Simulation Cheat Sheet

Nick Huntington-Klein

April 5, 2019

The Purpose of Simulations

- Data is *simulated* if we randomly generate it, and then test its properties
- We do this because, when we make our own data, *we know the process that generated the data*
- That way, we can know whether the *method* we use is capable of getting the right result, because we know what the right result is

Generating Random Data

We can generate categorical or logical random data with `sample()` with the `replace=TRUE` option. This will choose, with equal probability (or unequal probabilities if you specify with the `prob` option), between the elements of the vector you give it.

We can generate *normally* distributed data with `rnorm()`. Normal data is symmetrically distributed around a mean and has no maximum or minimum. By default, the `mean` is 0 and the standard deviation `sd` is 1.

We can generate *uniformly* distributed data with `runif()`. Uniformly distributed data is equally likely to take any value between its `minimum` and `maximum`, which are by default 0 and 1.

In each case, remember to tell it how many observations you want (1000 in the below example).

```r
library(tidyverse)
df <- tibble(category = sample(c('A','B','C'),1000,replace=T),
             normal = rnorm(1000),
             uniform = runif(1000))

table(df$category)
```

```
##
##   A   B   C
## 317 365 318
```

```r
library(stargazer)
stargazer(as.data.frame(df %>% select(normal, uniform)),type='text')
```

```
##
## ============================================================
## Statistic   N    Mean  St. Dev.  Min   Pctl(25) Pctl(75)  Max
## ------------------------------------------------------------
## normal    1,000 -0.029  0.990   -2.869  -0.720   0.674   3.858
## uniform   1,000 0.496   0.289   0.001   0.230    0.752   1.000
## ------------------------------------------------------------
```

Determining the Data Generating Process

You can make sure that `X -> Y` in the data by using values of `X` in the creation of `Y`. In the below example, `W -> X`, `X -> Y`, and `W -> Y`. You can tell because `W` is used in the creation of `X`, and both `X` and `W` are used in the creation of `Y`.

You can incorporate categorical variables into this process by turning them into logicals.

Remember that, if you want to use one variable to create another, you need to finish creating it, and then create the variable it causes in a later `mutate()` command.

```r
data <- tibble(W = sample(c('C','D','E'),2000,replace=T)) %>%
  mutate(X = .1*(W == 'C') + runif(2000)) %>%
  mutate(Y = 2*(W == 'C') - 2*X + rnorm(2000))
```

For Loops

A for loop executes a chunk of code over and over again, each time incrementing some index by 1. Remember that the parentheses and curly braces are required, and also that if you want it to show results while you're running a loop, you must use `print()`.

If you are interested in getting further into R, I'd recommend trying to learn about the `apply` family of functions, which perform loops more quickly.

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

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

Putting the Simulation Together

In a simulation, we want to:

- Create a blank vector to store our results
- (in loop) Simulate data
- (in loop) Get a result
- (in loop) Store the result
- Show what we found

We can create a blank vector to store our results with `c(NA)`. Then, in the loop, whatever calculation we do we can store using the index we're looping over.

Note that, if you like, you can make a blank data frame or tibble instead of a vector, and use `add_row` in dplyr to store results.

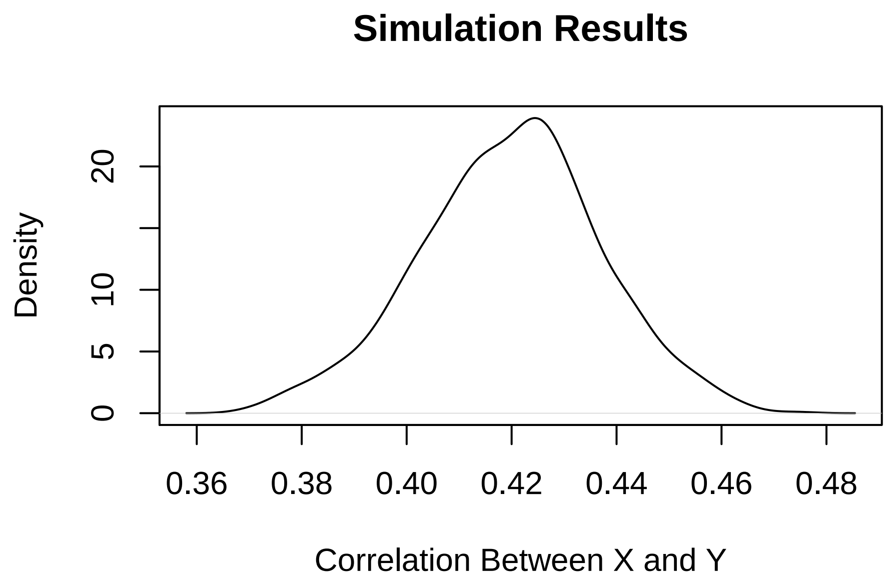
```r
results <- c(NA)

for (i in 1:1000) {
  data <- tibble(W = sample(c('C','D','E'),2000,replace=T)) %>%
    mutate(X = 1.5*(W == 'C') + runif(2000)) %>%
    mutate(Y = 2*(W == 'C') - .5*X + rnorm(2000))

  results[i] <- cor(data$X,data$Y)
}
```

After you're done you can show what happened using standard methods for analyzing a single variable, like creating a summary statistics table with `stargazer()` (don't forget to use `as.data.frame()`) or plotting the density. Make sure you're thinking about what the *true* answer is and how far you are from it. For example, here, we know that `X` negatively affects `Y`, and we might be interested in how often we find that.

```r
library(stargazer)
stargazer(as.data.frame(results),type='text')
```

```
##
## ============================================================
## Statistic   N   Mean  St. Dev.  Min  Pctl(25) Pctl(75)  Max
## ------------------------------------------------------------
## results   1,000 0.420  0.017   0.369  0.409    0.431   0.474
## ------------------------------------------------------------
```

```r
plot(density(results),xlab='Correlation Between X and Y',main='Simulation Results')
abline(v=0)
```


Simulation Results
Density
Correlation Between X and Y