# Recitation 1: Generating Fake Data

Seamus Wagner

August 26, 2021

## Intro to Fake Data

Typically, in quantitative social science, we use real data since we are seeking to contribute to our understanding of the "true" social world. However, using simulated or fake data can be useful to model specific DGPs with certainty along with the relative accessibility of fake data for students from all subfield backgrounds. This means we can create models that hold true and compare them to our expectations given our theoretical considerations

## Packages and Options

```r
## Load packages
library(data.table)
library(dplyr)

## Set seed number
set.seed(16803)

# Sets number of decimals to include
options(digits = 3)

#Set your working directory and create a workable file management system
setwd("E:/Quant 3 TA/Recitations/Scripts")
```

## Part 1

First, we will generate a fake dataset with 100 observations and two potential outcomes, Y1 and Y2. Y1 will be drawn from the standard normal distribution (mean = 0, sd = 1), while Y2 will be drawn from the standard uniform distribution (min = 0, max = 1).

```r
# Set number of observations as 100
n <- 100

# Create a data table with Y1 coming from the normal dist. and
# with Y2 coming from the uniform dist.
#Check the ?runif and ?rnorm for defaults
goddata <- data.table(Y1 = rnorm(n),
                      Y2 = runif(n))

head(goddata)
```

```
##       Y1     Y2
## 1: 0.496 0.2882
## 2: 1.163 0.2039
## 3: 1.352 0.9115
## 4: 0.931 0.5763
## 5: 0.121 0.3651
## 6: 1.207 0.0122
```

Next, we're going to make a third outcome, Y3, that combines Y1 and Y2 by sampling each with probability .5 (as in, there is an equal chance that each observation of Y3 is drawn from either Y1 or Y2). To do this, we're going to make an assignment variable, D, that tell us whether to copy Y1 or Y2 over to Y3.

```
# Make assignment variable
  # Note that the := operator in data.table is used to create new variables
goddata[, D := rbinom(n, 1, .5)]

# check the distribution of D
  # note that it won't necessarily be exactly 50/50, but it should be close
table(goddata$D)
```

```
##
##  0  1
## 47 53
```

```
# Make Y3
  # note how rowwise assignment works in data.table using [rows, newvar := operation] syntax
goddata[D==1, Y3 := Y1] # for rows where D = 1, Y3 takes on the value of Y1
goddata[D==0, Y3 := Y2] # for rows where D = 0, Y3 takes value of Y2
```

Note that we can do the same thing using tidyverse, with slightly different syntax. Tidyverse and data.table each have their respective strengths – I data.table is faster, while tidyverse can be easier to understand in plain English – and are often not mutually exclusive. For instance, you might find it useful to do your analyses in data.table but your visualization using ggplot, which is part of the tidyverse. Developers are also working on versions of tidy operations that incorporate data.table principles (if you want, you can check them out in the 'dtplyr' package) to get the best of both worlds.

In any case, here's what we just did – but in a tidy framework.

```
# Creating a "tibble" based on a data frame with Y1 and Y2 coming from
# the same distribution as before
goddata2 <- as_tibble(data.frame(Y1 = rnorm(n), Y2 = runif(n)))
head(goddata2)
```

```
## # A tibble: 6 x 2
##       Y1    Y2
##    <dbl> <dbl>
## 1 -0.238 0.193
## 2 -1.64  0.572
## 3 -0.184 0.329
## 4 -0.301 0.610
## 5  0.142 0.819
## 6  0.558 0.340
```

```
# Making D as variable that will say whether Y3 = Y1 or Y3 = Y2
  # In tidyverse, mutate() makes new variables
goddata2 <- goddata2 %>% mutate(D = rbinom(n, 1, .5))
head(goddata2)
```

```
## # A tibble: 6 x 3
##       Y1    Y2     D
##    <dbl> <dbl> <int>
## 1 -0.238 0.193     1
## 2 -1.64  0.572     0
## 3 -0.184 0.329     0
## 4 -0.301 0.610     0
## 5  0.142 0.819     1
## 6  0.558 0.340     0
```

```
# Setting Y3 to equal Y1 when D=1 and Y2 otherwise
goddata2 <- goddata2 %>% mutate(Y3 = ifelse(D == 1, Y1, Y2))
head(goddata2)
```

```
## # A tibble: 6 x 4
##       Y1    Y2     D     Y3
##    <dbl> <dbl> <int>  <dbl>
## 1 -0.238 0.193     1 -0.238
## 2 -1.64  0.572     0  0.572
## 3 -0.184 0.329     0  0.329
## 4 -0.301 0.610     0  0.610
## 5  0.142 0.819     1  0.142
## 6  0.558 0.340     0  0.340
```

Back to data.table. Next, we're going to take the mean of each of our potential outcomes, and then the mean of those means. This should confirm that our fake data has the properties we said it should. The mean of Y1 should be around zero, as per the standard normal, while the mean of Y2 should be around 0.5, as per the standard uniform. The mean of those means should be halfway in between them, and the mean of Y3 should be *about* halfway in between them (since, remember, we didn't have an exactly 50/50 split in our assignment variable).

```
# Q4: Means of Y1, Y2
my1 <- mean(goddata$Y1)
my2 <- mean(goddata$Y2)

# then the average of those means
Y1Y2mean_of_means <- (my1 + my2)/2

# followed by the mean of Y3 and difference between latter two values
my3 <- mean(goddata$Y3)

# check difference between mean of means and mean of Y3 (should be small)
Y1Y2mean_of_means - my3
```

```
## [1] 0.0347
```

Next, we're going to make a new variable, Y4, that samples from Y1 and Y2 unevenly. We do this by simply changing the distribution of our new assignment variable, D2, relative to the equal probability we gave to D.

Note that we can come fairly close to reconstructing the mean of this new variable, Y4, by manually weighting the means of Y1 and Y2. Pause here to make sure you understand why.

```
# Assign uneven probability of D = 1.
goddata[, D2 := rbinom(n, 1, .75)]
goddata[D2 == 1, Y4 := Y1]
goddata[D2 == 0, Y4 := Y2]

# Take mean of new Y4
my4 <- mean(goddata$Y4)

# weight means of Y1 and Y2 to match distribution of assignment for Y4.
#Check that the uneven value match the respective distributions.
Y1Y2_weighted <- .75*my1 + .25*my2

# check difference between weighted and actual
Y1Y2_weighted - my4
```

```
## [1] 0.0698
```

Quick knitr() note: you can pull objects into the text of your document by calling R directly. For example, here I'm printing the value of the mean of Y1 that we stored earlier: 0.103.

Finally, we can generate *more* fake data by making n larger, and repeat the whole process. Making n larger is typically better for reasons we know, but an added benefit of fake data is the ease of demonstration and low storage demands for larger simulated data.

```
n <- 10000
goddata <- data.table(Y1 = rnorm(n),
                      Y2 = runif(n))
goddata[, D := rbinom(n, 1, .5)]
goddata[D==1, Y3 := Y1]
goddata[D==0, Y3 := Y2]

mean(goddata$Y1)
```

```
## [1] 0.0024
```

```
mean(goddata$Y2)
```

```
## [1] 0.498
```

```
Y1Y2mean_of_means <- (mean(goddata$Y1) + mean(goddata$Y2))/2

mean(goddata$Y3)
```

```
## [1] 0.258
```

```
((mean(goddata$Y1) + mean(goddata$Y2))/2) - mean(goddata$Y3)
```

```
## [1] -0.00777
```

```
goddata[, D2 := rbinom(n, 1, .75)]
goddata[D2 == 1, Y4 := Y1]
goddata[D2 == 0, Y4 := Y2]

mean(goddata$Y4)
```

```
## [1] 0.138
```

```
Y1Y2mean_of_means
```

```
## [1] 0.25
```

```
Y1Y2_weighted <- .75*mean(goddata$Y1) + .25*mean(goddata$Y2)
```

If we want to simulate this a number of times, we can create looping or apply family functions.

```
# Keep a lower n and n_sims to shorten replication time while testing functions
n_sims <- 100
n <- 1000

# Create a function that takes anything (though we will use n and n_sims as inputs).
diffs <- function(...){
# Then a simple way to do this is to place the code from above into a function within,
  #and return the difference using some type of looping/apply operation
goddata <- data.table(Y1 = rnorm(n),
                      Y2 = runif(n))
goddata[, D := rbinom(n, 1, .5)]
goddata[D==1, Y3 := Y1]
goddata[D==0, Y3 := Y2]
Y1Y2mean_of_means <- (mean(goddata$Y1) + mean(goddata$Y2))/2
goddata[, D2 := rbinom(n, 1, .75)]
goddata[D2 == 1, Y4 := Y1]
goddata[D2 == 0, Y4 := Y2]
Y1Y2_weighted <- .75*mean(goddata$Y1) + .25*mean(goddata$Y2)
  Y1Y2_diff <- data.table(Y1Y2_weighted - my4)
}

#We can then use replicate() to complete the replication for n_sims times.
simulations <- replicate(n_sims, diffs(n,n_sims))

#We can then unlist the returned values and make them a data table.
simulations <- data.table(unlist(simulations))
head(simulations)
```

```
##          V1
## 1: -0.01438
## 2: -0.03271
## 3:  0.00383
## 4:  0.02230
## 5:  0.00106
## 6: -0.00586
```