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

11/16/2023

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

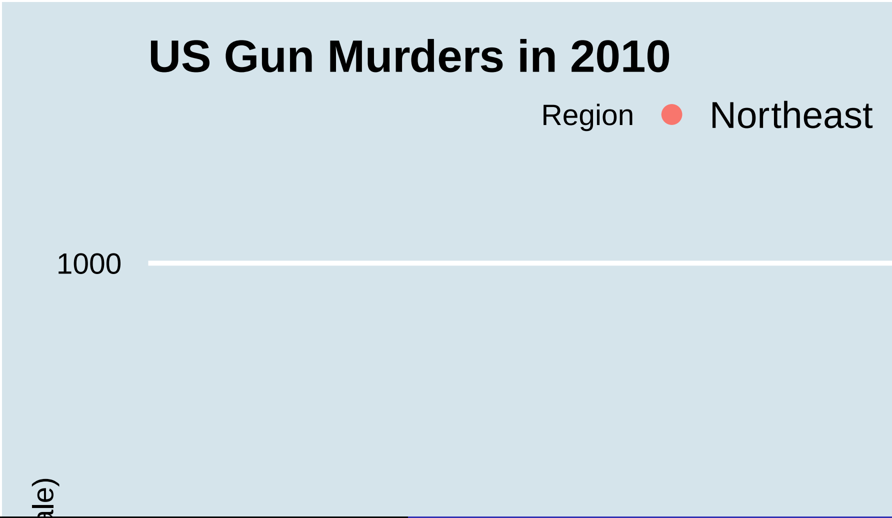
Throughout the book, we will be creating plots using the **ggplot2**[1] package.

```
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** in this book because it breaks plots into components in a way that permits beginners to create relatively complex and aesthetically pleasing plots using syntax that is intuitive and comparatively easy to

The components of a graph

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



ggplot objects

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

```r
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:

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

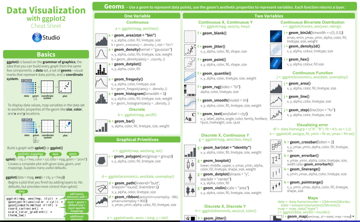
Geometries

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`.



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.

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)`:

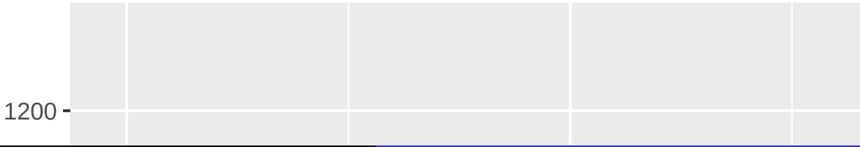
```
p + geom_point(aes(population/10^6, total))
```

Layers

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:

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



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:

```r
args(ggplot)
```

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

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

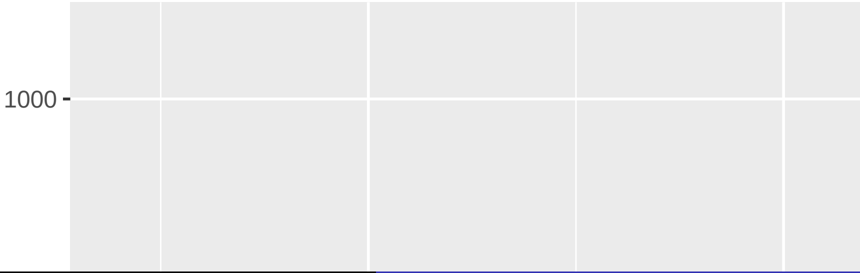
```r
p <- murders %>% ggplot(aes(population/10^6, total, label =
```

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

Scales

First, our desired scales are in log-scale. This is not the default, so this change needs to be added through a *scales* layer. A quick look at the cheat sheet reveals the scale_x_continuous function lets us control the behavior of scales. We use them like this:

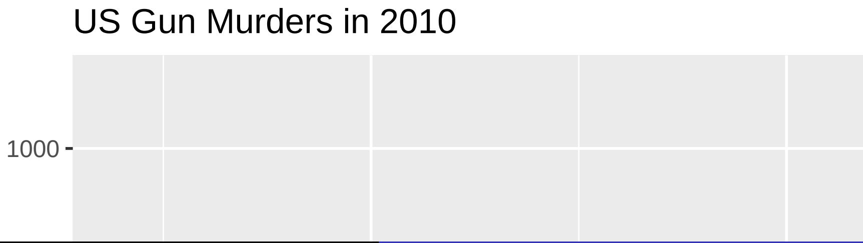
```
p + geom_point(size = 3) +
  geom_text(nudge_x = 0.05) +
  scale_x_continuous(trans = "log10") +
  scale_y_continuous(trans = "log10")
```



Labels and titles

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

```r
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")
```



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
  geom_text(nudge_x = 0.05) +
  scale_x_log10() +
  scale_y_log10() +
  xlab("Populations in millions (log scale)") +
  ylab("Total number of murders (log scale)") +
  ggtitle("US Gun Murders in 2010")
```

and then test out what happens by adding different calls to `geom_point`. We can make all the points blue by adding the `color` argument:

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

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 $\log(r)$. 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`). The default line has slope 1 and intercept 0 so we only

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()
```

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()
```

Putting it all together

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

```r
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 = "darkg
  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)") +
```

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 function `qplot`.

If we have values in two vectors, say:

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

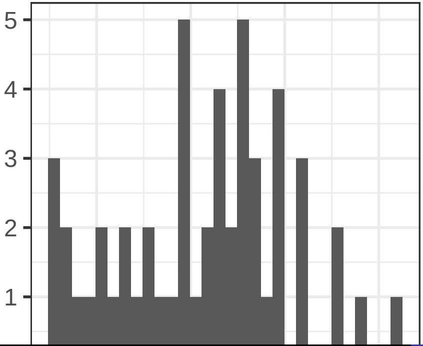
and we want to make a scatterplot with ggplot, we would have to type something like:

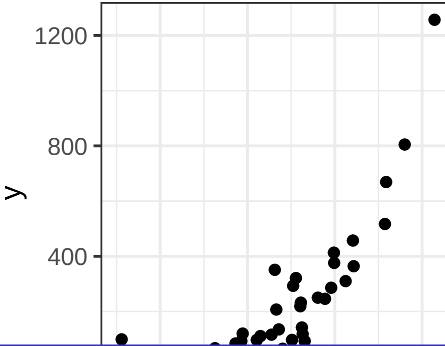
```r
data.frame(x = x, y = y) %>%
  ggplot(aes(x, y)) +
```

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)
```

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

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

Because `data` is the first argument we don't need to spell it out

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

and we can also use the pipe:

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

Session Info

```r
sessionInfo()
```

```
## R version 4.3.2 (2023-10-31)
## Platform: aarch64-apple-darwin20 (64-bit)
## Running under: macOS Ventura 13.5.1
##
## Matrix products: default
## BLAS:   /Library/Frameworks/R.framework/Versions/4.3-arm64/Resources/lib/libRblas.0.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/4.3-arm64/Resources/lib/libRlapack.dylib;  LAPACK vers
##
## 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_0.9.4   ggthemes_4.2.4  dslabs_0.7.6
##  [5] lubridate_1.9.3 forcats_1.0.0   stringr_1.5.0   dplyr_1.1.3
##  [9] purrr_1.0.2     readr_2.1.4     tidyr_1.3.0     tibble_3.2.1
## [13] ggplot2_3.4.4   tidyverse_2.0.0
##
## loaded via a namespace (and not attached):
##  [1] gtable_0.3.4      compiler_4.3.2    Rcpp_1.0.11       tidyselect_1.2.0
##  [5] scales_1.2.1      yaml_2.3.7        fastmap_1.1.1     R6_2.5.1
##  [9] labeling_0.4.3    generics_0.1.3    knitr_1.45        munsell_0.5.0
## [13] pillar_1.9.0      tzdb_0.4.0        rlang_1.1.2       utf8_1.2.4
## [17] stringi_1.8.1     xfun_0.41         timechange_0.2.0  cli_3.6.1
## [21] withr_2.5.2       magrittr_2.0.3    digest_0.6.33     grid_4.3.2
## [25] rstudioapi_0.15.0 hms_1.1.3         lifecycle_1.0.4   vctrs_0.6.4
## [29] evaluate_0.23     glue_1.6.2        farver_2.1.1      fansi_1.0.5
## [33] colorspace_2.1-0  rmarkdown_2.25    tools_4.3.2       pkgconfig_2.0.3
```