Lab 1 Exercise

Theodore Landsman

8/26/2021

FIRST, SOME TIPS & HELPFUL NOTES...

```
# Place a hashtag (or several!) before text to write yourself a note as I'm doing here
#   ALWAYS LEAVE YOURSELF NOTES!
```

- Coding in R is case-sensitive!
- Run lines of code by holding down "ctrl" + "r" OR "ctrl" + "enter" on Windows; "cmd" + "return" on Mac
- There is also a button up top that says "Run" and you can use that too
- In RMarkdown, run code `chunks` by clicking the green play button at the top right of the `chunk` when you hit the blue yarn `knit` button at the top left of the file, all r code will be executed as part of generating your output file.

Some RMD tips

Setting Eval to False means that R won't evaluate the R code at all, but will display it.

```
summary(cars)
```

Setting Echo to False means that R will run the code but won't display it. Not that if we combine echo = FALSE and eval = FALSE nothing about the code will effect the final document at all.

```
##      speed           dist
##  Min.   : 4.0   Min.   :  2.00
##  1st Qu.:12.0   1st Qu.: 26.00
##  Median :15.0   Median : 36.00
##  Mean   :15.4   Mean   : 42.98
##  3rd Qu.:19.0   3rd Qu.: 56.00
##  Max.   :25.0   Max.   :120.00
```

```
##      speed           dist
##  Min.   : 4.0   Min.   :  2.00
##  1st Qu.:12.0   1st Qu.: 26.00
##  Median :15.0   Median : 36.00
##  Mean   :15.4   Mean   : 42.98
##  3rd Qu.:19.0   3rd Qu.: 56.00
##  Max.   :25.0   Max.   :120.00
```

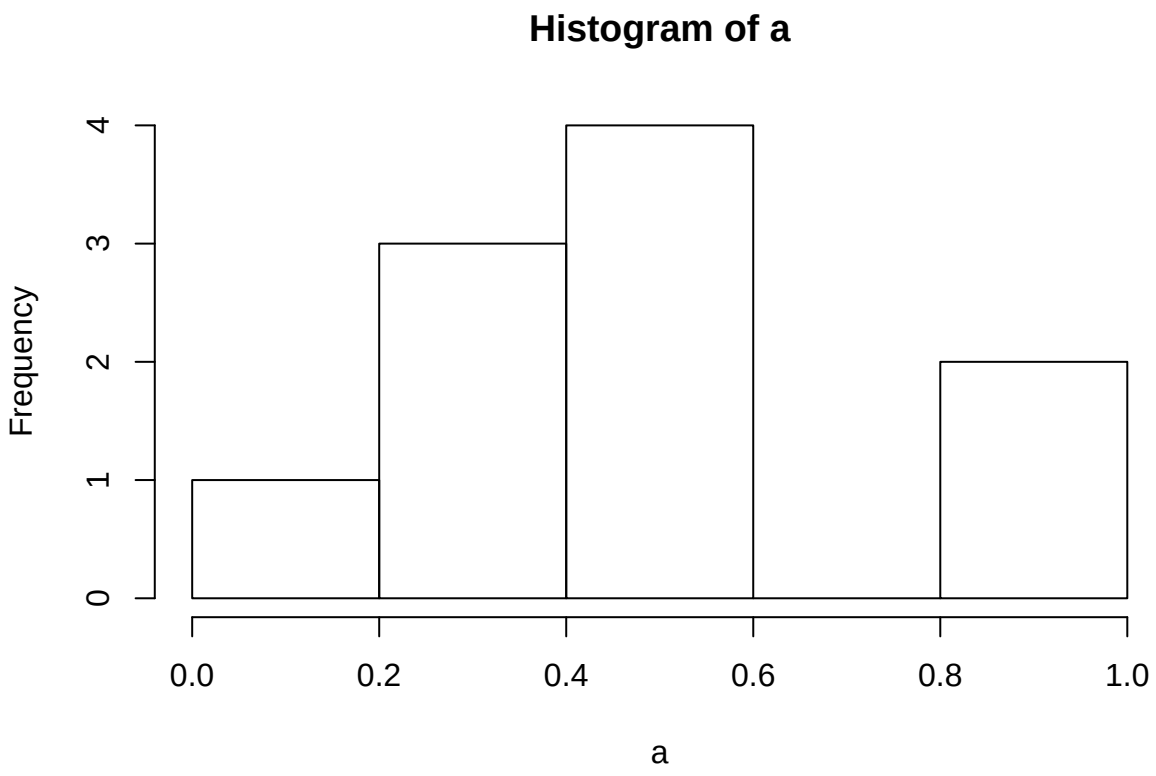
Setting Message and Warning to False means that R will run the code and display it but won't display any warnings or messages that the code creates as part of running, this is particularly useful for data import since import often generates a lot of uninteresting messages and warnings and is often combined with echo = FALSE because import code is also often not worth displaying.

Let's Create Some Fake Data!

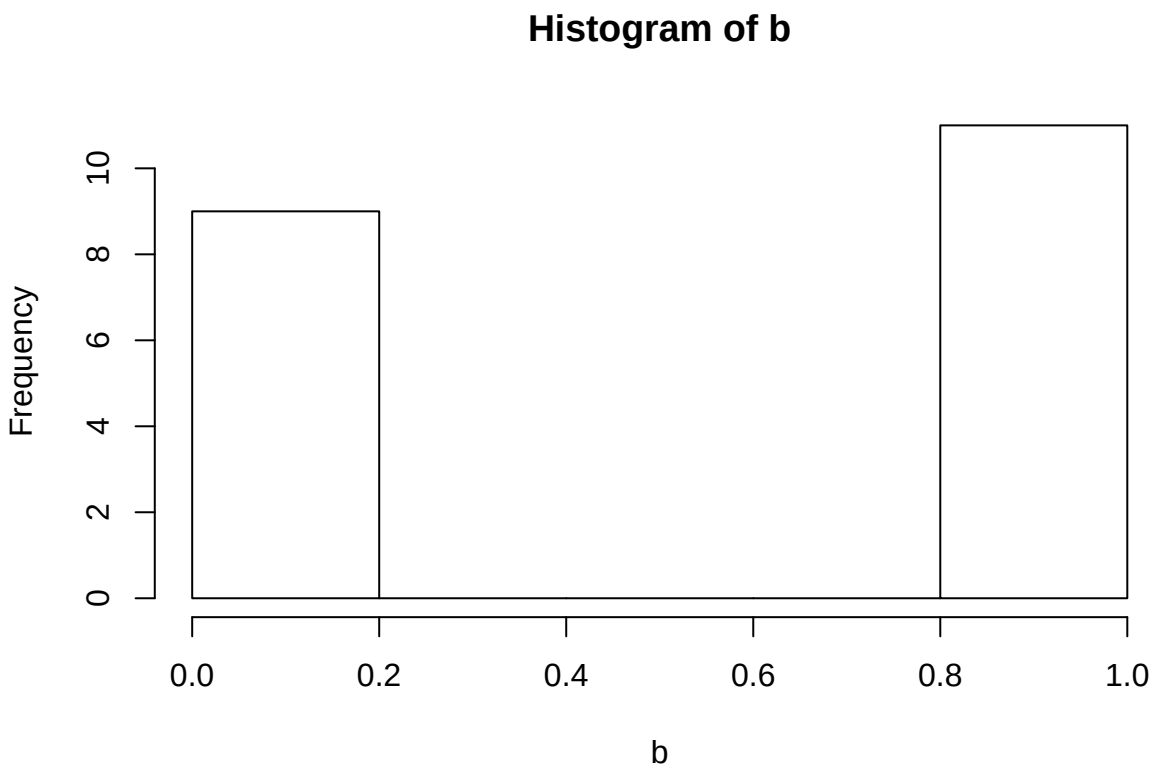
```
a <- runif(10) # vector of random selection of 10 numbers from 0 to 1.
b <- rbinom(20,1,.5) # random vector generated from a know probability of success (1) or failure (0), i
# with 20 elements, 1 run of the probabilistic outcome and a .5 chance of success, basically a coin fli
c <- rnorm(15) # random vector R generates from a standard normal distribution (mean=0, sd=1)
d <- rnorm(100, mean=5, sd=2) #vector of random selection of
# 100 numbers from standard distribution with mean=5, sd=2
```

Let's Interpret Some Fake Data

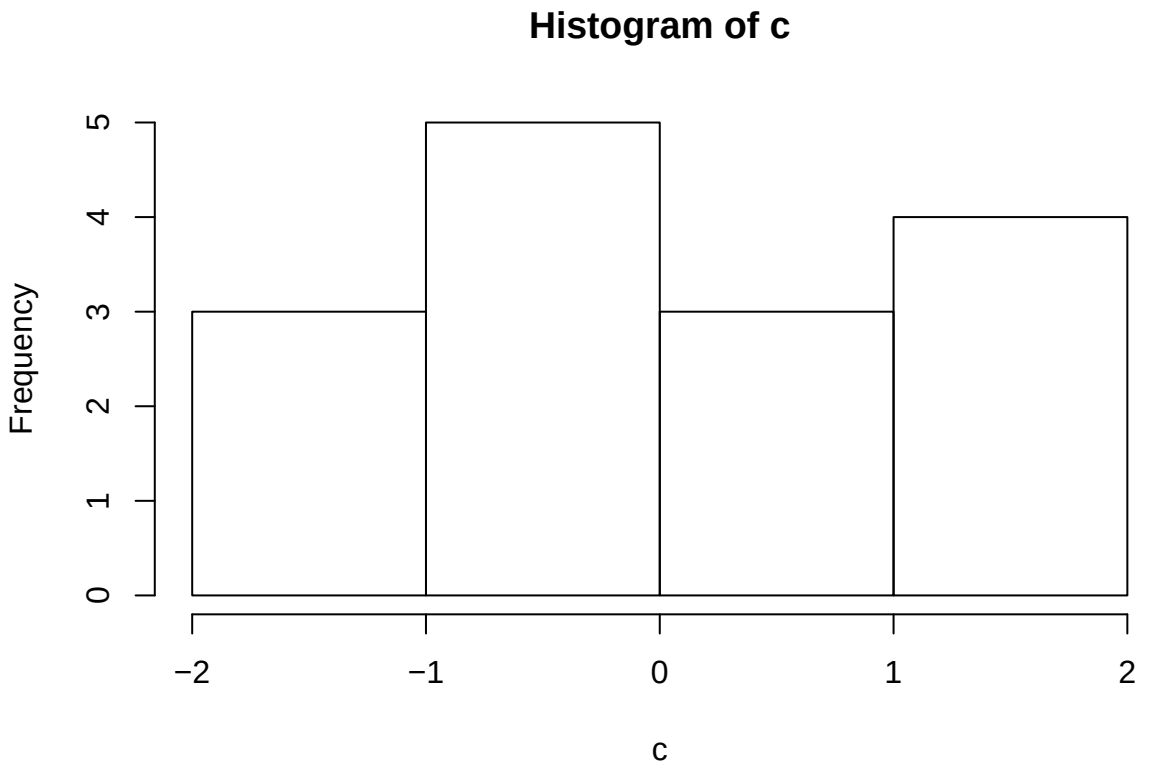
```
hist(a)
```



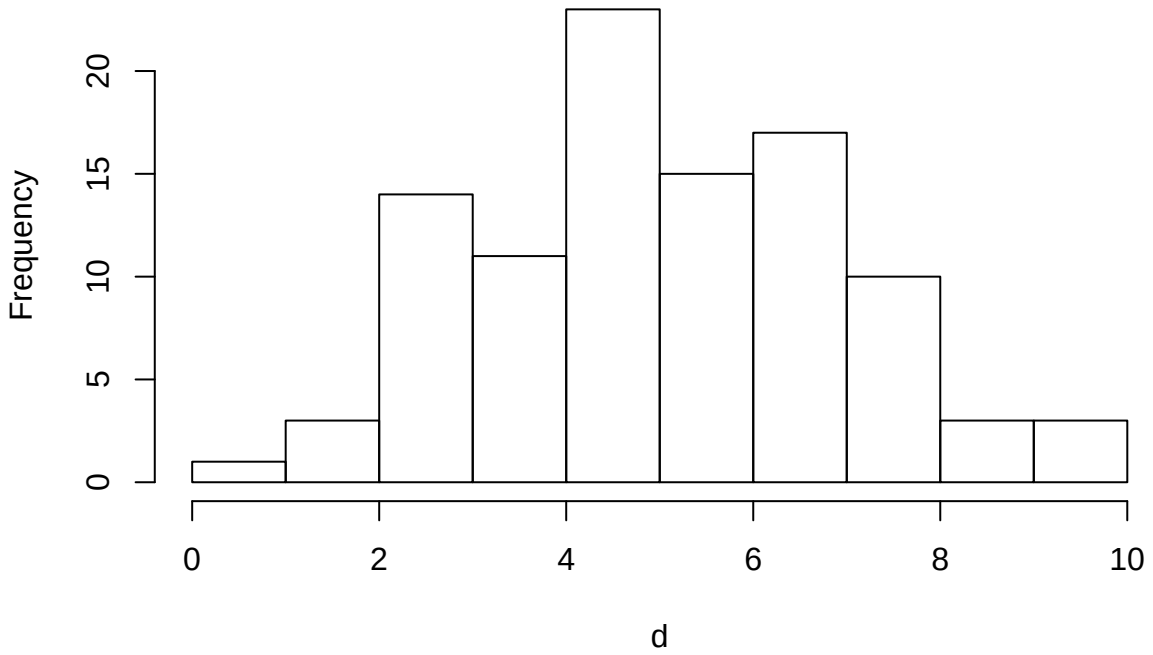

```
hist(b)
```


Histogram of b
Frequency
0
2
4
6
8
10
0.0
0.2
0.4
0.6
0.8
1.0
b

```
hist(c)
```


Histogram of c
Frequency
0
1
2
3
4
5
−2
−1
0
1
2
c

```
hist(d)
```




```
summary(d)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.8934  3.8013  4.9639  5.0793  6.4744  9.8509
```

```
sd(d)
```

```
## [1] 1.87416
```

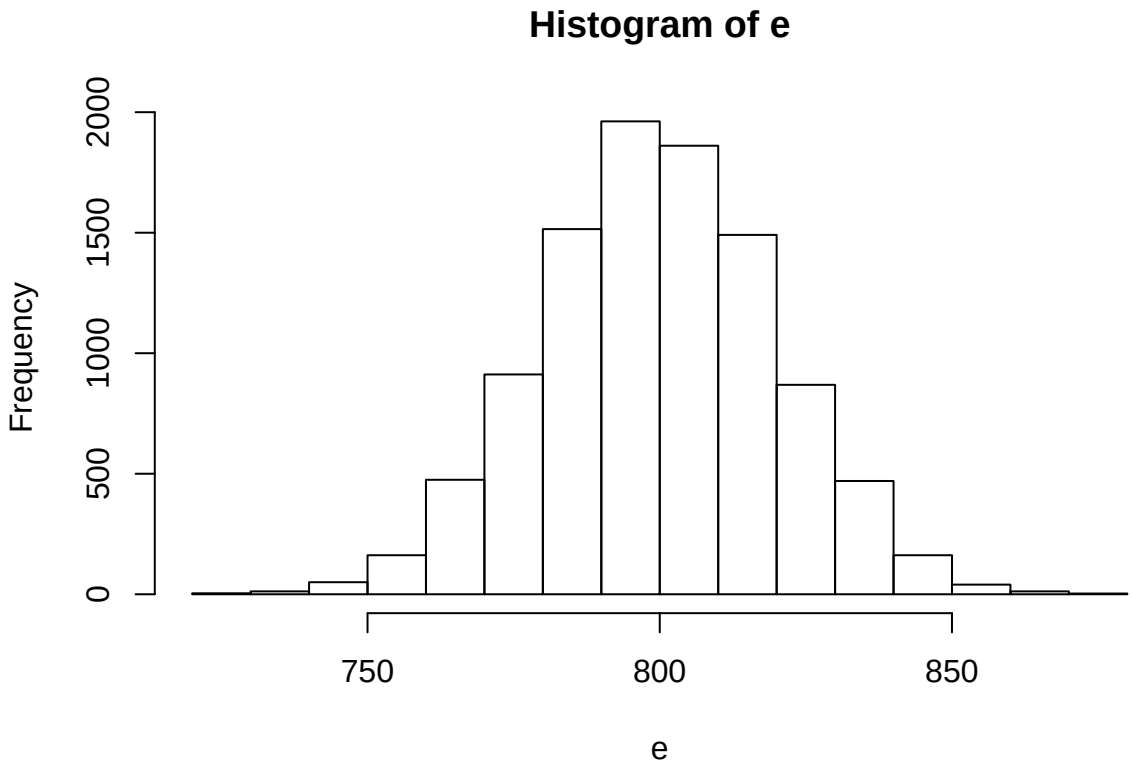
Understanding check! Why is the mean 4.450 instead of 5? Why is the standard deviation 2.19 instead of 2?

Fun with binomials

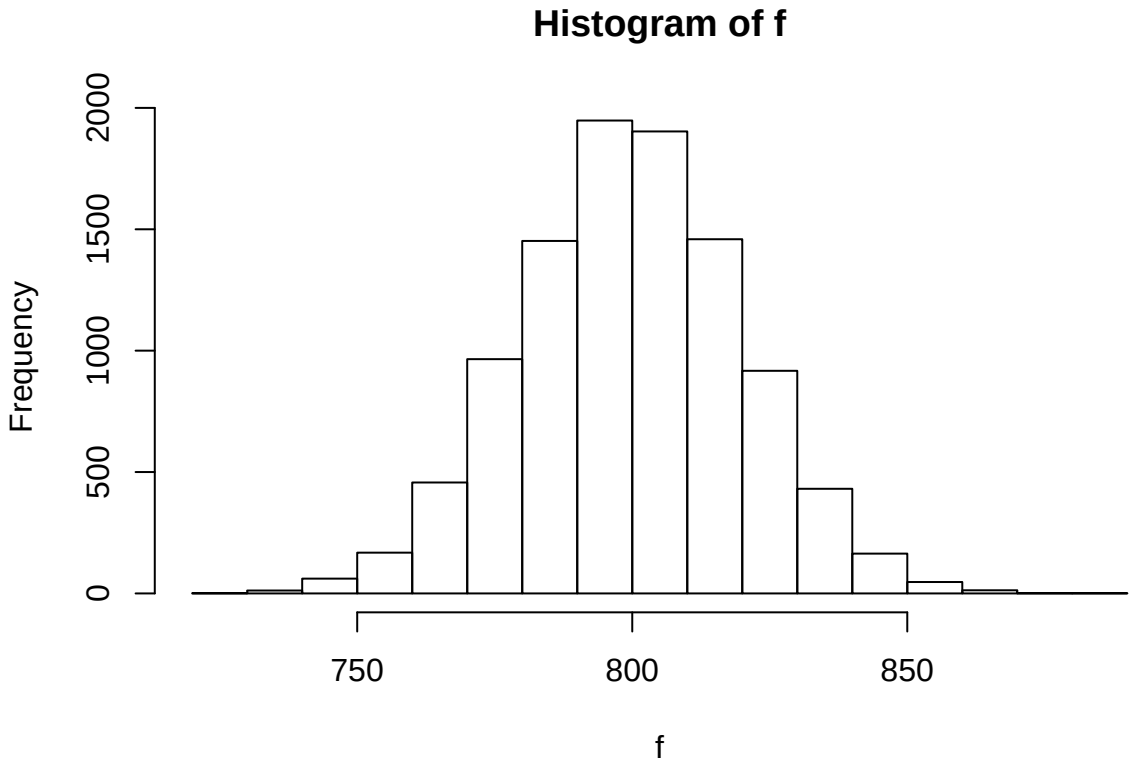
```
e <- rbinom(10000, 1600, .5)
sqrt(1600*.25)
```

```
## [1] 20
```

```
f <- rnorm(10000, mean =800, sd = 20)
hist(e)
```




```
hist(f)
```



Note that as we increase the number of trials (eg, two coin flips, which could produce 2 heads (2), a heads and a tails (1), or two tails (2)) the binomial distribution converges to a normal distribution where the mean is the probability of success times the number of trials, and the standard deviation is the square root of the number of trials times the probability of success times the probability of failure. sd = sqrt(n p (1-p)).

In other words, the normal distribution is the ideal type for a random process iterated across enough trials!

Combining Fake Variables into a Fake Dataframe

Let's start with two normally distributed variables with slightly different properties

```
g <- rnorm(100, mean = 5, sd = 2)

fakeData <- data.frame(d,g)
```

Now we're going to relate them to each other by creating a new variable.

```
fakeData$h <- g+d
```

Let's see if we could pursuade a social scientist that these two variables are related.

```
cov <- cov(fakeData$g, fakeData$h)
cov # >0 (positive), <0 (negative)
```

```
## [1] 3.463718
```

```
cor_1 <- cor(fakeData$d, fakeData$h)
cor_2 <- cor(fakeData$g, fakeData$h)
cor_1 # -1 (strong neg cor) to 1 (strong pos cor)
```

```
## [1] 0.6744108
```

```
cor_2
```

```
## [1] 0.6939
```

Both are pretty strongly correlated! What if we wanted to fake a specific correlation?

```
fakeData$i <- fakeData$g*1 + fakeData$d*.5
cov_2 <- cov(fakeData$g, fakeData$i)
cor_3 <- cor(fakeData$d, fakeData$i)
cor_4 <- cor(fakeData$g, fakeData$i)
cor_3
```

```
## [1] 0.3909729
```

```
cor_4
```

```
## [1] 0.89363
```

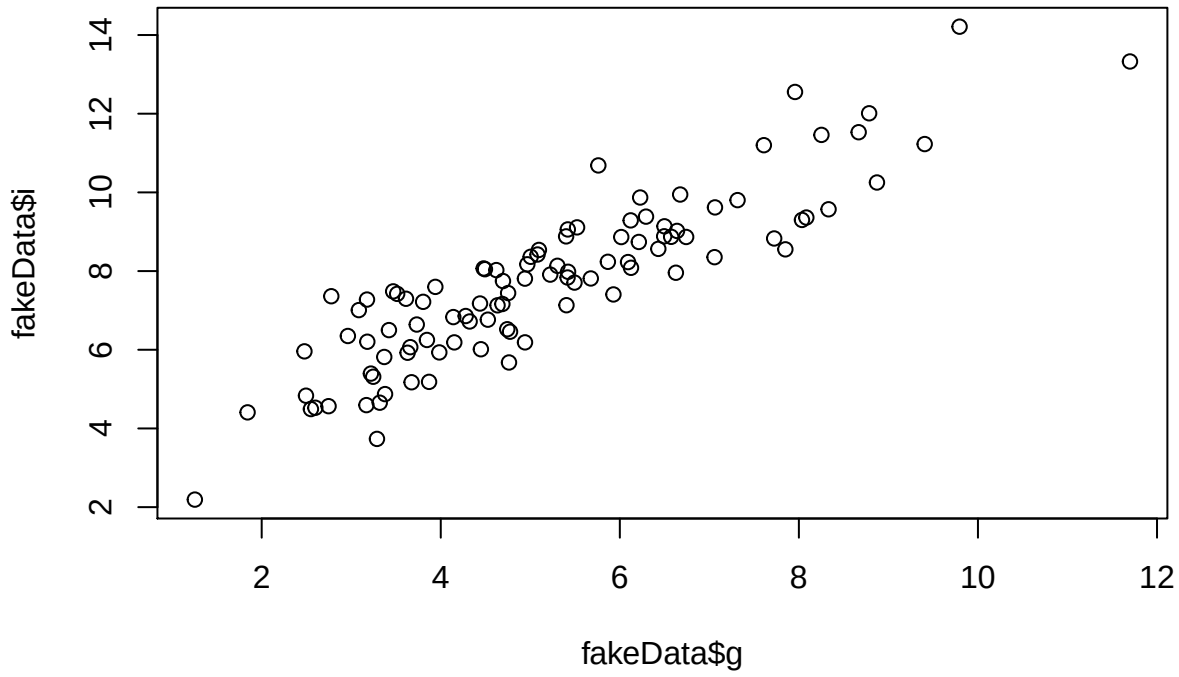
Not bad for only 100 pieces of data!

Understanding Check! What is equation we are using for variable assignment starting to look like?

Graphing Data With Two-Way Plots

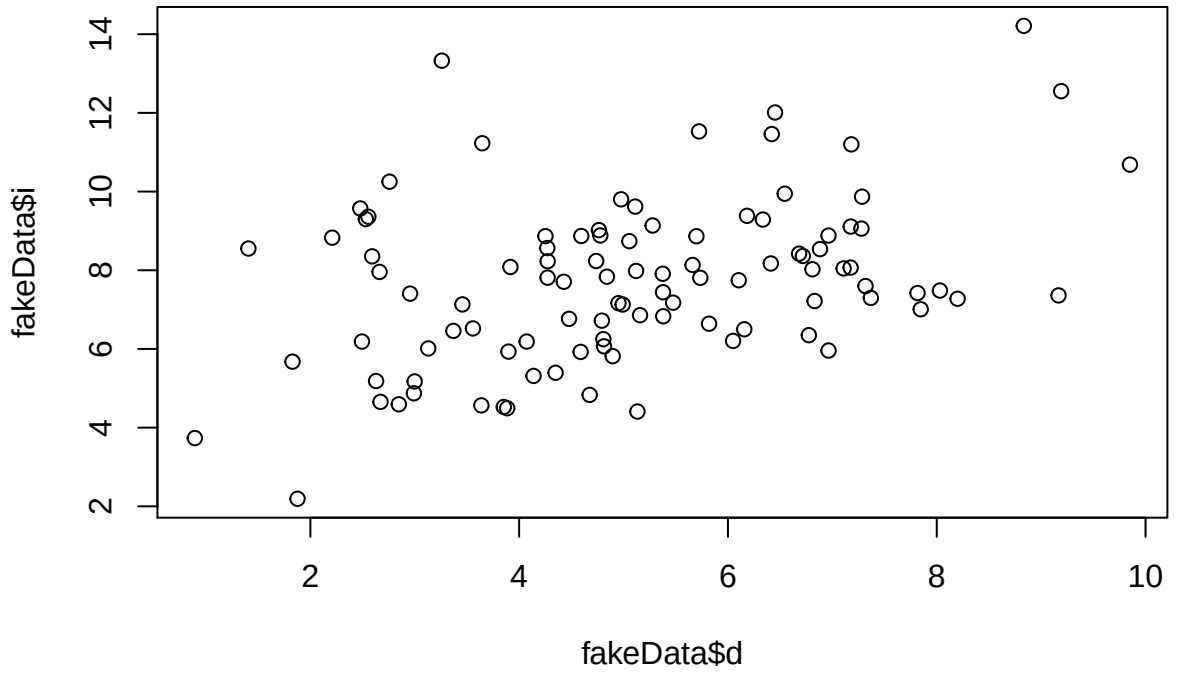
What would a scatterplot of these variables look like?

```
plot(fakeData$g, fakeData$i)
```



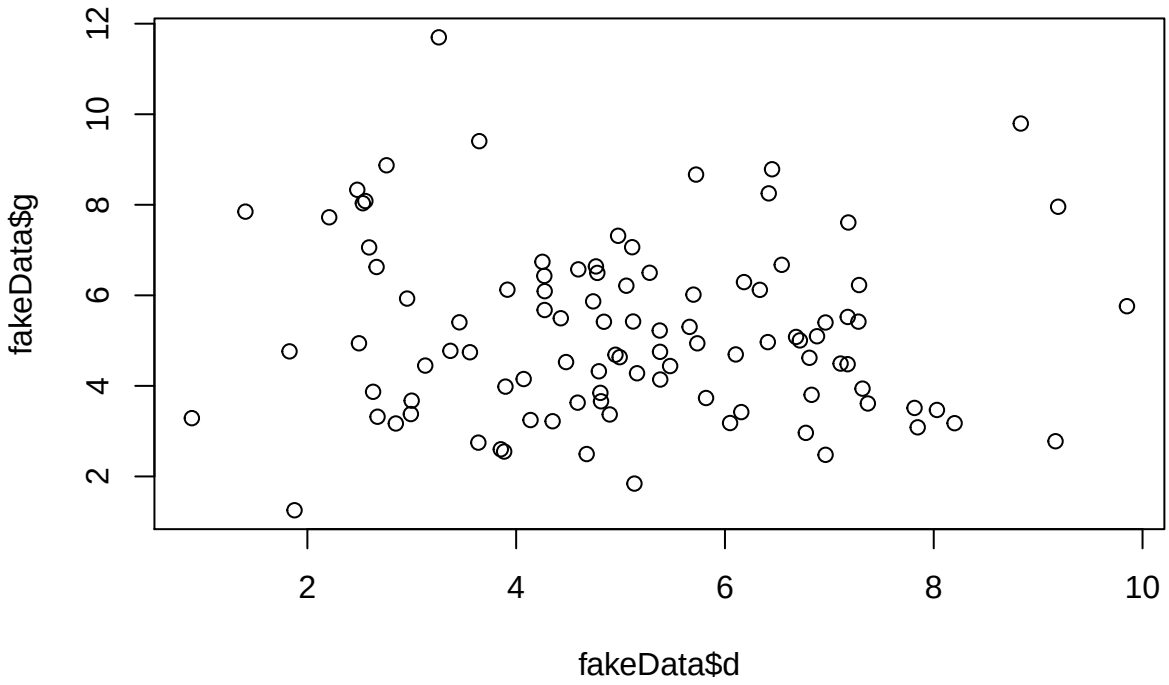
How about for our weaker correlation?

```
plot(fakeData$d, fakeData$i)
```



How about for two variables that are uncorrelated?

```
plot(fakeData$d, fakeData$g)
```



Practice Exercise:

Note that there may be different code that works!

1) Create a variable that is negatively correlated with d, demonstrate this correlation.
2) Create a variable that is negatively correlated with d and positively correlated with g, demonstrate both correlations seperately
3) Create a variable that is negatively correlated with d and g, where this correlation is made 'noisy' by the addition of another random variable (which we can think of as the error term).

End

Note: when closing R, save your RProject file so that you can return to your datasets with all of your relevant code loaded. If you are not using an RProject, it is generally better to NOT save the "workspace image" i.e., the objects in the Environment) Starting with a clean workspace each session makes it easier to keep track of your objects and prevent programming issues. Be intentional about the project space you are working in!