Homework 1

Zachary Lazerick

2023-01-31

Use R Markdown to write up your solutions to the following problems (See Q10). After you "knit" the Markdown file, you can upload the html (or pdf) file containing your solutions to Canvas.

1) Write a function that reverses a list of numbers. Test out your function by generating a random set of integers using sample(1:10, size = 10) and then reversing the list you get.

```
## Code for 1

Reverse <- function(x) {
  b <- rep(0, length(x))
  for (i in 1:length(x)) {
    b[i] <- x[(length(x)+1) - i]
  }
  return(b)
}

## Test for 1
x <- sample(1:10, size = 10)
print(x)
```

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

```
Reverse(x)
```

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

2) Write a function that merges two sorted lists into a new sorted list. Use runif(10) to generate your two lists and the sort()function to initially order your two lists.

```
## Code for 2

MergeSort <- function(x, y) {
  i = 1; j = 1; k = 1;
  z <- rep(0, length(x) + length(y))
  while (k <= length(x) + length(y)) {
    if (i <= length(x) & j <= length(y)) {
      if(x[i] >= y[j]) {
        z[k] <- y[j]
        j <- j + 1
        k <- k + 1
      }
      else {
        z[k] <- x[i]
        i <- i + 1
        k <- k + 1
      }
    }
    else if (i <= length(x) & j > length(y)) {
      z[k] <- x[i]
      i <- i + 1
      k <- k + 1
      }
    else if (i > length(x) & j <= length(y)) {
      z[k] <- y[j]
      j <- j + 1
      k <- k + 1
      }
    }
  return(z)
}

## Test for 2

x <- sort(runif(10)); y <- sort(runif(10))

MergeSort(x, y)
```

```
##  [1] 0.04009443 0.12903764 0.19751395 0.28057312 0.32012124 0.34128769
##  [7] 0.35604209 0.42967139 0.43029694 0.47165937 0.50816327 0.54749947
## [13] 0.60976918 0.66737853 0.68759883 0.69719390 0.70703121 0.78682182
## [19] 0.81213513 0.84189768
```

3) Write a program that combines two lists by alternating the elements in the two lists. For example: [1, 2, 3] and [a, b, c] becomes [1, a, 2, b, 3, c]. If the two lists are of different size, the remaining elements of the longer list can be appended to the end of your new list. Test out your program on [1, 2, 3] and [4, 5, 6, 7, 8] or something similar.

```
## Code for 3

ZipperSort <- function(x, y) {
  i = 1; j = 1; k = 1;
  z <- rep(0, length(x) + length(y))
  while (k <= length(x) + length(y)) {
    if (i <= length(x) & j <= length(y)) {
      if(i <= j) {
        z[k] <- x[i]
        i <- i + 1
        k <- k + 1
      }
      else {
        z[k] <- y[j]
        j <- j + 1
        k <- k + 1
      }
    }
    else if (i <= length(x) & j > length(y)) {
      z[k] <- x[i]
      i <- i + 1
      k <- k + 1
    }
    else if (i > length(x) & j <= length(y)) {
      z[k] <- y[j]
      j <- j + 1
      k <- k + 1
    }
  }
  return(z)
}

## Test for 3

x <- c(1, 2, 3)
y <- c(4, 5, 6, 7, 8)

ZipperSort(x, y)
```

```
## [1] 1 4 2 5 3 6 7 8
```

4) Write a program that calculates the sum of the number from 1 to n such that only numbers which are multiples of 3 or 5 are considered in the sum. Use n = 20, 50, and 100. Hint: You can use x%%y for x mod y. The “mod” function returns the remainder when you divide y into x. So 10 mod 7 = 3 and 42 mod 5 = 2.

```
## Code for 4

nSum <- function(x) {
  i = 1;
  sum = 0
  while (i <= x) {
    if (i %% 3 == 0) {
      sum = sum + i
      i <- i + 1
    }
    else if (i %% 5 == 0) {
      sum = sum + i
      i <- i + 1
    }
    else {
      i <- i + 1
    }
  }
  return(sum)
}

## Test for 4

nSum(20)
```

```
## [1] 98
```

```
nSum(50)
```

```
## [1] 593
```

```
nSum(100)
```

```
## [1] 2418
```

5) Given a pair of numbers, determine the set of common divisors. For example, the numbers 4 and 10 have 1 and 2 as common divisors. Test out your program using the numbers 125 and 169 as well as 120 and 160. Print out the common divisors for each pair of numbers. If the numbers are relatively prime, say so!

```r
##Code for 5

CommonDivisors <- function(x, y) {
  z <- c(1)
  i <- 2
  while(i <= x & i <= y) {
    if (x %% i == 0 & y %% i == 0) {
      z <- append(z, i)
      i <- i + 1
    }
    else {
      i <- i + 1
    }
  }
  if (length(z) != 1) {
    return(cat("The common factors of", x, "and", y, "are", z))
  }
  else {
    return(cat(x, "and", y, "are relatively prime."))
  }
}

## Test for 5

CommonDivisors(125, 169)
```

```
## 125 and 169 are relatively prime.
```

```r
CommonDivisors(120, 160)
```

```
## The common factors of 120 and 160 are 1 2 4 5 8 10 20 40
```

6) Write a function that takes a number and returns a list of its digits. Hint: How might you get R to return the ones digit of a number? This is related to questions 4 and 5, although maybe not in an obvious way.

```
## Code for 6

DigitFinder <- function(x) {
  z <- c()
  while(x > 0) {
    z <- append(z, x %% 10)
    x <- floor(x / 10)
  }
  z <- Reverse(z)
  return(z)
}

## Test for 6

DigitFinder(325)
```

```
## [1] 3 2 5
```

```
DigitFinder(123456)
```

```
## [1] 1 2 3 4 5 6
```

7) A positive integer is called a palindrome if its representation in the decimal system is the same when read from left to right and from right to left. Modify the function from the previous question to check whether or not a positive integer is a palindrome. Then, for a given positive integer K of not more than 1000000, output the value of the smallest palindrome larger than K. Use K = 45678 and K = 9876. Note that numbers are always displayed without leading zeros. Hint: You may find the function that you created in question #1 useful here.

```
## Code for 7

Palindrome <- function(x) {
  read <- x
  while (x <= 1000000) {
    if(identical(DigitFinder(x), Reverse(DigitFinder(x)))) {
      return(cat(x, "is the closet palindrome greater than", read))
    }
    else {
      x <- x + 1
    }
  }
}


## Test for 7

Palindrome(45678)
```

```
## 45754 is the closet palindrome greater than 45678
```

```
Palindrome(9876)
```

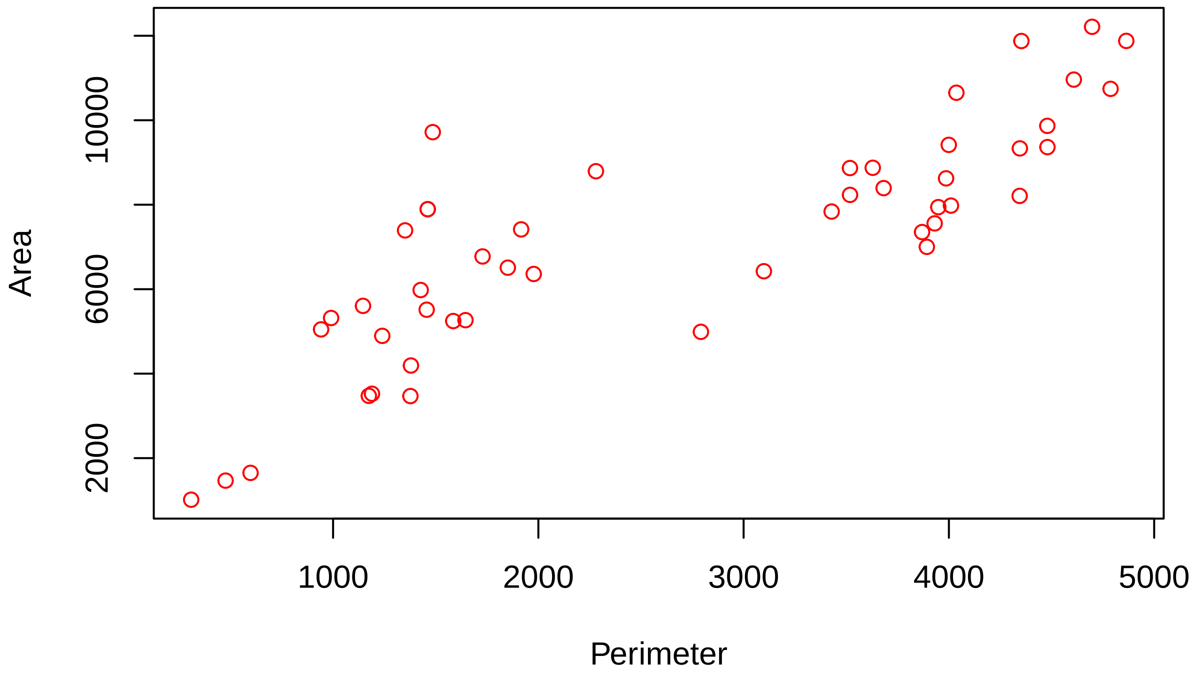
```
## 9889 is the closet palindrome greater than 9876
```

8) [Bonus] Write a program that outputs all possibilities to put + or − or nothing between the numbers 1, 2, . . . , 9 (in this order) such that the result is 100. For example $1 + 2 + 3 - 4 + 5 + 6 + 78 + 9 = 100$ [Note: This problem is kind of complicated. . . I think that I have figured out a good solution though and may provide a hint at some point. Twelve such sequences exist, including the example given above.]

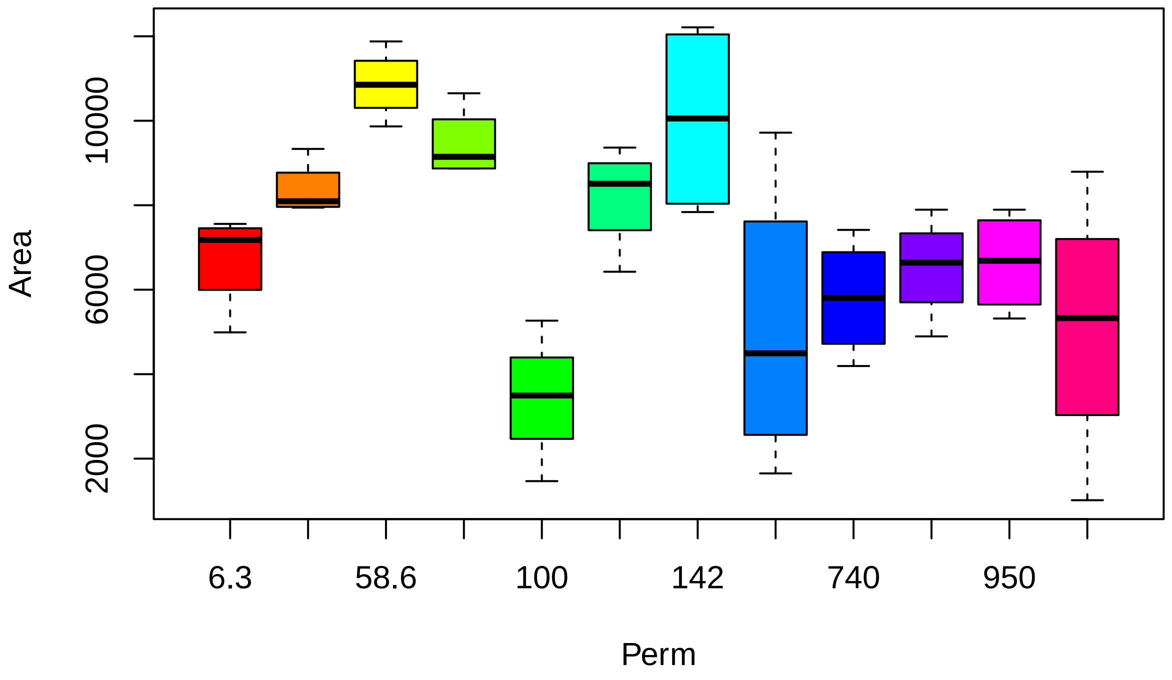
Graphing:

9) Go to: http://www.harding.edu/fmccown/r/. Try and reproduce some of the graphs. Initially, you can copy the examples verbatim, but try changing color schemes, axis labels, titles, measurement scales, etc. The goal is to get you comfortable creating graphs. Type “data()” into the R console (without the quotes) to see a list of all pre-installed data sets, or you can use the Class_Data data set that I have uploaded to Canvas. This data comes from my STAT 220 class in the fall. Choose one of these data sets and make three different graphs (e.g. bar chart, line chart, pie chart, etc.) based on what you learned from the website and what is most appropriate for the data set you selected.

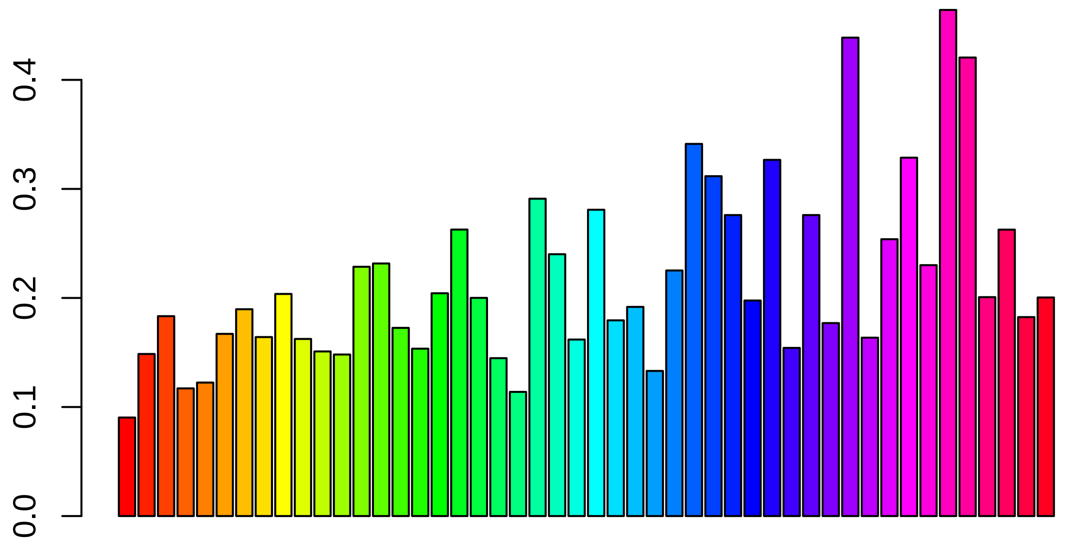
```
## Code for 9

## Building a Graph

rock <- rock ## Load in built-in Data set 'Rock'
attach(rock)

## Line plot
plot(peri, area, type = "p", col = 'red',
     xlab = "Perimeter", ylab = "Area")
```



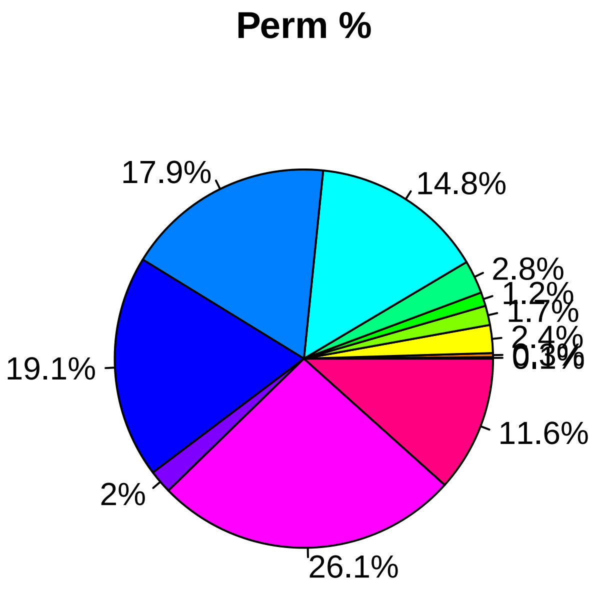
```r
## Boxplot

boxplot(area~perm, col = rainbow(12),
        xlab = "Perm", ylab = "Area")
```



```r
## Bar Chart
barplot(shape, col = rainbow(length(shape)))
```



```r
##Pie Chart

rock_labels <- c(6.3, 17.1, 119, 82.4, 58.6,
                 142, 740, 890, 950, 100, 1300, 580)

rock_labels2 <- round(rock_labels/sum(rock_labels) * 100, digits = 1)

rock_labels2 <- paste(rock_labels2, "%", sep = "")

pie(rock_labels, main = "Perm %", col = rainbow(length(rock_labels)),
    labels = rock_labels2)
```


Perm %