Intro to Data Science - HW 4

```
# Enter your name here: Rebecca Candee
```

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

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

(Chapters 4, 5, and 6 of the textbook)

Reminders of things to practice from previous weeks:
Descriptive statistics: mean( ) max( ) min( )
Sequence operator: : (For example, 1:4 is shorthand for 1, 2, 3, 4)
Create a function: myFunc <- function(myArg) { }
?command: Ask R for help with a command

This module: Sampling is a process of **drawing elements from a larger set**. In data science, when analysts work with data, they often work with a sample of the data, rather than all of the data (which we call the **population**), because of the expense of obtaining all of the data.

One must be careful, however, because **statistics from a sample rarely match the characteristics of the population**. The **goal of this homework** is to **sample from a data set several times and explore the meaning of the results**. Before you get started make sure to read Chapters 8-10 of *An Introduction to Data Science*. Don't forget your comments!

Part 1: Write a function to compute statistics for a vector of numeric values

A. Create a new function which takes a numeric vector as its input argument and returns a dataframe of statistics about that vector as the output. As a start, the dataframe should have the **min**, **mean**, and **max** of the vector. The function should be called **statsCalculator**:

```
statsCalculator <- function(x = c()){
return(data.frame(mean(x), min(x), max()))
}
```

B. Test your function by calling it with the numbers **one through ten**:

```
statsCalculator(1:10)
```

```
Warning message in max():
“no non-missing arguments to max; returning -Inf”
        A data.frame: 1 × 3
 mean.x.  min.x.  max..
   <dbl>   <int>  <dbl>
     5.5       1   -Inf
```

C. Enhance the statsCalculator() function to add the **median** and **standard deviation** to the returned dataframe.

```
statsCalculator <- function(x = c()){
return(data.frame(mean(x), min(x), max(x), median(x), sd(x)))
}
```

D. Retest your enhanced function by calling it with the numbers **one through ten**:

```
statsCalculator(1:10)
```

```
                    A data.frame: 1 × 5
 mean.x.  min.x.  max.x.  median.x.    sd.x.
   <dbl>   <int>   <int>      <dbl>    <dbl>
     5.5       1      10        5.5  3.02765
```

Part 2: Sample repeatedly from the New York State COVID Testing Dataset from HW 3

A. Load the dataset from the following URL: https://data-science-intro.s3.us-east-2.amazonaws.com/NYS_COVID_Testing.csv

```
library(tidyverse)
testDF <- read_csv("https://data-science-intro.s3.us-east-2.amazonaws.com/NYS_COVID_Testing.c
```

```
Rows: 7383 Columns: 5
```

```
Rows: 7383 Columns: 5
── Column specification ────────────────────────────────────────────────────────
Delimiter: ","
chr (3): TestDate, AgeGroup, AgeCategory
dbl (2): PositiveCases, TotalTests

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.
```

B. Use **head(testDF)** and **tail(testDF)** to show the data. Add a comment that describes what each variable in the data set contains.

```
head(testDF)  #head(testDF) contains the least amount of tests and positive cases
tail(testDF)
```

A tibble: 6 × 5

TestDate	AgeGroup	PositiveCases	TotalTests	AgeCategory
<chr>	<chr>	<dbl>	<dbl>	<chr>
3/2/2020	45 to 54	1	1	middle-aged_adults
3/3/2020	25 to 34	0	2	young_adults
3/3/2020	35 to 44	0	1	middle-aged_adults
3/3/2020	45 to 54	0	1	middle-aged_adults
3/3/2020	55 to 64	0	2	senior_citizens
3/3/2020	65 to 74	0	2	senior_citizens

A tibble: 6 × 5

TestDate	AgeGroup	PositiveCases	TotalTests	AgeCategory
<chr>	<chr>	<dbl>	<dbl>	<chr>
1/3/2022	5 to 19	9923	38977	children
1/3/2022	55 to 64	5739	27019	senior_citizens
1/3/2022	65 to 74	2759	14498	senior_citizens
1/3/2022	75 to 84	1141	6519	senior_citizens
1/3/2022	85 +	680	4028	senior_citizens
1/3/2022	< 1	717	2074	children

C. Sample ten observations from **testDF$TotalTests**.

```
sample(testDF$TotalTests, 10, replace=TRUE)
```

```
12636 · 23826 · 282 · 6247 · 3662 · 8080 · 31790 · 565 · 507 · 10056
```

D. Call your statsCalculator( ) function with a new sample of ten observations from **testDF$TotalTests**, where the sampling is done inside the **statsCalculator** function call.

```
statsCalculator(sample(testDF$TotalTests, 10, replace=TRUE))
```

```
                    A data.frame: 1 × 5
 mean.x.  min.x.  max.x.  median.x.     sd.x.
   <dbl>   <dbl>   <dbl>      <dbl>     <dbl>
 10366.6     893   27991     8077.5  8615.233
```

E. Now use the **mean()** function, with another sample done inside the mean function. Is the mean returned from the **statsCalculator** function the same as the mean returned from the mean function on this sample? Why or why not? Explain.

```
mean(sample(testDF$TotalTests, 10, replace=TRUE))
```

```
17788.3
```

F. Use the **replicate( )** function to repeat your sampling of **testDF$TotalTests** twenty times, with each sample calling **mean()** on ten observations. The first argument to **replicate( )** is the number of repeats you want. The second argument is the little chunk of code you want repeated.

```
replicate(20, mean(sample(testDF$TotalTests, 10, replace=TRUE)))
```

```
18895.2 · 13213.2 · 7382.9 · 14926.6 · 6630.8 · 15208.9 · 10893.6 · 13676.9 · 10824.8 · 7964.2 ·
8353.3 · 15781 · 6071.9 · 15727.2 · 10916.6 · 17608.6 · 15264.2 · 14534.2 · 9415.2 · 11659.3
```

G. Write a comment describing why every replication produces a different result.

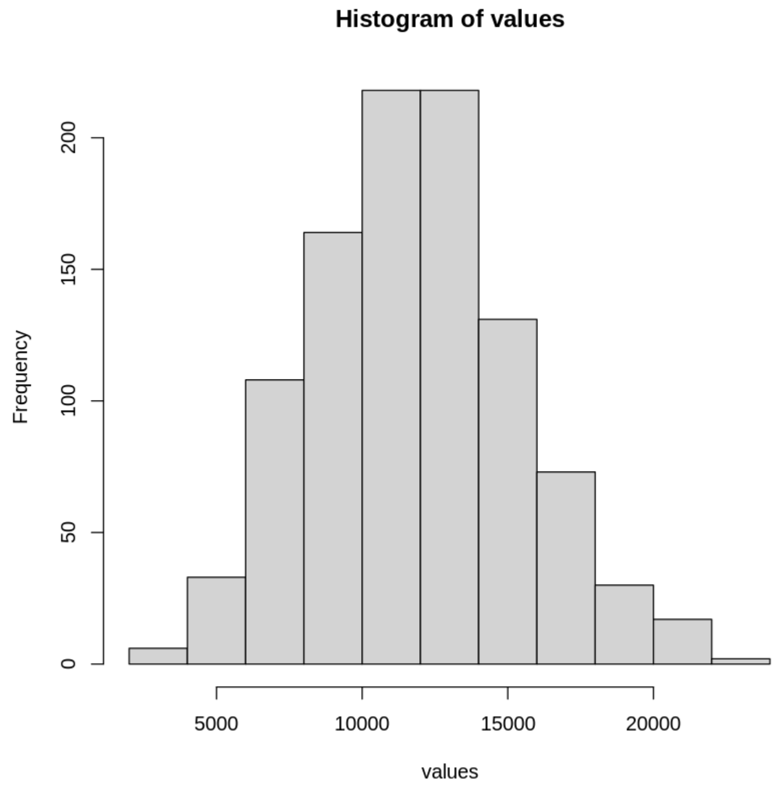
```
#every replication produces a different result. It pulls a new collection of numbers every ti
```

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

```
values <- replicate(1000, mean(sample(testDF$TotalTests, 10, replace=TRUE)))
```

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

```
hist(values)
```



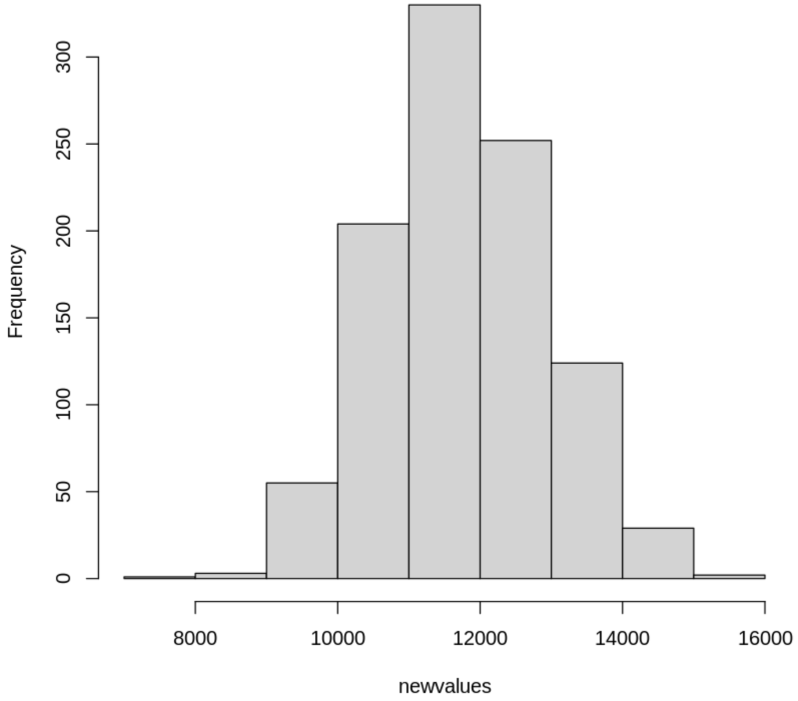
J. Repeat the replicated sampling, but this time, raise your sample size from **10 to 100**.

```
newvalues <- replicate(1000, mean(sample(testDF$TotalTests, 100, replace=TRUE)))
```

K. Compare the two histograms - why are they different? Explain in a comment.

```
hist(newvalues)
```

Histogram of newvalues



✓ 0s completed at 4:51 PM