# Intro to Data Science - HW 3

```
# Enter your name here: Mark Cappiello
```



**Attribution statement: (choose only one and delete the rest)**

```
# 1. I did this homework by myself, with help from the book and the professor.
```

**Reminders of things to practice from last week:**

Make a data frame data.frame( ) Row index of max/min which.max( ) which.min( ) Sort value or order rows sort( ) order( ) Descriptive statistics mean( ) sum( ) max( ) Conditional statement if (condition) "true stuff" else "false stuff"

**This Week:**

Often, when you get a dataset, it is not in the format you want. You can (and should) use code to refine the dataset to become more useful. As Chapter 6 of Introduction to Data Science mentions, this is called "data munging." In this homework, you will read in a dataset from the web and work on it (in a data frame) to improve its usefulness.

## Part 1: Use read_csv( ) to read a CSV file from the web into a data frame:

A. Use R code to read directly from a URL on the web. Store the dataset into a new dataframe, called dfComps. The URL is: "https://intro-datascience.s3.us-east-2.amazonaws.com/companies1.csv" **Hint:** use read_csv( ), not read.csv( ). This is from the **tidyverse package**. Check the help to compare them.

```
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.5
## v forcats   1.0.0     v stringr   1.5.1
## v ggplot2   3.5.0     v tibble    3.2.1
## v lubridate 1.9.3     v tidyr     1.3.1
## v purrr     1.0.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```
dfComps <- read_csv(file = "https://intro-datascience.s3.us-east-2.amazonaws.com/companies1.csv")
```

```
## Rows: 47758 Columns: 18
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (16): permalink, name, homepage_url, category_list, market, funding_tota...
## dbl  (2): funding_rounds, founded_year
##
## 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.
```

## Part 2: Create a new data frame that only contains companies with a homepage URL:

E. Use **subsetting** to create a new dataframe that contains only the companies with homepage URLs (store that dataframe in **urlComps**).

```
urlComps <- dfComps[!is.na(dfComps$homepage_url),]
```

D. How many companies are missing a homepage URL?

```
sum(is.na(dfComps$homepage_url))
```

```
## [1] 3323
```

## Part 3: Analyze the numeric variables in the dataframe.

G. How many **numeric variables** does the dataframe have? You can figure that out by looking at the output of **str(urlComps)**.

H. What is the average number of funding rounds for the companies in **urlComps**?

```
sumNumericVariables <- sum(sapply(urlComps, is.numeric))
sumNumericVariables
```

```
## [1] 2
```

```
avgFundingRounds <- mean(urlComps$funding_rounds,na.rm = TRUE)
avgFundingRounds
```

```
## [1] 1.725194
```

I. What year was the oldest company in the dataframe founded? **Hint:** If you get a value of "NA," most likely there are missing values in this variable which preclude R from properly calculating the min & max values. You can ignore NAs with basic math calculations. For example, instead of running mean(urlComps$founded_year), something like this will work for determining the average (note that this question needs to use a different function than 'mean'.

```
#mean(urlComps$founded_year, na.rm=TRUE)

#your code goes here
min(urlComps$founded_year, na.rm=TRUE)
```

```
## [1] 1900
```

## Part 4: Use string operations to clean the data.

K. The **permalink variable** in **urlComps** contains the name of each company but the names are currently preceded by the prefix "/organization/". We can use str_replace() in tidyverse or gsub() to clean the values of this variable:

```
urlComps$permalink <- str_replace(urlComps$permalink ,'/organization/','')
head(urlComps$permalink, 10)
```

```
##  [1] "waywire"            "tv-communications"  "rock-your-paper"
##  [4] "in-touch-network"   "n-plusn"            "club-domains"
##  [7] "fox-networks"       "0-6-com"            "004-technologies"
## [10] "01games-technology"
```

L. Can you identify another variable which should be numeric but is currently coded as character? Use the as.numeric() function to add a new variable to **urlComps** which contains the values from the char variable as numbers. Do you notice anything about the number of NA values in this new column compared to the original "char" one?

```
#--- Yes,funding_total_usd
urlComps$funding_total_usd_num <- as.numeric(urlComps$funding_total_usd)
```

```
## Warning: NAs introduced by coercion
```

```
#--- it converted all values to NA
```

M. To ensure the char values are converted correctly, we first need to remove the spaces between the digits in the variable. Check if this works, and explain what it is doing:

```
library(stringi)
urlComps$funding_new <- stri_replace_all_charclass(urlComps$funding_total_usd,"\\p{WHITE_SPACE}", "")
head(urlComps$funding_new, 10)
```

```
##  [1] "1750000" "4000000" "40000"   "1500000" "1200000" "7000000" "4912393"
##  [8] "2000000" "-"       "41250"
```

```
#--- This command finds all values in urlComps$funding_total_usd and replaces
#--- the WHITE_SPACE characters with nothing as represented by ""
```

```
Error in stri_replace_all_charclass(urlComps$funding_total_usd, "\\p{WHITE_SPACE}", : object 'urlComps'
Traceback:
```

```
1. stri_replace_all_charclass(urlComps$funding_total_usd, "\\p{WHITE_SPACE}",
 .     "")
```

N. You are now ready to convert **urlComps$funding_new** to numeric using as.numeric().

Calculate the average funding amount for **urlComps**. If you get "NA," try using the **na.rm=TRUE** argument from problem I.

```
urlComps$funding_new <- as.numeric(urlComps$funding_new)
```

```
## Warning: NAs introduced by coercion
```

```
avgFundingAmount <- mean(urlComps$funding_new, na.rm = TRUE)
```

Sample three unique observations from urlComps$funding_rounds, store the results in the vector 'observations'

```
observations <- sample(urlComps$funding_rounds, 3, replace = FALSE)
observations
```

```
## [1] 1 4 1
```

Take the mean of those observations

```
mean(observations)
```

```
## [1] 2
```

Do the two steps (sampling and taking the mean) in one line of code

```
mean(sample(urlComps$funding_rounds, 3, replace = FALSE))
```

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

Explain why the two means are (or might be) different

Use the replicate( ) function to repeat your sampling of three observations of urlComps$funding_rounds observations five times. The first argument to replicate( ) is the number of repeats you want. The second argument is the little chunk of code you want repeated.

```
#--- the two means are diferent because they were taken from different samples

replicate(5,sample(urlComps$funding_rounds, 3, replace = FALSE), simplify = TRUE)
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    2    1    1    8    3
## [2,]    2    1    5    1    1
## [3,]    1    3    1    2    1
```

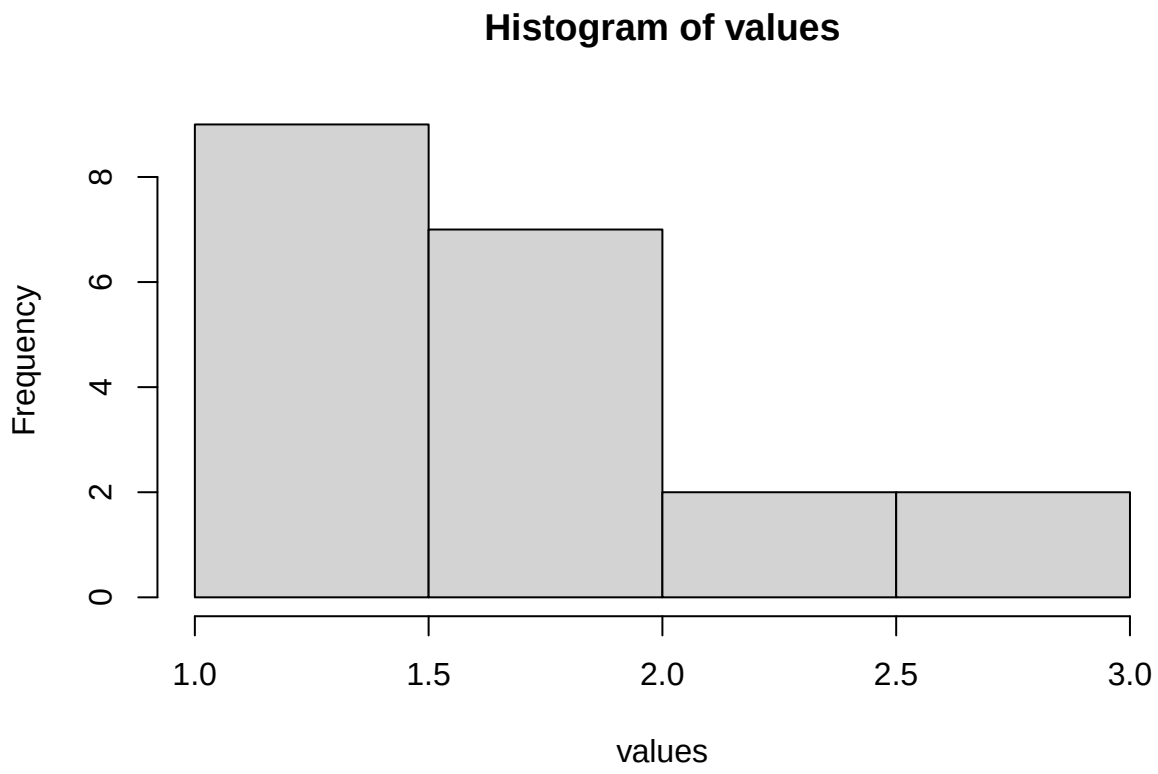
Rerun your replication, this time doing 20 replications and storing the output of replicate() in a variable called **values**.

```
values <- replicate(20,mean(sample(urlComps$funding_rounds, 3, replace = FALSE), simplify = TRUE))
values
```

```
##  [1] 2.666667 1.333333 1.666667 1.333333 1.666667 1.333333 1.000000 1.000000
##  [9] 2.333333 1.666667 1.000000 1.666667 1.666667 3.000000 1.000000 1.333333
## [17] 1.000000 2.000000 2.333333 1.666667
```

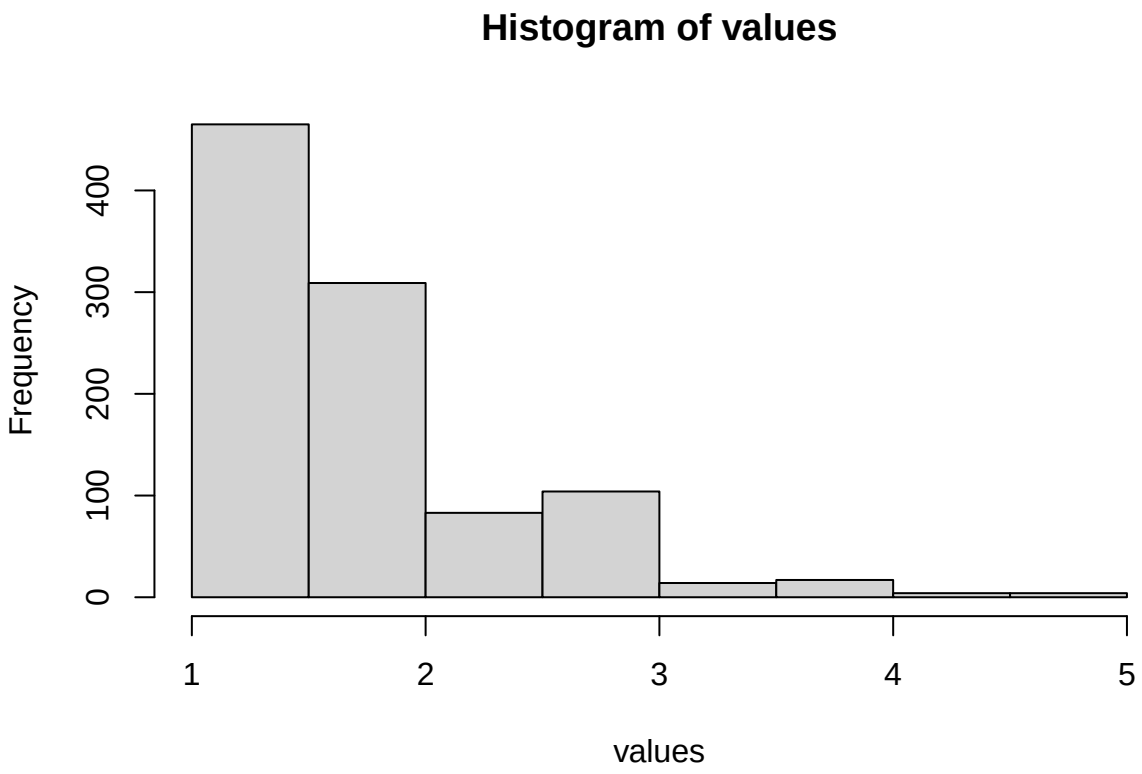
Generate a **histogram** of the means stored in **values**.

```
hist(values)
```



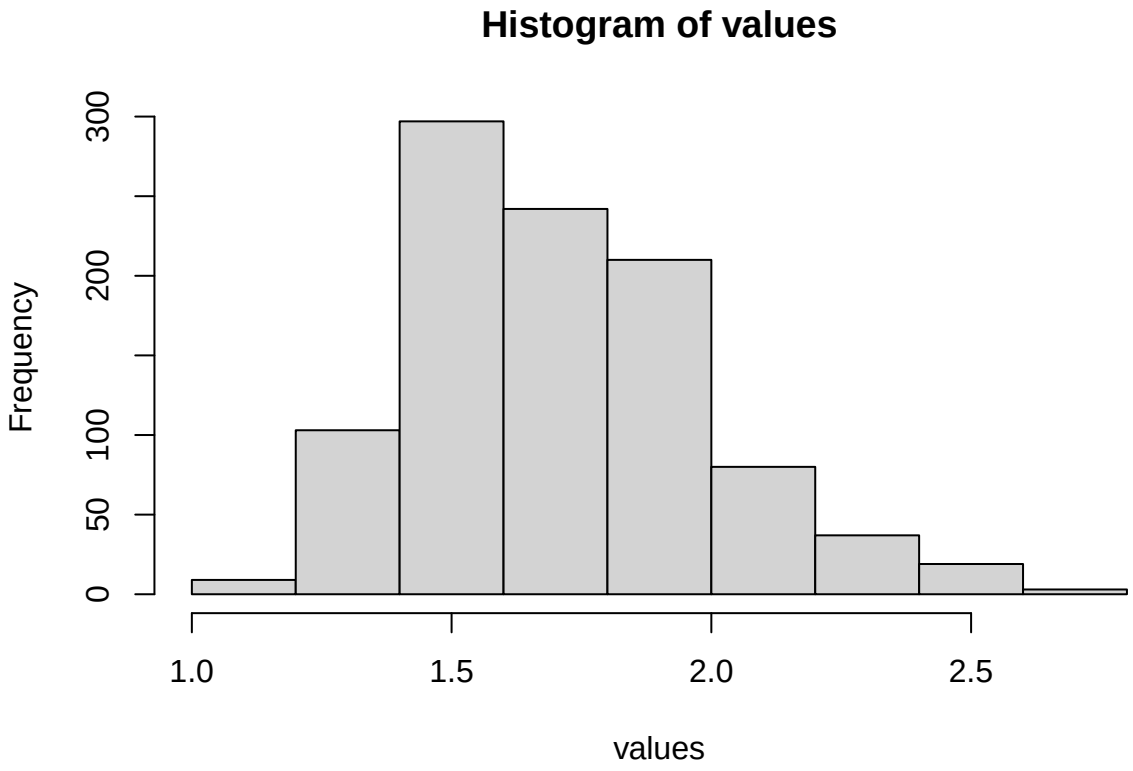

Rerun your replication, this time doing 1000 replications and storing the output of replicate() in a variable called **values**, and then generate a histogram of **values**.

```
values <- replicate(1000,mean(sample(urlComps$funding_rounds, 3, replace = FALSE), simplify = TRUE))
hist(values)
```




Repeat the replicated sampling, but this time, raise your sample size from 3 to 22. How does that affect your histogram? Explain in a comment.

```
values <- replicate(1000,mean(sample(urlComps$funding_rounds, 22, replace = FALSE), simplify = TRUE))
hist(values)
```




Explain in a comment below, the last three histograms, why do they look different?

```
#---The more data that you have and the more observations that you have the more likely
print("\n")
```

```
## [1] "\n"
```

```
#--- it is that you would have a normal distribution.  In the three examples above the
print("\n")
```

```
## [1] "\n"
```

```
#--- last example has the most data to work with and the histogram is moving closer to normal.
```