Homework 3

Eric Zou

Honor code:

"I have neither given nor received unauthorized assistance on this assignment." EZ

I receive help from " " and give help to " ".

Problem 1

Implement a function that sorts a dataframe based on the alphabetical order of its row names. The function should return a new dataframe where the row names are arranged in alphabetical order. Use the `mtcars` dataset to demonstrate your function.

Hints: 1. The `rownames()` function can be employed to retrieve the row names. 2. The `order()` function is useful for obtaining the alphabetical sequence.

```
dataAlph = function(df){
  orderedRows = order(rownames(df))
  print(orderedRows)
  df = df[orderedRows,]
  return(df)
}
```

```
df = mtcars
dataAlph(df)
```

```
##  [1] 23 15 24 17  3 22  7 30 18 26 29 19  4  5 16 28 31  1  2  9  8 10 11 12 13
## [26] 14 25 27 20 21  6 32
```

```
##                      mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## AMC Javelin         15.2   8 304.0 150 3.15 3.435 17.30  0  0    3    2
## Cadillac Fleetwood  10.4   8 472.0 205 2.93 5.250 17.98  0  0    3    4
## Camaro Z28          13.3   8 350.0 245 3.73 3.840 15.41  0  0    3    4
## Chrysler Imperial   14.7   8 440.0 230 3.23 5.345 17.42  0  0    3    4
## Datsun 710          22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
## Dodge Challenger    15.5   8 318.0 150 2.76 3.520 16.87  0  0    3    2
## Duster 360          14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
## Ferrari Dino        19.7   6 145.0 175 3.62 2.770 15.50  0  1    5    6
## Fiat 128            32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
## Fiat X1-9           27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1
## Ford Pantera L      15.8   8 351.0 264 4.22 3.170 14.50  0  1    5    4
## Honda Civic         30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2
## Hornet 4 Drive      21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout   18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
## Lincoln Continental 10.4   8 460.0 215 3.00 5.424 17.82  0  0    3    4
## Lotus Europa        30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
## Maserati Bora       15.0   8 301.0 335 3.54 3.570 14.60  0  1    5    8
```

```
## Mazda RX4           21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag       21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
## Merc 230            22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
## Merc 240D           24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
## Merc 280            19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
## Merc 280C           17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
## Merc 450SE          16.4   8 275.8 180 3.07 4.070 17.40  0  0    3    3
## Merc 450SL          17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3
## Merc 450SLC         15.2   8 275.8 180 3.07 3.780 18.00  0  0    3    3
## Pontiac Firebird    19.2   8 400.0 175 3.08 3.845 17.05  0  0    3    2
## Porsche 914-2       26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2
## Toyota Corolla      33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
## Toyota Corona       21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1
## Valiant             18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
## Volvo 142E          21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2
```

Problem 2

Initiate an empty vector named "mpg_discrete" with a length corresponding to the number of rows in the mtcars dataset. Construct a for loop that traverses from 1 to the total number of rows in the mtcars dataset. If the mpg value at position i is greater than or equal to the average mpg, set mpg_discrete[i] to 0. If the mpg value at position i is below the average, set mpg_discrete[i] to 1.

```
meanMPG = mean(mtcars$mpg)
mpg_discrete = vector(length = nrow(mtcars))
for(x in 1:nrow(mtcars)){
  if(mtcars$mpg[x]>meanMPG){
    mpg_discrete[x] = 0
  }
  if(mtcars$mpg[x]<meanMPG){
    mpg_discrete[x] = 1
  }
}
mpg_discrete
```

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

Problem 3

Utilize the apply function to generate the same output as in Problem 2 without resorting to a for loop.

Hint: Construct a custom function that takes a row vector as input and returns the transformed mpg value, either 0 or 1.

```
meanMPG = mean(mtcars$mpg)
mpg_discrete = vector(length = nrow(mtcars))
setBits = function(x){
  return(as.numeric(x>meanMPG))
}
apply(mtcars[1],2,setBits)
```

```
##       mpg
##  [1,]   1
##  [2,]   1
##  [3,]   1
```

```
##  [4,]   1
##  [5,]   0
##  [6,]   0
##  [7,]   0
##  [8,]   1
##  [9,]   1
## [10,]   0
## [11,]   0
## [12,]   0
## [13,]   0
## [14,]   0
## [15,]   0
## [16,]   0
## [17,]   0
## [18,]   1
## [19,]   1
## [20,]   1
## [21,]   1
## [22,]   0
## [23,]   0
## [24,]   0
## [25,]   0
## [26,]   1
## [27,]   1
## [28,]   1
## [29,]   0
## [30,]   0
## [31,]   0
## [32,]   1
```

Problem 4

Write a while loop to calculate pi with accuracy 0.001 i.e. |yourestimatepi - pi|<0.001 where true pi is a default variable in R. Use Gregory-Leibniz series to estimate pi, i.e., est_pi = 4[1 - 1/3 + 1/5 -1/7 + 1/9 ...].

Hint: Within the while loop, set a condition based on the desired accuracy. Allow the loop to progress with an incrementing variable 'n' indicating the number of terms used in est_pi, and terminate the loop once the target accuracy has been met.

```
gregLeiPi = function(acc){
  i = 0
  n = 4
  denom = -3
  while(abs(pi-4*n)>acc){
    n = n+ 1/denom
    denom = (abs(denom)+2)*-1
    i= i+1
  }
  return(i)
}
gregLeiPi(0.001)
```

```
## [1] 642
```

Problem 5

Initialize a random seed using the set.seed() function. Employ the Monte Carlo method, as outlined in Lecture Note 6, to approximate the value of pi to the same level of precision as described in Problem 4. Compare the time taken to compute pi using both the Gregory-Leibniz series (from Problem 4) and the Monte Carlo method by leveraging the system.time() function.

Bonus: Earn an additional 2 bonus points for the assignment by presenting one or more alternative techniques for estimating pi and contrasting their computational times.

```
monteCarloPi = function(acc){
  set.seed(123)
  count = 0
  n = 1
  while(abs(pi-4*sqrt(1-n**2))>acc){
    n = runif(1, 0.0,1.0)
    count = count+1
  }
  return(count)
}
```

Bonus: calculate pi using Nilakantha series.

```
nilakanPi = function(acc){
  i = 0
  estim = 3
  denom = 2
  polarity = 1
  while(abs(pi-estim)>acc){
    estim = estim+ polarity*4/(denom*(denom+1)*(denom+2))
    denom = denom+2
    i= i+1
    polarity = polarity*(-1)
  }
  return(i)
}
nilakanPi(0.001)
```

```
## [1] 6
```

Comparing

```
acc = 0.001
system.time(monteCarloPi(acc))
```

```
##    user  system elapsed
##   0.006   0.000   0.005
```

```
system.time(gregLeiPi(acc))
```

```
##    user  system elapsed
##       0       0       0
```

```
system.time(nilakanPi(acc))
```

```
##    user  system elapsed
##       0       0       0
```

Monte carlo seems to be the fastest, by a small amount at first but if I turn up the accuracy to 0.0001 the difference is many times faster.

Problem 6

Create a user-defined function to compute the sum of two vectors. The function should return an error message in the following situations:

1. The two vectors are of different lengths.
2. Either of the vectors contains non-numerical values.

```
myvecsum = function(a,b){
  if(length(a)!=length(b)){
    stop("Lengths unequal!")
  }
  if(!(is.numeric(a[1])&&is.numeric(b[1]))){
    stop("Vectors not numeric!")
  }
  newVec= vector(length = length(a))
  for(x in 1:length(a)){
    newVec[x] = a[x]+b[x]
  }
  return(newVec)
}
```

Verify the functionality of your function using these test cases:

```
try(myvecsum(c(1,2), c(2,3,4)))
```

```
## Error in myvecsum(c(1, 2), c(2, 3, 4)) : Lengths unequal!
```

```
try(myvecsum(c('a','b'), c(1,2)))
```

```
## Error in myvecsum(c("a", "b"), c(1, 2)) : Vectors not numeric!
```

```
myvecsum(c(1,2),c(2,3))
```

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