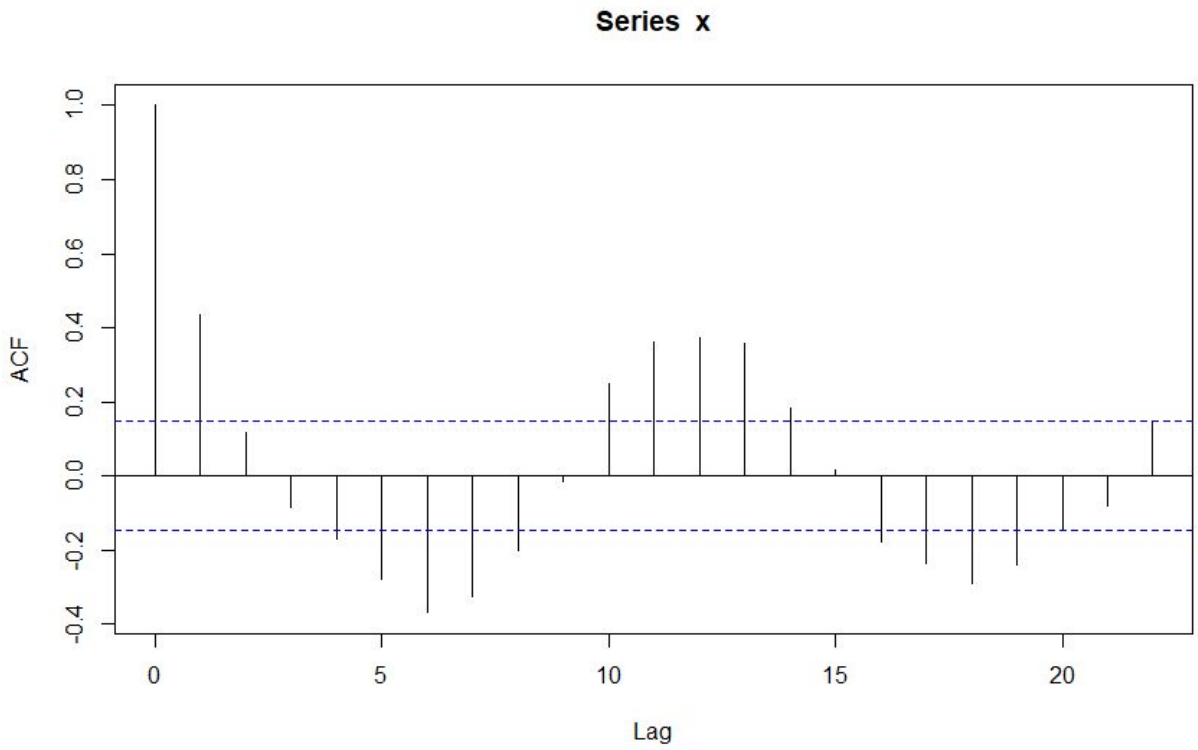
Homework 4

Due: Friday 12/6, 5:30pm on Titanium. Prepare your answers as a single PDF file.
Group work: You may work in groups of 1-3. Include all group member names in the PDF file. Only one person in the group needs to submit to Titanium.

1. Download the linked text file, `seattle_rainfall.txt`, which is the amount of monthly rainfall in a location in Seattle between November 2002 and May 2017[1]. The goal is to convert this to a R `ts` object with the appropriate frequency and then decompose it into trend, seasonal, and random components. Write R code to do the following tasks:
   a) Load the file into a R variable. Call `scan('seattle_rainfall.txt')` [show code]
      x <- scan("seattle_rainfall.txt")
      Read 175 items
   b) Find the frequency of the seasonal component by plotting the autocorrelation (plot the output of the `acf` function and pick the peak away from x=0) [show code and plot]
      acf(x)



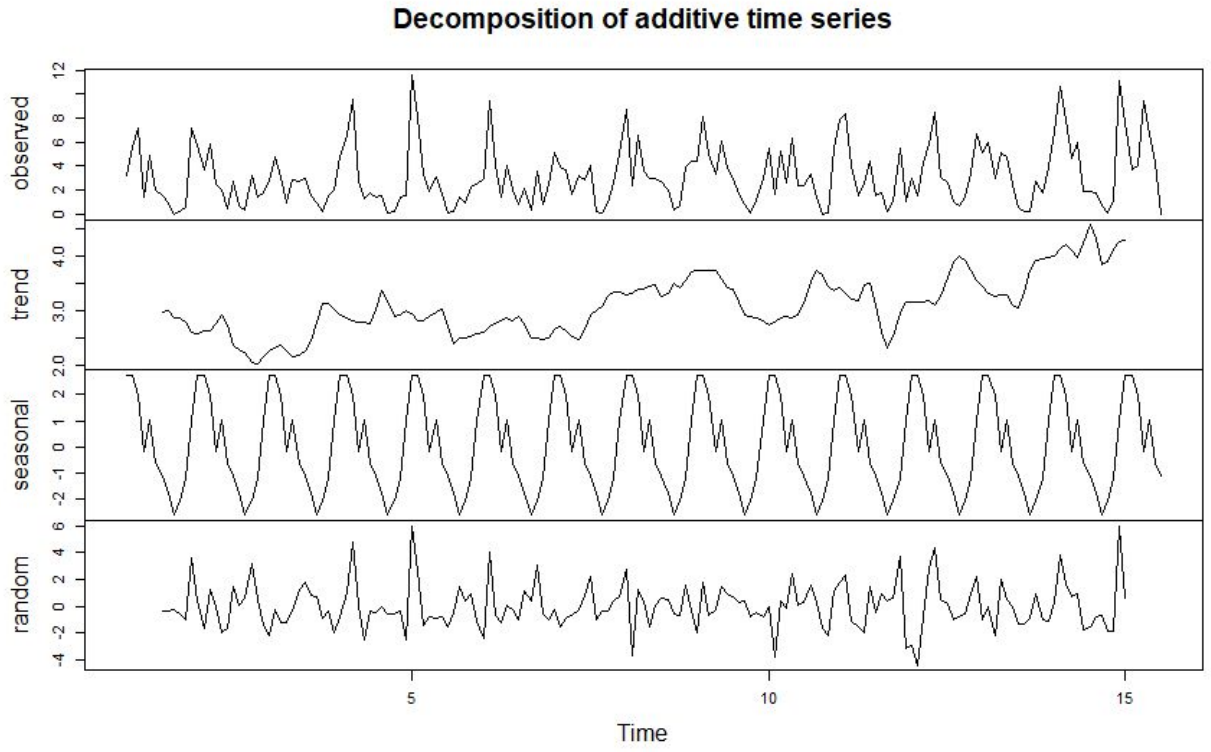
      Second highest peak is at x = 12. So frequency will be 12.
   c) Convert to a `ts` object (call the constructor `ts` with frequency parameter) [show code]

[1] https://catalog.data.gov/dataset/observed-monthly-rain-gauge-accumulations

xts <- ts(x, frequency = 12)

d) Decompose the ts dataset (call `decompose` function). Plot the output showing the trend, seasonal, random). [show code and plot]

plot(decompose(xts))



2. Compute the Dynamic Time Warping distance between the two time series:

   a. X = [1, 3, 4, 4]
   b. Y = [1, 2, 5]

Let the cost function be $cost(x_i, y_i) = (x_i - y_i)^2$.

a) Show the cost matrix. This is partially complete below.

0	1	16
4	1	4
9	4	1
9	4	1

X2, y3 = (3-5)^2 = 4
X3, Y2 = (4-2)^2 = 4
X3. Y3 = (4-5)^2 = 1

X4, Y1 = (4-1)^2 = 9
X4, Y2 = (4-2)^2 = 4
X4, Y3 = (4-5)^2 = 1

b) Show the DTW matrix. This is partially complete below.

0	1	17
4	1	5
13	5	2
22	9	3

C(4, 1) = 13 + 9 = 22
C(3, 2) = min(13, 4, 1) + 4 = 5
C(4, 2) = min(22, 13, 5) + 4 = 9
C(2, 4) = min(1, 1, 17) + 4 = 5
C(3, 4) = min(5, 1, 5) + 1 = 2
C(4, 4) = min(9, 5, 2) + 1 = 3

c) The DTW distance between the two time-series is 3.

3. Complete the R function below to compute the DTW distance between two vectors, v1 and v2.

```
dtw <- function (v1, v2) {
  M <- length(v1)
  N <- length(v2)
  Cost <- matrix(0,M,N) # Initialize with zeros
  for (i in 1:M) {
      for (j in 1:N) {
      Cost[i,j] <- (v1[i] - v2[j])*(v1[i] - v2[j]) # distance function
      }
  }
  C <- matrix(0,M,N) # Initialize with zeros
  C[1,1] <- Cost[1,1] # Initialize top left cell
  for (i in 2:M) { # Initialize first column
      C[i,1] <- C[i-1,1] + Cost[i,1]
  }
  for (j in 2:N) { # Initialize first row
      C[1,j] <- C[1,j-1] + Cost[1,j]
  }
  # Complete the main loop
```

```
  for (i in 2:M) {
    for (j in 2:N) {
      C[i,j] <- min(C[i-1,j], C[i, j-1], C[i-1,j-1]) + Cost[i,j]
    }
  }
  return (C[M,N])
}
```

Verify your answer to Q3 using the above function.

```
> dtw(x,y)
[1] 3
```

4. Consider the following three short documents:

Document 1: *see spot*
Document 2: *see spot run*
Document 3: *run spot run*

The following questions are meant to be done by hand though you can also use R. Note that some of the R functions normalize/don't normalize.

a) Show the Document-Term matrix weighted by Term-frequency (Tf). Do not normalize (i.e., use the word counts**).**

V = {see, spot, run}

DTM = d1: (1,1,0)
d2: (1,1,1,)
d3: (0,1,2)

b) Show the Document-Term matrix weighted by Term-frequency (Tf). Normalize by the number of words in each document.

d1: (½, ½, 0)
d2: (⅓, ⅓, ⅓)
d3: (0, ⅓, ⅔)

c) What is the inverse document frequency (Idf) of each word?

IDF("see") = $\log_2(\frac{3}{2})$ = 0.5849625007211562
IDF("spot") = $\log_2(\frac{3}{3})$ = 0
IDF("run") = $\log_2(\frac{3}{2})$ = 0.5849625007211562

d) Show the Document-Term matrix weighted by Tf-Idf for this dataset.

answer:

	d1	d2	d3
see	**½*0.5849625=.29248**	**⅓*0.5849625=.194975**	**0*0.5849625=0**
spot	**½*0=0**	**1/3*0=0**	**1/3*0=0**
run	**0*0.5849625=0**	**1/3*0.5849625=.1949875**	**2/3*0.5849625=.389975**

e) Which word has the most "information" based on Tf-idf? Which is the least informative?

The word "run" has the most information on it since it has less frequency. It also has the highest TF-IDF total. "Spot" is least informative since it appears in every document. "spot" and "run" has a zero TF-IDF total.

5. Download the linked CSV file that contains tweets relating to airline flights[2]. Each tweet also has an associated "sentiment" - whether the expressed opinion in the tweet is positive, negative, or neutral. The goal is to determine the sentiment of the first tweet by only comparing the first tweet to the remaining tweets. The sentiment of the most similar tweet will then be the predicted sentiment of the first tweet.
The goal is to get familiar with fundamental text processing steps; so do not install and run any sentiment analysis library or function for this question.

Write R code to do the following tasks (please refer to the sample text processing R code posted on Titanium):

a) Load the given CSV file and convert it to a VCorpus data type.

```
# workDir <- "C:/temp/cpsc375hw4/"
# setwd(workDir)

filename <- "airline_tweets_sentiment2.csv"
# filename <- "short.csv"
docs <- read.csv(filename)
docArray <- unlist(docs[,4])

twitterData <- VCorpus(VectorSource(docArray))
```

[2] Modified from https://www.kaggle.com/crowdflower/twitter-airline-sentiment

b) Print the full text of the first tweet.

```
> twitterData[[1]][["content"]][1]
[1] "@united yes. We waited in line for almost an hour to do so. Some passengers just left not wanting to wait past 1am."
```

c) Transform all text to remove numbers, remove punctuation, and remove stopwords

```
# twitterData <- tm_map(twitterData, content_transformer(tolower)) # to lower case
twitterData <- tm_map(twitterData, removeNumbers)
twitterData <- tm_map(twitterData, removePunctuation)
twitterData <- tm_map(twitterData, removeWords, stopwords("english")) # remove stop words
```

d) Convert the document to a Document-Term matrix with Tf-Idf weighting. How many words are present?

```
dtm <- DocumentTermMatrix(twitterData, control=list(weighting=weightTfIdf))
inspect(dtm) # DocumentTermMatrix with Tf-Idf weighting
vocabs <- dtm$dimnames$Terms
length(vocabs)
```

output:

```
> length(vocabs)
[1] 14744
```

e) Remove all words from the Document-Term matrix with sparsity >= 99%. How many words are now present?

```
> sparsity <- .99
> dtm <- removeSparseTerms(dtm, sparsity)
> vocabs <- dtm$dimnames$Terms
> length(vocabs)
[1] 169
```

f) Compute the cosine similarity between the first tweet and the remaining tweets. It is easiest to write R code from scratch instead of installing a library just for this:

a) Define a function that takes two vectors as input and computes their cosine similarity: $cosine(A,B) = \frac{sum(A_i B_i)}{\sqrt{sum(A_i^2) \times sum(B_i^2)}}$

```
cosine <- function (A, B) {
  sumAB = 0
  for (i in 1:length(A)){
    sumAB = sumAB + A[i]*B[i]
  }

  sumA2 = 0
  for (i in 1:length(A)){
    sumA2 = sumA2 + A[i]^2
  }

  sumB2 = 0
  for (i in 1:length(B)){
    sumB2 = sumB2 + B[i]^2
  }

  denom = sqrt(sumA2*sumB2)
  if (denom > 0) {
    return (sumAB/sqrt(sumA2*sumB2))
  }
  else {
    return (0)
  }
}
```

b) Convert the Document-Term sparse matrix to a regular matrix, like so:

i) mymatrix <- as.matrix(mydtm)

```
mymatrix <- as.matrix(dtm)
```

c) Call the cosine function with mymatrix[1,:] and every other row. Find the index which gives the maximum value.

```
dimMatrix <- dim(mymatrix)

maxValue = 0
maxIndex = 0
current = 0
for (i in 2:dimMatrix[1]) {
  current = cosine(as.vector(mymatrix[1,]), as.vector(mymatrix[i,]))
  if (current > maxValue) {
    maxValue = current
```

```
    maxIndex = i
  }
}
```

g) Print the full text of the tweet with the maximum similarity

```
> print(twitterData[[1]][["content"]][1])
[1] "united yes We waited  line  almost  hour    Some passengers just left  wanting  wait
past "
> print(twitterData[[maxIndex]][["content"]][1])
[1] "AmericanAir Err  hour wait time   Exec Plat line"
> print(maxValue)
[1] 0.5765516
```

h) Print the corresponding sentiment of the above tweet. This is the predicted sentiment of the first tweet. Does this match the given sentiment (from row 1)?

```
> print(docs[['airline_sentiment']][maxIndex])
[1] negative
Levels: negative neutral positive
> print(docs[['airline_sentiment']][1])
[1] negative
Levels: negative neutral positive
```

answer: They do match

i) Is this bag-of-words approach in general a good way to predict sentiment in tweets? Why or why not? Answer in 2-3 sentences.

answer:
I believe that it is a good way to predict sentiment in tweets because the similarity of the tweets does communicate the same sentiment. The match in similarity means that the words and structure very closely matches when the tweets have high similarity indexes. The match might not be a 100% but it is very close.