Data Visualization

Leili Mortazavi

8/3/2020

Contents

Many people have contributed to developing and revising the tutorial material over the years:

Anna Khazenzon Cayce Hook Paul Thibodeau Mike Frank Benoit Monin Ewart Thomas Michael Waskom Steph Gagnon Dan Birman Natalia Velez, Kara Weisman Andrew Lampinen Joshua Morris Yochai Shavit Jackie Schwartz Russ Poldrack

In this modulw, you'll learn the basics of how to plot with `ggplot2`,
a powerful package that enables the construction of highly
customizable and visually pleasing graphs.

Setting up

First, let's import our dataset.

Even though we already have the dataset loaded in our environment, we want each script to be standalone. So let's first clea out orur workspace with
`rm(list=ls())`. Enter this command into your console. (Remember Console is typically the bottom left window). You could also click on the broom icon in you environment pane (top right).

Then, we need to load the packages that we want to use in this script. We'll need `tidyverse`, which includes `ggplot2`.

```
library(tidyverse)
```

```
## -- Attaching packages ----------------------------------------------------------------------------

## v ggplot2 3.3.2     v purrr   0.3.4
## v tibble  3.0.3     v dplyr   1.0.1
## v tidyr   1.1.1     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.5.0

## -- Conflicts -------------------------------------------------------------------------------------
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

Now import the tidy dataset that we saved out in the last module.

```
data <- read_csv("../data/prepost_tidy.csv")
```

```
## Parsed with column specification:
## cols(
##   subject = col_double(),
##   gender = col_character(),
##   age = col_double(),
##   condition = col_character(),
##   diff = col_double(),
##   prepost = col_character(),
##   score = col_double()
## )
```

```
head(data)
```

```
## # A tibble: 6 x 7
##   subject gender   age condition  diff prepost  score
##     <dbl> <chr>  <dbl> <chr>     <dbl> <chr>    <dbl>
## 1       1 f         20 drug          2 pretest     45
## 2       1 f         20 drug          2 posttest    47
## 3        2 m          45 drug           4 pretest     65
## 4        2 m          45 drug           4 posttest    69
## 5        3 f          36 drug          10 pretest     45
## 6        3 f          36 drug          10 posttest    55
```

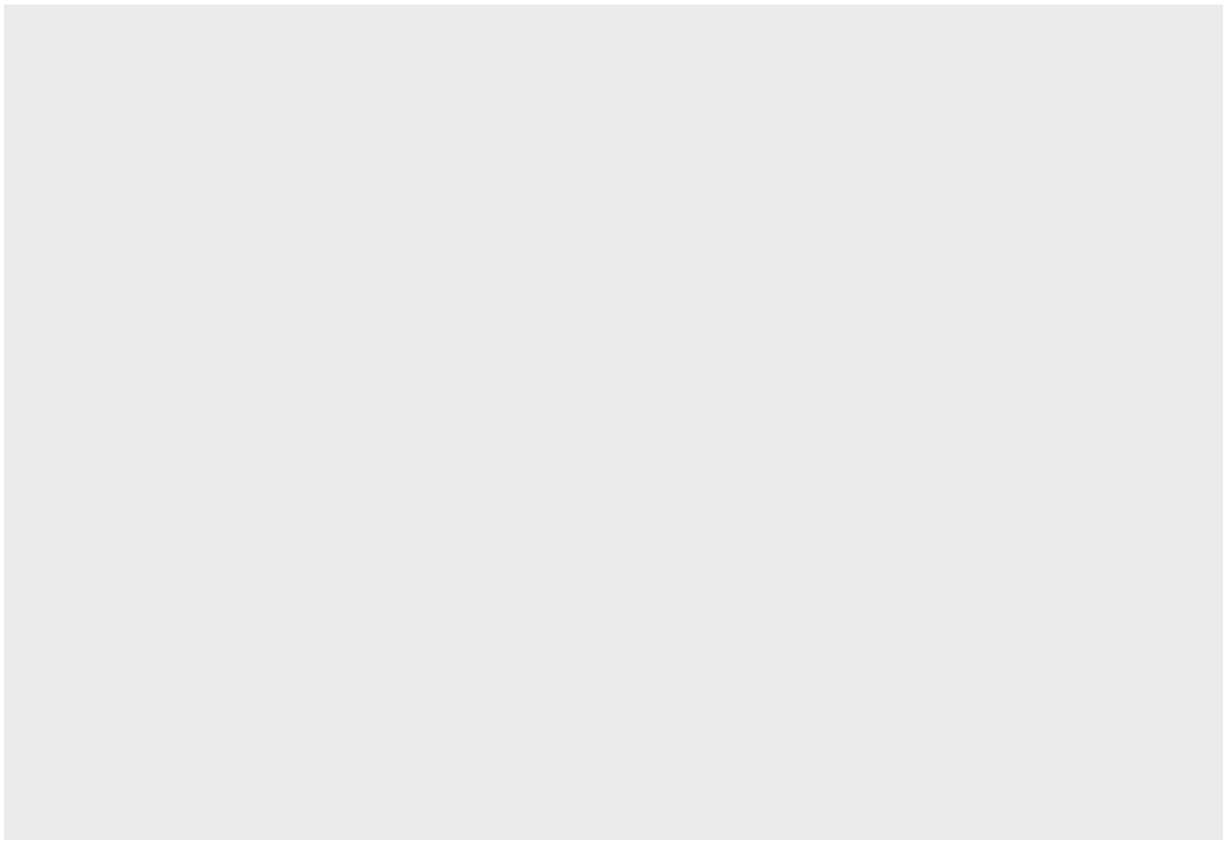
How does ggplot work?

Like dplyr, ggplot takes an object and adds "layers" to it.
However, while dplyr use piping ("%>%") ggplot uses the plus sign ("+").

Basic layers

Let's see what that means.

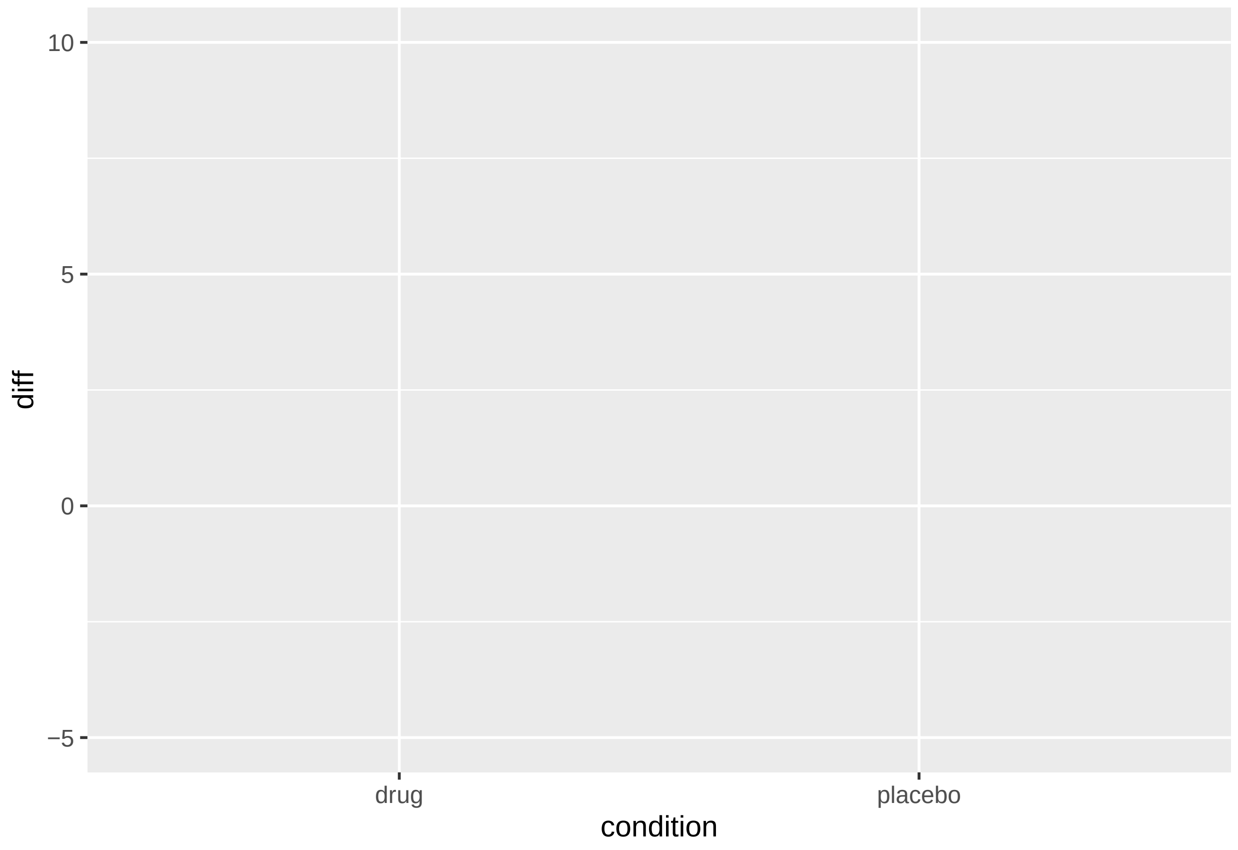
```
ggplot()
```



As you see, ggplot alone (i.e., without any arguments) provides a space for your plot to sit in.

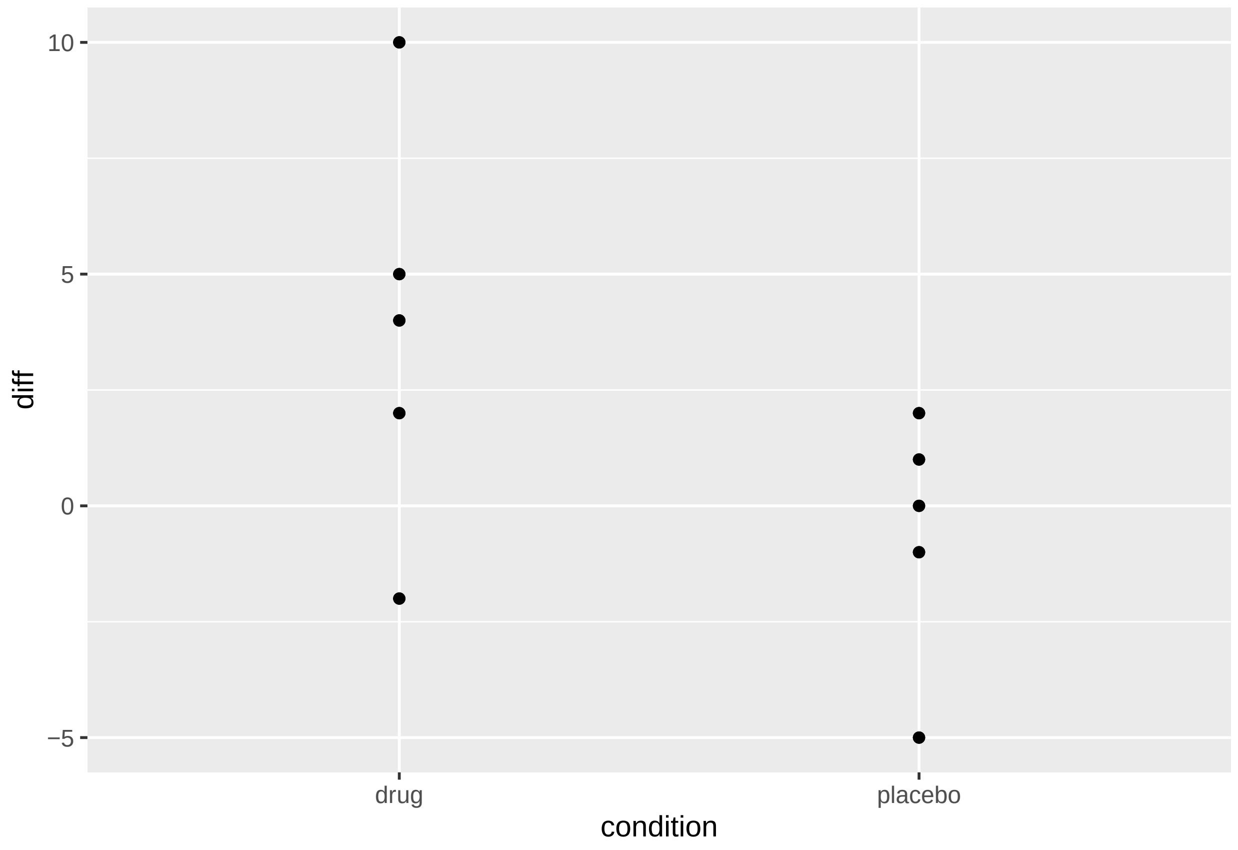
Now let's add axes.

```
ggplot(data = data,
       aes(x = condition, y = diff))
```



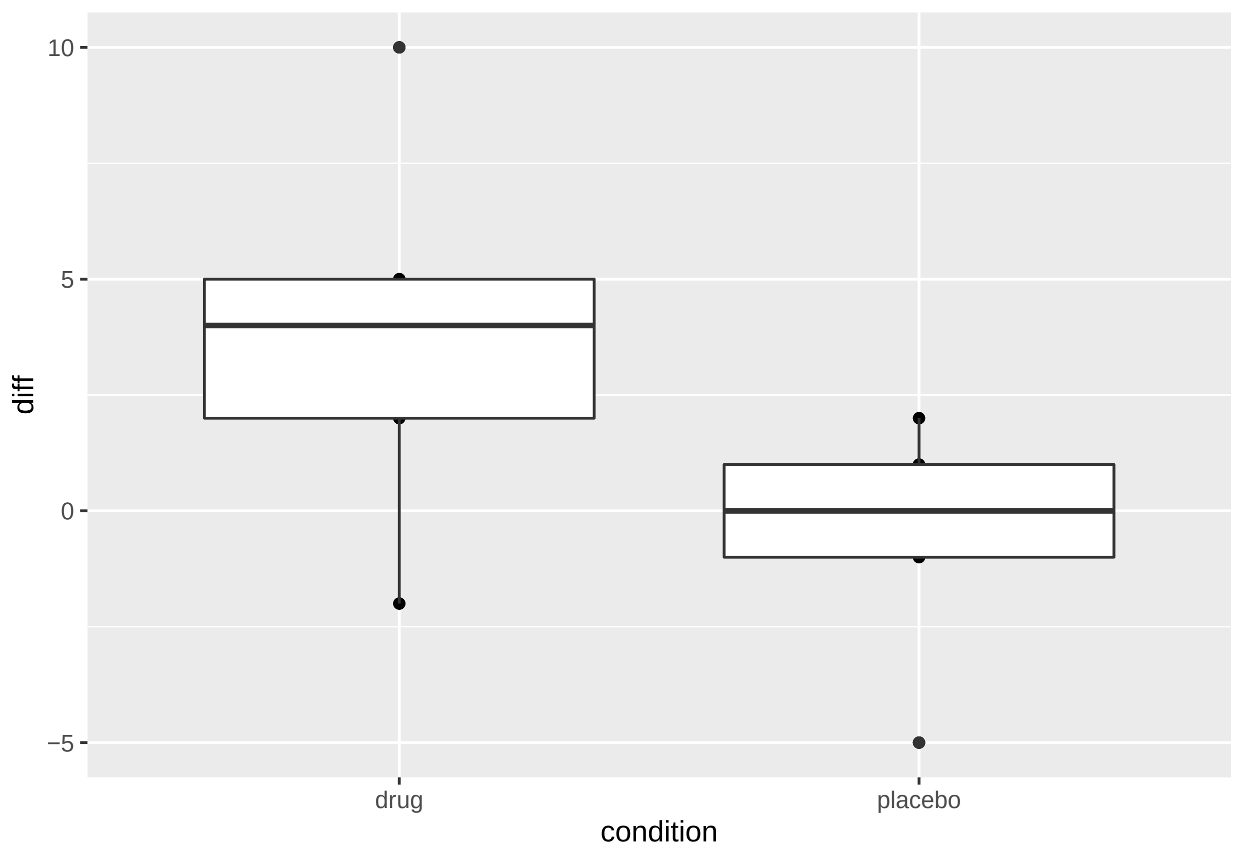
This just sets up the axes, but doens't actually put any data on the plot. To do that, let's add a "layer" with `+`

```
ggplot(data = data,
       aes(x = condition, y = diff)) +
  geom_point()
```



Let's experiment with some more layers:

```r
ggplot(data = data,
       aes(x = condition, y = diff)) +
  geom_point() +
  geom_boxplot()
```

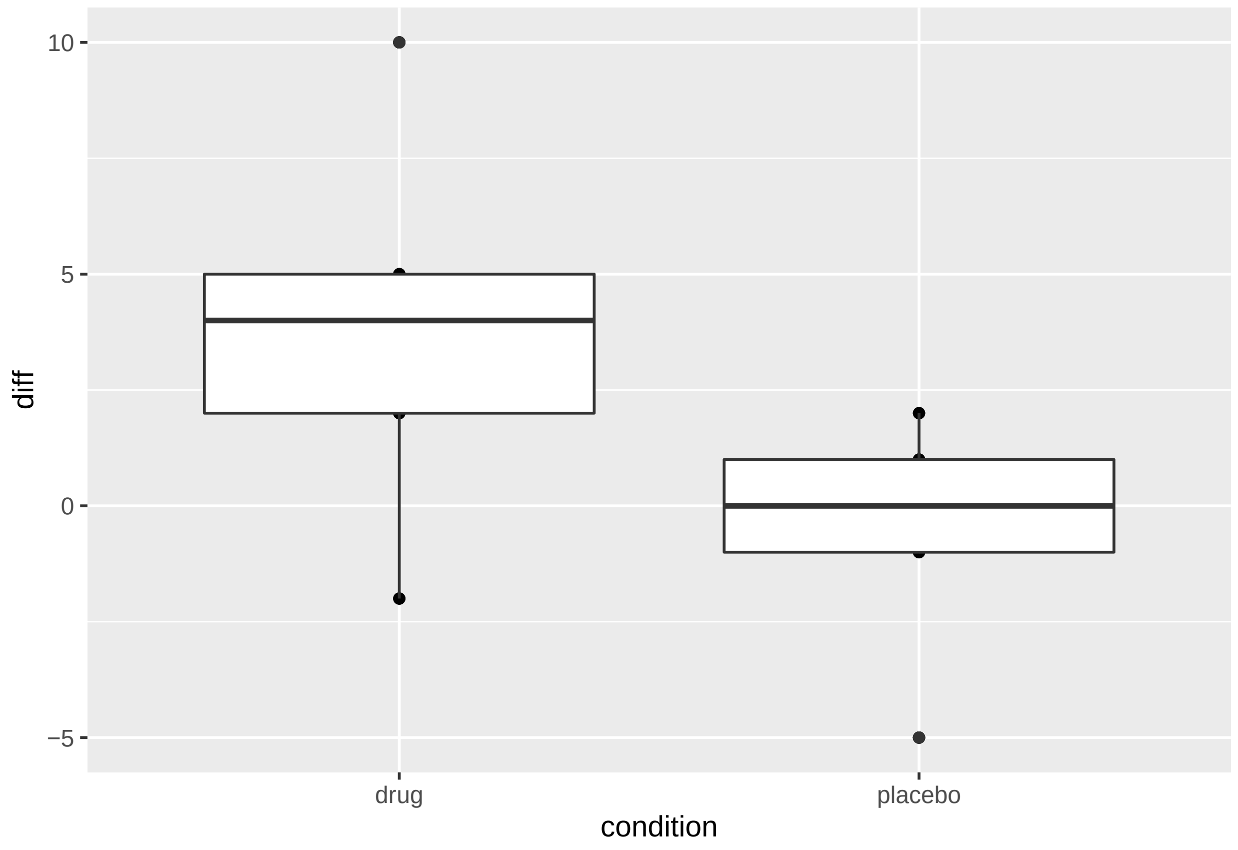


ggplot is very versatile. It can take many many layers, and each layer can take some arguments to customize it. But this is really the skeleton it what it does.

Basically, the first argument is 'data=' so that ggplot knows what dataset to work with.
Then, 'aes()' tells ggplot what the basic "aesthetics" of the plot are.
These include the x-axis, y-axis (when relevant) and color/fill settings
(if we want lines or bars in different colors in their outlines or filled
with different colors respectively).

Remember piping `%>%`? Let's pipe `data` into ggplot.

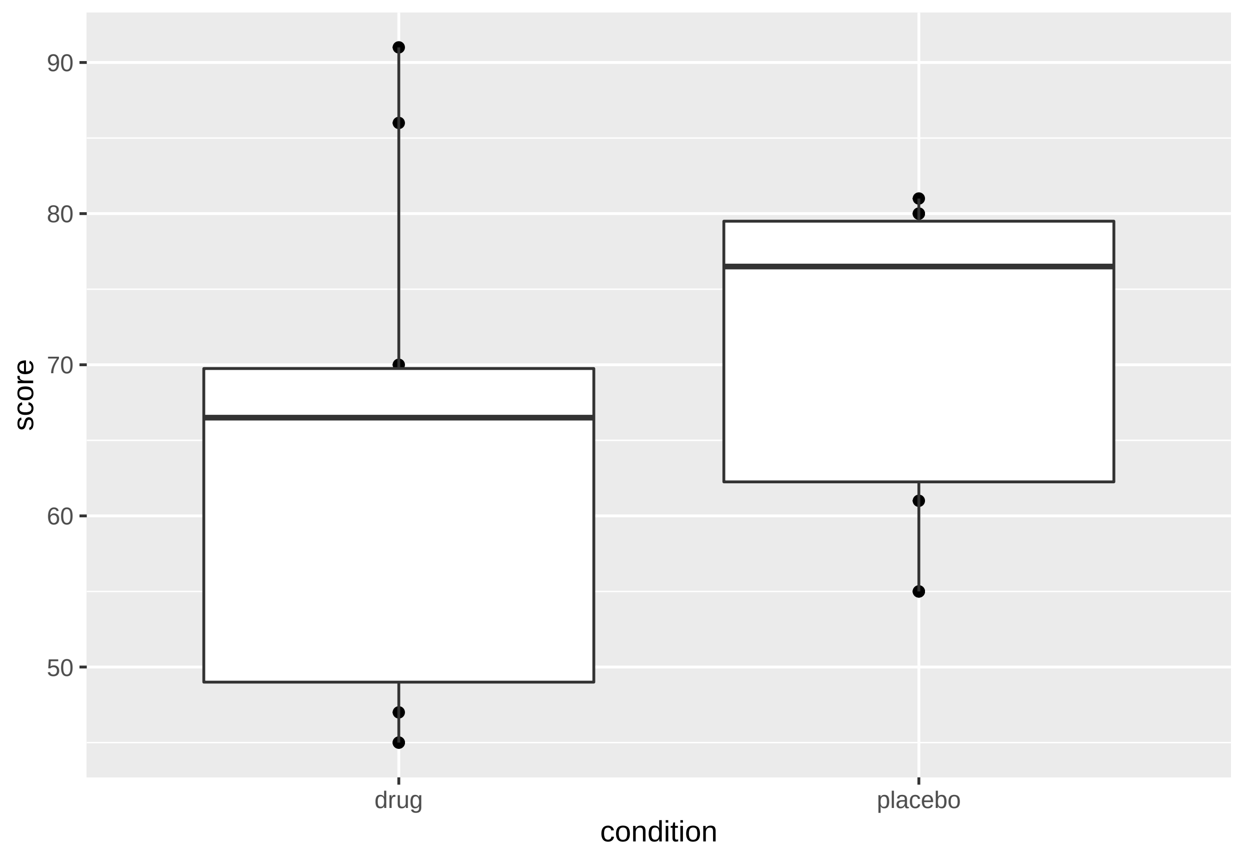
```
# take the dataset and feed it into ggplot's data argument
data %>%
  ggplot(data = .,
         aes(x = condition, y = diff)) +
  geom_point() +
  geom_boxplot()
```



We may want to save this plot into a variable.

```r
figure1 <- data %>%
  ggplot(data = .,
         aes(x = condition, y = score)) +
  geom_point() +
  geom_boxplot()

# just type the name of the plot to see it in here
figure1
```



Saving out a plot

This plot is now saved in our workspace as figure1. But what if I want to save it on my computer as a .png ?

```
ggsave(filename = "figure1.png",
       plot = figure1,
       path = "../figures/.")
```

```
## Saving 6.5 x 4.5 in image
```

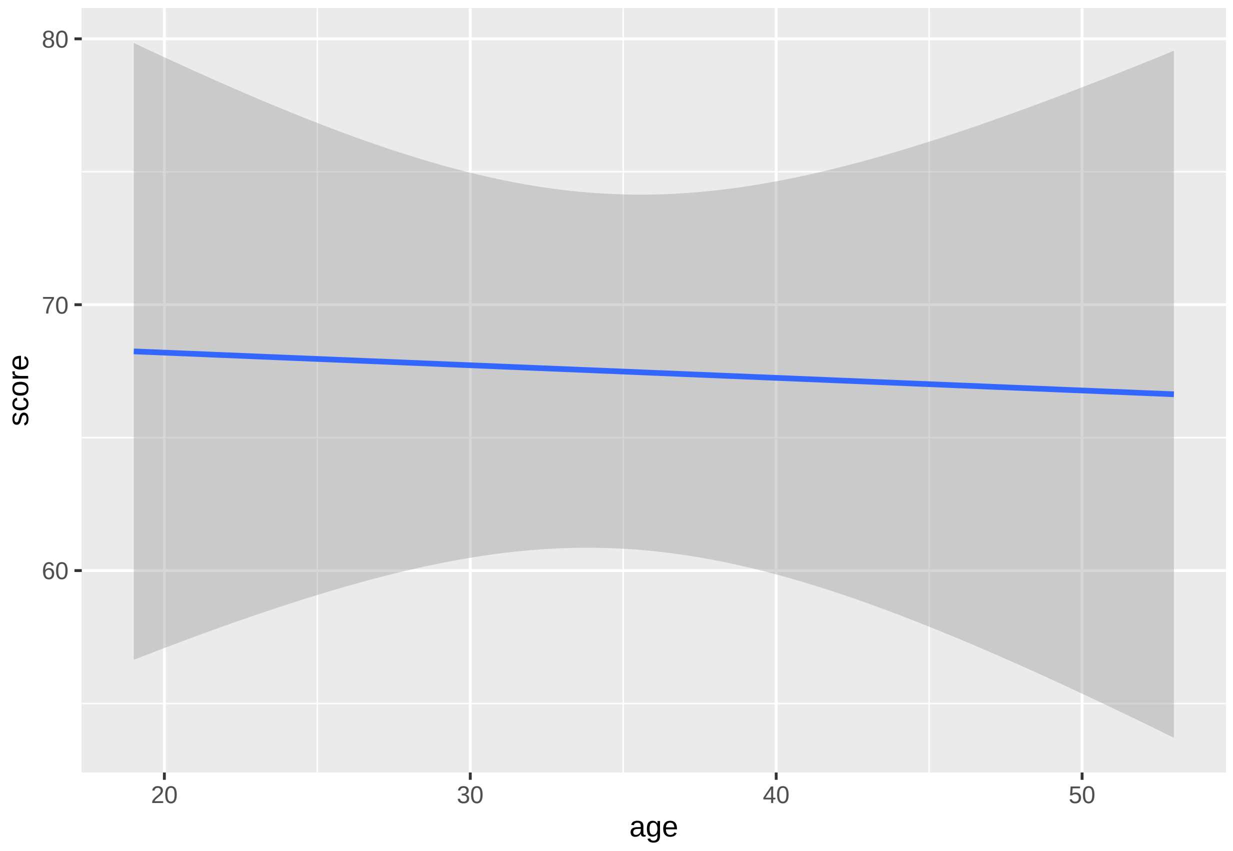
```
# if your plot's proportions look weird when saved,
# you can customize its dimensions as below
ggsave(filename = "figure1.png",
       plot = figure1,
       path = "../figures/.",
       width = 7, height = 4)
```

Geom objects

We added points and a line with `geom_point()` and `geom_boxplot()` But we could also add bars and boxplots and many other types of plots. To do this, ggplot uses `geom` objects, such as `geom_smooth()`

```
data %>%
  ggplot(data = .,
         aes(x = age, y = score)) +
  geom_smooth(method = "lm")
```

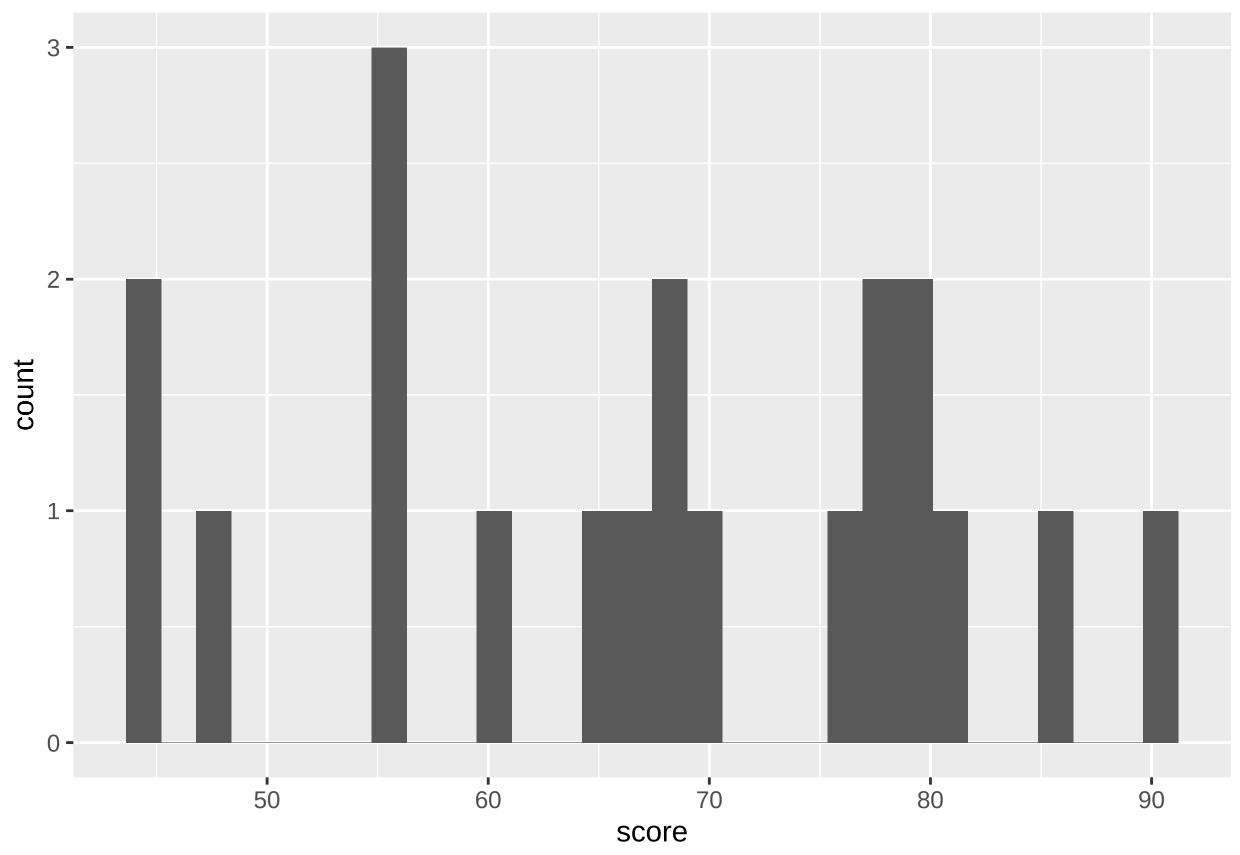
```
## `geom_smooth()` using formula 'y ~ x'
```



Or a histogram with:

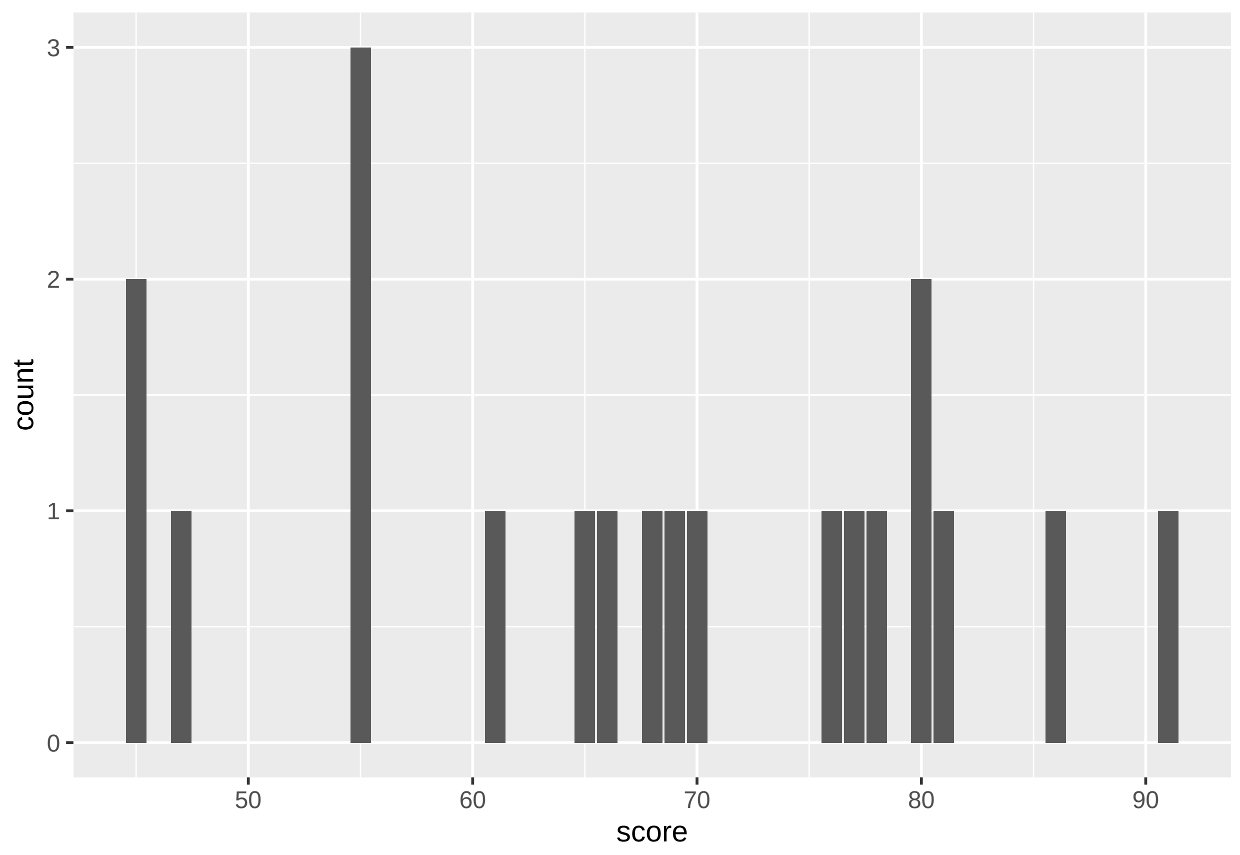
```
data %>%
  ggplot(data = .,
         aes(x = score)) +
  geom_histogram()
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



Or a bar plot:

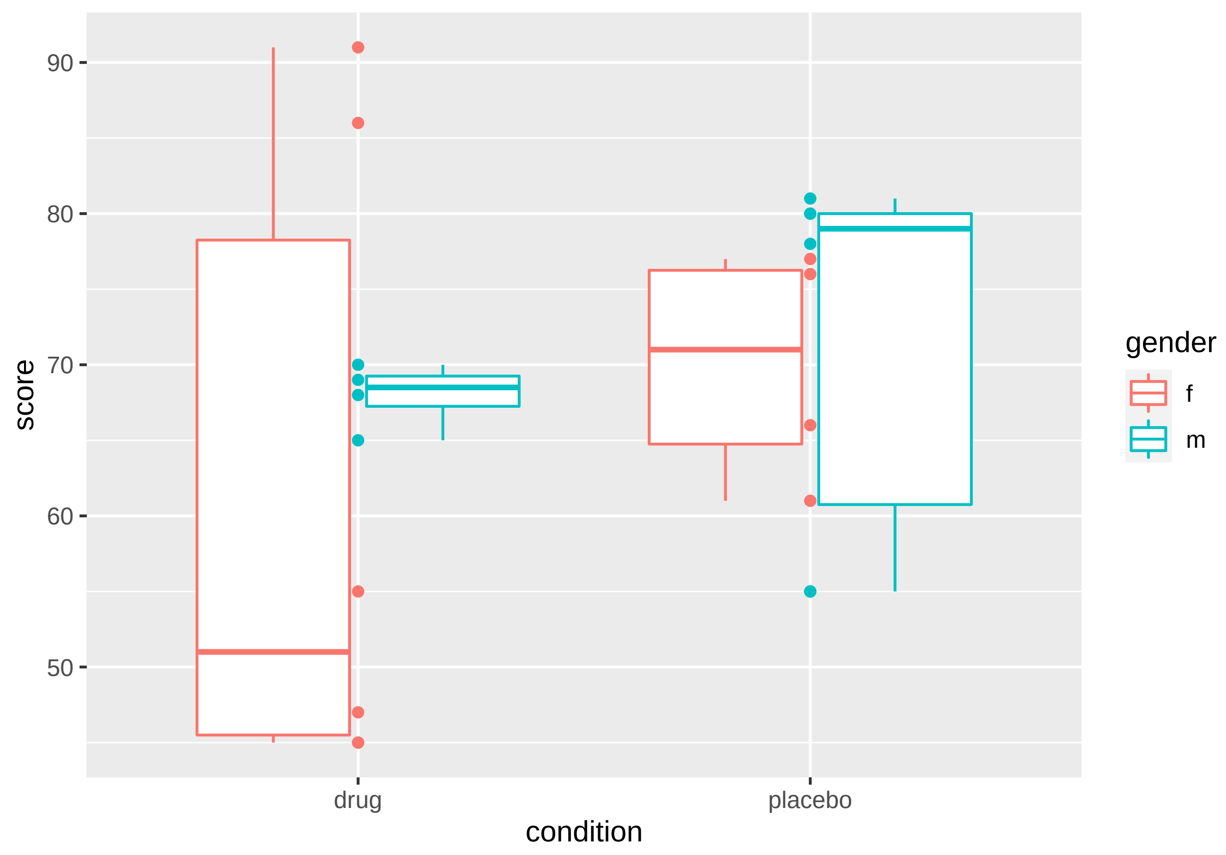
```
data %>%
  ggplot(data = .,
         aes(x = score)) +
  geom_bar()
```



Using facets and colors

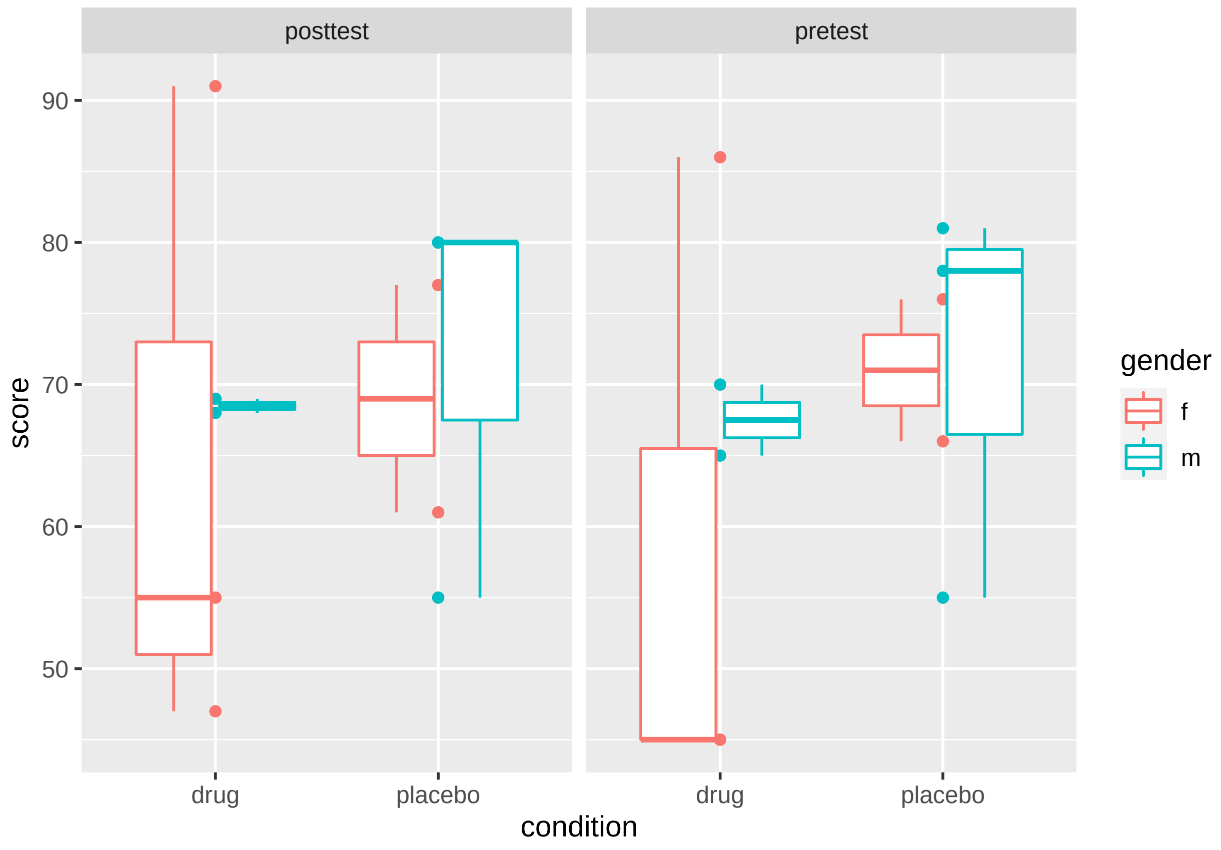
We may want to get a sense of how men and women are different. Let's take our initial plot, and show men and women with a different color.

```
data %>%
  ggplot(data = .,
         aes(x = condition, y = score, color = gender)) +
  geom_point() +
  geom_boxplot()
```



Now let's plot pretest and posttest separately.

```
data %>%
  ggplot(data = .,
         aes(x = condition, y = score, color = gender)) +
  geom_point() +
  geom_boxplot() +
  facet_grid(. ~ prepost)
```



```
# data %>%
#   ggplot(data = .,
#          aes(x = condition, y = score, color = gender, fill = gender)) +
#   geom_point() +
#   geom_boxplot() +
#   facet_grid(. ~ prepost)
```

(What's that ~ about? It means row by column.)

There's many more things you can do with ggplot.

- Here's a reference for `ggplot()`'s options, with examples.

To get a sense of all the possibilities, checkout: R graoh gallery

A few extra resources

- Take a moment to learn how to write R code with good style–that is, code that's readable and pleasant to look at.
- Check out dplyr's documentation for more info on what you can do with dplyr (and how to do it).
- For a reference guide to plotting data with ggplot, try the R Cookbook. Personally, I've found the author's published book, the *R Graphics Cookbook*, particularly helpful.

- For a general reference guide to using R for data science, i.e. all the sorts of stuff we've been doing, look at Grolemund & Wickham's eponymous book. (The whole book is online!)
- `tidyr` is a handy package for reshaping data from wide to long format and back.
- `broom` is a handy package for turning default output from regressions and t-tests into organized data frames.
- It's possible (and can be convenient) to set up a git pane and make commits directly from R.