LeaRning Week 3

LeaRning Team

6/15/2021

Introduction

```
##
##  -----
## Get ready to wrangle!
##  ------
##     \
##      \
##         (.)_(.)
##       _ (   _   ) _
##      / \/`-----'\/ \
##    __\ ( (     ) ) /__
##    )   /\ \._./ /\   (
##     )_/ /|\   /|\ \_(  [nosig]
##
```

Welcome back LeaRners! In week 1 we got the hang of basic syntax. Then in Week 2 we demystified "tidy data", read in our first data sets, and leaRned some basic operations to familiarize ourselves with our data. In Week 3, we will keep building our data literacy to hone down some essentials in handling data in R. These include: 1. Basic Vector Math 2. Filtering your data set 3. Performing pairwise comparisons (revisited from lesson 1) 4. Plotting our data: an introduction Plotting is a big topic in R. We will spend most of Week 4 on this.

Part 1: Vector Math

Now that we have some background about our CFU data set, we know that we've got some work to do before we analyze this data. Luckily, all this math is simple to perform in R, and more importantly, can be saved and applied to similar data sets in the future. Let's start by reading in our data with read.csv().

```
# load data
dat <- read.csv("Week2_data frame basics/LeaRning_week2_cfu_data.csv")
```

We have three calculations to do in a stepwise order. Let's tackle these one-by-one.

Calculations part 1: CFU per mL

To calculate CFU per mL: number of colonies / volume plated * the dilution factor. What columns represent these?

```
colnames(dat)
```

```
##  [1] "animal.id"          "cage"               "experiment.date"
##  [4] "sex"                "experimental.group" "tissue"
##  [7] "dilution.factor"    "homogenate.volume"  "plated.volume"
## [10] "tissue.weight"      "no.cfu"
```

Number of CFU is cfu.no Volume plated is plated.volume Dilution factor is dilution.factor

```
# let's make a new column called cfu.per.ml
dat$cfu.per.ml <- dat$no.cfu / dat$plated.volume * dat$dilution.factor
```

Calculation Part 2: Total number of CFU

To calculate total CFU: CFU per mL * sample volume CFU per mL is cfu.per.ml sample volume is homogenate.volume

```
# let's make a new column called cfu.total
dat$cfu.total <- dat$cfu.per.ml * dat$homogenate.volume
```

Calculation 3: cfu per gram tissue

To calculate CFU per gram tissue: CFU total / tissue weight in grams CFU total is cfu.total Tissue weight in grams is tissue.weight

```
# let's make a new column called cfu.g.tissue
dat$cfu.g.tissue <- dat$cfu.total / dat$tissue.weight
```

Now we've got it! We can start to proceed with data analysis, AFTER doing one more thing: Filtering our data!!!

Part 2: Filtering Data

Here, we will look at four different ways to filter the data set by tissue type.

The first way is to do this in "base" R, using the `which()` function in combination with our square brackets for navigation::

```
# let's create a new df containing only the liver data

liver_data <- dat[which(dat$tissue == "liver"),]

# now one for the spleen data

spleen_data <- dat[which(dat$tissue == "spleen"),]

# and finally one for lung

lung_data <- dat[which(dat$tissue == "lung"),]
```

The "==" is used here to say that "keep rows in the data frame where the values in the tissue column **exactly equal to** to liver/spleen/lung".

Q: What if we just wanted to exclude "liver" samples? How can we achieve that? Recall the logical operators from week 2:

```
non_liver <- dat[which(dat$tissue != "liver"),]
```

The second way is to use the in-built base R function called `subset()`. Here is an example of how we could use this instead of the `which()` function to subset the data into tissue types:

```
liver_data <- subset(dat, dat$tissue == "liver")

spleen_data <- subset(dat, dat$tissue == "spleen")

lung_data <- subset(dat, dat$tissue == "lung")
```

```
# notice how we don't need to navigate to particular rows for this function to work
# i.e. no square brackets
```

The third way is to use the package `dplyr` within the `tidyverse` - the tidyverse is a collection of packages that are very popular for data manipulation, exploration and visualization https://www.tidyverse.org/. If you haven't already installed it, let's do it now:

```
install.packages("tidyverse")
```

Now, load the package so that we can access the functions inside:

```
library(tidyverse)
```

```
## -- Attaching packages --------------------------------- tidyverse 1.2.1 --
## v ggplot2 3.1.0       v purrr   0.3.1
## v tibble  2.0.1       v dplyr   0.8.0.1
## v tidyr   0.8.3       v stringr 1.4.0
## v readr   1.3.1       v forcats 0.4.0
## -- Conflicts ------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

Now, we will use the `filter()` function within the `dplyr` package to subset the data:

```
liver_data <- filter(dat, tissue == "liver")

spleen_data <- filter(dat, tissue == "liver")

lung_data <- filter(dat, tissue == "liver")

# similar to the subset() function, but with even less code!
```

Finally, we can use a function called `split()` within base R, to divide our object "dat" into three based on the column "tissue":

```
dat_split <- split(dat, dat$tissue)
```

Even less code! But what format of data has this returned?

If we want to return each element of the list to the global environment as separate objects we can do so:

```
list2env(dat_split,globalenv())
```

```
## <environment: R_GlobalEnv>
```

Part 3: Performing pairwise comparisons (revisited from lesson 1)

Let's use the data that we generated in Part 1 (CFU per gram of tissue) to compare our Treatment and Control samples for each tissue using a Wilcoxon test.

First, visit the help page for a simple Wilcoxon test comparing the means of two independent samples:

```
?wilcox.test
```

Now, perform a Wilcoxon test for each tissue, and print the results. **Note, you can supply two vectors as arguments "x" and "y", or you can :

```
## SPLEEN
```

```
res.spleen <- wilcox.test(cfu.g.tissue ~ experimental.group, data = spleen,
                   exact = FALSE)
res.spleen
```

```
##
##  Wilcoxon rank sum test with continuity correction
##
## data:  cfu.g.tissue by experimental.group
## W = 255, p-value = 1.649e-06
## alternative hypothesis: true location shift is not equal to 0
```

If we just want to print the p-value:

```
res.spleen$p.value
```

```
## [1] 1.648756e-06
```

```
## LIVER

res.liver <- wilcox.test(cfu.g.tissue ~ experimental.group, data = liver,
                   exact = FALSE)
res.liver
```

```
##
##  Wilcoxon rank sum test with continuity correction
##
## data:  cfu.g.tissue by experimental.group
## W = 196, p-value = 0.01055
## alternative hypothesis: true location shift is not equal to 0
```

```
## LUNG

res.lung <- wilcox.test(cfu.g.tissue ~ experimental.group, data = lung,
                   exact = FALSE)
res.lung
```

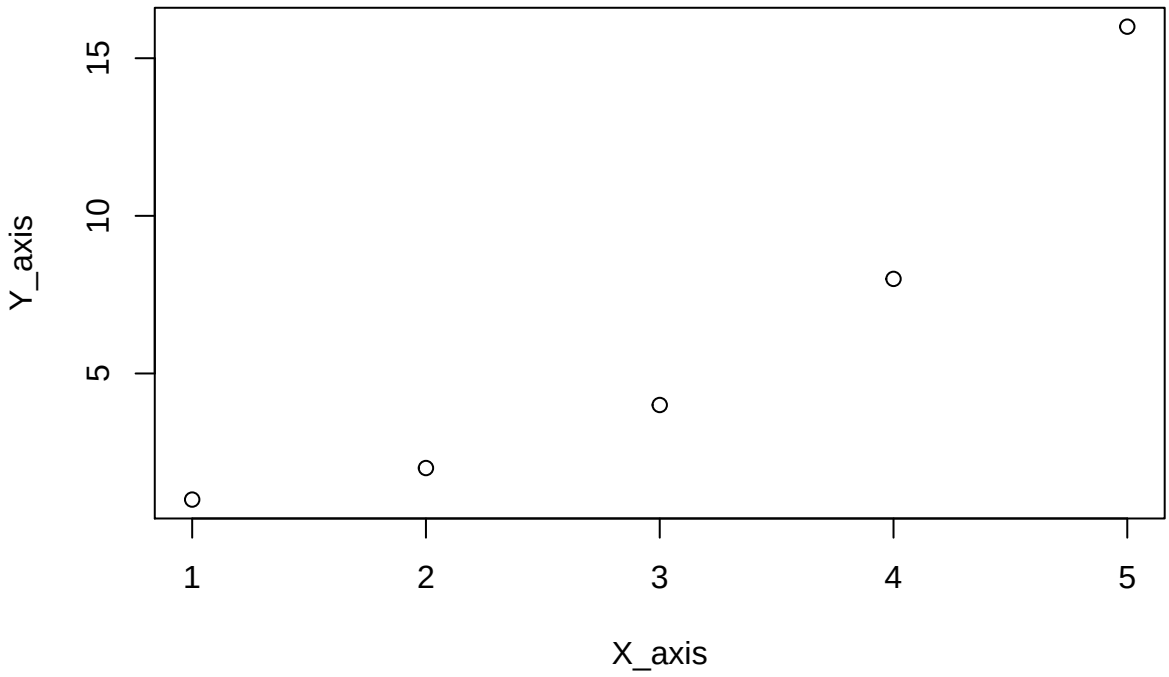
```
##
##  Wilcoxon rank sum test with continuity correction
##
## data:  cfu.g.tissue by experimental.group
## W = 229, p-value = 1.609e-05
## alternative hypothesis: true location shift is not equal to 0
```

Are these results expected? A great way to check is to visualise the data, let's look a couple of ways to plot the data for each tissue.

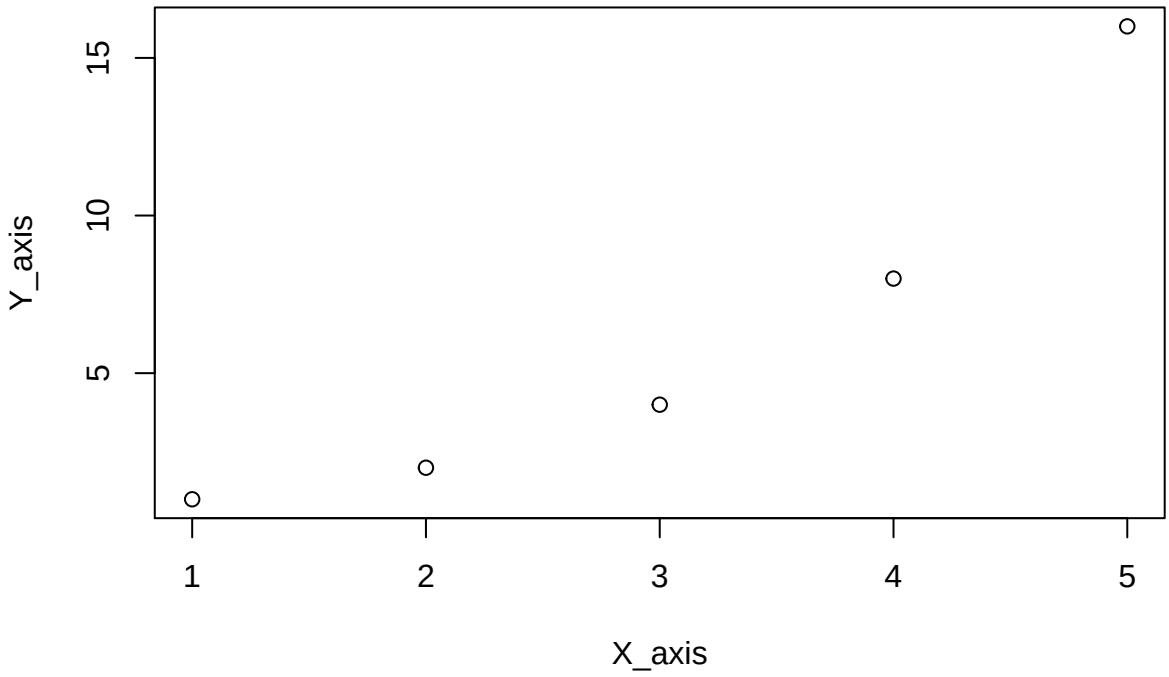
Part 4: Plotting our data: an introduction*

First, let's revisit basic plots using base R.

```
# Reminder
X_axis <- c(1,2,3,4,5)
Y_axis <- c(1,2,4,8,16)
plot(x = X_axis, y = Y_axis)
```

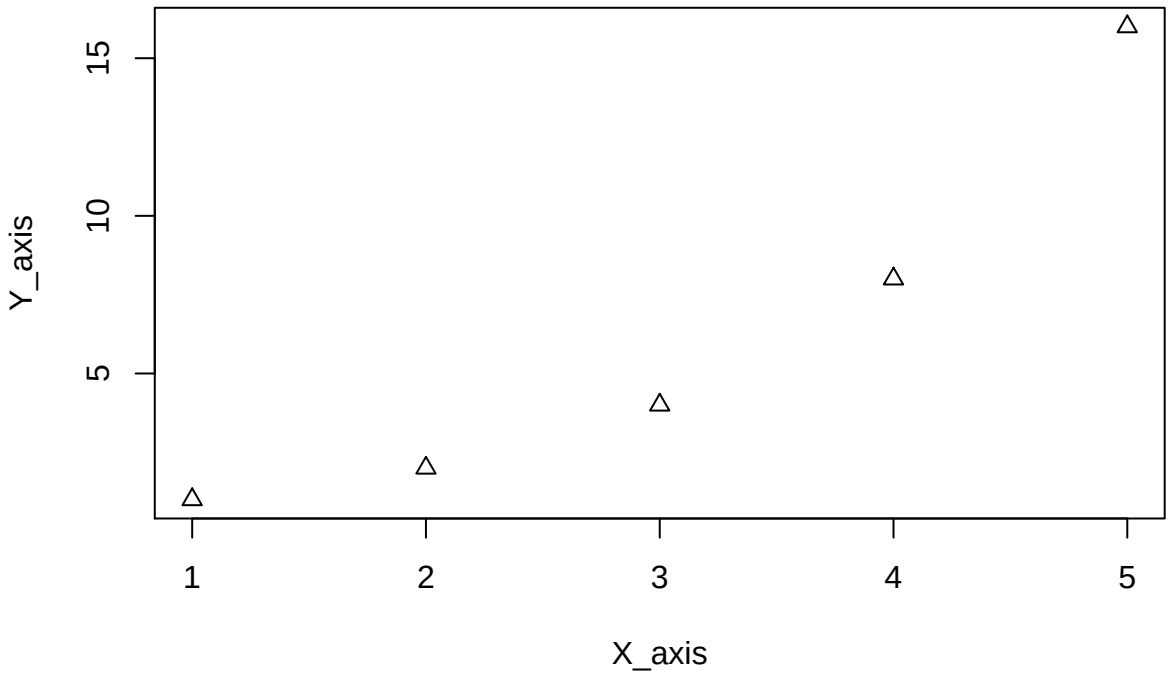



```
# Which can be shortened to:
plot(X_axis,Y_axis)
```

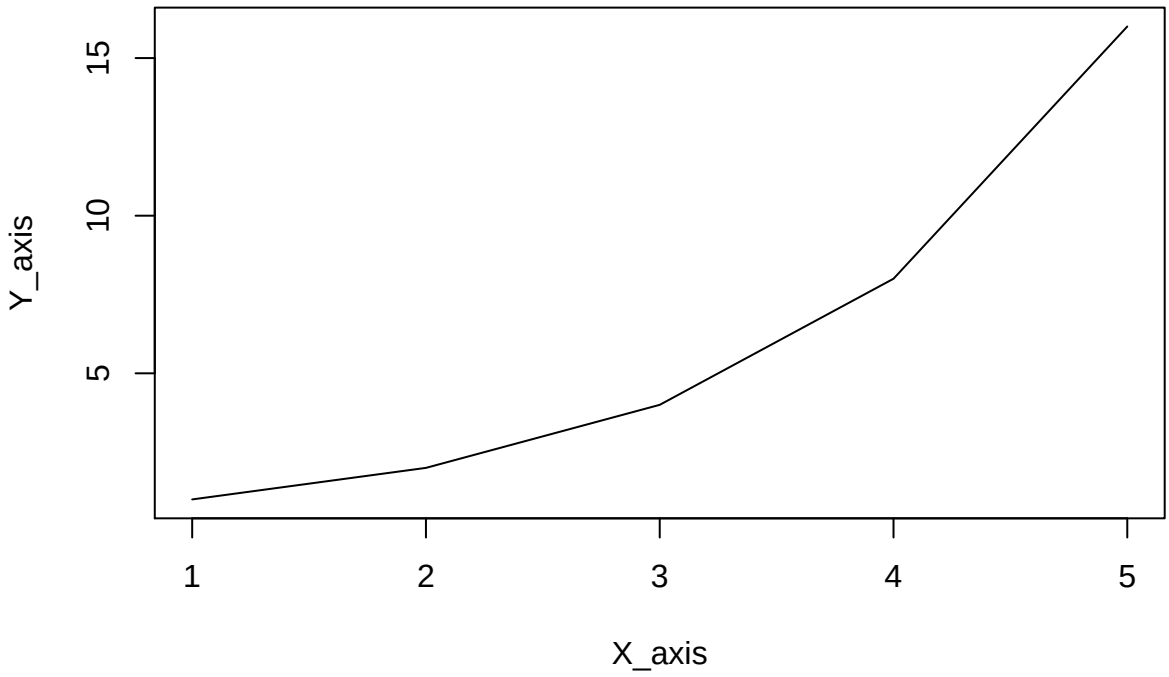


When having a look at the documentation for the plot function using ?plot(), we can see that there are a few things we can use to modify our plot. Have a look:

```
plot(X_axis,Y_axis, pch = 2)
```

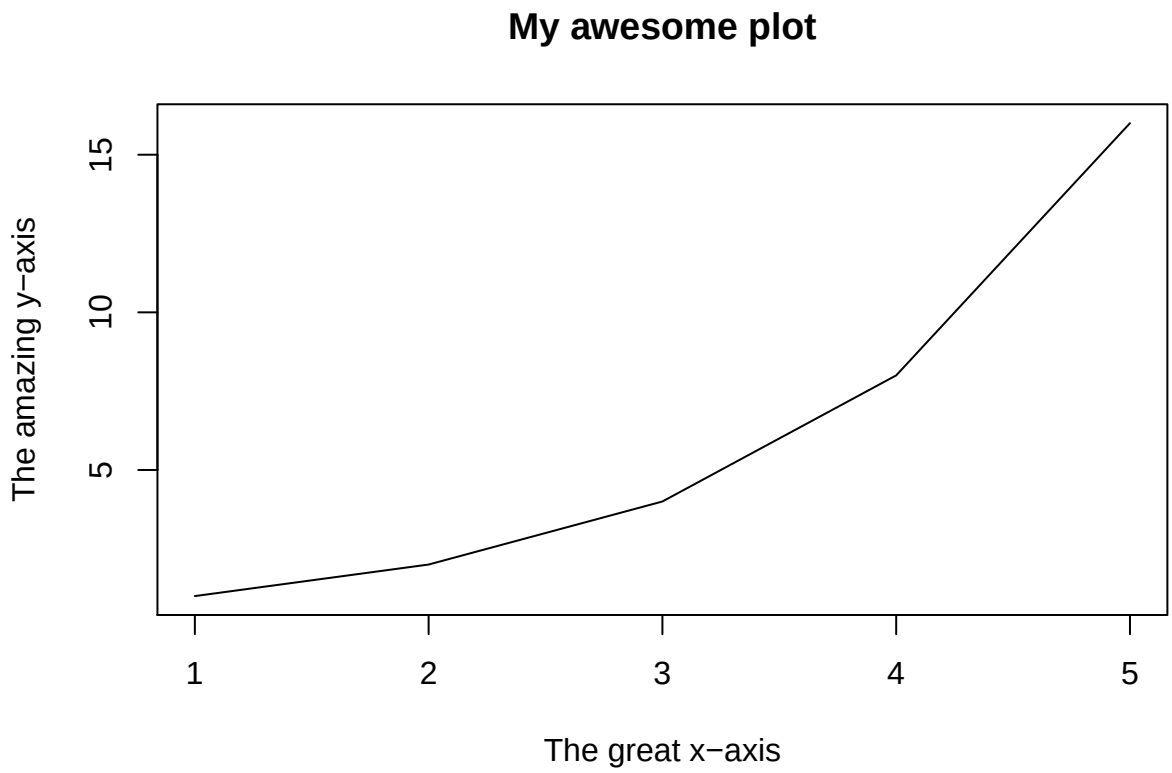



```r
# or
plot(X_axis,Y_axis, type = 'l')
```



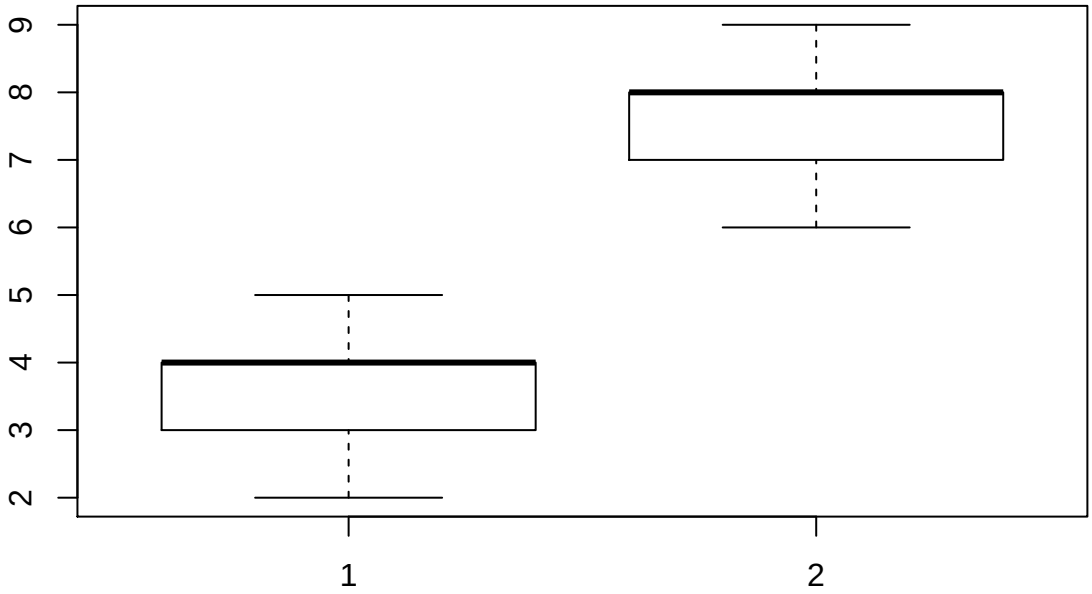

```r
# Adding some labels:
plot(
  X_axis,
  Y_axis,
  type = 'l',
  main = 'My awesome plot',
  xlab = 'The great x-axis',
  ylab = 'The amazing y-axis'
```

```
)
```



As previously shown in our first lesson, base R also has a basic boxplot() function.

```
a <- c(2,4,3,4,5)
b <- c(8,7,8,9,6)
boxplot(a,b)
```



However, plotting using base R functions often lacks intuition for more complex datasets and does not support the more advanced customization that is sometimes needed for them.

One of the best graphics packages in R which can generate publication-worthy figures is called ggplot2. Let's

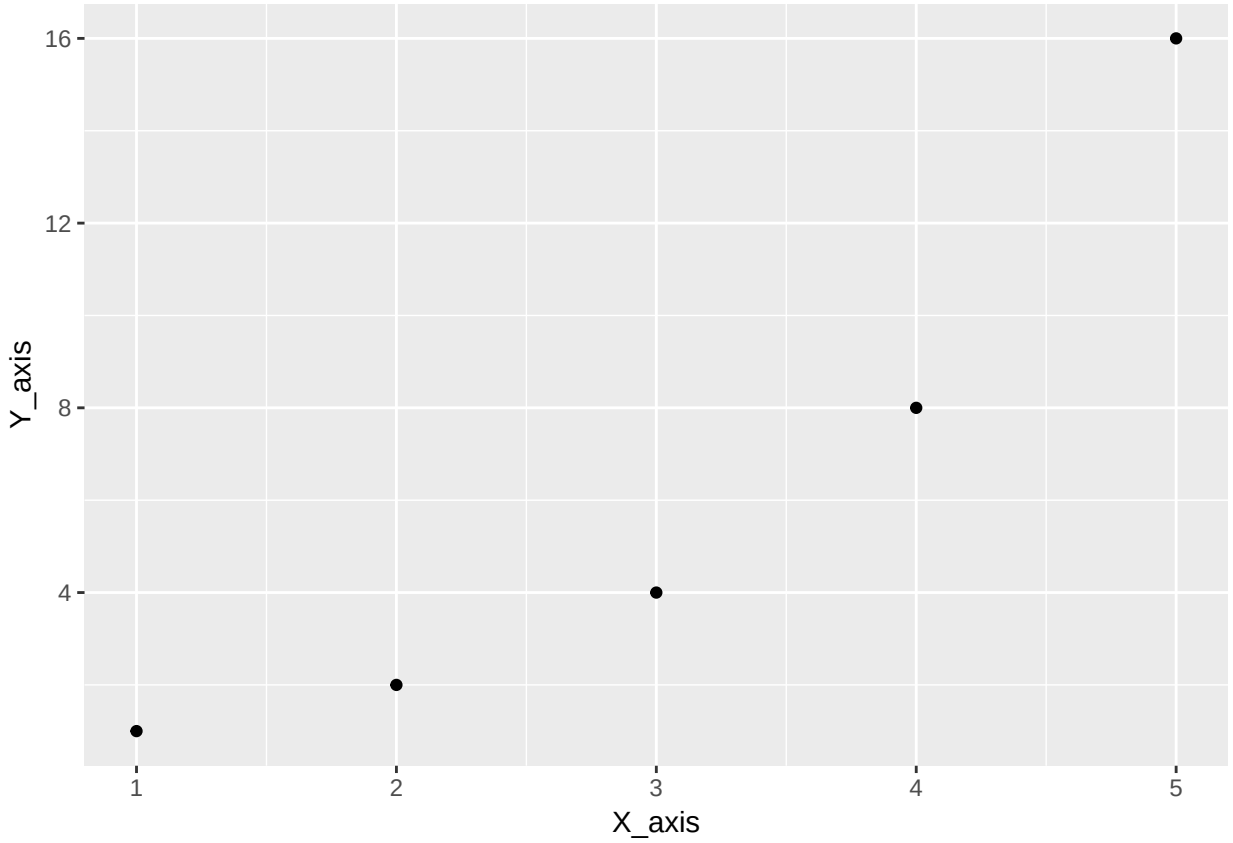
install ggplot2 and load it to our current session. Note: installing the package tidyverse also installs ggplot2, think of tidyverse as a package of packages. As we have already installed tidyverse, we only need to load ggplot2 into our current session.

```
library(ggplot2)
```

The package ggplot2 comes with a wide variety of functions, but really there are two functions that can be used for *almost* anything. qplot() and ggplot(). 'qplot' stands for 'Quick plot' and is a shortcut function designed for people who are more familiar with base plot(), while allowing for more intuitive graph customization. As its name indicates, it is very handy for quick visualizations of your data and it is elegant and easy to use.

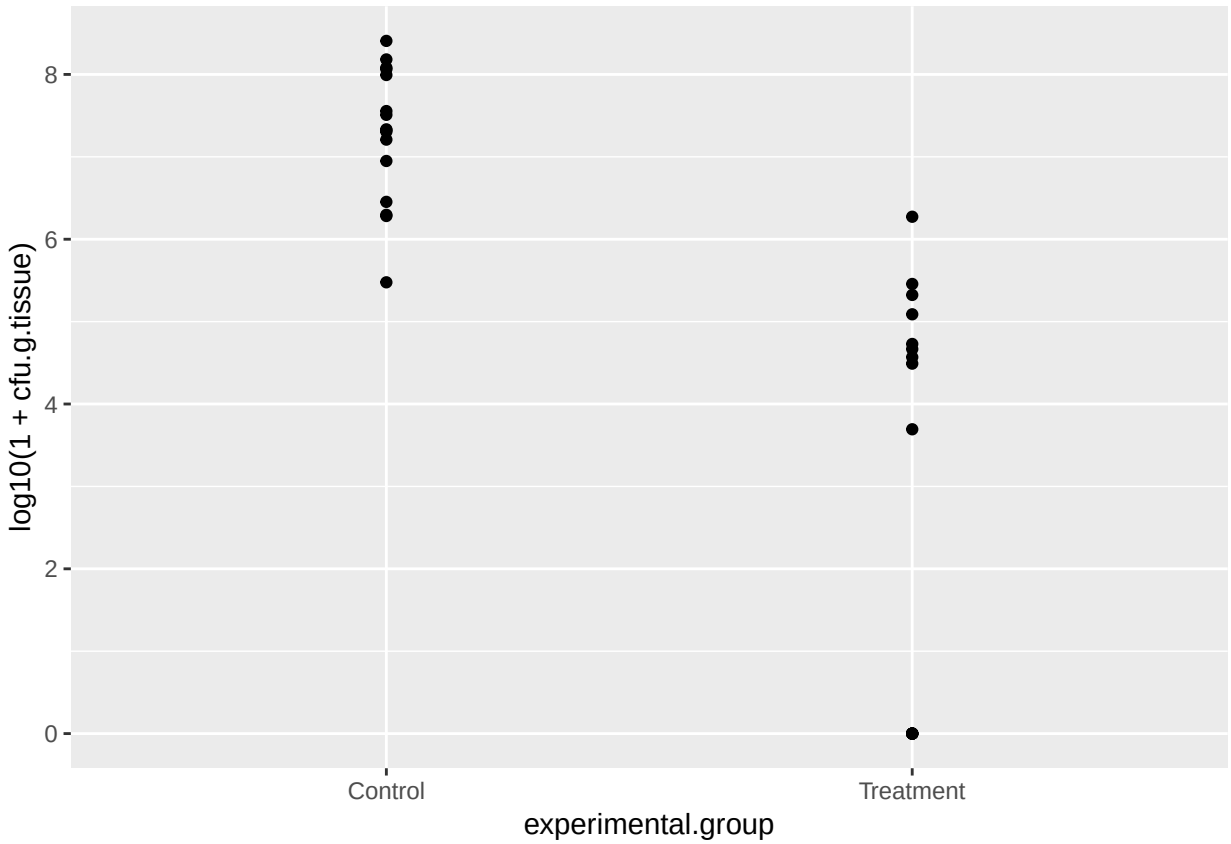
```
# Have a look at the documentation
?qplot()
```

```
# Using our X_axis and Y_axis:
qplot(X_axis, Y_axis)
```

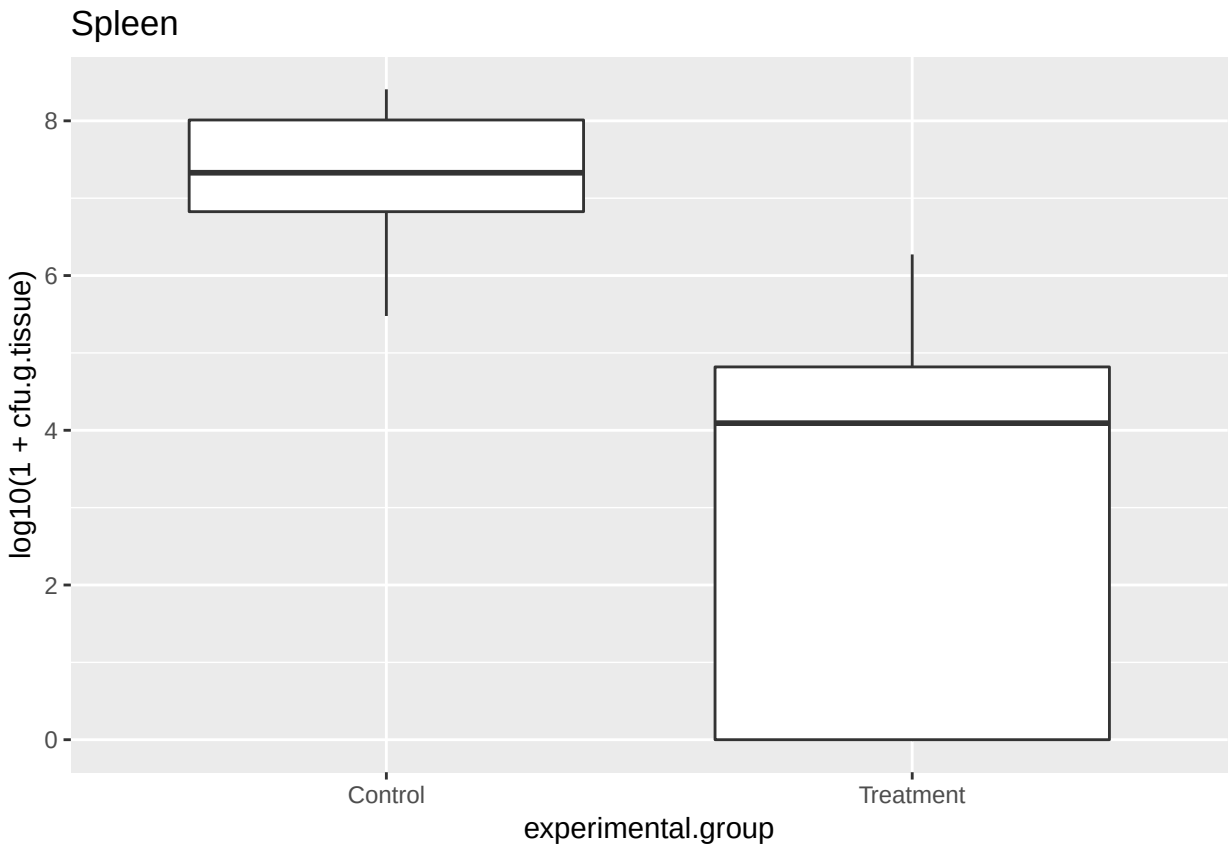



```
# We can see that ggpot2 brings in some nice default graphics compared to base R.
```

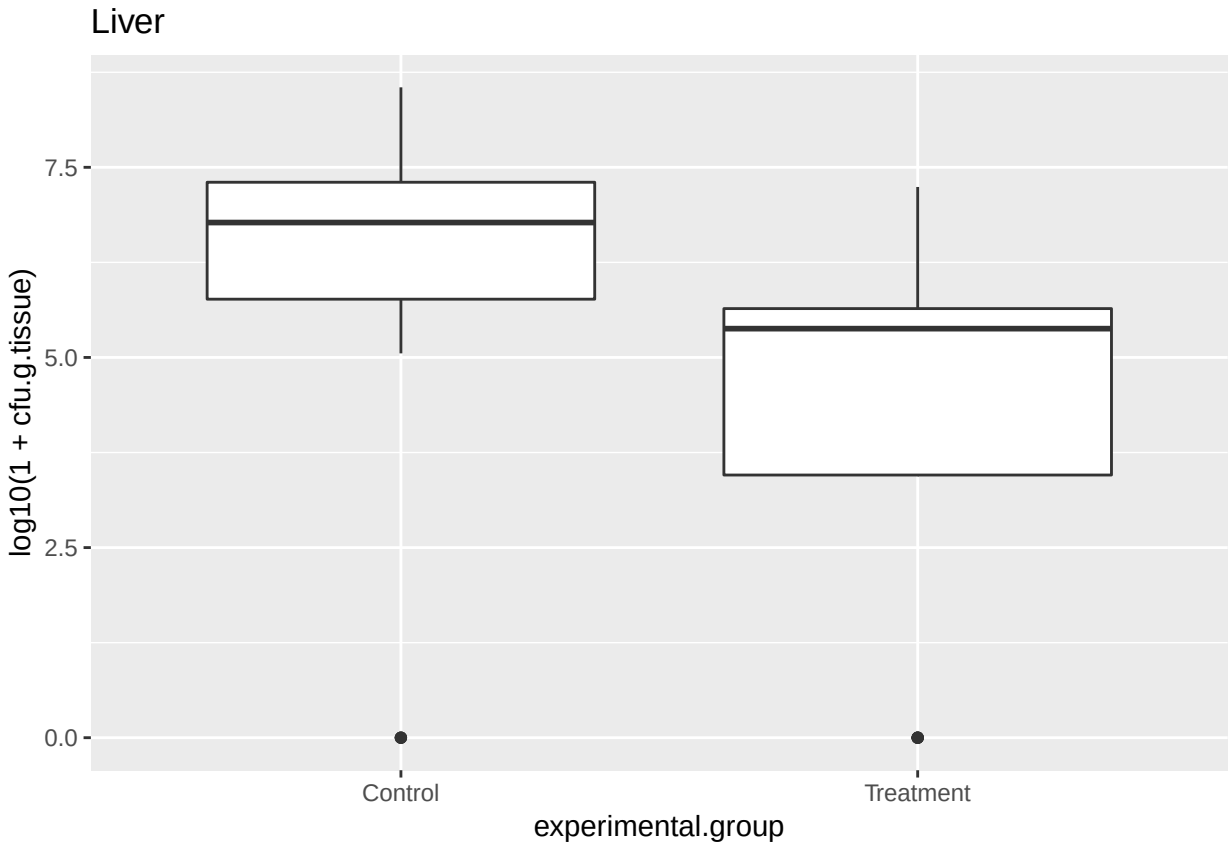
```
# Let's have a look at our data now!
qplot(experimental.group,log10(1+cfu.g.tissue), data = spleen)
```




```
# Let's make it a boxplot and add a title.
qplot(experimental.group,log10(1+cfu.g.tissue), data = spleen, geom = 'boxplot', main = 'Spleen')
```

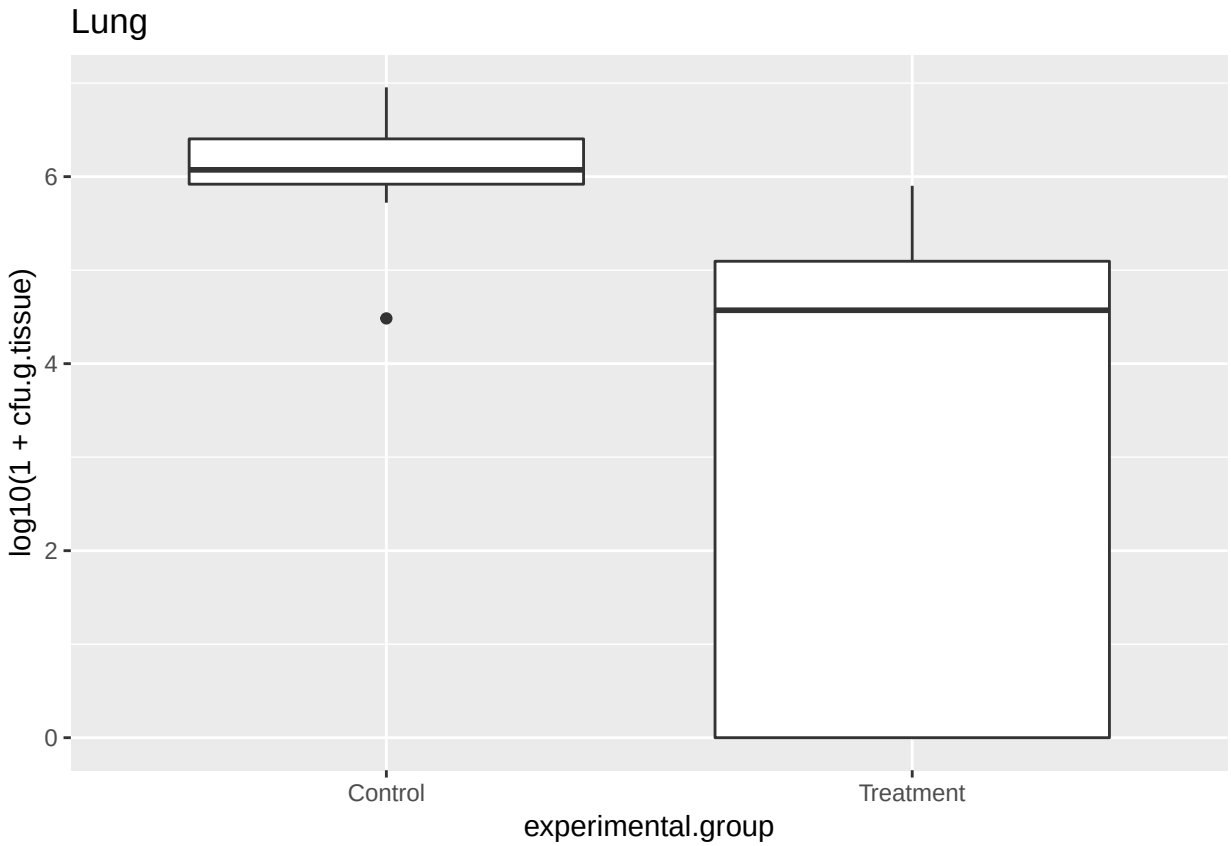



```
# For each group:
qplot(
  experimental.group,
  log10(1 + cfu.g.tissue),
  data = liver,
  geom = 'boxplot',
  main = 'Liver'
  )
```




```
qplot(
  experimental.group,
  log10(1 + cfu.g.tissue),
  data = lung,
  geom = 'boxplot',
  main = 'Lung'
  )
```

```
## Warning: Removed 1 rows containing non-finite values (stat_boxplot).
```

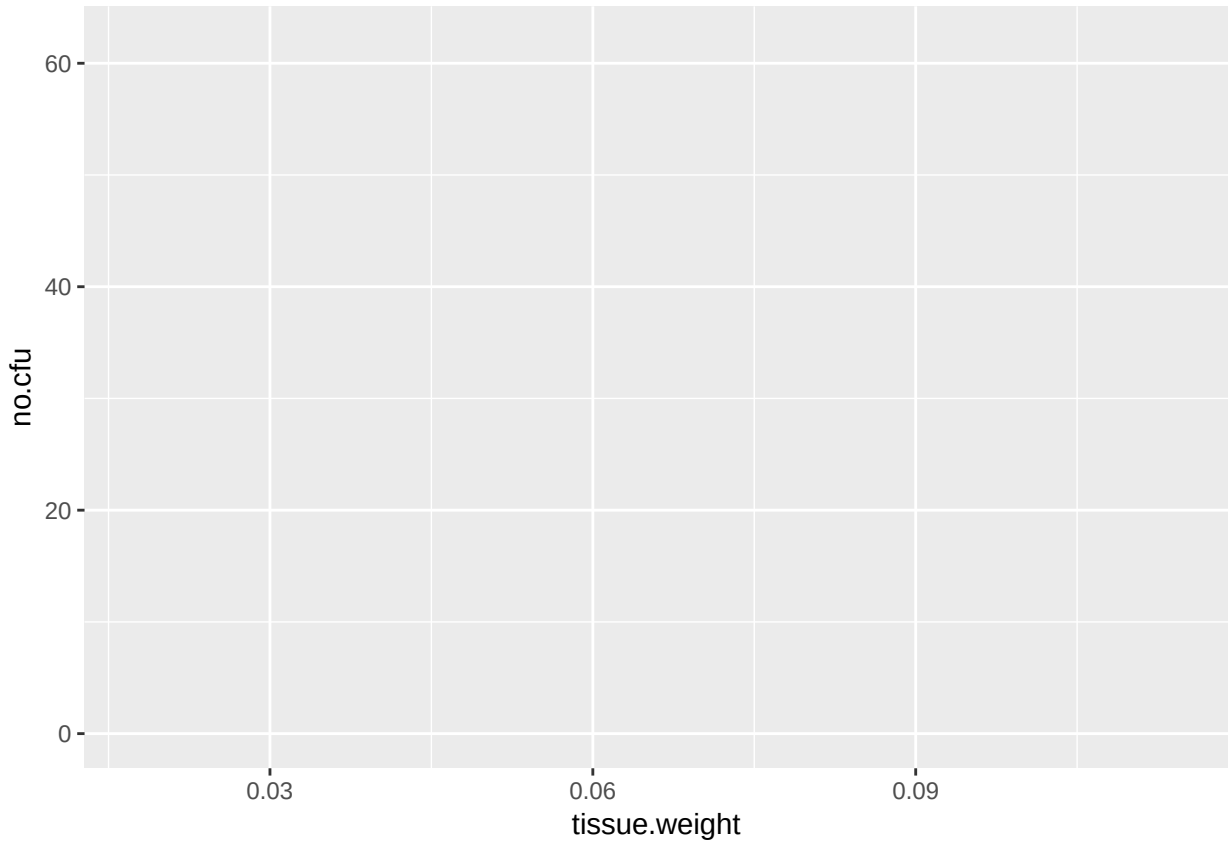


Now let's look at the real superpower of the ggplot2 package. The mother of all plots, ggplot()!

```
?ggplot()
```

Where qplot() could interpret what you meant in a single function, ggplot requires some more details. ggplot() works by creating an 'empty' plot, essentially creating the boundaries of the graph using the data you gave it. After having created the aesthetics of your plot, you can subsequently add 'layers' to it.

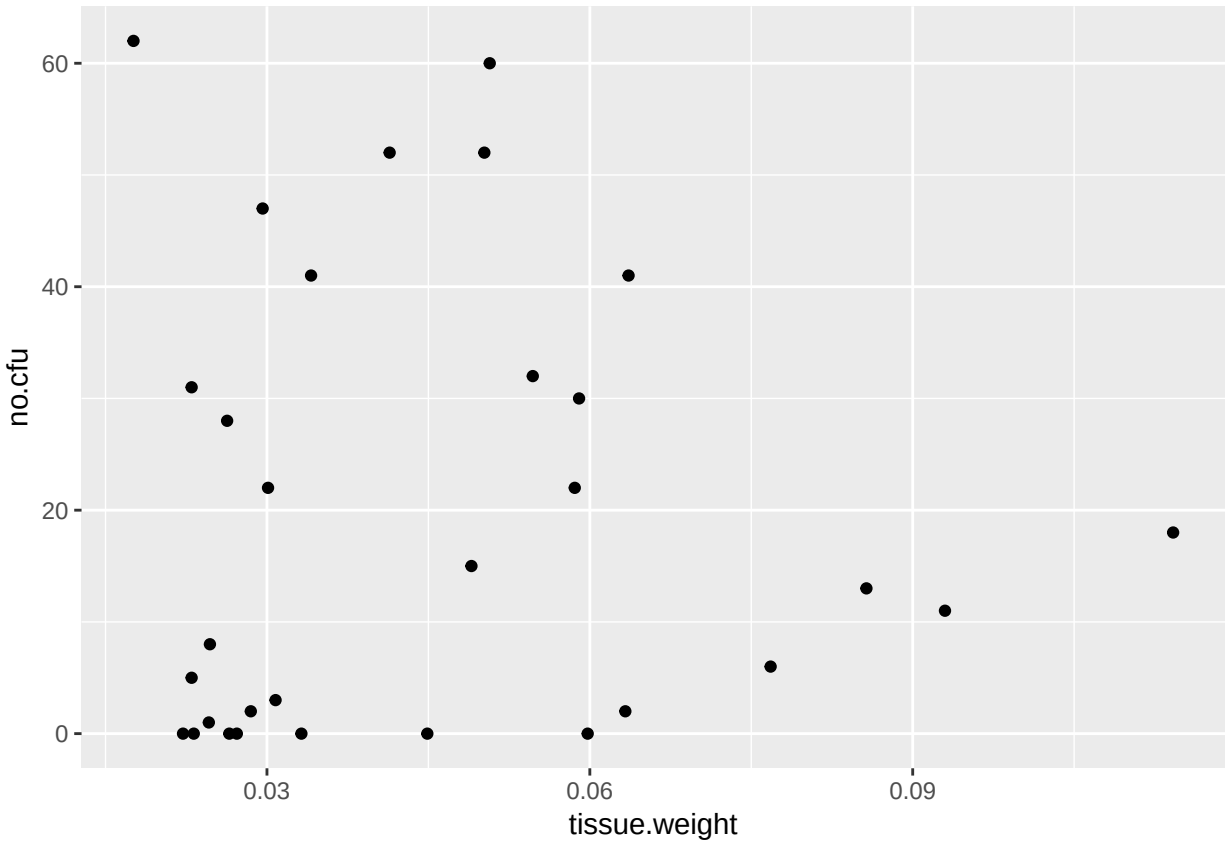
```
# Let's have a look at it by making a simple scattergraph of no.cfu by tissue weight.
# We first use ggplot() to initiate the aesthetics of our plot and tell it which data to use for it.
ggplot(data = lung, aes(x = tissue.weight, y = no.cfu))
```



We can see that the boundaries of our plot were created, but our points were not plotted.

```
# Let's plot them!
# First we define our boundaries, then we add our points.
ggplot(data = lung, aes(x = tissue.weight, y = no.cfu)) + geom_point()
```

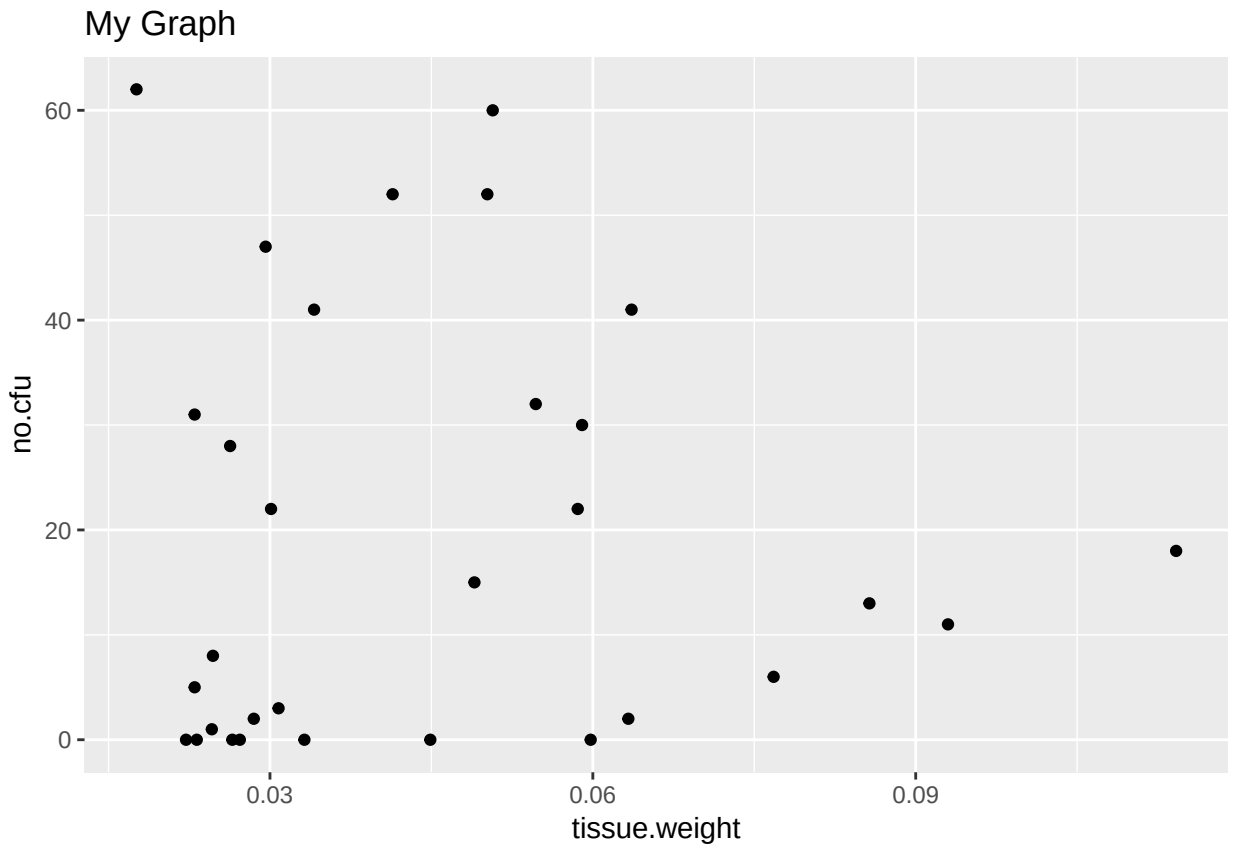
```
## Warning: Removed 1 rows containing missing values (geom_point).
```



Essentially, the ggplot2 package comes with a wide variety of 'support' functions which can be added together to tell a story about how you want your graph to look like.

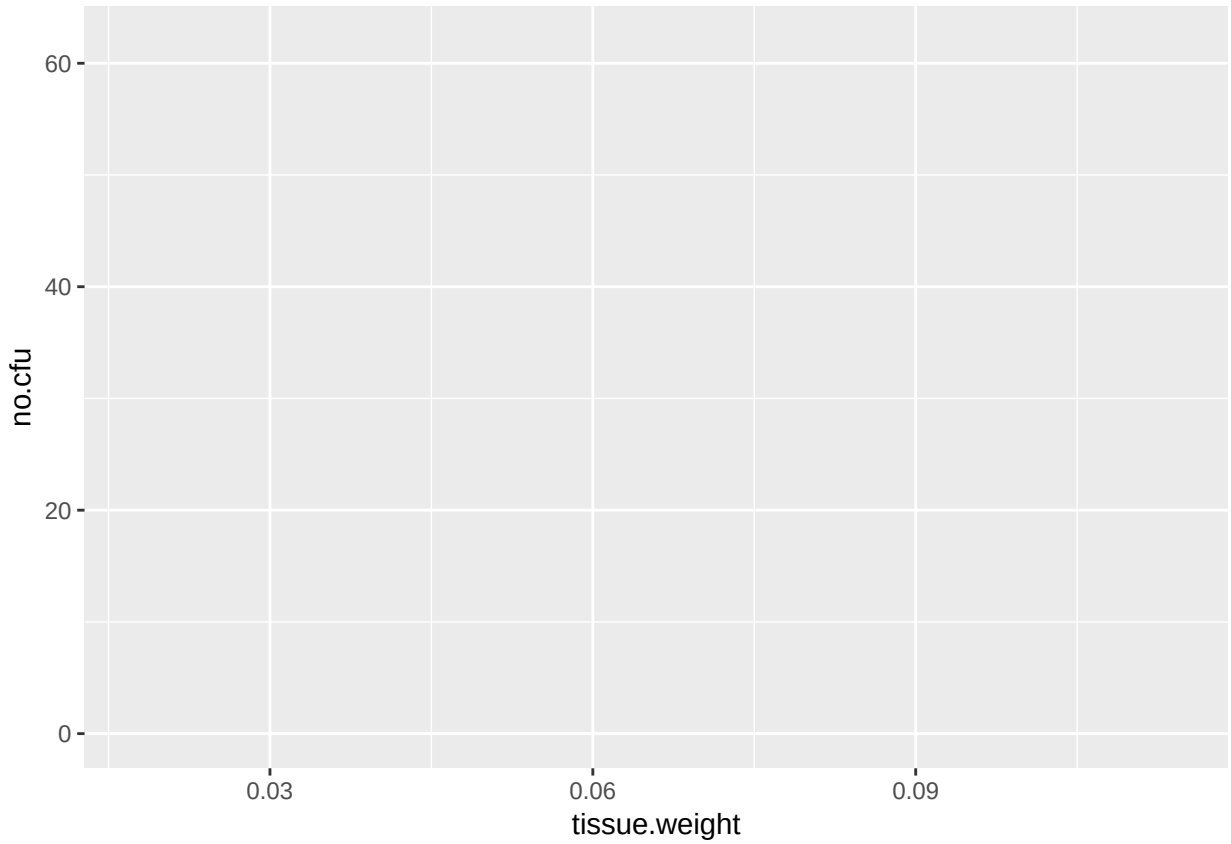
```
# Adding a title using a 'support' title function.
ggplot(data = lung, aes(x = tissue.weight, y = no.cfu)) + geom_point() + ggtitle('My Graph')
```

```
## Warning: Removed 1 rows containing missing values (geom_point).
```



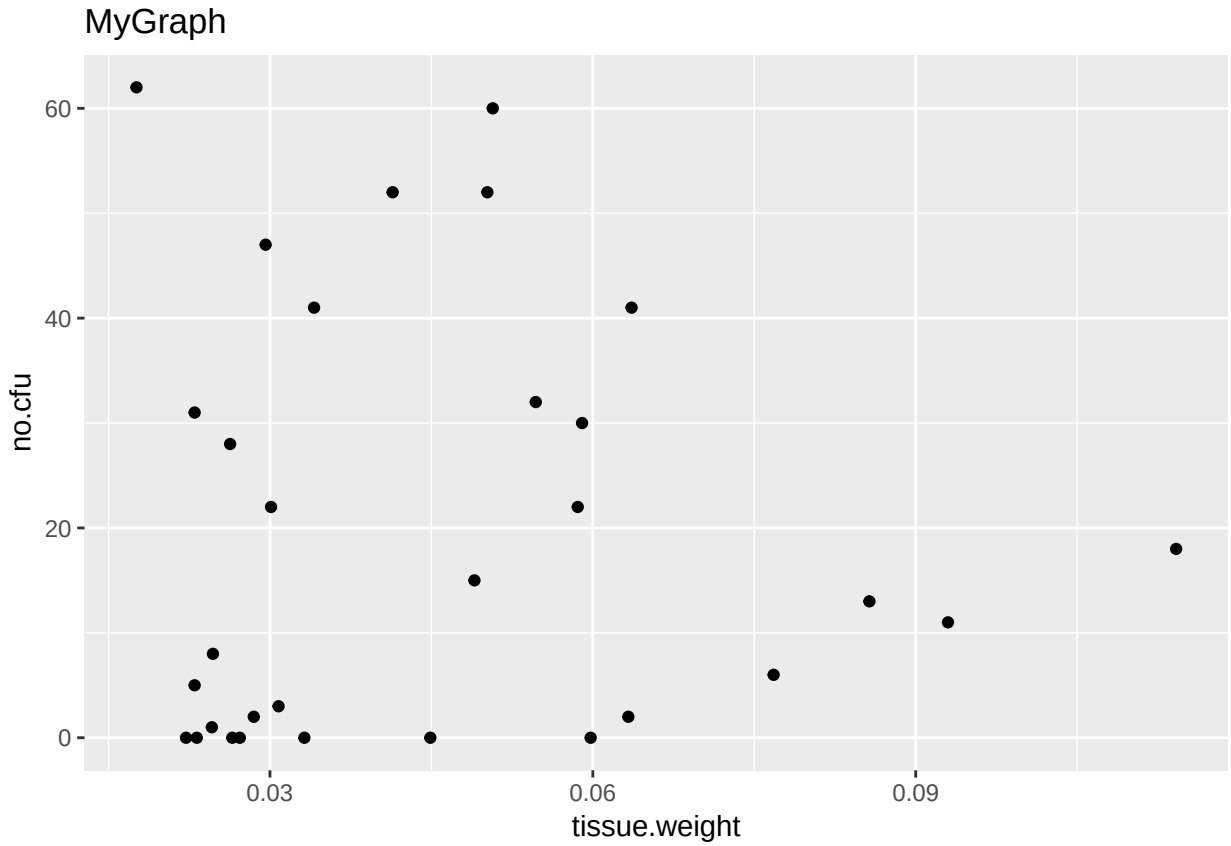
Generally, it can become redundant to re-write the whole thing each time. Instead, we can store our graph instructions inside an object/variable.

```
p <- ggplot(data = lung, aes(x = tissue.weight, y = no.cfu))
p
```



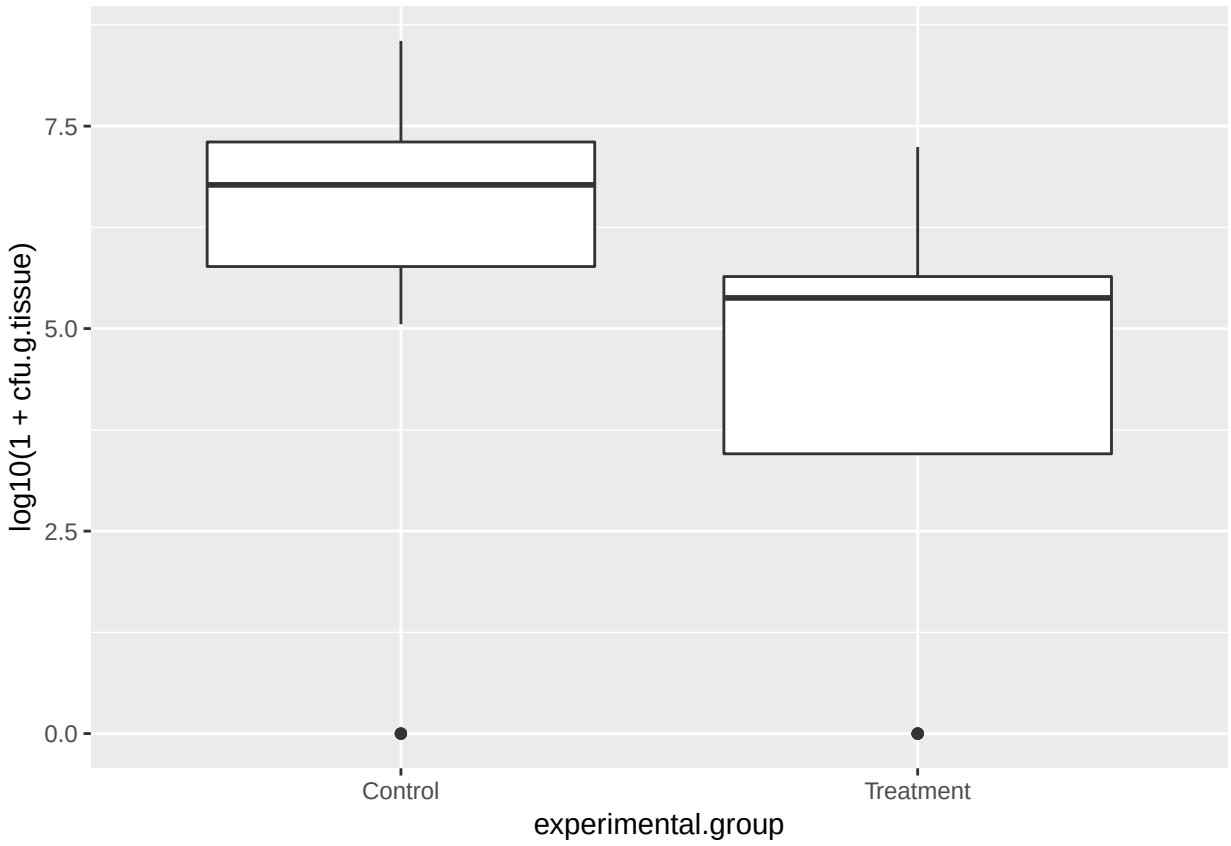

```
# I can now keep adding layers to it.
p <- p + geom_point()
p <- p + ggtitle('MyGraph')
p
```

```
## Warning: Removed 1 rows containing missing values (geom_point).
```

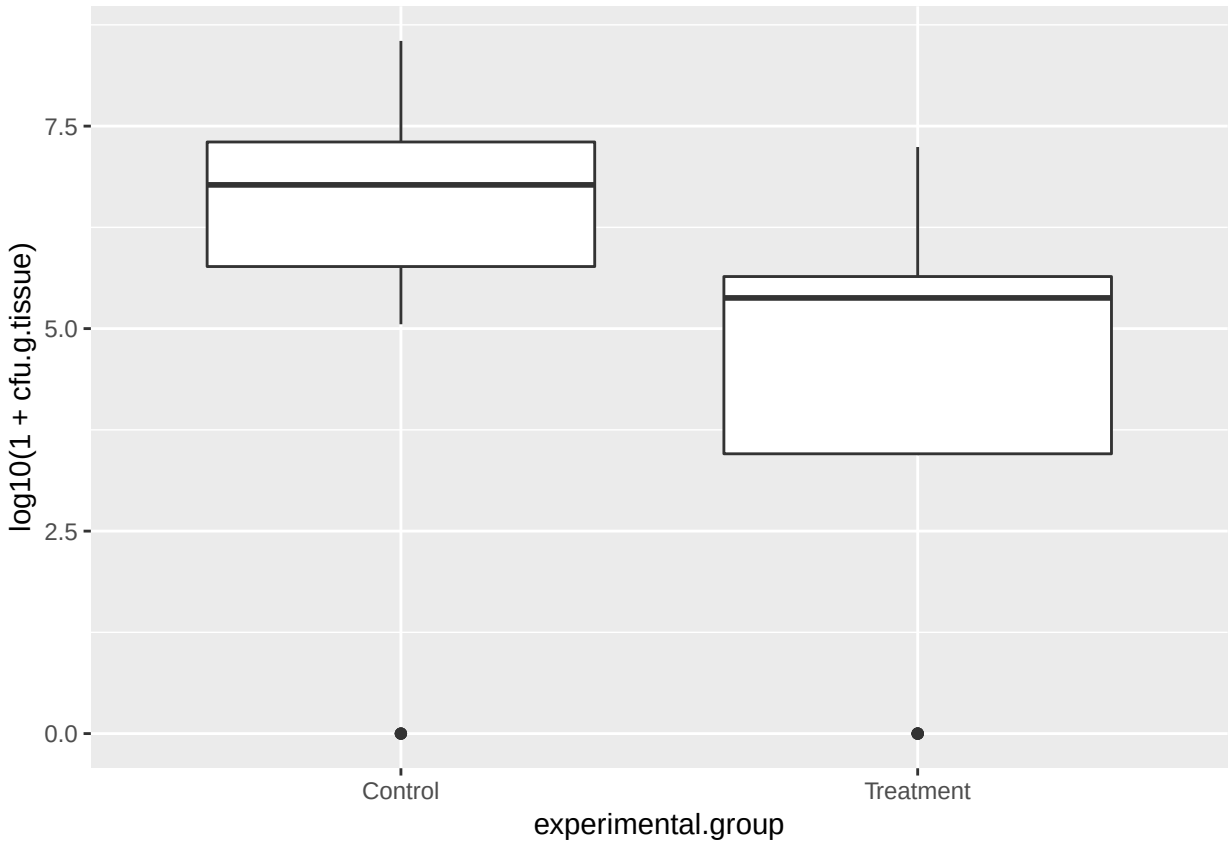


Notice that in this case below, qplot() and ggplot() give the same output.

```
qplot(experimental.group,
      log10(1 + cfu.g.tissue),
      data = liver,
      geom = 'boxplot')
```

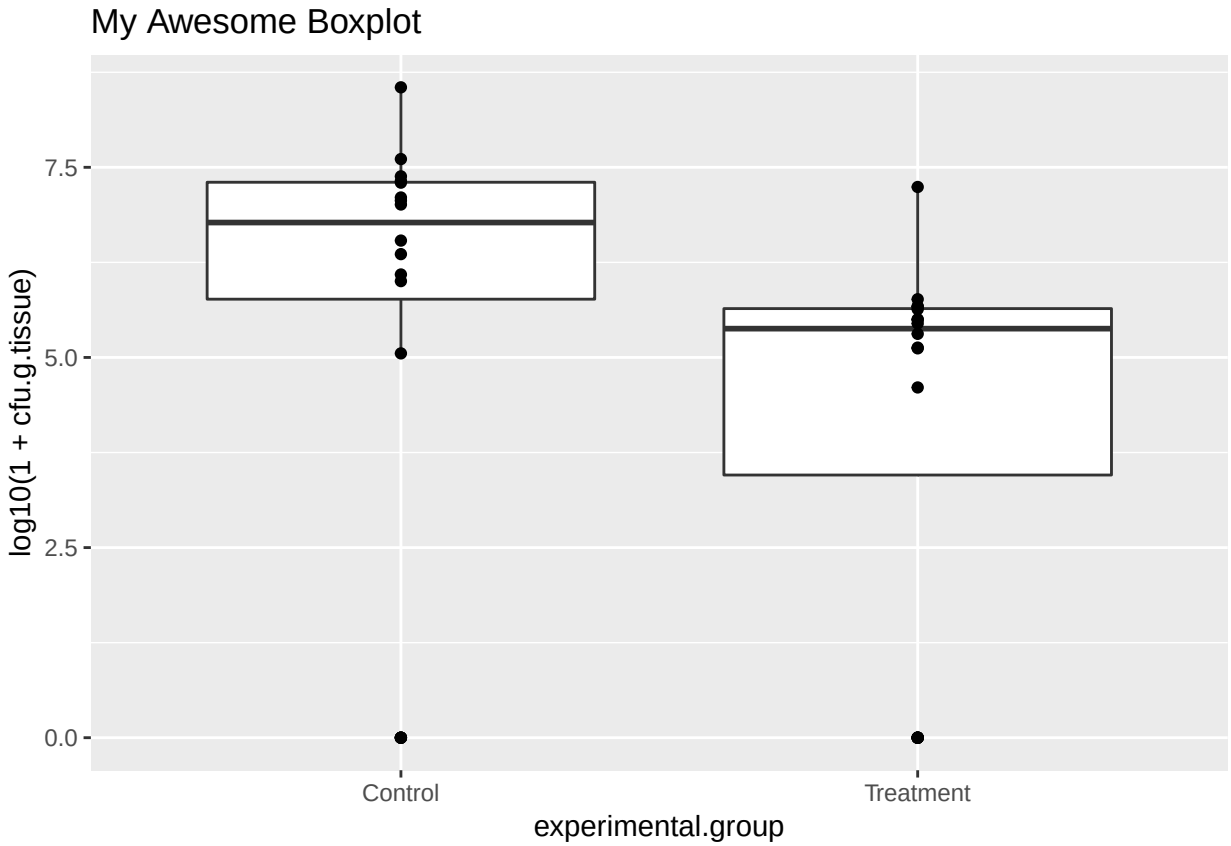



```
ggplot(liver, aes(experimental.group, log10(1 + cfu.g.tissue))) + geom_boxplot()
```



With ggplot() however, we can keep adding more layers which increases the overall complexity of our graph.

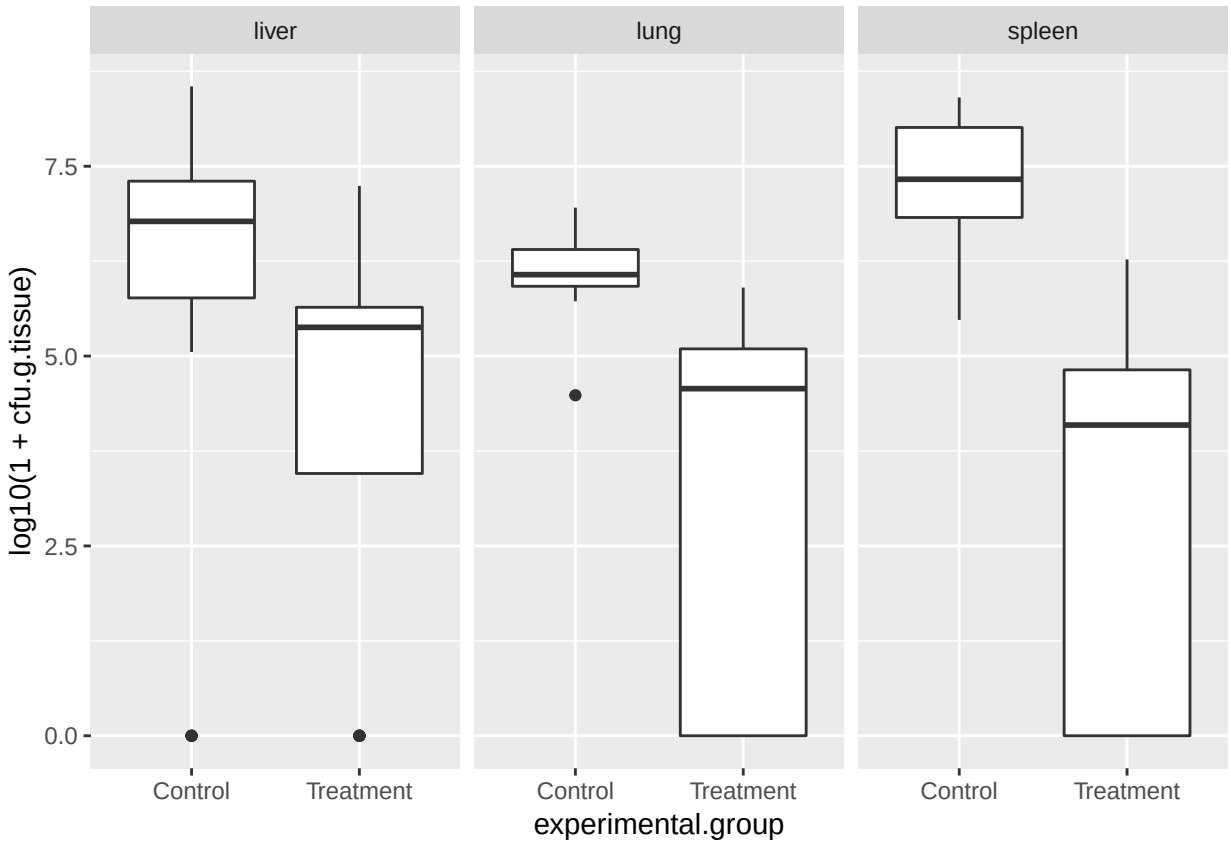
```
ggplot(liver, aes(experimental.group,log10(1+cfu.g.tissue))) +
  geom_boxplot() +
  geom_point() +
  ggtitle('My Awesome Boxplot')
```



ggplot() allows for an immense variety of graphs to be made and customised. For example, here we are using our original dataset without having to subset it. We can plot everything side by side with facet_wrap().

```
ggplot(dat, aes(x = experimental.group, y = log10(1+cfu.g.tissue))) +
  geom_boxplot() +
  facet_wrap(~tissue)
```

```
## Warning: Removed 1 rows containing non-finite values (stat_boxplot).
```



Hopefully you now understand the logic behind some of the plotting functions of R. It's totally fine if you cannot use ggplot() on your own or know all the functions by heart. The goal here is to give you enough understanding of basics plotting so that you can now practice creating your first plots and visualising your data.

In Conclusion

You know have a basic arsenal at your fingertips to perform basic data analysis. You can: 1. Perform some basic vector math 2. Subset or filter your long data 3. Graph your data using either base R, qplot, or ggplot2 Next week, we will build on this skill set to learn more ways to manipulate our data. For example, you can run the statistics for all your groups in one go and save the p-values to a convenient spread sheet, and print your p-values directly on your plot. There are other basic data wranging skills that will help you handle heterogeneous data sets. These include: 1. Merging two data sets 2. Reshaping your data to go from long to wide and back again 3. Summarize your data in all kinds of ways (e.g. mean, median, sample size, and standard deviation across experimental groups, to name a few) All of this will be covered in Week 4!

```
##
##  -----
## Go forth and code!
##  ------
##     \
##      \
##           _
##         _/ }
##       `>' \
##       `|   \
##        |   /'-.     .-.
```

```
##          \'     ';`--' .'
##           \'.    `'-./
##             '.`-..-;`
##                `;-..'
##                _| _|
##                /` /` [nosig]
##
```