Central Limit Theorem

Nick Sumpter

2022-08-21

Today's Lab

In this lab we will finally start using R for its true purpose: statistical analysis. Our goal for today is to learn how to extract some statistics from our data and to use R to understand the concepts of the Central Limit Theorem.

Loading Packages and Data

In this lab, as in all labs, we will be using the `tidyverse` package. New to this lab, however, is the `sciplot` package, which contains the `se` function for easily calculating standard errors. You will need to install the `sciplot` package as you have not used it before. Remember, this only has to be done once per package:

```
install.packages("sciplot")
```

Now that both packages are installed, let's load them with the `library` function.

```
library(tidyverse)
library(sciplot)
```

For this lab, we will be working with data from the American Gut Project. This data has been cleaned up and is stored in the `microbiome.RData` file on Canvas. Download this file and save it in the directory you wish to work in. Go ahead and set your working directory to this directory as we did last lab. This file is in a format that enables it to easily be read into R. Once you have this file in your directory, we will use the `load` function to pull it into R. You can manually type the code below, or you can click on the file in your Files panel to load it in, then copy the corresponding code to your script.

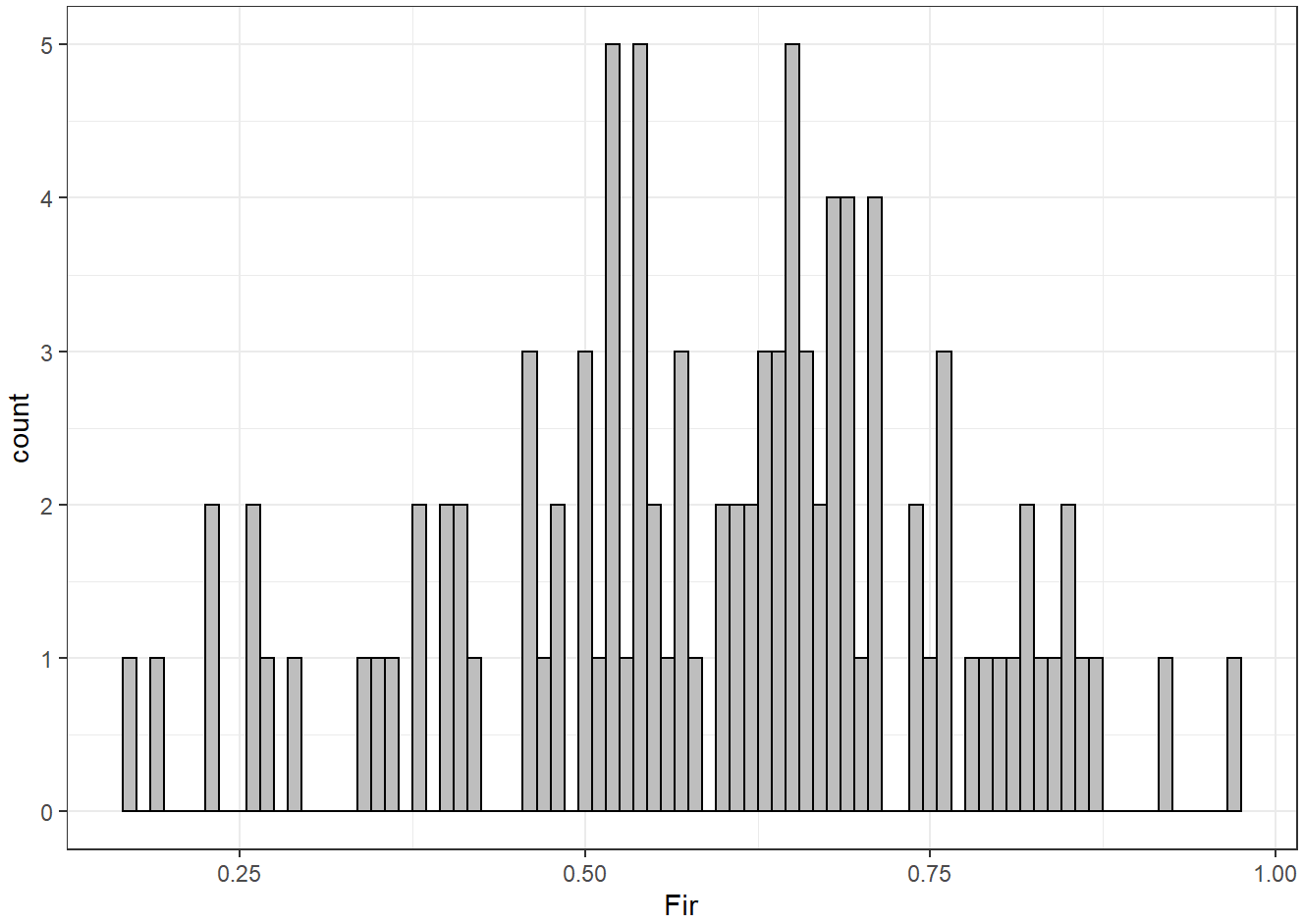
```
setwd("~/PhD/Teaching/GRD770/R Labs 2021/Lab 5 - Central Limit Theorem")

load("microbiome.RData")
```

Getting to Know Your Data

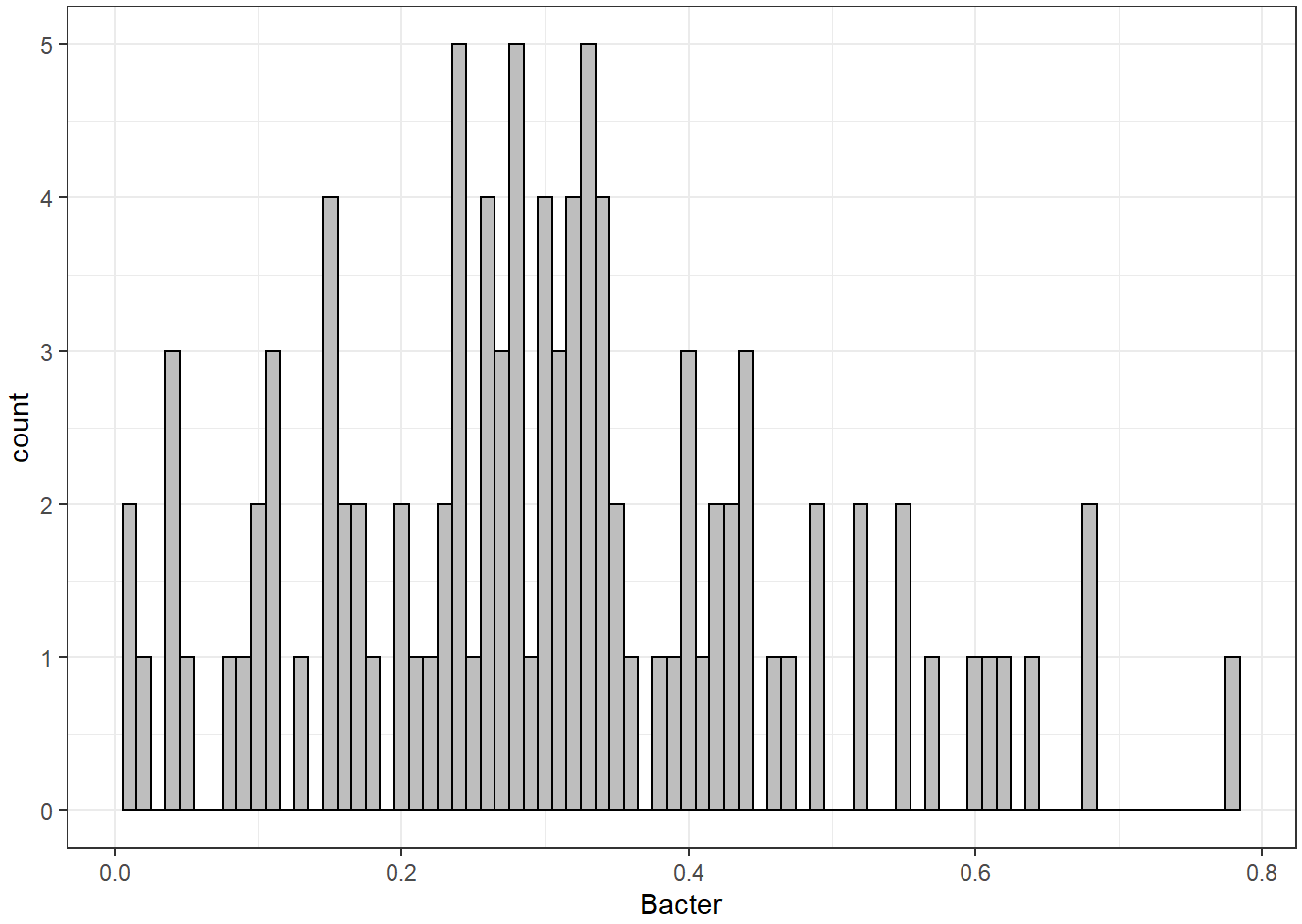
The first step in any statistical analysis is to get to know your data. As you can see, it is stored in an object called `microbiome` in our Environment panel. By clicking on the little blue arrow, we can see that it consists of 5 columns. This includes an ID column along with four other columns (Fir, Bacter, Actin, and Verru). These four columns indicate, for each individual, what percentage of their gut microbiome consisted of that respective phylum of bacteria.

When visualizing continuous data, a great place to start is a histogram. This is included with `ggplot` as the `geom_histogram` function. Let's go ahead and plot the distribution of each phylum. We will make the plots look a little nicer by setting the width of each bin to 0.01 using the `binwidth` argument. We will also set the fill of the bars to gray and the outlines to black using the `fill` and `color` arguments respectively. Let's make sure to first set the theme as mentioned at the end of last lab using the `theme_set` function.

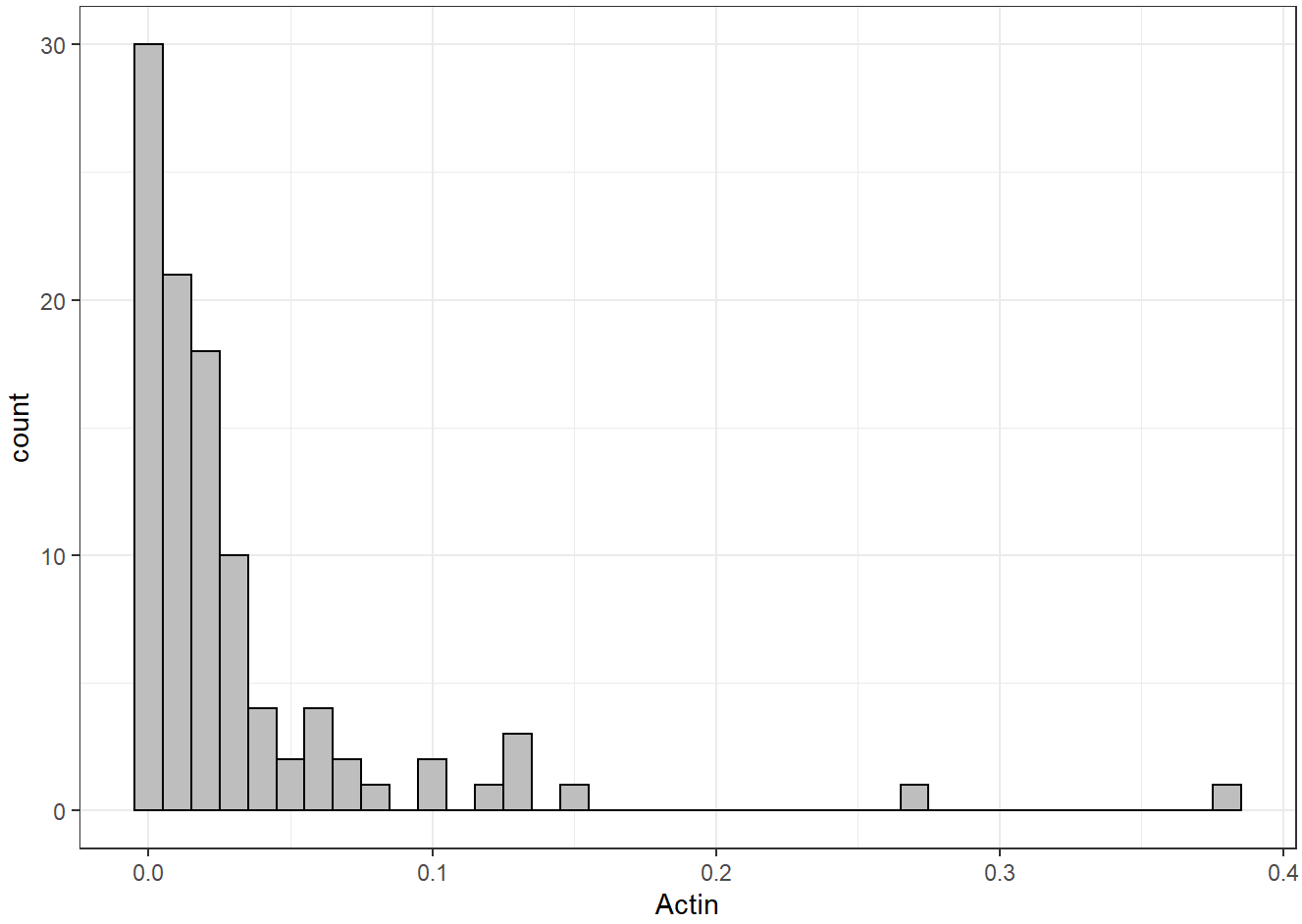
```
theme_set(theme_bw())

ggplot(data = microbiome, mapping = aes(x = Fir)) + geom_histogram(binwidth = 0.01, fill
= "gray", color = "black")
```



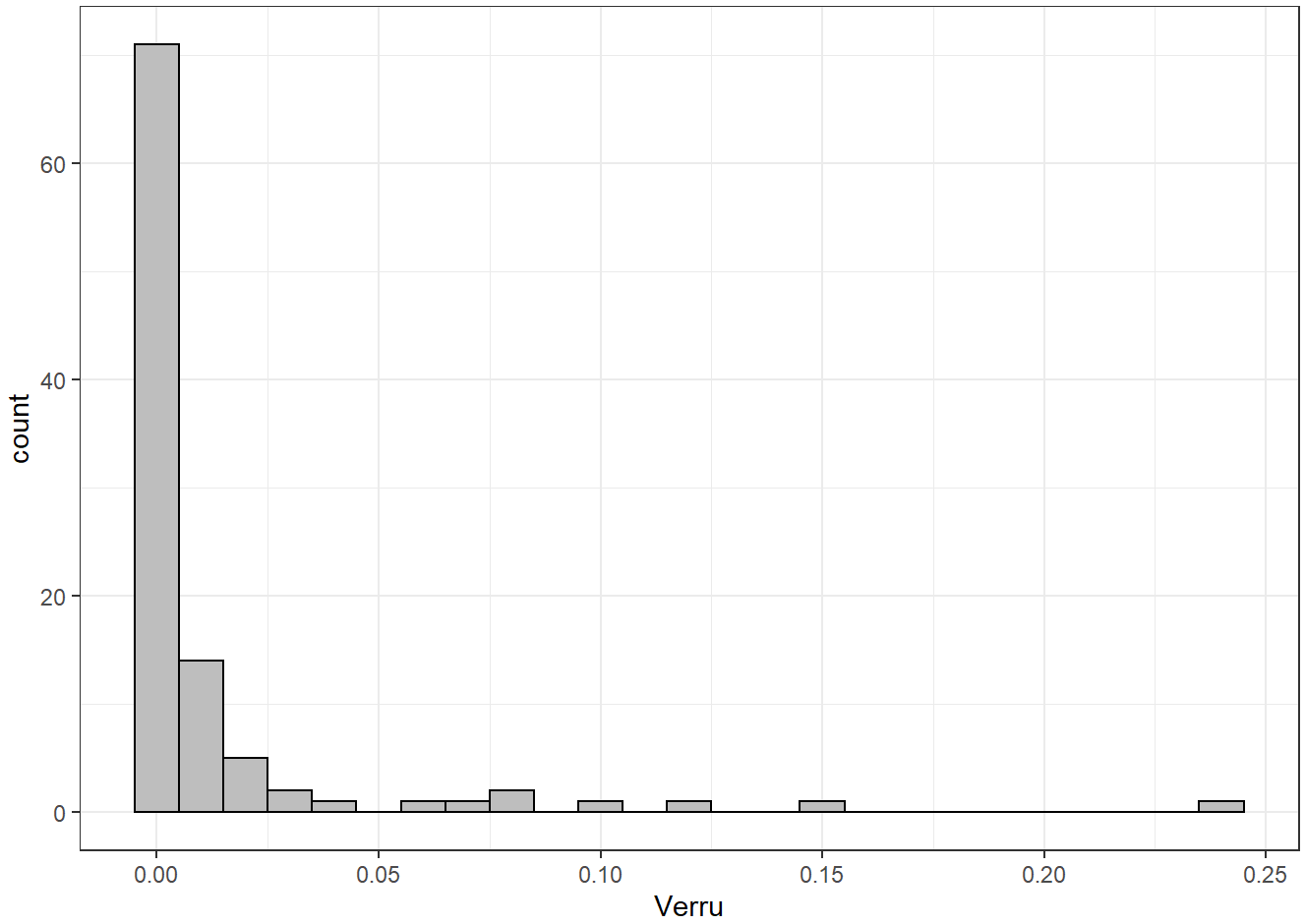

```
ggplot(data = microbiome, mapping = aes(x = Bacter)) + geom_histogram(binwidth = 0.01, f
ill = "gray", color = "black")
```




```
ggplot(data = microbiome, mapping = aes(x = Actin)) + geom_histogram(binwidth = 0.01, fi
ll = "gray", color = "black")
```




```
ggplot(data = microbiome, mapping = aes(x = Verru)) + geom_histogram(binwidth = 0.01, fi
ll = "gray", color = "black")
```



Great, so we can see the distribution of each of the four variables. At this stage, we want to work out some basic statistics for each phylum. Let's start with the mean, standard deviation, and standard error. For this, we will use the `mean`, `sd`, and `se` functions respectively. For example, lets just use the `Fir` phylum.

```
mean(microbiome$Fir)
```

```
## [1] 0.5916822
```

```
sd(microbiome$Fir)
```

```
## [1] 0.1698658
```

```
se(microbiome$Fir)
```

```
## [1] 0.01690228
```

These functions are useful, however they only provide information on one variable at a time. It quickly becomes laborious trying to repeat these functions over and over for each phylum of bacteria.

Wide vs Long Data

Briefly, I want to introduce you to the concept of wide vs long format datasets. Long format = more rows, wide format = more columns. For reference, our microbiome dataset is currently in a wide format. This essentially means that there is information stored in multiple columns that could be condensed into fewer columns. We know that the Fir, Bacter, Actin, and Verru columns contain the same sort of information: proportion of the microbiome belonging to X phylum. This could in theory all be stored in two columns, one for the name of the phylum, and one for the proportion. The `pivot` functions are very useful for quickly converting a dataset from a wide to long format (or vice versa). First, let's start by using `pivot_longer` on the microbiome dataset to elongate it. There are a few ways to use this function, but I will show you what I believe to be the easiest:

```
microbiome_long <- pivot_longer(data = microbiome,
                                cols = Fir:Verru,
                                names_to = "Phylum",
                                values_to = "Proportion")
```

Have a look at the output dataset. You will notice first that it now has 404 rows with only 3 columns, versus 101 rows and 5 columns in the original dataset. We have elongated the dataset by moving the variable name itself (i.e. Fir) into a column called 'Phylum', and its corresponding value into the 'Proportion' column. We can break this down by looking at each argument above:

1. data = the input dataset
2. cols = the columns you wish to merge into fewer columns
    - Note that we have used a shortcut to define these, by using the `:` function which essentially says "all columns from Fir to Verru inclusive" - this only works in `tidyverse` functions
3. names_to = the name of the new column that contains the names of the columns that are being merged
4. values_to = the name of the new column that contains the values from the columns that are being merged

For reference, if we have a long format dataset that we want to convert back into a wide format, we can use the opposite function: `pivot_wider`.

```
microbiome_wide <- pivot_wider(data = microbiome_long,
                               names_from = "Phylum",
                               values_from = "Proportion")
```

Compare this to the `pivot_longer` function above and you will see the symmetry, and as you can see in your Environment, `microbiome_wide` is identical to `microbiome`.

Grouped Summaries

With our newly elongated dataset, we can now perform a couple of functions (`summarize` and `group_by`) that allow us to summarize subsets of a dataset. In this case, these functions will allow us to tell R to calculate the mean, standard deviation (sd), and standard error (se) for each phylum. The `group_by` function groups the `microbiome_long` dataset based on values of a specified column (in this case, 'Phylum'). The `summarize` function takes this grouped tibble and makes a summary tibble with each row representing a phylum and each column specified by the user. Feel free to look these up in the Help panel for more information.

```
microbiome_grouped <- group_by(microbiome_long, Phylum)

microbiome_summary <- summarize(microbiome_grouped,
                                mean = mean(Proportion),
                                sd = sd(Proportion),
                                se = se(Proportion))

microbiome_summary
```

```
## # A tibble: 4 × 4
##   Phylum   mean     sd      se
##   <chr>   <dbl>  <dbl>   <dbl>
## 1 Actin  0.0314 0.0537 0.00534
## 2 Bacter 0.306  0.162  0.0161
## 3 Fir    0.592  0.170  0.0169
## 4 Verru  0.0127 0.0343 0.00342
```

The Pipe Function: %>%

In the interest of making the remainder of the labs easier to code, I will show you one of the most powerful functions for making your code both easier to read and generally easier to write. This function is the pipe function `%>%` from the `tidyverse`. Essentially, what this does is say: **"take the output of everything on the left of the pipe and input it into the function on the right"**. This will make your code much easier to follow, as it shows the order that functions apply and is much easier to code (especially as you add more functions). RStudio provides a nice keyboard shortcut for the pipe (`Ctrl + Shift + M` for Windows, `Cmd + Shift + M` for Mac). I cannot stress how much easier you will find R if you learn to use this function. Let's use the pipe function to make all of the above code easier to read and write:

```
microbiome_summary <- microbiome %>%
  pivot_longer(cols = Fir:Verru,
               names_to = "Phylum",
               values_to = "Proportion") %>%
  group_by(Phylum) %>%
  summarize(mean = mean(Proportion),
            sd = sd(Proportion),
            se = se(Proportion))

microbiome_summary
```

```
## # A tibble: 4 × 4
##   Phylum   mean     sd      se
##   <chr>   <dbl>  <dbl>   <dbl>
## 1 Actin  0.0314 0.0537 0.00534
## 2 Bacter 0.306  0.162  0.0161
## 3 Fir    0.592  0.170  0.0169
## 4 Verru  0.0127 0.0343 0.00342
```

Breaking down what you see above:

1. Start with what you want to name the output of your entire coding operation (in this case `microbiome_summary`)
2. Next, use the assign function `<-` with your starting dataset on the right
3. Then put the pipe function at the end of this line of code and press `Enter` / `Return` to start a new line (starting a new line is is not necessary but it makes it easier to read/write)
4. Input your first function, removing the `data` argument from it as the pipe will automatically insert whatever is output from the left hand side of the pipe into the function
5. After each function, end the line of code with a pipe and string the next function into the next line of code, again making sure not to manually tell it the input data

I highly recommend spending a little bit of time trying to learn this function, mainly because it stops you from having to make a bunch of different objects in your Environment that you have to keep track of, and it also condenses your code a lot.

Note: This function works best when used alongside `tidyverse` functions because it requires the first argument of the following function to be the input data. If you wish to specify whatever is on the left side of the pipe in the following function, you can reference it using the `.` symbol like so:

```
microbiome_summary <- microbiome %>%
  pivot_longer(data = .,
               cols = Fir:Verru,
               names_to = "Phylum",
               values_to = "Proportion") %>%
  group_by(.data = ., Phylum) %>%
  summarize(.data = .,
            mean = mean(Proportion),
            sd = sd(Proportion),
            se = se(Proportion))

microbiome_summary
```

```
## # A tibble: 4 × 4
##   Phylum   mean     sd      se
##   <chr>   <dbl>  <dbl>   <dbl>
## 1 Actin  0.0314 0.0537 0.00534
## 2 Bacter 0.306  0.162  0.0161
## 3 Fir    0.592  0.170  0.0169
## 4 Verru  0.0127 0.0343 0.00342
```

Central Limit Theorem

Next, we are going to use the `microbiome` dataset to help us understand the Central Limit Theorem. This theorem states: regardless of the distribution of a variable in a population, if you repeatedly sampled enough individuals from this population and calculated the sample means, the distribution of the sample means would be normally distributed.

To test this theorem, we will need to do the following:

1. Randomly sample from our dataset
2. Calculate the mean of that sample
3. Repeat 1000 times (using R)
4. Plot a histogram of our total population and the sample means

Calculating Random Sample Means

To randomly sample the data, we will use the `sample` function. Let's look it up in the Help panel. It will require three arguments:

1. `x` = data to sample from
2. `size` = sample size (must be less than the size of x)
3. `replace` = TRUE/FALSE indicating whether to sample with or without replacement

First, let's get a random sample of `Fir` of size 30 without replacement

```
sample(x = microbiome$Fir, size = 30, replace = FALSE)
```

```
##  [1] 0.1693 0.6617 0.6930 0.4608 0.3647 0.6030 0.6498 0.8600 0.7061 0.5834
## [11] 0.6054 0.5448 0.7564 0.6283 0.2601 0.6074 0.4598 0.3958 0.7564 0.2265
## [21] 0.6015 0.7964 0.7536 0.5413 0.5724 0.4003 0.3540 0.7423 0.6353 0.7064
```

With Pipes: We can use the pipe function here by combining what we know with the `pull` function, which is just a way to pull out a column from the dataset (exactly like `$` but it works better with pipes):

```
microbiome %>%
  pull(Fir) %>%
  sample(size = 30, replace = FALSE)
```

```
##  [1] 0.6602 0.8229 0.6054 0.4598 0.2911 0.7564 0.6074 0.7061 0.9736 0.4128
## [11] 0.6483 0.2640 0.2715 0.5246 0.7551 0.5240 0.6168 0.3773 0.3647 0.6316
## [21] 0.5709 0.5448 0.7064 0.6283 0.3401 0.5724 0.6030 0.8096 0.4795 0.7096
```

Now, the power of the pipe will make it super easy to calculate the mean of this sample. We could just put a `mean` function around the whole above function and get an average of a random sample, or even better, we can use the pipe function. Note that these will come out as slightly different numbers because this code is randomly sampling your input data each time you run it.

```
# Without pipes
mean(sample(x = microbiome$Fir, size = 30, replace = FALSE))
```

```
## [1] 0.6083767
```

```
# With pipes
microbiome %>%
  pull(Fir) %>%
  sample(size = 30, replace = FALSE) %>%
  mean()
```

```
## [1] 0.5781667
```

Now that we can calculate the mean of a random sample, we need to be able to do it over and over again to create a distribution of sample means. We can do that using the `replicate` function. If we search this up in the Help panel, we will see that it is part of a family of functions that apply another function over a list or vector (a vector is just a variable in R). In our case, we wish to apply the `mean` function over 1000 random samples of `Fir`. The `replicate` function requires two inputs:

1. `n` = the number of times to repeat the expression
2. `expr` = the expression you are going to repeat

Let's repeat the mean sample from before 1000 times and store it in variable `Fir_means`.

```
# Without pipes
Fir_means <- replicate(n = 1000,
                       expr = mean(sample(x = microbiome$Fir, size = 30, replace = FALS
E)))

# With pipes
Fir_means <- replicate(n = 1000,
                       expr = microbiome %>% pull(Fir) %>% sample(size = 30, replace = F
ALSE) %>% mean())
```

Let's do this for `Bacter`, `Actin`, and `Verru` as well.

```
# Without pipes
Bacter_means <- replicate(n = 1000,
                          expr = mean(sample(x = microbiome$Bacter, size = 30, replace =
FALSE)))
Actin_means <- replicate(n = 1000,
                         expr = mean(sample(x = microbiome$Actin, size = 30, replace = F
ALSE)))
Verru_means <- replicate(n = 1000,
                         expr = mean(sample(x = microbiome$Verru, size = 30, replace = F
ALSE)))

# With pipes
Bacter_means <- replicate(n = 1000,
                        expr = microbiome %>% pull(Bacter) %>% sample(size = 30, replace
= FALSE) %>% mean())
Actin_means <- replicate(n = 1000,
                         expr = microbiome %>% pull(Actin) %>% sample(size = 30, replace =
FALSE) %>% mean())
Verru_means <- replicate(n = 1000,
                         expr = microbiome %>% pull(Verru) %>% sample(size = 30, replace =
FALSE) %>% mean())
```

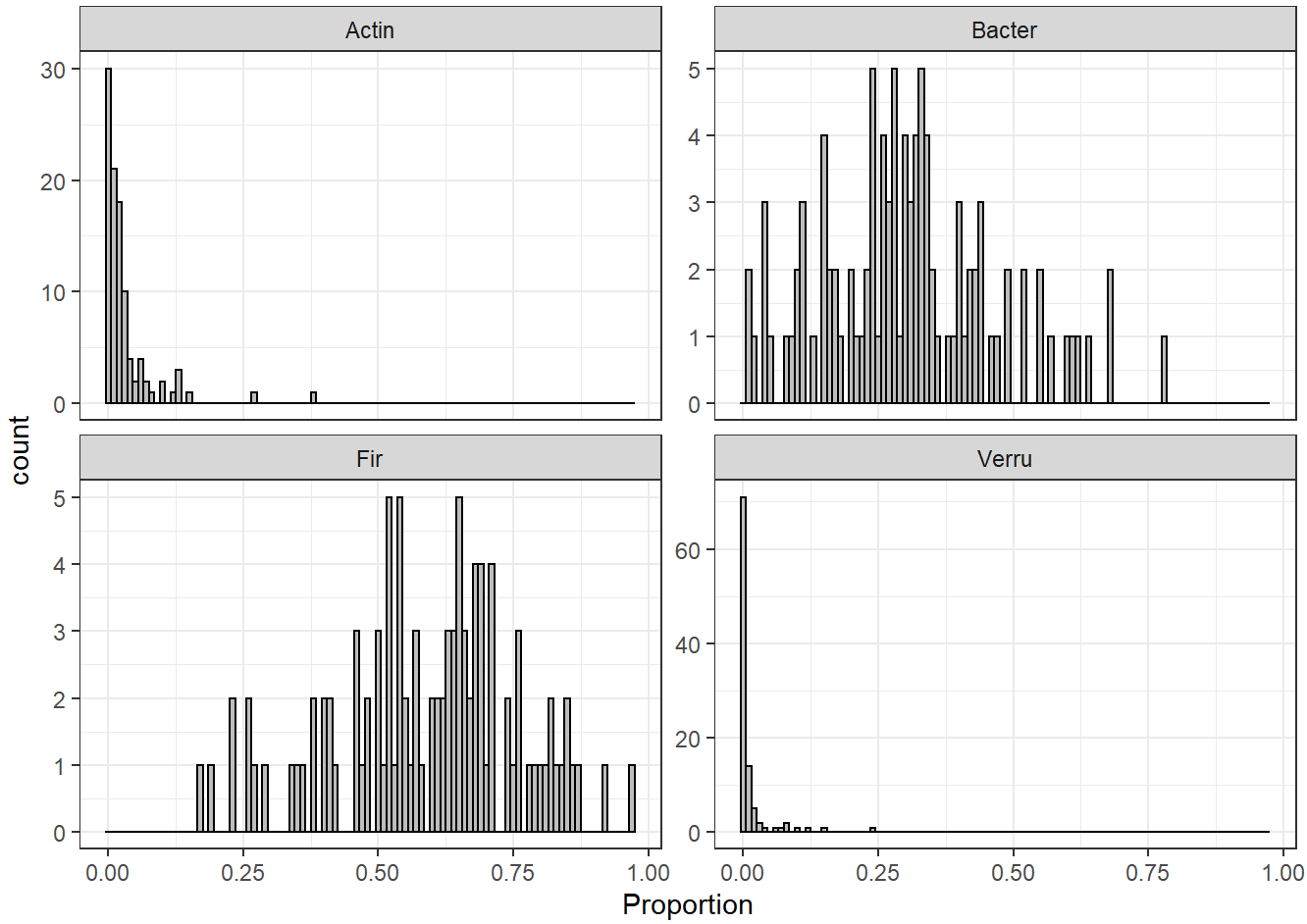
To plot the sample means more easily, let's store all of these sample distributions in a tibble (using the `tibble` function). This function just requires you to specify the column name and its corresponding values, for each column. It requires the data to be the same length (i.e. each column should be the same length).

```
microbiome_means <- tibble(Fir = Fir_means,
                           Bacter = Bacter_means,
                           Actin = Actin_means,
                           Verru = Verru_means)
```

Plotting Sample Means

Earlier, we created histograms to show the distribution of each variable in the `microbiome` tibble. Using the long version of the tibble, we can use the `facet_wrap` function to make all four histograms more easily. Additionally, we need to make it so the y-axis scale of each faceted plot is free to change between plots. We do this by setting the `scales` argument to `"free_y"` .

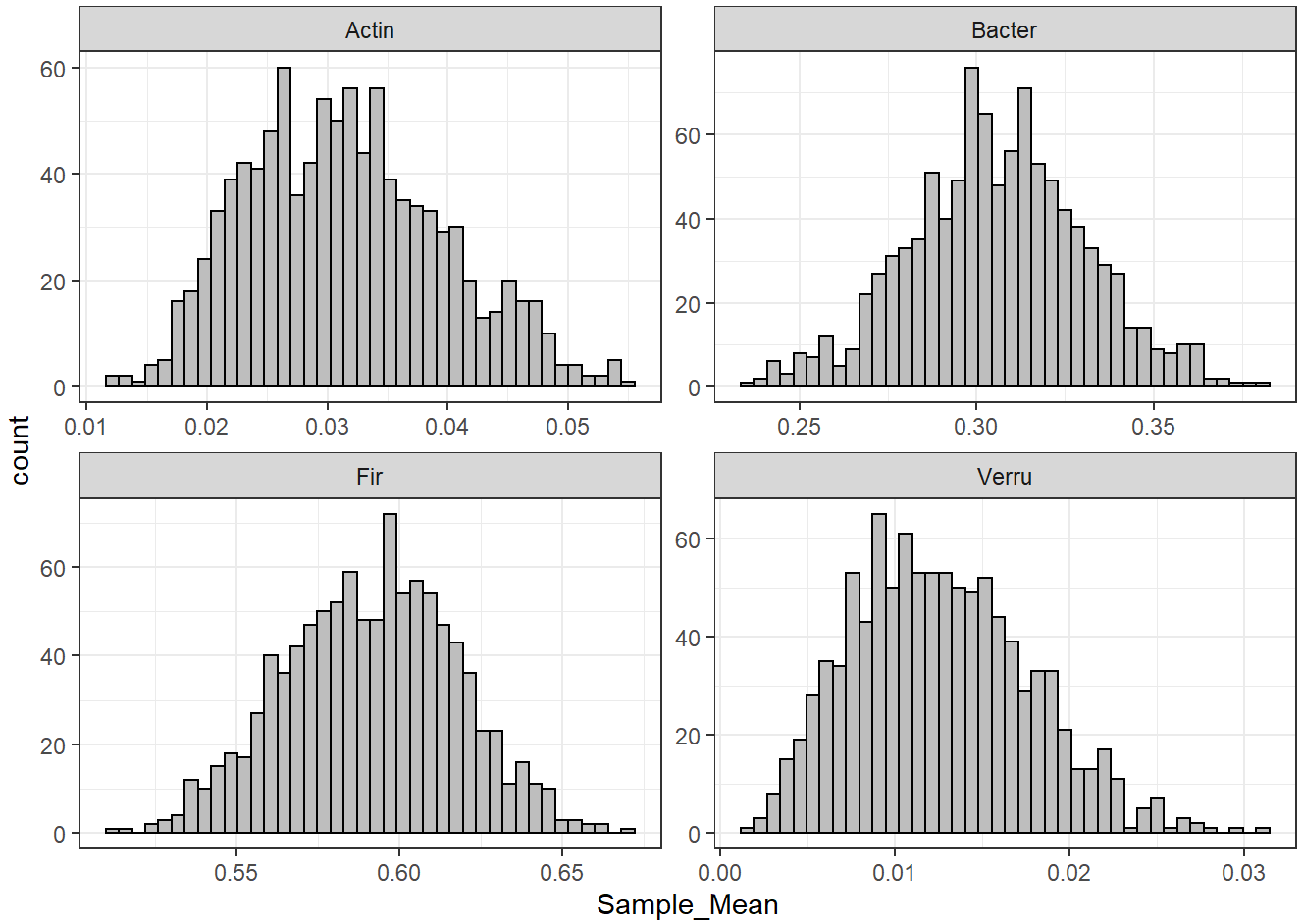
```
ggplot(data = microbiome_long, mapping = aes(x = Proportion)) +
  geom_histogram(binwidth = 0.01, fill = "gray", color = "black") +
  facet_wrap(~ Phylum, scales = "free_y")
```



Now let's plot our sample means for each variable and attempt to empirically verify the Central Limit Theorem. We will do this using the long version of the `microbiome_means` tibble, and facet based on the `Phylum` column. Instead of setting the `bin_width` to 0.01, we will set the total number of bins for each plot to 40. Additionally, we need to make it so the scale of both axes on each faceted plot are free to change between plots. We do this by setting the `scales` argument to `"free"`.

Let's lengthen the data using `pivot_longer` to make it easier to plot, as before.

```
microbiome_means_long <- pivot_longer(microbiome_means,
                                      cols = Fir:Verru,
                                      names_to = "Phylum",
                                      values_to = "Sample_Mean")

ggplot(data = microbiome_means_long, mapping = aes(x = Sample_Mean)) +
  geom_histogram(bins = 40, fill = "gray", color = "black") +
  facet_wrap(~ Phylum, scales = "free")
```



You can see that despite a couple of our original distributions being extremely skewed, the distributions of sample means came out roughly normally distributed. Therefore we have empirically verified the Central Limit Theorem. Bear in mind that this only applies to relatively large datasets (N > 30 per group)

Independent Practice

1. Using the iris dataset, attempt to verify the Central Limit Theorem.
2. Try using the pipe function to plot a ggplot function.