KATHLEEN ZHEN

999210972

STA 141

HOMEWORK 1

The codes and results derived by using these codes constitute my own work. I have consulted the following resources regarding this assignment: TA Haoran Li helped with number 1 during office hours on 10/6/2016

NUMBER 1

Script name: HW 1 PROB 1.R

Function name: markov_chain()

```
markov_chain(c(0.5, 0.5, 0.5, 0.5), 10, 0)
```

```
output: 0 0 1 1 1 0 0 1 1 0
```

```
markov_chain()
```

```
output: 0 0 1 1 1 0 0 0 1 1 0 0 1 1 1 0 1 0 0 0 0 0 1 1 0 0 0 1 0 1 0 1 0 0 1 1 0
```

In this question, I assumed that the function takes in 3 arguments: probability matrix, the length of the sequence, and the initial value. My function is called markov_chain() and takes in the above 3 arguments respectively. For the first parameter, probability matrix, please enter the matrix in vector form. My function will transform that to a matrix automatically. The sequence length must be greater than 1 since the initial value will count as first position of the sequence. If the user doesn't input anything and simply calls the function, the probability matrix will automatically be set to 0.5's. The length of the sequence will be a random number between 5 and 50. Lastly, the initial value is set at 0.

In my code, I first transfer the vector into a matrix. After that I used a for loop to iterate through the length of sequence and recursively, it uses the previous sequence number to predict the next number with rbinom.

NUMBER 2

Script name: HW 1 PROB 2.R

Function name: sequence_location()

```
a = c(1,1,1,1,0,0,0,0)

b = 3

store_sequence = sequence_location(a, b)
```

output:

```
"Start 0 locations: "
"5" "6"
```

```
"Start 1 locations: "
"1" "2"
```

The code prints out the locations but it’s also stored in a matrix.

If you call store_sequence, it will show the following matrix:

```
     start0 start1
[1,]      5      1
[2,]      6      2
```

In this problem, I named my function sequence_location() in HW 1 PROB 2.R. This function takes in two arguments: sequence and the length of runs, respectively. An example of how to use my function is shown above.

If you decide to call sequence_location() without any arguments, it will generate a random sequence of 0’s and 1’s with a length of 20. The sequence run length will be a random number between 1 and 5.

I made a couple assumptions in this question. I counted ALL the subsequences that you can find sequence. In the above example [1, 1, 1, 1, 0, 0, 0, 0] k = 3, the starting location of 1’s will be 1 and 2 because when you start and 1 or 2, you can still get a length of 3 1’s. If you remove position 1, position 2 is still a run of 3, therefore, I counted it.

In my code, I used a for loop to loop through the length of the inputted sequence. I then used another loop for the length of run to find locations of the runs. If I can find a sequence of 3, I would store the starting location in a vector. After that, my first loop will iterate through the rest of the sequence repeating the process. This process is repeated for 0 and 1’s separately. At the end of my code, I printed the locations and stores it in a matrix. I printed the locations because if you just call the function without using a variable, you can still view the locations. If you call the function to be a variable, the variable will be in a matrix. Since the vectors are not the same length because the runs of 0 and 1 can be different lengths, the vector with the shorter length will report the other values as NA.

<u>**NUMBER 3**</u>

Script name: HW 1 PROB 3.R

Function name: startend()

```
ss = c(1,0,1,0,0,0,1,0,1,1,1,1,0)
st = c(0,0)
se = c(1,1)
startend(ss, st, se)
```

Output:

```
[1] "Start motif locations: 4"
[1] "End motif starting locations:"
[1] "9"  "10" "11"
[1] "Start motif locations: 5"
[1] "End motif starting locations:"
[1] "9"  "10" "11"

     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    4    4    4    5    5    5
[2,]    9   10   11    9   10   11
```

In this problem, my function is named startend(). It can take 4 arguments: sequence, start motif, end motif, and length of motif. An example of how to call the function is show above. In the above example, I only had three arguments and chose not to have the 4 one which is the length of the motif. If the length of motif isn't specified, the code assumes that you want to know the start and end locations of all lengths. In my function, motifs can overlap. For example, 0 0 starts at position 4 and 5 and I assumed that those are two different starting locations because they are different subsequences if a user were to extract just one motif from the sequence.

If you decide to call the function without any arguments, it will generate a random sequence of length 20, a random start motif of length 3, a random start motif of length 3, and no length for the length of motif. The length of motif is used mainly for question 4.

In my code, I used a for loop to iterate through the sequence. I used another for loop and if statements to check if the iterations of the sequence equal the start or end motif. If they match, I stored the start and positions in a vector. At the end of the function, I printed out the start motif location and all the possible end possible locations. The answers are stored in a matrix. Each pair means it is a complete motif.

NUMBER 4

Script name: HW 1 PROB 4.R

PART I

A)

In this part, I simply called my functions markov_chain() with the 0.5 probability matrix, length of 10000, and starting number of 0. Afterwards, I input the markov chain into sequence_location() with lengths 1 – 10. I did a for loop in this script to iterate the lengths. I DID NOT incorporate this into my function because I did not want the user to input a certain length that she wants but then gets an output all the previous lengths as well.

I did not include all the outputs from lengths 1 to 10 because it would be really long and redundant on this write up. Below is the output for runs of length 10:

```
[1] "Start 0 locations: "
[1] "1877" "1878" "1879" "1880" "1881" "4529" "5042" "5654" "5835"
[1] "Start 1 locations: "
[1] "790"  "7917" "7918"
```

B)

In this part, I simply called my functions markov_chain() and sequence_location() like the above problem but with matrix B.

Since each output for the lengths are super long, I only included length 10 in this writeup.

Since the output of this part is really big, the following a snippet of the output for length 10:

```
[1] "Start 0 locations: "
  [1] "24"   "44"   "126"  "127"  "128"  "145"  "146"  "253"  "254"  "255"  "256"  "281"  "282"
"283"
[1] "Start 1 locations: "
   [1] "34"   "54"   "55"   "56"   "57"   "58"   "59"   "60"   "61"   "62"   "63"   "64"   "65"
```

In matrix A, the probabilities are all 0.5 and thus the chances of the next number being 0 or 1 when the previous number was 0 or 1, is 0.5. With this in mind, each number is as equal as the other to be chosen. When the lengths get bigger, the chances of that length existing diminishes. When the lengths is small like 2 or 3, majority of the positions show up. For matrix A, regardless of the length, the number of positions that start with 0 or 1 should be roughly equal or symmetric. In matrix B, the probabilities are more favored for the next number being 0 if the previous number was 0 and for the next number being 1 if the previous number is 1. For this data, the number of positions also decreased as the lengths got bigger but the number of starting positions increased a lot in comparison to matrix A. For matrix B, I noticed that number of starting positions increased compared to matrix A because if the previous number is 0 then the probability that the next number is 0 is 0.8. When the previous number is 1, the probability of the next number being 1 is 0.9. There is a bigger chance that the number will be the same as the previous number. When the number changes from previous to next, the next number gets repeated a couple times before it changes back to the other number. The patterns I noticed for runs of 0 and 1’s is that its more common when the probability isn’t equal.

PART II

A)

In this part, I called my function startend() using the sequence chain from the previous part and motifs 0000 and 1111 and a length of 200.

Since the output is really large, the following is a snippet of the output:

```
   [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13] [,14] [,15] [,16] [,17]
[1,]  357  642  643  814  815 1201 1754 1883 2410  2411  2412  2539  2540  2541  2845  2846
3195
[2,]  554  839  840 1011 1012 1398 1951 2080 2607  2608  2609  2736  2737  2738  3042  3043
3392
```

Each pair is a 200 length motif that starts with 0000 and ends with 1111.

B)

In this part, I called my functions startend() like the previous part.

Since the output is really large, the following is a snippet of the output:

```
   [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13] [,14] [,15] [,16] [,17]
[1,]   15   16   24   25   26   27   28   44   45    46   126   145   160   161   162   163   164
[2,]  212  213  221  222  223  224  225  241  242   243   323   342   357   358   359   360   361
```

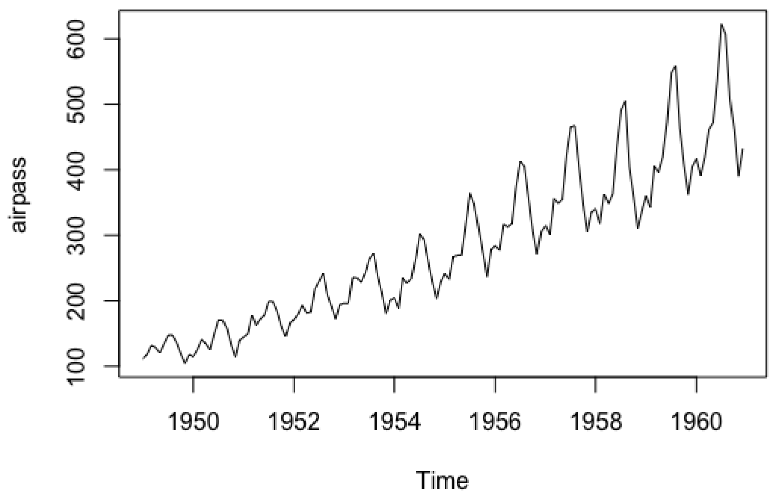
Each pair is a 200 length motif that starts with 0000 and ends with 1111.

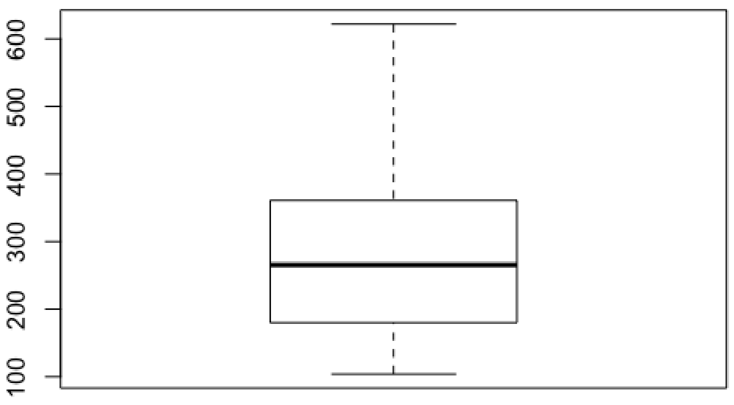
PART III

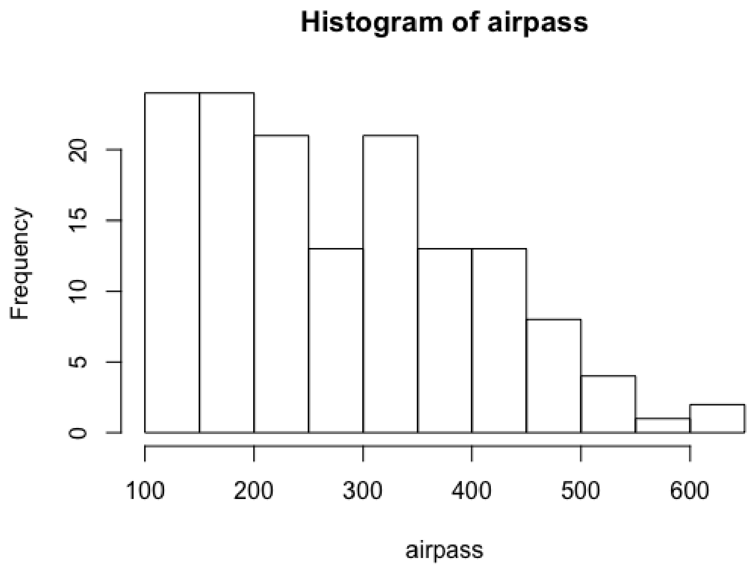
My findings are consistent for both matrix A and B from part i and ii. In part I, matrix A had less output than matrix B because the probability matrix for A is all 0.5. When the probability is at 0.5, it is not skewed to either side, therefore, each number has an equal chance of being either 0 or 1. For matrix B, there are two dominating probabilities at PI(00) = 0.8 and PI(11) = 0.9. There is a higher chance that the next number will be 0 if the previous number was 0. There is an even higher chance that the next number will be 1 if the previous number was 1. In matrix B, most of the number replicate for a long time, thus, creating longer sequences. This is consistent to part II because a length of 200 with starting motifs 0000 and ending motif 1111 is common in matrix A but it is even more common in matrix B. In matrix B, most numbers replicate, therefore it is easy to find motifs of 0000 and 1111. Overall, if the probabilities are equal or close to equal, the numbers in the sequence will fluctuate between 0 and 1 more commonly. If the probabilities are weighted heavily on one side, then that number will be favored more created a sequence with longer subsequences of the same number.

NUMBER 5

PART I AIRPASSENGERS

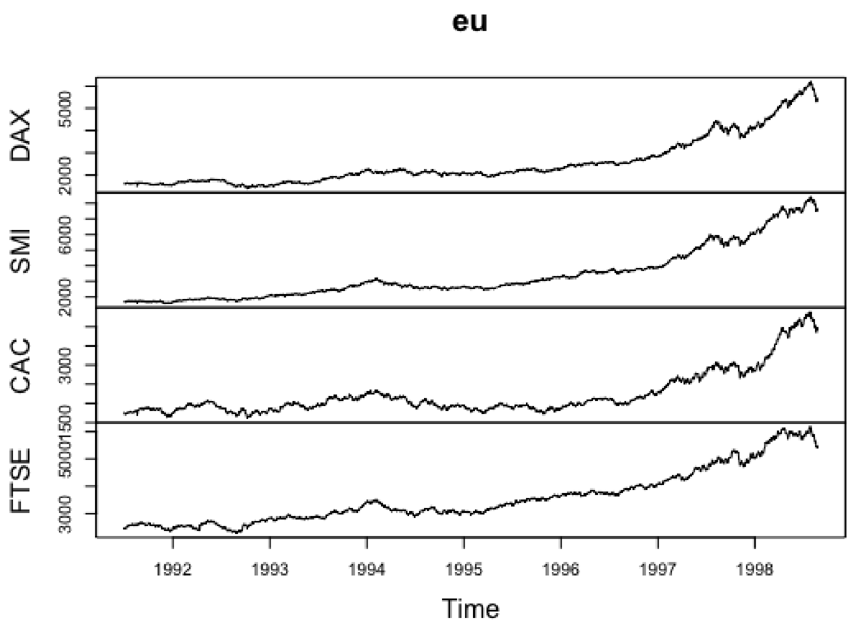


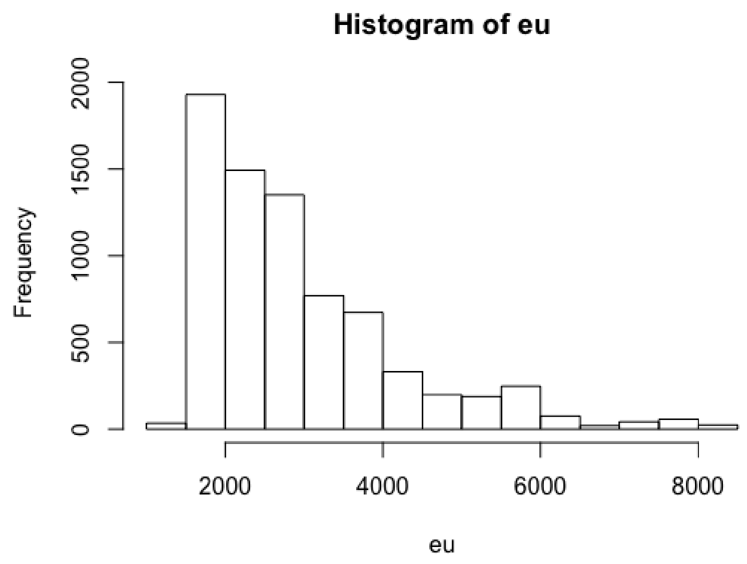


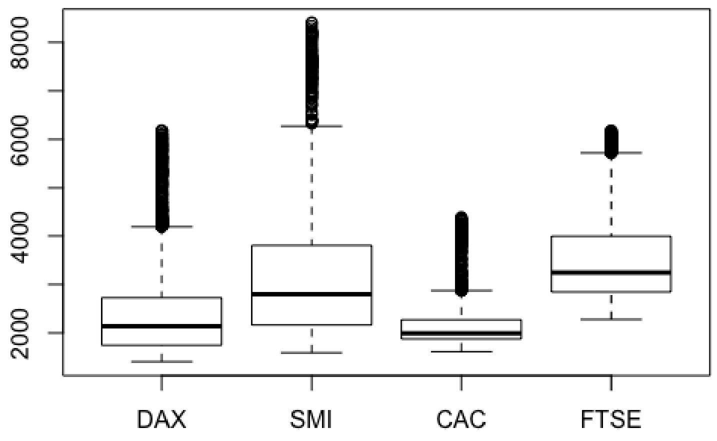


In the AirPassengers data regular line plot, it shows that there's an increase in passengers as the years went by. It also looks like every year at the beginning and the end of the year, there is a decline in passengers. These decreases and increases are more much more significant in the latter years. According to the boxplot, the maximum number of passengers is around 600 and the lowest amount of passengers is around 100. In the histogram 100-350 number of passengers is more common than a greater number of passengers.

PART II EUSTOCKMARKETS

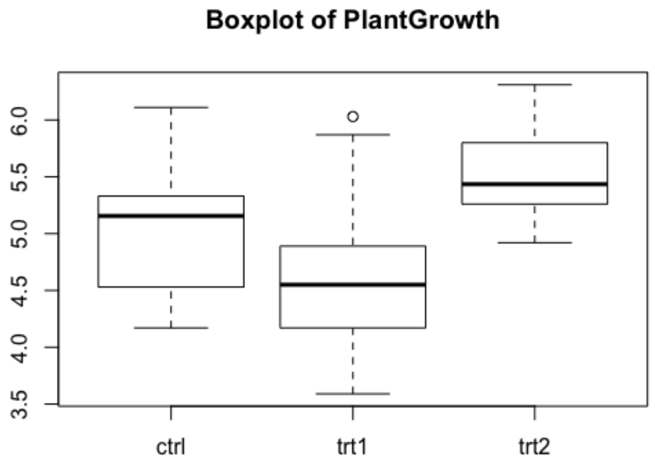






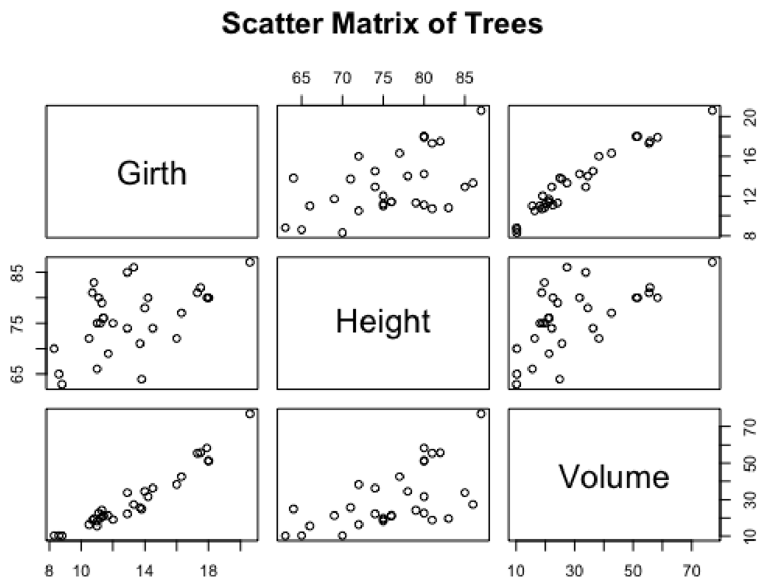
In the EuStockMarket data set, all the stocks fluctuate minimally but they increase more; thus creating an upward trend for all the stocks through the years. Currently, all the stocks are decreasing right now. From 1992 – 1997, the stocks were pretty plateau with a lot of minimal fluctuations thus the histogram shows a lot of data on the left side. This is also shown in the boxplot because all the high values that occurred between 1997 – 1998, they are outliers.

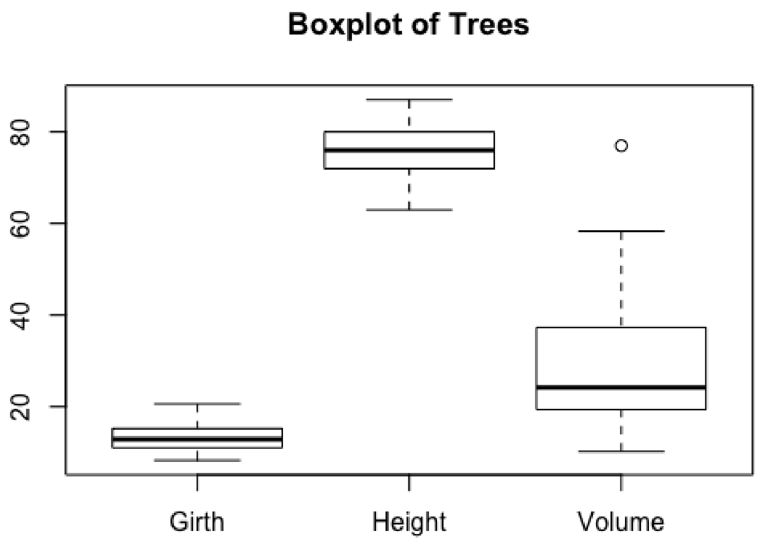
PART III PLANTSGROWTH



In PlantGrowth, there is a group and weight. In group, it is divided into 'crtl', 'trt1', and 'trt2'. None of the groups have outliers except 'trt1'. The values for 'trt2' is the largest set out of all the groups. The maximum value is greater than 6 but less than 7. 'trt1' has the lowest values and median. The control group is right in the middle of 'trt1' and 'trt2'.

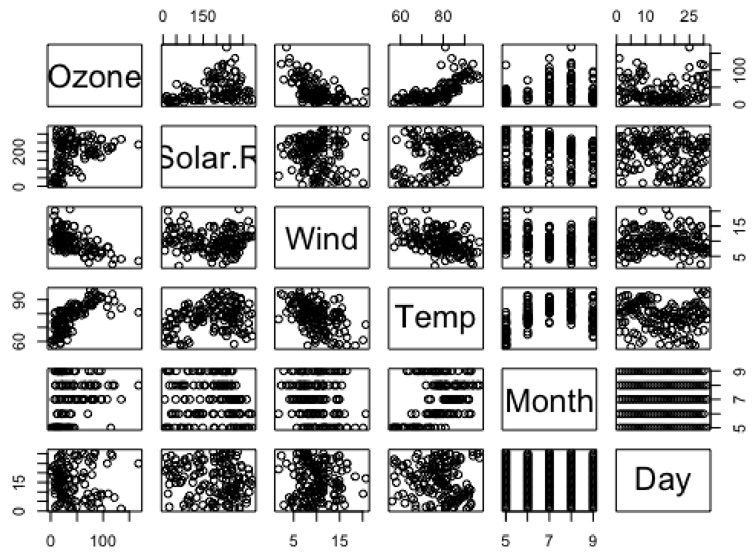
PART IV TREES

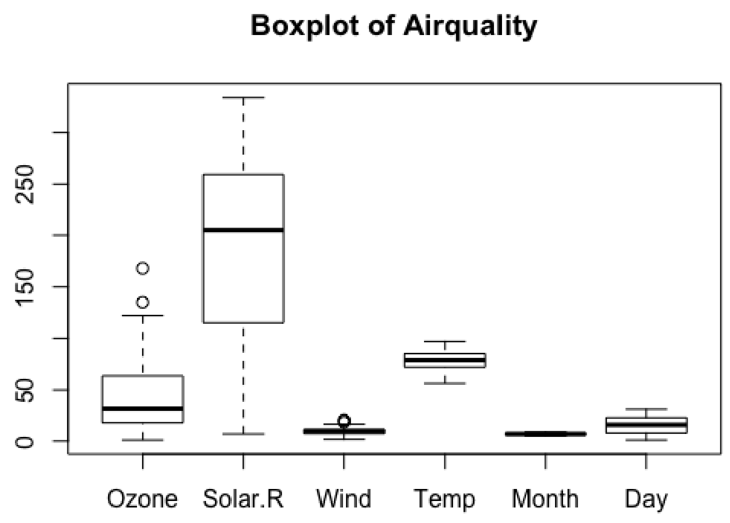




According to this boxplot, Girth's data has the lowest one and they're all within 0 and 30. Volume's data varies more. Height's data is greater which is true since trees are taller. In the scatter plot matrix, girth and volume is more correlated and it has a positive upward trend. It is also more linear than the other pairs. Girth and Height is very scatter and it doesn't look correlated.

PART V AIRQUALITY





In Ozone and Wind, there is 2 and 1 outliers, respectively which doesn't seem to affect the data a lot. In the scatter plot matrix, Ozone and Wind is a good pair because it is negatively correlated. Ozone and Temp is positively correlated. All the other data and pairs are widely distributed so they have no correlation. Solar has the widest range of data whereas all the other data values are roughly similar to each other per month and day.

APPENDIX

HW 1 PROB 1. R

```
## NUMBER 1
## markov_chain takes in 3 parameters
## 1) probability matrix in the form of a vector. The first two numbers must add to 1.The last 2
must also add to 1
## 2) sequence length that you want the output to be
## 3) Initial value = 0 or 1


markov_chain = function(probmatrix = c(0.5, 0.5, 0.5, 0.5), seqlen = sample(5:50, 1), initval = 0){
  probmatrix = matrix(probmatrix, nrow = 2, ncol = 2)
  mc = numeric(seqlen) # empty vector
  mc[1] = initval

  for (i in 2:(seqlen)){
    if (mc[i - 1] == 1){ # checks previous number
      mc[i] = rbinom(1, 1, probmatrix[2,2]) # calculates the next number based on the prob
    }
    else {
      mc[i] = rbinom(1, 1, probmatrix[1,2])
    }
  }

  return (mc)
}
```

HW 1 PROB 2.R

```
sequence_location = function(num_seq = sample(c(0,1), replace = TRUE, size = 20), seq_len =
sample(1:5, 1)) {
  start0 = numeric() #empty vector
  start1 = numeric()
  start0counter = 1 #counters for 0
  start1counter = 1
  less1 = seq_len - 1
  num = (length(num_seq))

  #for loop to iterate through the whole sequence
  for (i in 1:num) {
    count0 = 1;
    count1 = 1;
```

```
#check if location i is 0
if (num_seq[i] == 0){
  if (seq_len == 1) {
      start0[start0counter] = i #if length is 1 then store it and reiterate
      start0counter = start0counter + 1;
  }
  else {
    # loop through length of sequence length requested
    for (j in 1:less1){
      n2 = i + j

      #check if i and j are in bounds
      if (n2 > num) {
      num2 = 'a'
      }
      else {
        num2 = num_seq[n2]
      }
      if (num2 == num_seq[i]) {
        count0 = count0 + 1
      }

      if (count0 == seq_len){
        start0[start0counter] = i #store in vector
        start0counter = start0counter + 1; #update count
      }
    }
  }
}
#repeat process for 1's
if (num_seq[i] == 1) {
  if (seq_len == 1) {
    start1[start1counter] = i
    start1counter = start1counter + 1;
  }
  else {
    for (k in 1:less1) {
      n1 = i + k
      if (n1 > num) {
        num1 = 'a'
      }
      else {
        num1 = num_seq[n1]
        }

        if (num1 == num_seq[i]) {
          count1 = count1 + 1;
        }

        if (count1 == seq_len){
          start1[start1counter] = i
          start1counter = start1counter + 1;
        }
      }
    }
  }
}

#print locations
print(sprintf("Start 0 locations: "))
if (length(start0) == 0){
  print(sprintf("No runs starting with 0"))
}
else {
  print(sprintf("%d",start0))
}

print(sprintf("Start 1 locations: "))

if (length(start1) == 0){
  print(sprintf("No runs starting with 1"))
}
else {
  print(sprintf("%d",start1))
}

#check lengths and create empty spaces with NA so vectors can cancatenate
max_len = max(length(start0), length(start1))
start_0 = c(start0, rep(NA, max_len - length(start0)))
start_1 = c(start1, rep(NA, max_len - length(start1)))

return(cbind(start_0, start_1))

}
```

HW 1 PROB 3.R

```
## NUMBER 3
## startend() takes in at max 4 arguments
## 1) sequence of 0 and 1's
## 2) start motif of 0 and 1's
## 3) end motif of 0 and 1's
## 4) length of total motif
## KATHLEEN ZHEN 999210972

startend = function(sequence_num = sample(c(0,1), replace = TRUE, size = 20), start =
sample(c(0,1), replace = TRUE, size = 3), end = sample(c(0,1), replace = TRUE, size = 3),
motiflength = NULL){
  start_pos = numeric()
  end_pos = numeric()
  start_length = length(start)
  end_length = length(end)
  start_length_less1 = start_length - 1
  end_length_less1 = end_length - 1
  seq_length = ((length(sequence_num)))
  startcount = 1;
  endcount = 1;

  # iterate through sequence length
  for (i in 1:seq_length) {
    countstart = 1;
    countend = 1;
    #check if location i matches start motif[1]
    if (sequence_num[i] == start[1]) {
      for (j in 1:start_length_less1) {
        n = i + j;
        #check bounds
        if (n > seq_length) {
          num = 'a'
        }
        else {
          num = sequence_num[n]
        }
        #check if rest of motif exists
        if (num == start[j+1]){
          countstart = countstart + 1;
          if (countstart == start_length) {
            start_pos[startcount] = i #store in vector
            startcount = startcount + 1;
          }
        }
      }
    }
    #repeat for end motif
    if (sequence_num[i] == end[1]){
      for (k in 1:end_length_less1) {
        nn = i + k;
        if (nn > seq_length) {
          numm = 'a'
        }
        else {
          numm = sequence_num[nn]
        }

        if (numm == end[k+1]) {
          countend = countend + 1;
          if (countend == end_length) {
            end_pos[endcount] = i;
            endcount = endcount + 1;
          }
        }
      }
    }
  }

  x = length(start_pos)
  y = length(end_pos)
  new_store_start = numeric()
  new_store_end = numeric()
  start_locator = 1
  end_locator = 1
  asd = numeric()
  print(sprintf("%d", sequence_num))
  print(sprintf("%d", start))
  print(sprintf("%d", end))

  #print and store locations if specified length of a motif sequence - parameter 4
  if (length(motiflength) == 1) {
    for (j in 1:y){
      for (k in 1:x) {
        poslength = (end_pos[j] + end_length_less1) - start_pos[k]  #check matching length
        if (poslength == motiflength){
          new_store_start[start_locator] = start_pos[k] #store
          new_store_end[end_locator] = end_pos[j]
          start_locator = start_locator + 1
```

```
          end_locator = end_locator + 1
          asd = cbind(asd, c(start_pos[k], end_pos[j]))
        }
      }
    }
    print(sprintf("%d", new_store_start)) #print
    print(sprintf("%d", new_store_end))
  }
  else { #for all other instances where length of motif is specified
    for ( q in 1:x) {
      if (length(start_pos) == 0){
        print(sprintf("No runs starting with motif"))
        if (length(end_pos) == 0){
          print(sprintf("No ends starting with motif"))
        }
      }
      else {
        print(sprintf("Start motif locations: %d",start_pos[q]))
        o = 1;
        p = numeric()
        tick = 1;
        for (u in 1:y) {
          if (end_pos[u] > start_pos[q]) { #finding the pairs
            p[o] = end_pos[u] #store
            o = o + 1;
            asd = cbind(asd, c(start_pos[q], end_pos[u])) #store in a matrix
          }
        }
        print(sprintf("End motif starting locations:"))
        print(sprintf("%d", p))
      }
    }
  }
  return(asd)
}
```

HW 1 PROB 4. R

```
## NUMBER 4
## KATHLEEN ZHEN 999210972

##PART I
num4ilength = 10000
```

```
##a

pmatrix = c(0.5, 0.5, 0.5, 0.5)
chain = markov_chain(pmatrix, num4ilength, 0)

#iterate from lengths 1 to 10
for ( fgh in 1:10) {
  print(sprintf("Lengths of %d", fgh))
  a = sequence_location(chain, fgh)

}

##b
pmatrixb = c(0.8, 0.10, 0.2, 0.9)
chainb = markov_chain(pmatrixb, num4ilength, 0)

for ( fgh1 in 1:10) {
  print(sprintf("Lengths of %d", fgh1))
  b = sequence_location(chainb, fgh1)

}

##PART II
number4iilength = 200
num4startmotif = c(0,0,0,0)
num4endmotif = c(1,1,1,1)

##a
fouriia = startend(chain, num4startmotif, num4endmotif, number4iilength)


##b
fouriib = startend(chainb, num4startmotif, num4endmotif, number4iilength)
```

HW 1 PROB 5.R

```
## NUMBER 5
## KATHLEEN ZHEN 999210972

airpass = AirPassengers
plot(airpass) #scatter plot
boxplot(airpass) #boxplot
hist(airpass) #histogram
```

```
eu = EuStockMarkets
plot(eu)
hist(eu)
boxplot(eu, main = "Boxplot of EuStockMarkets")

plants = PlantGrowth
plot(plants$group, plants$weight, main = "Boxplot of PlantGrowth")


treedata = trees
boxplot(treedata, main = "Boxplot of Trees")
plot(treedata, main = "Scatter Matrix of Trees")

aq = airquality
plot(aq) #scatter plot matrix
boxplot(aq, main = "Boxplot of Airquality")
```