Subsection 4

Useful R Functions

table()

table() is quick and dirty way to build a contingency table of the counts at each combination of factor levels

- Using the iris data set, how many of each type of Species are there?

```
> table(iris$Species)

    setosa versicolor  virginica
        50         50         50
```

- Using the ToothGrowth data set, what is the count of dose by supp?

```
> table(ToothGrowth$dose, ToothGrowth$supp)

      OJ VC
  0.5 10 10
  1   10 10
  2   10 10
```

attach() & detach()

- Don't use these!
- `attach()` allows you to semi-permanently attach a data frame, thereby not having to enclose statement in a `with()`
- Can you guess what issue might arise when attaching multiple data frames?
- Again...DON'T USE THESE!

unique()

- Identifies unique values in data, with duplicate elements removed

Cars have how many unique cylinder sizes in mtcars?

```
> unique(mtcars$cyl)

[1] 6 4 8
```

- A similar function, `dplyr::distinct()`, will remove all duplicate rows from a data frame

which()

- Returns the row number (index) of a subset of data

Which observations in mtcars have cars with 6 cylinders and 100 horsepower?

```
> which(mtcars$cyl == 6 & hp == 110)

[1] 1 2 4 # rows (observations) 1, 2 and 4
```

paste()

- Converts arguments to characters and then concatenates them

```
> library(magrittr)

> iris %>%
   dplyr::filter(Sepal.Length >= 7.7) %>%
      dplyr::mutate(paste("The ", Species, " has a petal length of ",
      Petal.Length, sep = ""))


[1] "The virginica has a petal length of 6.7"
[2] "The virginica has a petal length of 6.9"
[3] "The virginica has a petal length of 6.7"
[4] "The virginica has a petal length of 6.4"
[5] "The virginica has a petal length of 6.1"
```

seq()

- seq() and its variants are a quick way to generate sequences
- Common arguments include from, to, and by

```
> 1:10
 [1]  1  2  3  4  5  6  7  8  9 10

> seq(1:10)
 [1]  1  2  3  4  5  6  7  8  9 10

> seq(1, 10, 2) # the last argument is equivalent to by = 2
[1] 1 3 5 7 9

> seq(from = 1, by = 7, length.out =  10)
 [1]  1  8 15 22 29 36 43 50 57 64
```

seq_len() & seq_along()

- two stylized variants of seq() which are abbreviated versions of seq()and very fast results
- seq_len() only take one argument, length.out, and generates a sequence of integers beginning from one to length.out
- seq_along() is (ideally) passed a vector and generates a sequence of integers from 1 to length(myVector)

```
> seq_len(15)
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15

> (myVec <- c(LETTERS[4:8]))
[1] "D" "E" "F" "G" "H"

> seq_along(myVec)
[1] 1 2 3 4 5
```

rep()

- rep() replicates the value(s) passed to it *n* times
- Setting the each equal to a positive integer repeats each entry sequentially that integer number of times

```
> rep(LETTERS[2:4], times = 3)
[1] "B" "C" "D" "B" "C" "D" "B" "C" "D"

> rep(letters[2:4], each = 2)
[1] "b" "b" "c" "c" "d" "d"

> rep(letters[2:4], times = 2, each = 2)
[1] "b" "b" "c" "c" "d" "d" "b" "b" "c" "c" "d" "d"
```

any()

- Given a set of logical vectors, is at least one of the values TRUE?
- Returns a single logical value
- If relevant, can use na.rm option

```
> myAtomicVector <- c(1 ,2, 99.99, NA, sqrt(2))

> is.na(myAtomicVector)
[1] FALSE FALSE FALSE  TRUE FALSE

> any(is.na(myAtomicVector))
[1] TRUE
```

- Passing a data structure to anyNA() will return a single logical indicating whether or not NAs are in said data structure; this is an alternative to the equivalent compound statement any(is.na())

sample()

- sample() takes a sample of the specified size from the elements of a vector passed to it
- replace permits for sampling with or without replacement (default is FALSE, i.e., w/o replacement)
- n is non-negative, integral sample size

```
> sample(LETTERS, 5)
[1] "D" "R" "J" "I" "L"

> mtcars[sample(nrow(mtcars), 5), ]
                   mpg cyl  disp  hp drat    wt  qsec vs am gear carb
Chrysler Imperial 14.7   8 440.0 230 3.23 5.345 17.42  0  0    3    4
Merc 450SLC       15.2   8 275.8 180 3.07 3.780 18.00  0  0    3    3
Merc 280          19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
Merc 230          22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
Datsun 710        22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
```

source()

- In its simplest form, `source()` permits running an external R script while inside another R script

E.g. The file `firstFile.R` contains a single line of code:

```
titanicData <- read.csv("~/titanic.csv")
```

A second file, `secondFile.R` can call `firstFile.R` via the `source()` function to run all code in `firstFile.R`

```
source("~/firstFile.R")
```

which, in this case, would result in loading the csv file and storing it in a data frame `titanicData`, where code from `secondFile.R` could then manipulate said data frame