Loopy for Loops and Silly for Samples - Hands-on!

YOUR NAME

2023-10-28

Part 1: Loops

`for` loops

First, we are going to be focusing on "for loops". A for loop applies a command to each value provided then stops. In other words, the `for` loop runs for a preset number of times.

Basic example:

```
for (i in 1:5) {
  print(i)
}
```

```
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
```

- You don't actually have to use i for the value there but most people do. `i` is presumably short for "index", which makes sense. But you could just as well call it `banana`.

Ex.1) run the same command above replacing `i` with `banana`.

```
for (banana in 1:5) {
  print(banana)
}
```

```
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
```

So this loop printed each value for `i` (or `banana`). How would you write code that printed the values 6-10?

```
for (i in 6:10) {
  print(i)
}
```

```
## [1] 6
## [1] 7
## [1] 8
## [1] 9
## [1] 10
```

It's often useful to define a placeholder variable before running your loop. For example, this loop calculates the mean of a given data set.

```
numbers <- c(4, 22, 6, 13, 19, 2, 11)
sum <- 0
for(num in numbers) {
  sum <- sum + num
}
mean <- sum / length(numbers)
print(mean)
```

```
## [1] 11
```

You can also write loops with character vectors. Here's a character vector: `days<-c("Mon", "Tues", "Wednes", "Thurs", "Fri", "Satur", "Sun")`

Ex.2) Write a loop that adds day to each day of the week so that the output of `days` gives you "Monday", "Tuesday", etc, one in each line.

Hint: if you add the argument sep = " " to paste you don't get a space in between)

```
#First create the vector days as show above

#now write a for loop that adds "day" to each of the elements and print each one as you
↪  go

days<-c("Mon", "Tues", "Wednes", "Thurs", "Fri", "Satur", "Sun")

for (day in days){
 print(paste(day, "day", sep = ""))

}
```

```
## [1] "Monday"
## [1] "Tuesday"
## [1] "Wednesday"
## [1] "Thursday"
## [1] "Friday"
## [1] "Saturday"
## [1] "Sunday"
```

`while` loops

Ex.3) A while loop repeats code until a condition is met. It's really important to include the condition because the loop might continue forever if you don't.

Here's an example of a loop that prints x then multiplies it by two, as long as x is under 16.

```r
x <- 2
while (x < 16) {
  print(x)
  x <- x * 2
}
```

```
## [1] 2
## [1] 4
## [1] 8
```

Now write a while loop that prints the value, then doubles it, starting at 1 and stopping at 100.

```r
x <- 1
while (x <= 100) {
  print(x)
  x <- x * 2
}
```

```
## [1] 1
## [1] 2
## [1] 4
## [1] 8
## [1] 16
## [1] 32
## [1] 64
```

Just for fun try running that loop without with while(x). This will run forever. Make sure you remember to add the condition!

Say you have a number and you want to find the smallest divisor that isn't 1. You could use a while loop to do this.

```r
#your code here
number <- 91
divisor <- 2
while (number %% divisor != 0) {
  divisor <- divisor + 1
}
print(divisor)
```

```
## [1] 7
```

You can do a lot more with loops once you learn about functions. But alas, that's for next lab. For now that's probably enough for loops.

Recommended: This guide someone wrote on the internet is really good if you want to also read it. https://intro2r.com/loops.html

---

Part 2: Sampling

There is a basic function in R for taking samples very appropriately named sample. You need three arguments for it to work (there is a 4th also, probability). The first is the data set, the second is the number of samples, and the third is with or without replacement.

Ex.4) Let's with some basic sampling. Imagine you have a coin, equal chances of heads and tails. You want to flip it 10 times and see how many heads and how many tails. First make a vector called coin, with heads and tails. then sample it 10 times, with replacement

```
coin_toss<-c("heads", "tails")

sample(coin_toss<-c("heads", "tails"), 10, replace = TRUE)
```

```
##  [1] "heads" "heads" "heads" "heads" "tails" "heads" "tails" "heads" "heads"
## [10] "heads"
```

Now lets try and do this same thing with a for loop. Make an empty vector called all_tosses and use it to generate 10 coin tosses. Hint:Because you are looping over the sample you are only sampling one each time

```
all_tosses <- c()

for (i in 1:10) {
  all_tosses[i] <- sample(c("Heads", "Tails"), 1)
}
```

Now I'm being sneaky. I challenge you to flip a coin but its weighted. It lands on heads 60% of the time. Use the same loop and add the probabilities to the sample function. Hint: make them decimals in a vector.

```
all_tosses <- c()

for (i in 1:10) {
  all_tosses[i] <- sample(c("Heads", "Tails"), 1, prob = c(0.6, 0.4))
}
```

Now let's see what happens if we slip both coins even more times. Use your original loop with the fair coin and toss is 100 times. Then take the weighted coin, change the variable names to make them different (add the word weighted or rigged) and toss that 100 times. Then count how many heads and how many tails came from each coin. Make sure to make an empty vector for each loop. hint: you can use the table function to get a count of heads and tails

```
all_tosses <- c()

for (i in 1:100) {
  all_tosses[i] <- sample(c("Heads", "Tails"), 1)
}



all_tosses_weighted <- c()

for (i in 1:100) {
```

```
  all_tosses_weighted[i] <- sample(c("Heads", "Tails"), 1,  prob = c(0.6, 0.4))
}




toss_counts <- table(all_tosses)
print(toss_counts)
```

```
## all_tosses
## Heads Tails
##    56    44
```

```
weighted_counts<-table(all_tosses_weighted)
print(weighted_counts)
```

```
## all_tosses_weighted
## Heads Tails
##    71    29
```

This is why you shouldn't play games with other peoples strange coins

Ex.5) Read in the `human_genes.csv` file and name it `human_genes`. Play around with it using your strong exploratory data analysis skills. Get a good feel for the data set.

Overall goal of this exercise is to compare sample means at different n to the population mean.

Its big right? So let's take some samples.

NOTE: Your output will look different to the pdf because your samples will be different values.

5.1. First filter it to only keep the size and name column.

```
library(readr)
library(dplyr)
human_genes <- read_csv("human_genes.csv")
```

```
## Rows: 22385 Columns: 4
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (3): gene, name, description
## dbl (1): size
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
human_genes<-human_genes %>% select(name, size)
```

5.2. create a second data frame with summary statistics mean, median and SD of length. (what is interesting about this data??)

```
human_genes_summ <- human_genes %>%
  summarise(
    MeanLength = mean(size),
    MedianLength = median(size),
    SDLength = sd(size)
)
```

5.3. Now let's take some samples. Take 3 samples of gene size with replacement on. Use n = 10, 100, 1000. Calculate the mean of these samples, and save each mean as mean10, mean100, and mean1000.

```
mean10<-mean(sample(human_genes$size, 10, replace = TRUE))
mean100<-mean(sample(human_genes$size, 100, replace = TRUE))
mean1000<-mean(sample(human_genes$size, 1000, replace = TRUE))
#
```

5.4. Compare these means to the population mean. Which is closest?

```
abs(c(mean10, mean100, mean1000)-human_genes_summ$MeanLength) #abs is equivalent to |x|
```

```
## [1] 607.25669 143.74669  65.80869
```

```
#
```

Note: we should have gotten that mean1000 gives the smallest difference, but you MAY have found something different. That's because we did not filter this human_genes dataset to remove the few very long genes that exist. Remember we mentioned this in class. So when you take larger samples you're just more likely to get those values sampled. So without removing them there is a chance that in your random samples the closest mean to the true mean was 100 instead of 1000 (though it is more likely the 1000 sample). Let's filter the long genes out to show this:

```
human_genes2<-human_genes %>% filter(size<=15000) #filter long genes out
human_genes_summ_v2 <- human_genes2 %>% #get summaries for the pop
  summarise(
    MeanLength = mean(size),
    MedianLength = median(size),
    SDLength = sd(size),
)
#get the three smaples
mean10_v2<-mean(sample(human_genes2$size, 10, replace = TRUE))
mean100_v2<-mean(sample(human_genes2$size, 100, replace = TRUE))
mean1000_v2<-mean(sample(human_genes2$size, 1000, replace = TRUE))
abs(c(mean10_v2, mean100_v2, mean1000_v2)-human_genes_summ_v2$MeanLength) #dabs is
↪  equivalent to |x|
```

```
## [1] 627.6955  82.4545  51.7635
```

There we go! As expected, the greater sample size led to a smaller difference to the real mean. This may already have been true for you without the filtering, but with the filtering it is way less likely we might get a conflicting result here.

Ex.5) Congrats you did sampling! Now let's combine this with the skills you learned in loops. You are going to take 100 samples of each of those sample sizes to compare

6.1 First you want to create 3 vectors to store each of your results. Them mean_values_10 etc.

```
mean_values_10 <- numeric(100)
mean_values_100 <- numeric(100)
mean_values_1000 <- numeric(100)
```

6.2 Now you write a for loop like you did before. Execept you can include all three of your new vectors in the same loop.

```
#I will use human_genes2 from now on
for (i in 1:100) {
  mean_values_10[i] <- mean(sample(human_genes2$size, 10, replace = TRUE))
  mean_values_100[i] <- mean(sample(human_genes2$size, 100, replace = TRUE))
  mean_values_1000[i] <- mean(sample(human_genes2$size, 1000, replace = TRUE))
}
```

You have now bootstrapped. Lets graph!

Ex.7). We want to compare our samples to see which sample size got closest to our population mean. You are going to make a histogram of these means.

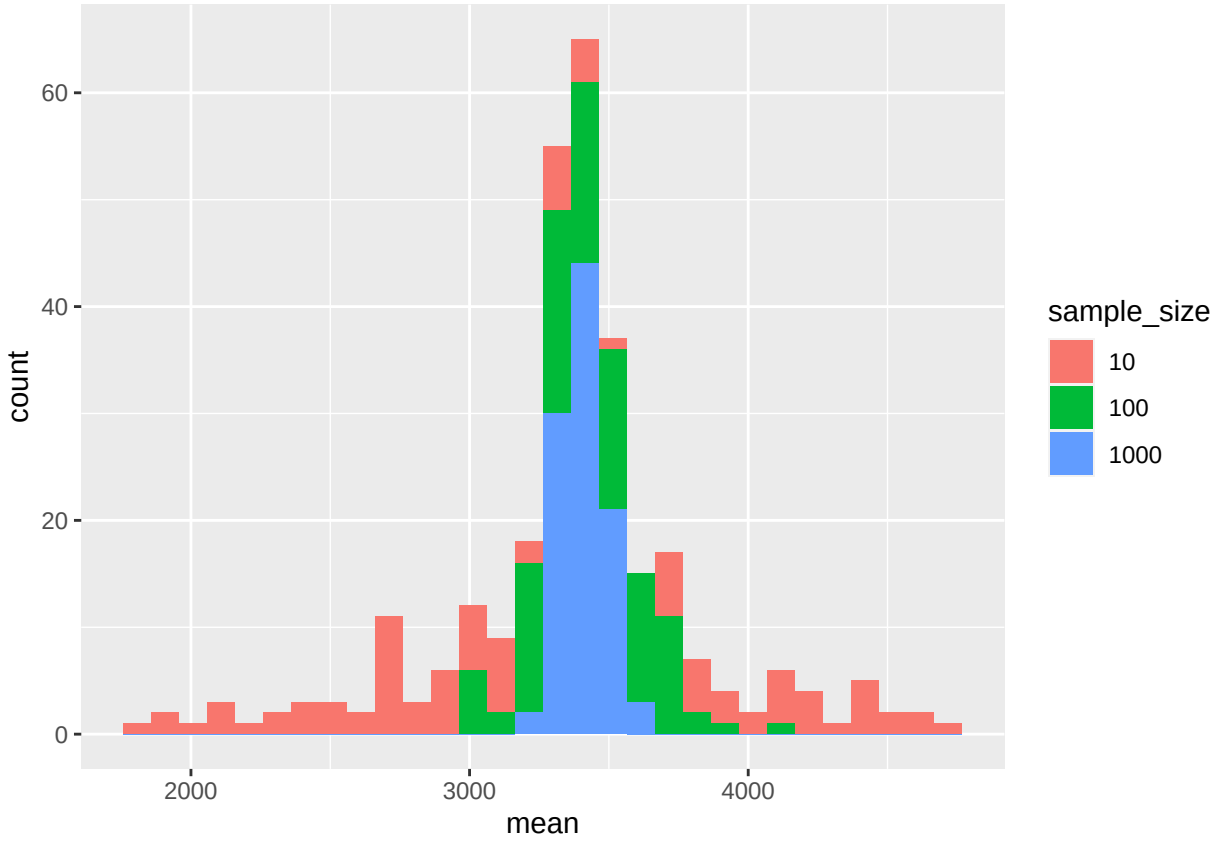
7.1 It's probably a good idea to start with making a data frame of our values. Take the vectors you just made and make one column called "mean" and another column with the corresponding sample size called "sample_size." You're going to have to make this a factor for the graphs to turn out right. You also need to nestle the functions rep inside your factor call, with the argument each = 100. Call your data frame mean_data.

```
mean_data <- data.frame(
  mean = c(mean_values_10, mean_values_100, mean_values_1000),
sample_size = factor(rep(c(10, 100, 1000), each = 100)))
```

7.2 Lets make a basic histogram. x axis is mean, fill by sample_size.

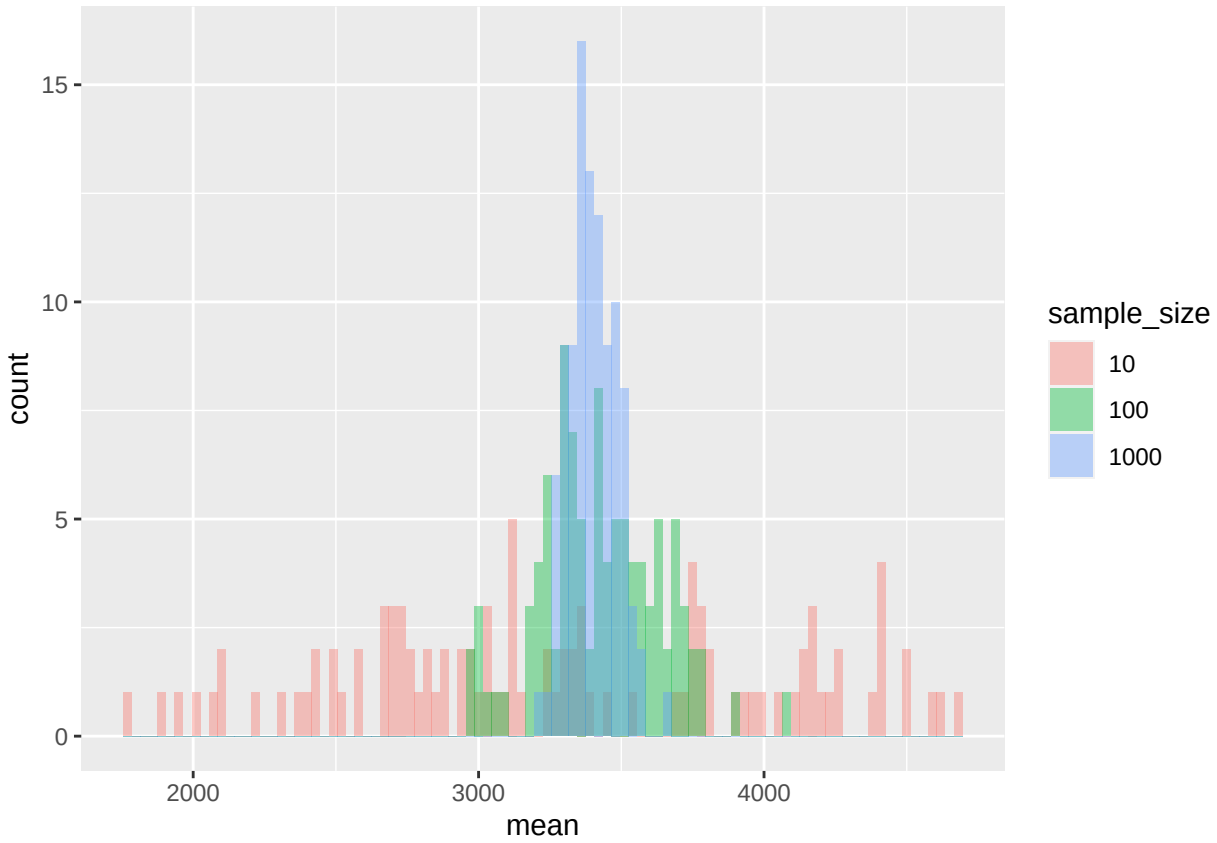
```
library(ggplot2)
plot<-ggplot(mean_data,
             aes(x = mean, fill = sample_size)) +
             geom_histogram()
print(plot)
```

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



7.3 This is a bit hard to read. Let's add some detail. inside geom_histogram, change position to "fill", and both alpha to 0.4 and binwidth to 30.

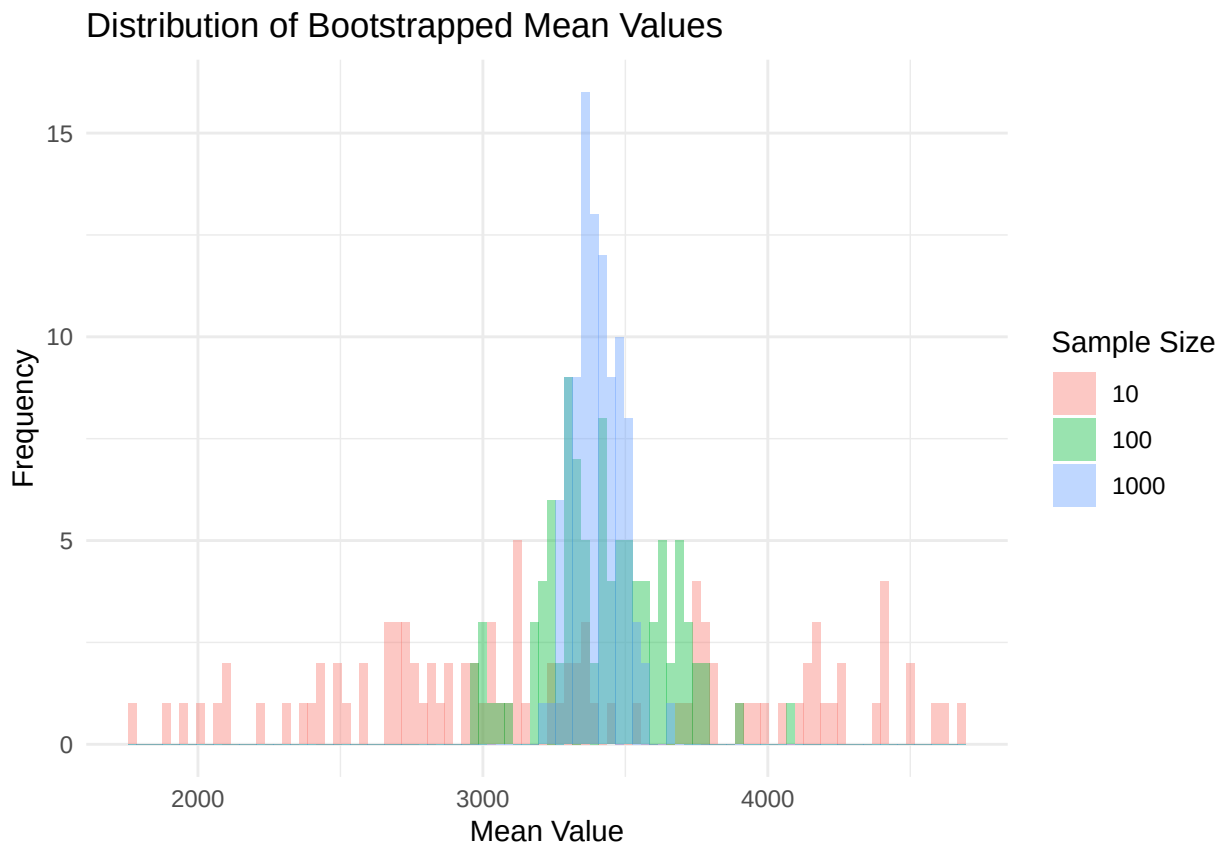
```
plot <- ggplot(mean_data,
               aes(x = mean, fill = sample_size)) +
               geom_histogram(position="identity", alpha=0.4, binwidth=30)
print(plot)
```



7.4 Much better! Ok add some labels on your axis and legend. perhaps even add a theme if you feel silly

```
library(ggplot2)
plot<- ggplot(mean_data, aes(x = mean, fill = sample_size)) +
  geom_histogram(position="identity", alpha=0.4, binwidth=30) +
  labs(x = "Mean Value",
       y = "Frequency",
       title = "Distribution of Bootstrapped Mean Values",
       fill = "Sample Size") +
  theme_minimal()

print(plot)
```



Are you tired of boring ggplot themes? Sick of theme_minimal? Run this code and go crazy devtools::install_github("https://github.com/MatthewBJane/ThemePark") ——

Data Camp activities

To strengthen the skills learned today:

- Intermediate R: Loops
- Sampling in R: Introduction to sampling
- Sampling in R: Bootstrap distributions

The end!

- Knit your document into a PDF using the button above this text editor and upload the PDF into Moodle.

References

[1] Kodable. What are loops? Link

[2] Intro2r. For loops. https://intro2r.com/loops.html