mapping = aes(xmin =`

`long, xmax = long + delta_long,`

`ymin = long, ymax = long + delta_long,`

`color, fill, linetype, size`

`+ geom_rect(mapping = aes(xmin =`

`long, xmax = long + delta_long,`

`ymin = long, ymax = long + delta_long,`

`color, fill, linetype, size`

`+ geom_rect(mapping = aes(xmin =`

`long, xmax = long + delta_long,`

`ymin = long, ymax = long + delta_long,`

`color, fill, linetype, size`

`+ geom_rect(mapping = aes(xmin =`

`long, xmax = long + delta_long,`

`ymin = long, ymax = long + delta_long,`

`color, fill, linetype, size`

`+ geom_rect(mapping = aes(xmin =`

`long, xmax = long + delta_long,`

`ymin = long, ymax = long + delta_long,`

`color, fill, linetype, size`

`+ geom_rect(mapping = aes(xmin =`

`long, xmax = long + delta_long,`

`ymin = long, ymax = long + delta_long,`

`color, fill, linetype, size`

`+ geom_rect(mapping = aes(xmin =`

`long, xmax = long + delta_long,`

`ymin = long, ymax = long + delta_long,`

`color, fill, linetype, size`

`+ geom_rect(mapping = aes(xmin =`

`long, xmax = long + delta_long,`

`ymin = long, ymax = long + delta_long,`

`color, fill, linetype, size`

`+ geom_rect(mapping = aes(xmin =`

`long, xmax = long + delta_long,`

`ymin = long, ymax = long + delta_long,`

`color, fill, linetype, size`

`+ geom_rect(mapping = aes(xmin =`

`long, xmax = long + delta_long,`

`ymin = long, ymax = long + delta_long,`

`color, fill, linetype, size`

`+ geom_rect(mapping = aes(xmin =`

`long, xmax = long + delta_long,`

`ymin = long, ymax = long + delta_long,`

`color, fill, linetype, size`

`+ geom_rect(mapping = aes(xmin =`

`long, xmax = long + delta_long,`

`ymin = long, ymax = long + delta_long,`

`color, fill, linetype, size`

`+ geom_rect(mapping = aes(xmin =`

`long, xmax = long + delta_long,`

`ymin = long, ymax = long + delta_long,`

`color, fill, linetype, size`

`+ geom_rect(mapping = aes(xmin =`

`long, xmax = long + delta_long,`

`ymin = long, ymax = long + delta_long,`

`color, fill, linetype, size`

`+ geom_rect(mapping = aes(xmin =`

`long, xmax = long + delta_long,`

`ymin = long, ymax = long + delta_long,`

`color, fill, linetype, size`

`+ geom_rect(mapping = aes(xmin =`

`long, xmax = long + delta_long,`

`ymin = long, ymax = long + delta_long,`

`color, fill, linetype, size`

`+ geom_rect(mapping = aes(xmin =`

`long, xmax = long + delta_long,`

`ymin = long, ymax = long + delta_long,`

`color, fill, linetype, size`

`+ geom_rect(mapping = aes(xmin =`

`long, xmax = long + delta_long,`

`ymin = long, ymax = long + delta_long,`

`color, fill, linetype, size`

`+ geom_rect(mapping = aes(xmin =`

`long, xmax = long + delta_long,`

`ymin = long, ymax = long + delta_long,`

`color, fill, linetype, size`

`+ geom_rect(mapping = aes(xmin =`

`long, xmax = long + delta_long,`

`ymin = long, ymax = long + delta_long,`

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

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:

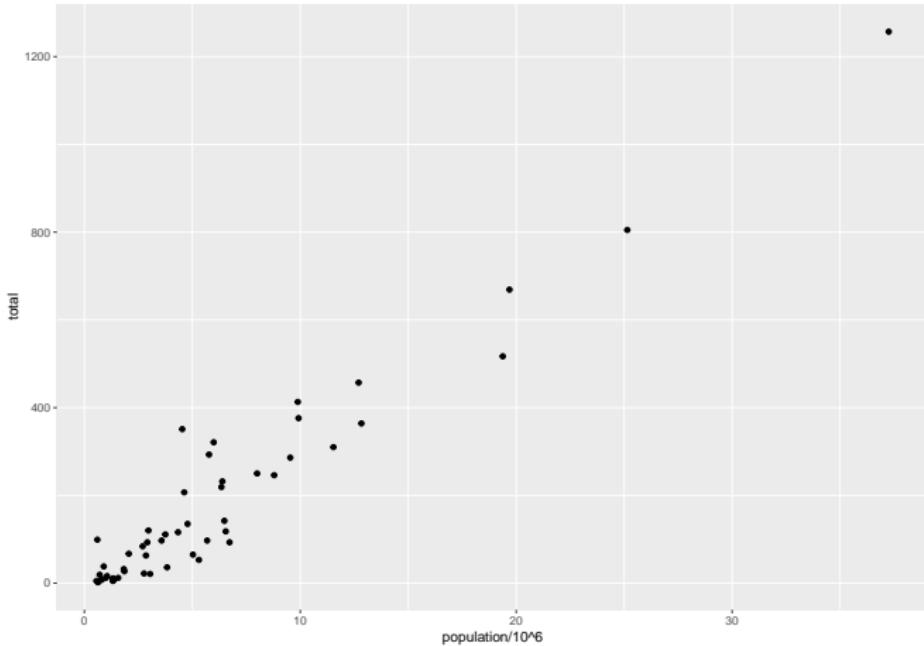
```
murders %>% ggplot() +  
  geom_point(aes(x = population/10^6, y = total))
```

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.

Aesthetic Mappings

We can also add a layer to the p object using `p <- ggplot(data = murders):`

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



Aesthetic Mappings

The scale and labels are defined by default when adding this layer. The aes function also uses the variable names from the object component: we can use population and total without having to call them as murders\$population, etc.

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.

Layers

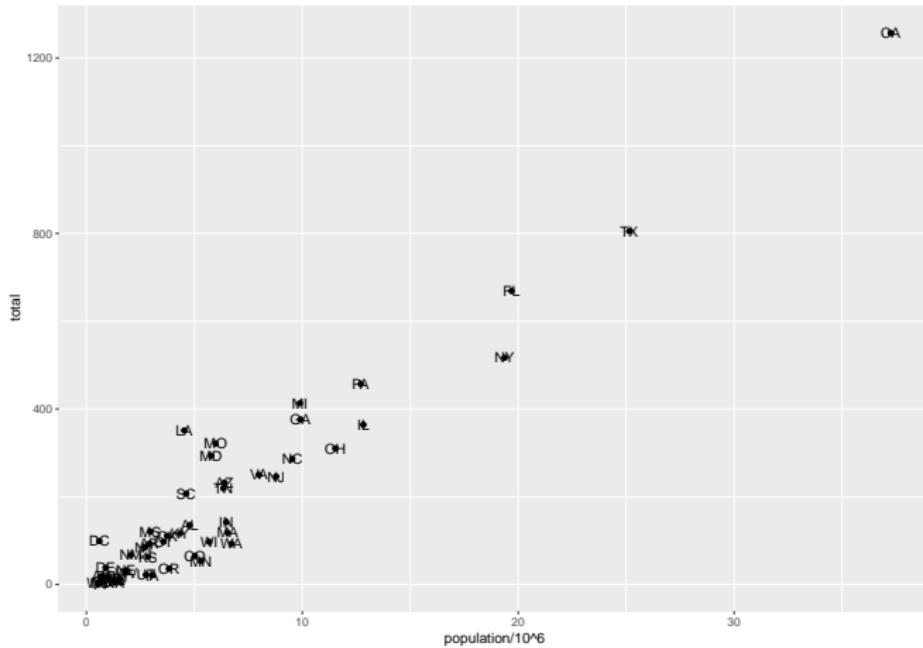
A second layer in the plot we wish to make involves adding a label to each point to identify the state. The `geom_label` and `geom_text` functions permit us to add text to the plot with and without a rectangle behind the text, respectively.

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`.

Layers

So the code looks like this:

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



Layers

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

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

is fine, whereas this call:

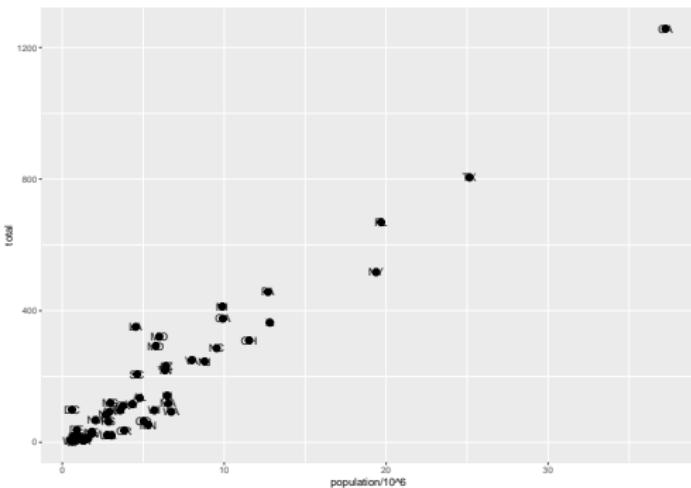
```
p_test <- p + geom_text(  
  aes(population/10^6, total), label = abb)
```

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.

Tinkering with Arguments

In the help file we see that size is an aesthetic. We can change it:

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



Tinkering with Arguments

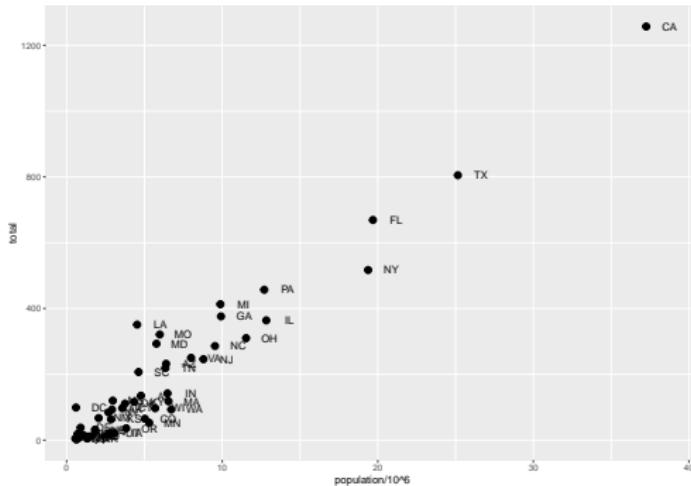
Each geometry function has many arguments other than aes and data. They tend to be specific to the function. For example, in the plot we wish to make, the points are larger than the default size.

Note that **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.

Tinkering with Arguments

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 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:

```
args(ggplot)
```

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

Global versus Local Aesthetic Mappings

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 = abb))
```

Global versus Local Aesthetic Mappings

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)
```

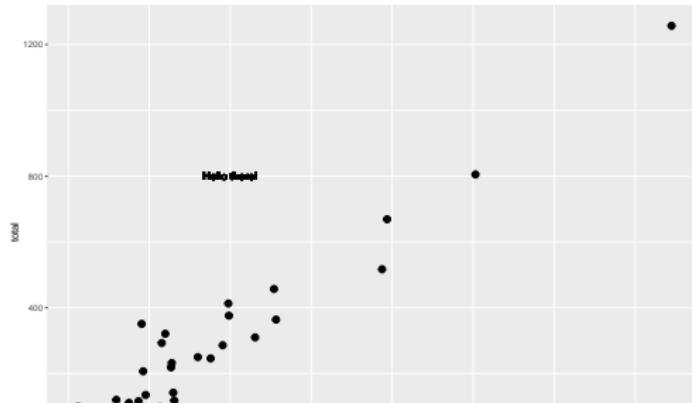
We keep the `size` and `nudge_x` arguments in `geom_point` and `geom_text`, respectively, because we want to only increase the size of points and only nudge the labels. If we put those arguments in `aes` then they would apply to both plots. Also note that the `geom_point` function does not need a `label`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

Global versus Local Aesthetic Mappings

If necessary, we can override the global mapping by defining a new mapping within each layer:

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

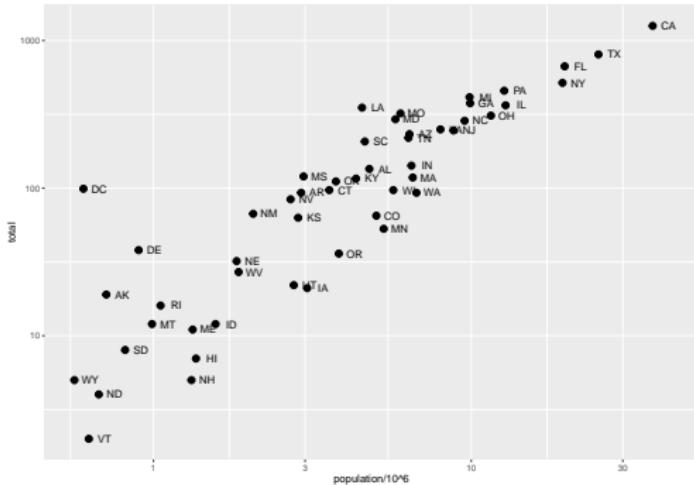
```
## Warning in geom_text(aes(x = 10, y = 800, label = "Hello there!")):
##   Please consider using `annotate()` or provide this layer
##   a single row.
```



Scales

If our desired scales are in log-scale, we can add this 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:

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



Scales

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:

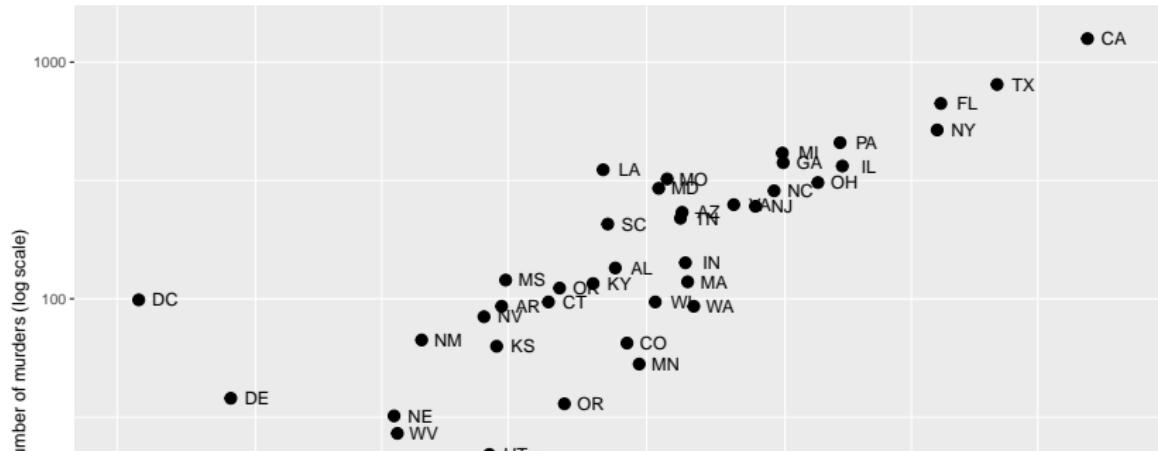
```
p + geom_point(size = 3) +
  geom_text(nudge_x = 0.05) +
  scale_x_log10() +
  scale_y_log10()
```

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")
```

US Gun Murders in 2010



Labels and Titles (categories as colors)

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

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:

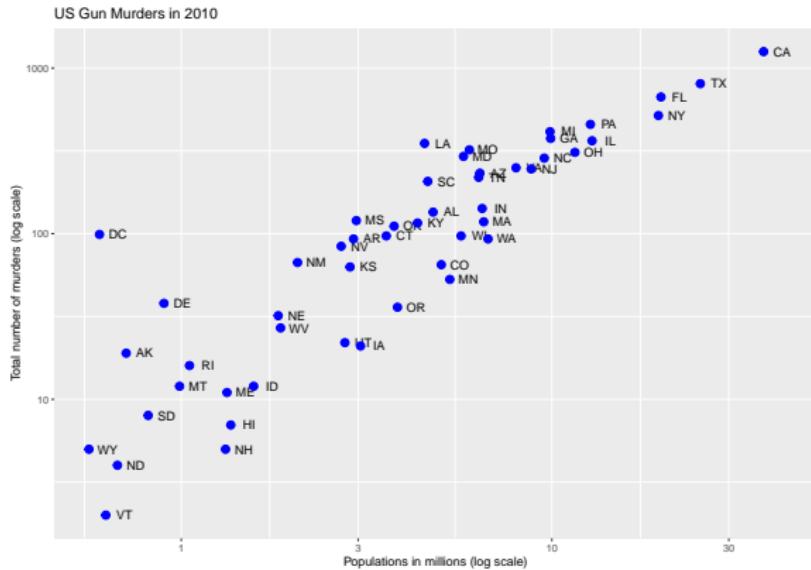
```
p <- murders %>% ggplot(aes(population/10^6,  
                                total, label = abb)) +  
  geom_text(nudge_x = 0.05) +  
  scale_x_log10() +  
  scale_y_log10() +  
  xlab("Populations in millions (log scale)") +  
  ylab("Total number of murders (log scale)") +  
  ggtitle("US Gun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

and then test out what happens by adding different calls to `geom_point`.

Labels and Titles (categories as colors)

We can make all the points blue by adding the color argument:

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

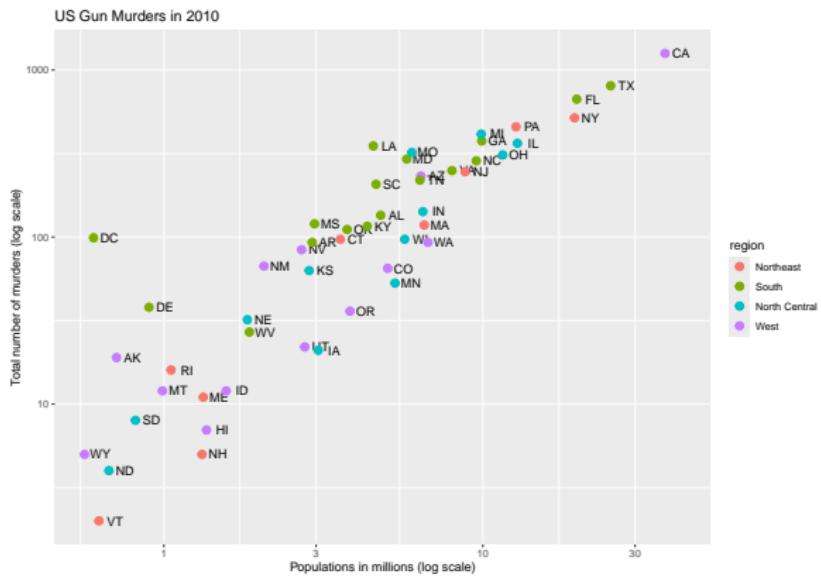


This, of course, is not what we want. We want to assign color depending on the geographical region. A nice default behavior of `ggplot2` is that if we assign a categorical variable to color, it

Labels and Titles (categories as colors)

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:

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



Labels and Titles (categories as colors)

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.

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: $y = rx$, 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 $\log(r)$.

Annotation, Shapes, and Adjustments

To compute this value, we use our **dplyr** skills:

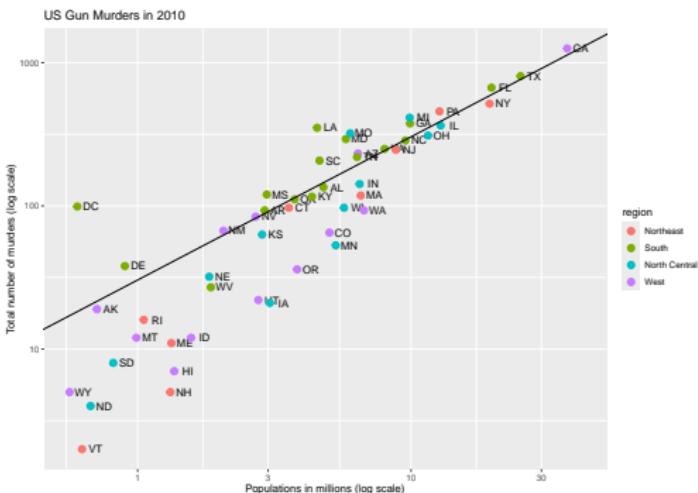
```
r <- murders %>%
  summarize(rate = sum(total) / sum(population) * 10^6) %>%
  pull(rate)
```

To add a line we use the `geom_abline` function. `ggplot2` uses `ab` in the name to remind us we are supplying the intercept (`a`) and slope (`b`).

Annotation, Shapes, and Adjustments

The default line has slope 1 and intercept 0 so we only have to define the intercept:

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



Annotation, Shapes, and Adjustments

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.

```
p <- p + geom_abline(intercept = log10(r), lty = 2, color = "red")  
      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.

Annotation, Shapes, and Adjustments

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:

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

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. In fact, for most of the plots in this book, 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. After installing the package, you can change the style by adding a layer like this:

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

Add-on Packages

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.

The final difference has to do with the position of the labels. In our plot, some of the labels fall on top of each other. 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`.

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)

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

Putting it All Together



Quick plots with qplot

We have learned the powerful approach to generating visualization with ggplot. However, there are instances in which all we want is to make a quick plot of, for example, a histogram of the values in a vector, a scatterplot of the values in two vectors, or a boxplot using categorical and numeric vectors. 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 `qplot`.

Quick plots with qplot

If we have values in two vectors, say:

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

and we want to make a scatterplot with ggplot, we would have to type something like:

```
data.frame(x = x, y = y) %>%
  ggplot(aes(x, y)) +
  geom_point()
```

Quick plots with qplot

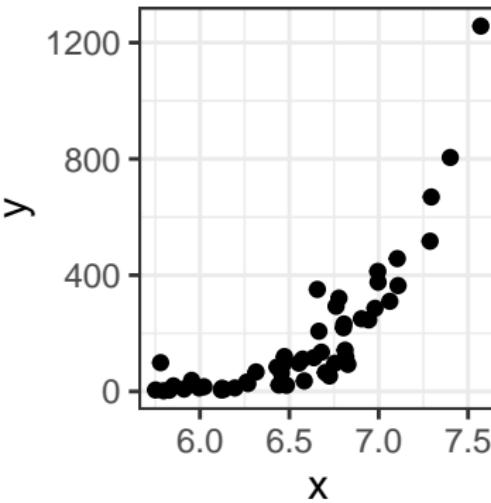
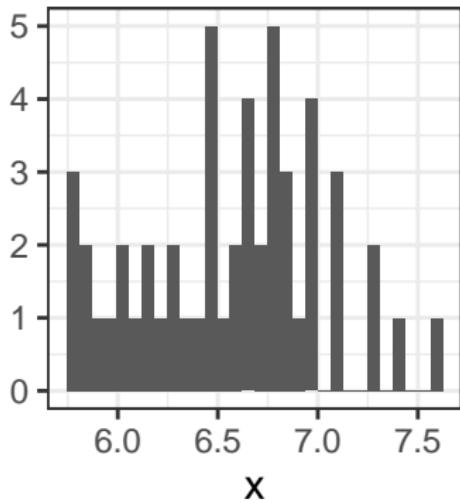
This seems like too much code for such a simple plot. The `qplot` function sacrifices the flexibility provided by the `ggplot` approach, but allows us to generate a plot quickly.

```
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

Now open the `ggplot2 Exercises` file and complete Exercises 1-16.

Session Info

```
sessionInfo()
```

```
## R version 4.4.0 (2024-04-24)
## Platform: aarch64-apple-darwin20
## Running under: macOS Sonoma 14.2.1
##
## Matrix products: default
## BLAS: /Library/Frameworks/R.framework/Versions/4.4-arm64/Resources/lib/libRblas.0.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/4.4-arm64/Resources/lib/libRlapack.dylib; LAPACK ver
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## time zone: America/Denver
## tzcode source: internal
##
## attached base packages:
## [1] stats      graphics   grDevices  utils       datasets   methods    base
##
## other attached packages:
## [1] gridExtra_2.3   ggrepel_0.9.5   ggthemes_5.1.0   dslabs_0.8.0
## [5] lubridate_1.9.3forcats_1.0.0   stringr_1.5.1   dplyr_1.1.4
## [9] purrr_1.0.2    readr_2.1.5    tidyr_1.3.1    tibble_3.2.1
## [13] ggplot2_3.5.1  tidyverse_2.0.0
##
## loaded via a namespace (and not attached):
## [1] gtable_0.3.5     compiler_4.4.0    tinytex_0.50    Rcpp_1.0.12
## [5] tidyselect_1.2.1  scales_1.3.0     yaml_2.3.8     fastmap_1.1.1
## [9] R6_2.5.1         labeling_0.4.3   generics_0.1.3  knitr_1.46
## [13] munsell_0.5.1    pillar_1.9.0     tzdb_0.4.0     rlang_1.1.3
## [17] utf8_1.2.4      stringi_1.8.3   xfun_0.43     timechange_0.3.0
## [21] cli_3.6.2       withr_3.0.0     magrittr_2.0.3  digest_0.6.35
## [25] grid_4.4.0       rstudioapi_0.16.0 hms_1.1.3     lifecycle_1.0.4
## [29] vctrs_0.6.5     evaluate_0.23   glue_1.7.0     farver_2.1.1
## [33] fansi_1.0.6     colorspace_2.1-0  rmarkdown_2.26  tools_4.4.0
```