PSTAT 10: Homework 3

Solution Sketches

If we find these solutions posted to the likes of Chegg, CourseHero, etc., we will cancel all future quizzes and move the corresponding weights of your final grade solely to the final exam.

- Instructions
  - Exercise 1: Signing Off
  - Exercise 2: Determining Divisibility
  - Exercise 3: Loopty-Loop
  - Exercise 4: Vroom Vroom

Please Note:

- There are often many ways of solving each problem. Just because the solution sketches below are different than your own personal approach doesn't mean your approach is invalid!
- Do not post these solutions anywhere (e.g. Chegg, CourseHero, etc.). If we find these solutions posted online, we will change the grading structure of the course to put 100% weight on the final exam.

Instructions

- Remember to always copy the necessary files to your working directory; never edit and/or knit the files directly in the `10f22-content` folder.
- Rename your `.Rmd` file as `hw03-your-netid.Rmd`. For example, my Rmd script file will be `hw03-umaravat.Rmd`.
- Make sure to include your identifying details in the Rmd file.
- **Please keep in mind that there are 5 additional Multiple Choice questions that appear on Canvas**, in the submission portal for this homework. These questions are still considered a part of the homework, and you must complete them in order to be eligible to earn full points.
- When submitting to Canvas, follow the provided submission instructions carefully.

Exercise 1: Signing Off

In mathematics, the so-called **sign** function is defined as follows:

$$\text{sgn}(x) = \begin{cases} +1 & \text{if } x > 0 \\ 0 & \text{if } x = 0 \\ -1 & \text{if } x < 0 \end{cases}$$

So, for example, $\text{sgn}(-2039) = -1$ and $\text{sgn}(\pi) = 1$.

(a) There is no built-in function in `R` that computes the sign of a number. Let's fix that! Write a function called `sgn()` that computes the sign of a number `x`.

Solutions:

```
# Write your code here
```

```
sgn <- function(x) {
  if(x > 0) {
    return(1)
  } else if (x == 0) {
    return(0)
  } else {
    return(-1)
  }
}
```

(b) Test that your `sgn()` function from part (a) works by calling it on three different numbers.

Solutions:

```
# Write your code here
```

```
## ANSWERS MAY VARY
sgn(pi)
```

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

```
sgn(0)
```

```
## [1] 0
```

```
sgn(-pi)
```

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

(c) Look up the `Vectorize()` function in `R`. Describe (in words) what it does. *No Coding Required for this part.*

Solutions:

Replace this line of text with your description.

The main idea is that `Vectorize()` takes in a function and returns a vectorized version of the function. Recall that a vectorized function is one that applies to vectors element-wise; by default, nearly all `R` functions are vectorized.

(d) Now, create a vectorized version of your `sgn()` function. Call this new function `sgn_vec`, and check that `sgn_vec(c(-2, 0, 2))` returns `-1 0 1`.

Solutions:

```
# Write your code here
```

```
sgn_vec <- Vectorize(sgn)

sgn_vec(c(-2, 0, 2))
```

```
## [1] -1  0  1
```

Remark: The sign function is actually very important in mathematics! For example, the derivative of the absolute value function is often regarded to be the sign function. If you're curious, you can read up about the sign function via this Wikipedia page: https://en.wikipedia.org/wiki/Sign_function (https://en.wikipedia.org/wiki/Sign_function)

Exercise 2: Determining Divisibility

(a) Recall the modulo operator in `R`, `%%`. In words, `x %% y` computes the remainder when `x` is divided by `y`; for example, `3 %% 2` returns `1`, since there is a remainder of 1 when 3 is divided by 2 ($3 = 1 \cdot 2 + \boxed{1}$). **Look up the help file for `%%`.** In case the usual way we call for help isn't working, take a look at this post: https://stat.ethz.ch/pipermail/r-help/2011-July/283608.html (https://stat.ethz.ch/pipermail/r-help/2011-July/283608.html)

Solutions:

```
# Write your code here
```

```
?"%%"
```

REMARK: It seems many people missed what we were asking for in this question. The goal was for you to realize that when looking up the help files for special characters (e.g. `%%`, or `==`, etc.) you need to enclose the character in quotation marks.

(b) The modulo operator is very useful for determining divisibility. What is true of `x %% y` if `x` is divisible by `y`? *No coding required for this question*.

Solutions:

Replace this line of text with your answer

`x %% y` must be 0 if `x` is divisible by `y`.

(c) In `R`, we can set default values for variables in user-defined functions in the following way:

```
<fnt. name> <- function(x = <default value>) {
  <code>
}
```

For example, if the function `foo()` is defined as:

```
foo <- function(y = 4) {
  <code>
}
```

then the variable `y` has a default value of 4. Give an example of a built-in function in `R` that has one or more variables with default values.

Solutions:

There are lots of examples! Two such examples are `mean()` and `read.csv()`.

(d) Now, write a function `is_divisible_by()` that takes in two arguments `x` and `y` that returns a value of `TRUE` if `x` is divisible by `y` and returns a value of `FALSE` if `x` is not divisible by `y`. For example, `is_divisible_by(4, 2)` should return `TRUE` whereas `is_divisible_by(3, 2)` should return `FALSE`. **Set the default value of `y` to be 2**, and check that `is_divisible_by(4)` returns `TRUE`. Make sure to test your function on several (at least 3) different inputs!

Solutions:

```
# Write your code here
```

```
is_divisible_by <- function(x, y = 2) {
  if(x %% y == 0) {
    return(TRUE)
  } else {
    return(FALSE)
  }
}

is_divisible_by(4, 2)
```

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

```
is_divisible_by(3, 2)
```

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

```
is_divisible_by(4)
```

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

(e) Write a function `is_even()` that checks whether a given number `x` is even or not. In your definition of `is_even()`, be sure to use `is_divisible_by()`. **Hint:** Your default value in part (d) should help save you a lot of work for this part! Test your function on at least 2 different outputs.

Solutions:

```
# Write your code here
```

```
is_even <- function(x) {
  return(is_divisible_by(x))
}

is_even(2)
```

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

Exercise 3: Loopty-Loop

We'd like to use loops to count the number of multiples of three between 1 and 100. For this exercise, you can (if you like) use your `is_divisible_by()` function from Exercise 2 above (but you do not need to). We will do this in three ways:

(a) Write a `for` loop to count the number of multiples in the vector `1:100`.

Solutions:

```
# Write your code here
```

```
x <- 1:100
count <- 0

for(i in x){
  if(is_divisible_by(i, 3)) {
    count <- count + 1
  }
}

count
```

```
## [1] 33
```

(b) Write a `while` loop to count the number of multiples in the vector `1:100`.

Solutions:

```
# Write your code here
```

```
count <- 0
i <- 1

while(i <= 100){
  if(is_divisible_by(i, 3)) {
    count <- count + 1
  }
  i <- i + 1
}

count
```

```
## [1] 33
```

(c) Write a `repeat` loop to count the number of multiples in the vector `1:100`.

Solutions:

```
# Write your code here
```

```
count <- 0
i <- 1

repeat{
  if(is_divisible_by(i, 3)) {
    count <- count + 1
  }
  i <- i + 1
  if(i > 100) {
    break
  }
}

count
```

```
## [1] 33
```

Exercise 4: Vroom Vroom

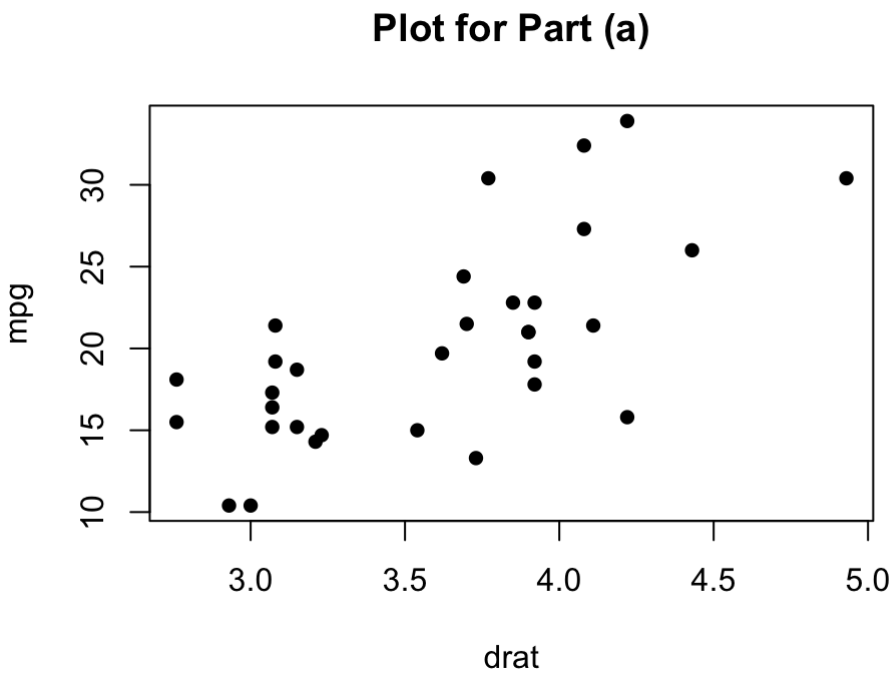
Using the `mtcars` data frame, and remembering to include axis labels and meaningful titles (and legends, wherever necessary),

(a) Plot Miles/(US) gallon versus Rear axle ratio using the `plot()` function. As a note: the language we use in Data Science is to say "plot `y` vs `x`," where `y` is on the $y-$axis and `x` is on the $x-$axis. Use a different plotting character (`pch` value) than the default.

Solutions:

```
# Write your code here
```

```
plot(mtcars$drat, mtcars$mpg,
     pch = 16,
     xlab = "drat", ylab = "mpg",
     main = "Plot for Part (a)")
```

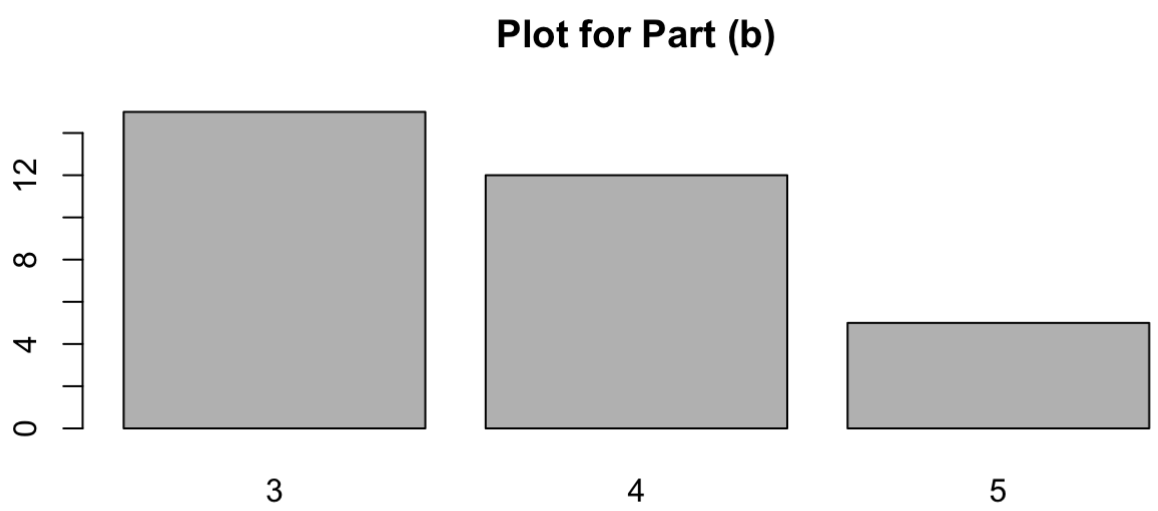


(b) Construct a barplot of the number of forward gears. Remember that you may need to use the `table()` function in conjunction with the `barplot()` function to be able to produce a meaningful plot.

Solutions:

```
# Write your code here
```

```
barplot(table(mtcars$gear),
     main = "Plot for Part (b)")
```



(c) Write code to return a numeric summary of the mtcars dataset, and interpret your results.

Solutions:

```
# Write your code here
```

```
summary(mtcars)
```

```
##       mpg             cyl             disp             hp       
##  Min.   :10.40   Min.   :4.000   Min.   : 71.1   Min.   : 52.0  
##  1st Qu.:15.43   1st Qu.:4.000   1st Qu.:120.8   1st Qu.: 96.5  
##  Median :19.20   Median :6.000   Median :196.3   Median :123.0  
##  Mean   :20.09   Mean   :6.188   Mean   :230.7   Mean   :146.7  
##  3rd Qu.:22.80   3rd Qu.:8.000   3rd Qu.:326.0   3rd Qu.:180.0  
##  Max.   :33.90   Max.   :8.000   Max.   :472.0   Max.   :335.0  
##       drat             wt             qsec             vs        
##  Min.   :2.760   Min.   :1.513   Min.   :14.50   Min.   :0.0000  
##  1st Qu.:3.080   1st Qu.:2.581   1st Qu.:16.89   1st Qu.:0.0000  
##  Median :3.695   Median :3.325   Median :17.71   Median :0.0000  
##  Mean   :3.597   Mean   :3.217   Mean   :17.85   Mean   :0.4375  
##  3rd Qu.:3.920   3rd Qu.:3.610   3rd Qu.:18.90   3rd Qu.:1.0000  
##  Max.   :4.930   Max.   :5.424   Max.   :22.90   Max.   :1.0000  
##        am              gear            carb      
##  Min.   :0.0000   Min.   :3.000   Min.   :1.000  
##  1st Qu.:0.0000   1st Qu.:3.000   1st Qu.:2.000  
##  Median :0.0000   Median :4.000   Median :2.000  
##  Mean   :0.4062   Mean   :3.688   Mean   :2.812  
##  3rd Qu.:1.0000   3rd Qu.:4.000   3rd Qu.:4.000  
##  Max.   :1.0000   Max.   :5.000   Max.   :8.000
```

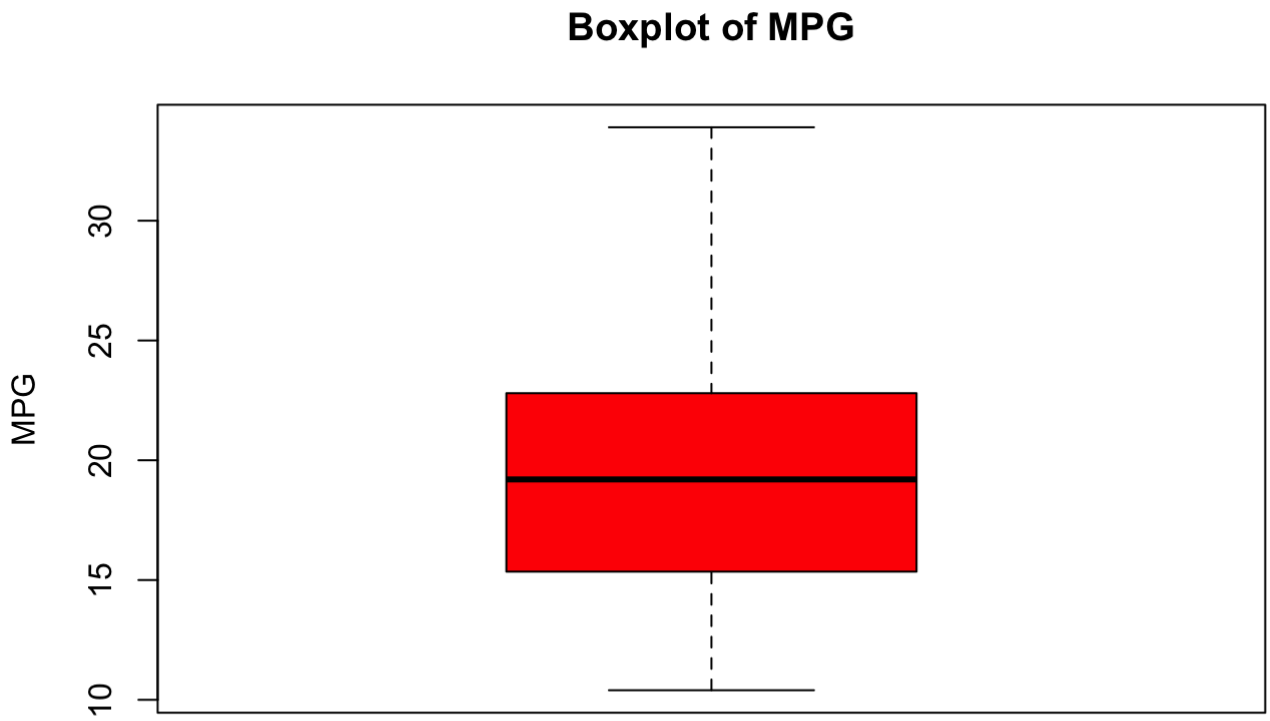
This is a fairly open-ended question, and was designed to get you actually *thinking* about the data you are analyzing, rather than just blindinly running it through `R`! As such, answers may vary; some examples may include comparisons of what students notice within each each variable (i.e., how close or far median is from the mean), or what values they notice of each variable (i.e., the mean of `hp` is `146.7`, etc…).

(d) Construct a boxplot of the miles per gallon.

Solutions:

```
# Write your code here
```

```
boxplot(mtcars$mpg,
        main ="Boxplot of MPG",
        ylab = "MPG", col = "Red")
```



(e) On one chart, produce side-by-side boxplots to compare miles per gallon to the number of gears. Use a different color for each category.

Solutions:

```
# Write your code here
```

```
boxplot(mtcars$mpg ~ mtcars$gear,
        xlab = "Number of Gears", ylab = "mpg",
        col = c(1, 2, 3),
        main = "Plot for part (e)")
```