week-2-homework

Question 4.1 Describe a situation or problem from your job, everyday life, current events, etc., for which a clustering model would be appropriate. List some (up to 5) predictors that you might use.

Clustering algorithms can be used in recommender systems to analyse user data. These clusters can be used to can be used to group similar users together based on their preferences and behaviors.

For example a recommender system for a streaming platform might implement clustering based on:

1. Genre of a show
2. Length, such as the length of one episode, or number of episodes, or number of seasons
3. Producer, director or production studio of a show
4. The cast of the show

Question 4.2 The iris data set `iris.txt` contains 150 data points, each with four predictor variables and one categorical response. The predictors are the width and length of the sepal and petal of flowers and the response is the type of flower. The data is available from the R library datasets and can be accessed with iris once the library is loaded. It is also available at the UCI Machine Learning Repository (https://archive.ics.uci.edu/ml/datasets/Iris). *The response values are only given to see how well a specific method performed and should not be used to build the model.* Use the R function `kmeans` to cluster the points as well as possible. Report the best combination of predictors, your suggested value of k, and how well your best clustering predicts flower type.

Ref: https://www.datacamp.com/tutorial/k-means-clustering-r

Ref: https://www.rdocumentation.org/packages/stats/versions/3.6.2/topics/kmeans

Ref: https://www.analyticsvidhya.com/blog/2021/01/in-depth-intuition-of-k-means-clustering-algorithm-in-machine-learning/

Ref: https://www.analyticsvidhya.com/blog/2019/08/comprehensive-guide-k-means-clustering/

Ref: https://vitalflux.com/kmeans-silhouette-score-explained-with-python-example/

For this question, we're going to use the `kmeans()` function to cluster the Iris dataset.

First, let's go ahead and load the dataset. The `iris` dataset is a built-in dataset which we can access via the `datasets` package in R, stored as the variable `iris`.

```r
# load dataset
library(datasets)
data("iris")
head(iris, 5)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
```

Remember that the `iris` dataset comes with a response variable column, which we do not want when training an unsupervised learning clustering model. So let's go ahead and remove that column from our dataset and store the modified dataset as `iris_unlabelled`.

```r
# remove response column
iris_unlabelled <- iris[,1:4]
head(iris_unlabelled,5)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width
## 1          5.1         3.5          1.4         0.2
## 2          4.9         3.0          1.4         0.2
## 3          4.7         3.2          1.3         0.2
## 4          4.6         3.1          1.5         0.2
## 5          5.0         3.6          1.4         0.2
```

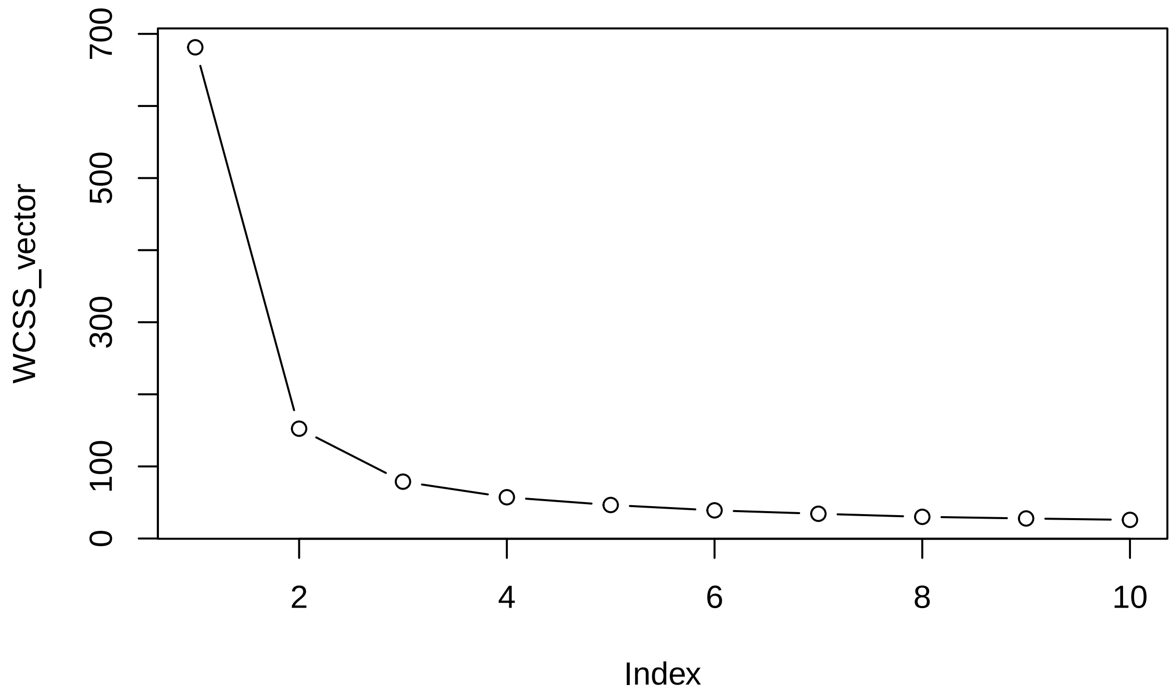
Now we implement our k-means model. We will loop through 10 values of k and calculate the within-cluster sum of squares (WCSS) for each value of k.

WCSS is the sum of squared distance between each data point and the centroid in the cluster. In the lectures, this is part of the formula that was given to us in lecture 4.3 $\sum_j (x_{ij} - z_{jk})^2$, and our overall aim is to minimize this value to the point where increasing the value of k no longer significantly affects the WCSS.

```r
# set up variables
WCSS_vector = vector("numeric")
k_vector <- 1:10

set.seed(42)

# loop over values of k
for (k in k_vector) {
  # train kmeans model
  model <- kmeans(iris_unlabelled,
                  centers = k,
                  nstart = 25)

  # store wss for each model
  WCSS_vector[k] <- model$tot.withinss
}
```

To identify this point, we will plot the WCSS against the values of k. As the number of clusters increases, the WCSS value will start to decrease. There will be a point on the graph where the curve will start to change very rapidly.

```r
# scree plot
plot(WCSS_vector, type='b')
```



From this graph, we can see that the optimal value of k is probably 2 or 3. The elbow method is not an exact method and there can be cases such as this where the bend in the "elbow" is not obvious, making it difficult to determine the exact optimal value of k.

To be more confident in our choice, we can turn to a second metric to evaluate the performance of our model.

Unlike in classification, accuracy is not going to be a very good metric for evaluating clustering. In practice, we are dealing with unlabelled data and will not know what the "correct" clustering should be. Instead, one of the metrics commonly used in evaluating the quality of a clustering model is the silhouette plot, which we can visualize with the help of the `cluster` and `factoextra` packages.

```r
# import packages
library(cluster)
library(factoextra)
```

```
## Loading required package: ggplot2
```

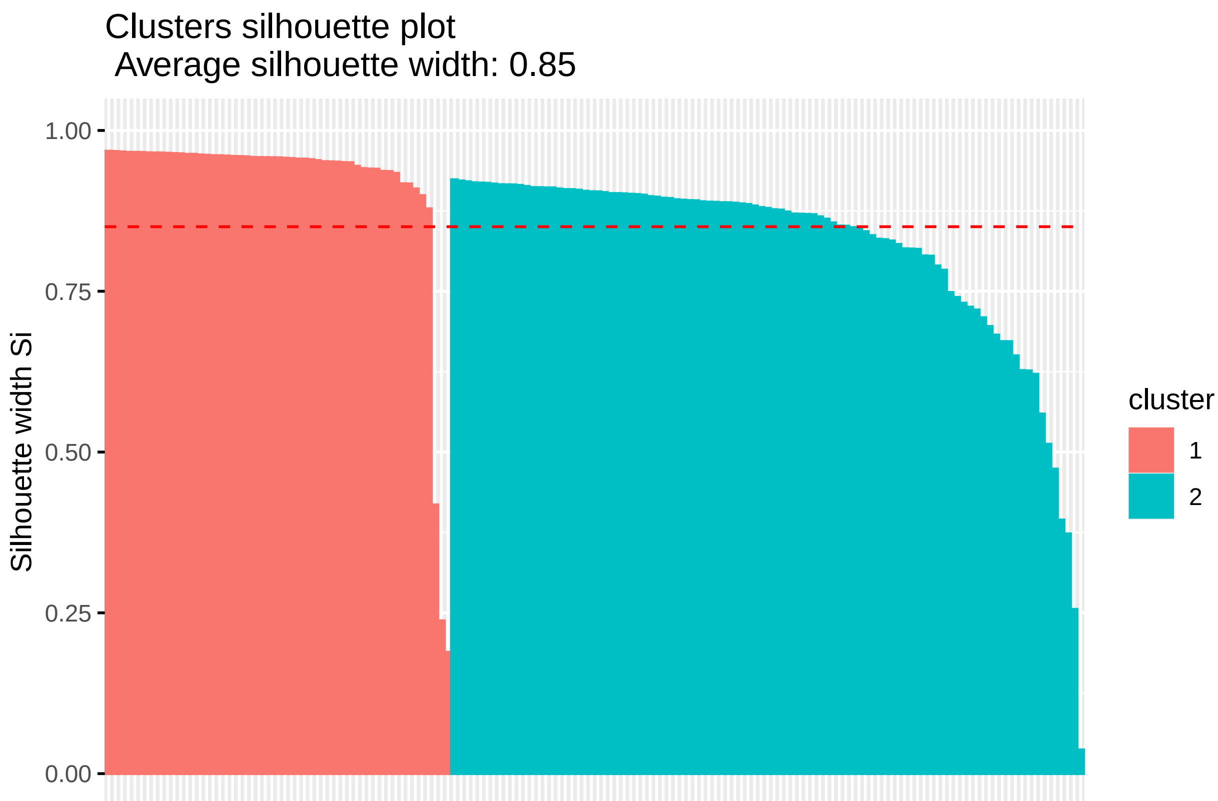
```
## Welcome! Want to learn more? See two factoextra-related books at https://goo.gl/ve3WBa
```

A 'silhouette' refers to a method of measuring consistency within clusters of data. The silhouette score is a measure of how similar a data point is to its own cluster, compared to other clusters. The silhouette score ranges from -1 to +1, where a high score indicates that the data point is well matched to its own cluster and poorly matched to other clusters.

Since we have already narrowed our optimal values for k down to either k=2 or k=3, let's plot a silhouette plot for each of these:

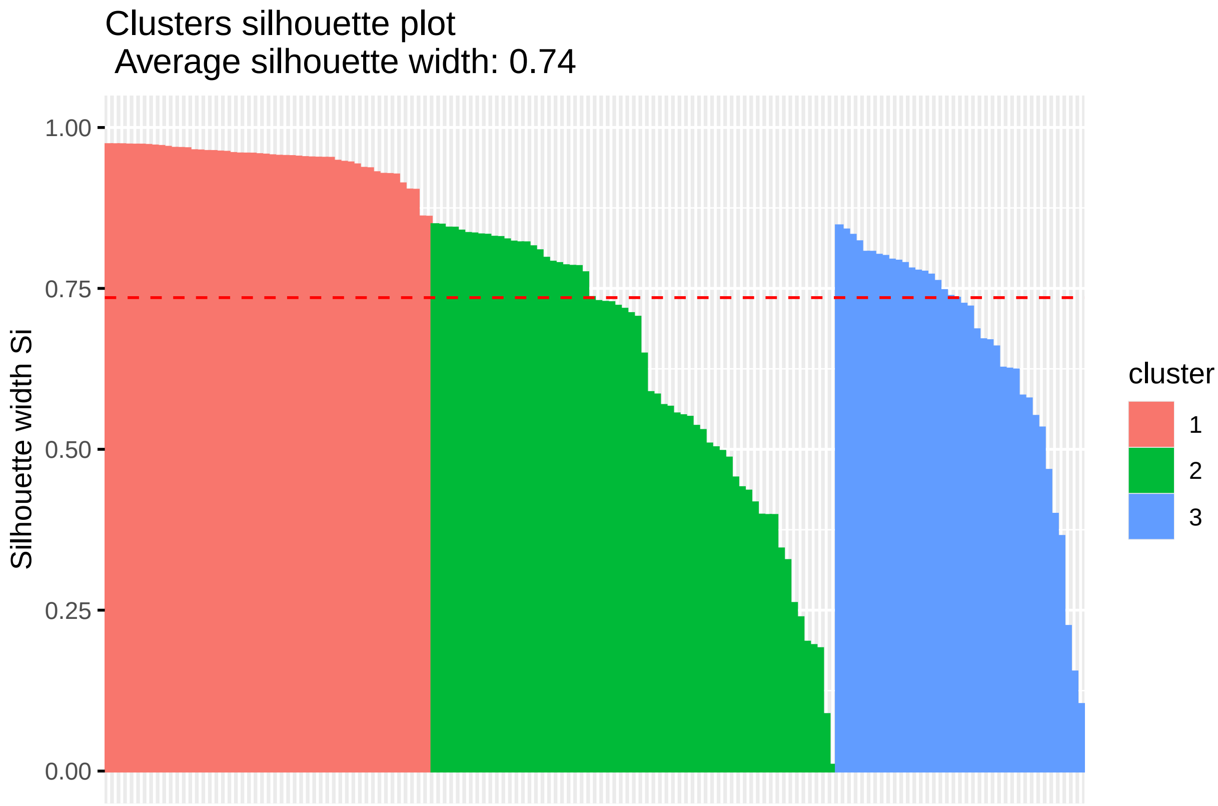
```r
# silhouette plot when k = 2
set.seed(42)
model <- kmeans(iris_unlabelled, centers=2, nstart=25)
ss <- silhouette(model$cluster, dist(iris_unlabelled)^2)
fviz_silhouette(ss)
```

```
##   cluster size ave.sil.width
## 1       1   53          0.91
## 2       2   97          0.82
```




```r
# silhouette plot when k = 3
set.seed(42)
model <- kmeans(iris_unlabelled, centers=3, nstart=25)
ss <- silhouette(model$cluster, dist(iris_unlabelled)^2)
fviz_silhouette(ss)
```

```
##   cluster size ave.sil.width
## 1       1   50          0.95
## 2       2   62          0.61
## 3       3   38          0.66
```

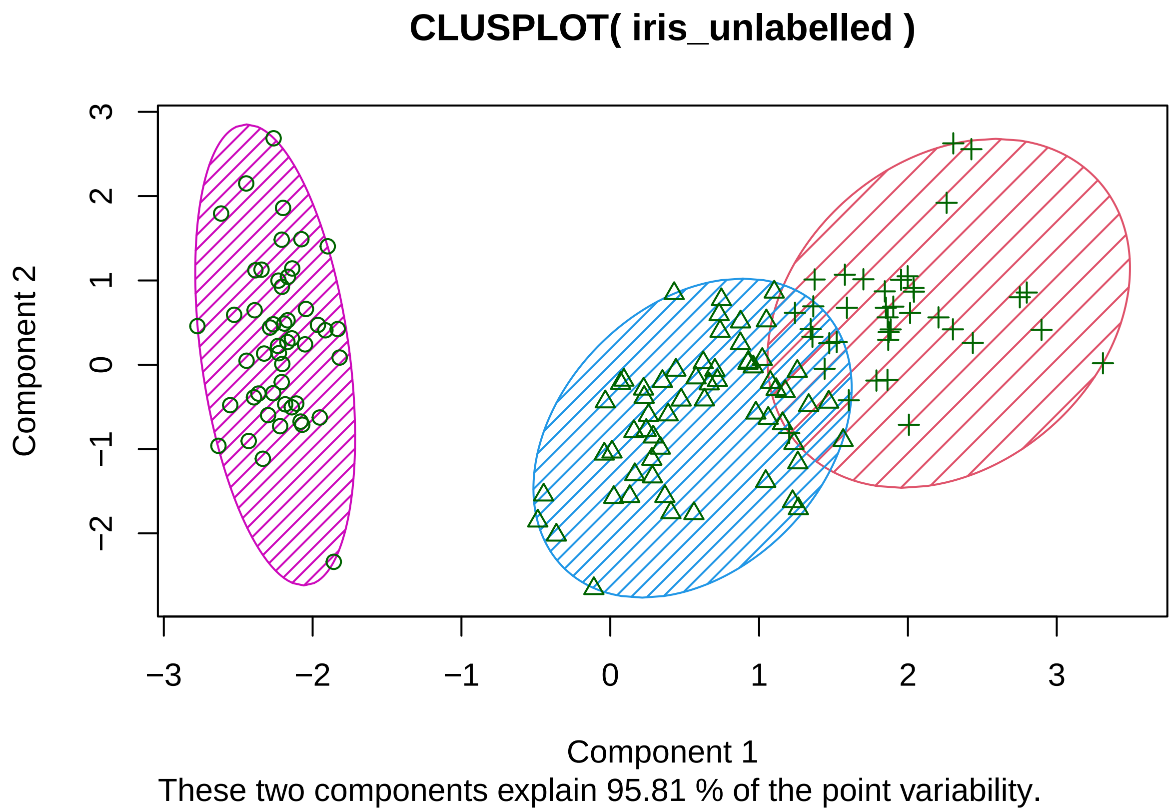


The average silhouette scores for both k=2 and k=3 are very high, as expected. Although the silhouette score for k=2 (0.85) is higher than that of k=3 (0.74), the width of the silhouette plot representing each cluster also is a deciding point. For plot with k=3, the width of each cluster is more uniform than the plot with k=2, which has one cluster much wider than the other. Therefore, we can select the optimal number of clusters as 3.

Let's re-train the model with our chosen value of k=3 and plot the clusters to see how well our clustering model performs on the `iris` dataset.

```
# train kmeans model with k=3
model <- kmeans(iris_unlabelled, centers=3, nstart=25)
```

```
# plot clusters
clusplot(iris_unlabelled, model$cluster, color=T, shade=T, lines=F)
```



As mentioned before, we do not typically have response variables available for clustering models. However, since the Iris dataset does contain labels, we might as well take advantage of that to observe how our model has separated the data points into clusters.

```
table(model$cluster, iris$Species)
```

```
##
##     setosa versicolor virginica
##   1     50          0         0
##   2      0         48        14
##   3      0          2        36
```

From this breakdown, we can see the setosa cluster perfectly explained, meanwhile virginica and versicolor have some noise between their clusters.

Question 5.1 Using crime data from the file `uscrime.txt` (http://www.statsci.org/data/general/uscrime.txt, description at http://www.statsci.org/data/general/uscrime.html), test to see whether there are any outliers in the last column (number of crimes per 100,000 people). Use the `grubbs.test` function in the `outliers` package in R.

Ref: https://www.rdocumentation.org/packages/outliers/versions/0.15/topics/grubbs.test

Ref: https://www.itl.nist.gov/div898/handbook/eda/section3/eda35h1.htm

Ref: https://www.statisticshowto.com/grubbs-test/

For this question, we will be using the Grubbs' test to detect outliers in the last column of the `uscrime` dataset. the `grubbs.test` function is found in the `outliers` package.

From the NIST Engineering Statistics Handbook:

Grubbs' test (Grubbs 1969 and Stefansky 1972) is used to detect a single outlier in a univariate data set that follows an approximately normal distribution.

Grubbs' test is defined for the hypothesis:

- H_0: There are no outliers in the dataset
- H_1: There is exactly one outlier in the dataset

Note: All hypothesis tests in this exercise is performed at 95% confidence-level

```
# import package
library(outliers)
```

```
# load data
crime <- read.delim("../week 2 data-summer/uscrime.txt")
head(crime)
```

```
##      M So   Ed  Po1  Po2    LF   M.F Pop   NW    U1  U2 Wealth Ineq     Prob
## 1 15.1  1  9.1  5.8  5.6 0.510  95.0  33 30.1 0.108 4.1   3940 26.1 0.084602
## 2 14.3  0 11.3 10.3  9.5 0.583 101.2  13 10.2 0.096 3.6   5570 19.4 0.029599
## 3 14.2  1  8.9  4.5  4.4 0.533  96.9  18 21.9 0.094 3.3   3180 25.0 0.083401
## 4 13.6  0 12.1 14.9 14.1 0.577  99.4 157  8.0 0.102 3.9   6730 16.7 0.015801
## 5 14.1  0 12.1 10.9 10.1 0.591  98.5  18  3.0 0.091 2.0   5780 17.4 0.041399
## 6 12.1  0 11.0 11.8 11.5 0.547  96.4  25  4.4 0.084 2.9   6890 12.6 0.034201
##      Time Crime
## 1 26.2011   791
## 2 25.2999  1635
## 3 24.3006   578
## 4 29.9012  1969
## 5 21.2998  1234
## 6 20.9995   682
```

```
# view last column
summary(crime[, 16])
```

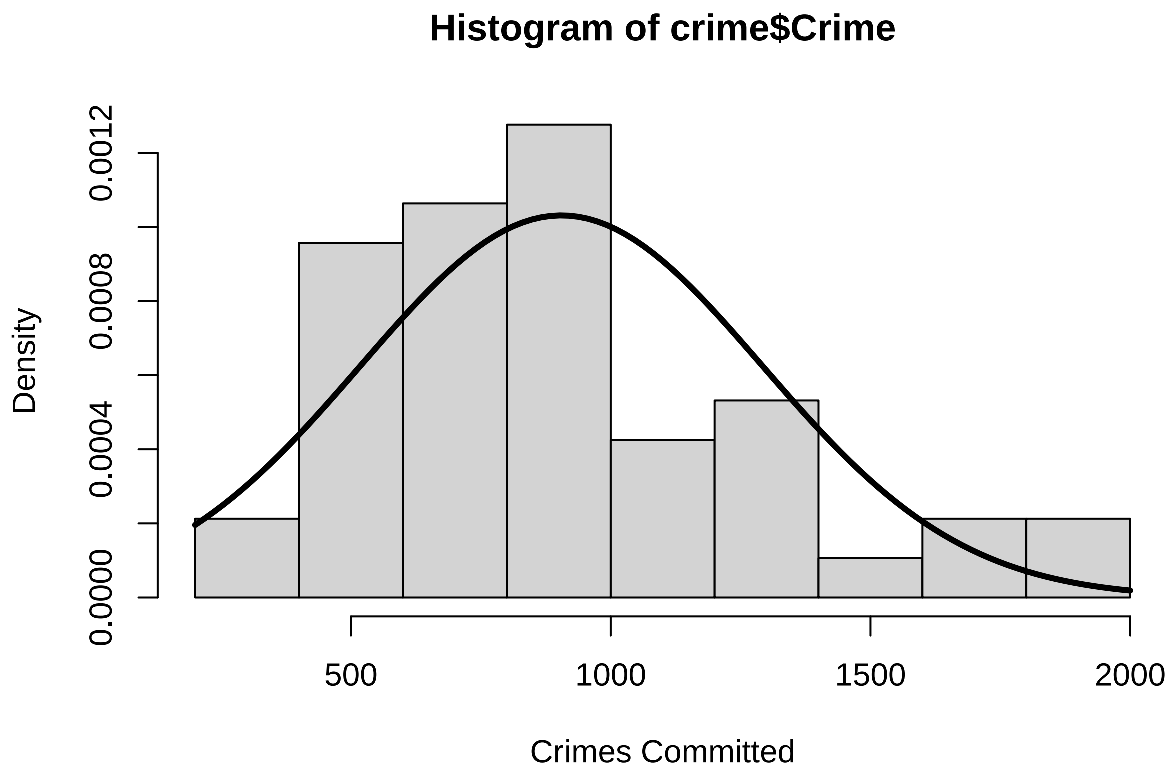
```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   342.0   658.5   831.0   905.1  1057.5  1993.0
```

Since the Grubbs' test is based on the assumption that the data is normally distributed, we should verify that our data can actually be approximated as a normal distribution before we apply the test.

We can do this through a couple of ways:

1. Histogram plot

```
# plotting histogram
hist(crime$Crime, xlab="Crimes Committed",probability=TRUE)
s = sd(crime$Crime)
m = mean(crime$Crime)
curve(dnorm(x, mean=m, sd=s), add=TRUE,lwd = 3)
```



Here we can see that that there is a clear right skew for the data in the `Crime` column. The distribution is not symmetrical and thus cannot be described as being normally distributed.

2. Shapiro-Wilk test

```
# shapiro-wilk test
shapiro.test(crime$Crime)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  crime$Crime
## W = 0.91273, p-value = 0.001882
```

With a p-value of less than 0.05, the Shapiro-Wilk test

Based on these 2 tests, it's pretty clear that we are not dealing with a univariate normal distribution.

There is a chance that the data is *mostly* normally distributed and is being seriously affected by an extreme outlier. To check this, we will use the Grubbs' test to identify one outlier, then run the Shapiro-Wilk test again to check if the distribution is normal. If the data still fails the Shapiro-Wilk test, then the Grubbs' test is not suited for it.

Use the parameter `type=10` to identify one outlier on one tail of the distribution. By default, this is the *right* tail.

```r
# identify one outlier (right tail)
grubbs.test(crime[,16], type=10)
```

```
##
##  Grubbs test for one outlier
##
## data:  crime[, 16]
## G = 2.81287, U = 0.82426, p-value = 0.07887
## alternative hypothesis: highest value 1993 is an outlier
```

The test has identified the data point 1993 as an outlier. This is accompanied by a p-value of 0.07887. Because the p-value is more than 0.05, we fail to reject the null hypothesis that this data point is an outlier; we do not have enough evidence to say that 1993 is an outlier.

Let's see what happens if we drop 1993 from the data set:

```r
# data frame with 1993 removed
crime2 <- subset(crime,Crime!=1993 )
head(crime2,5)
```

```
##      M So   Ed  Po1  Po2    LF   M.F Pop   NW    U1  U2 Wealth Ineq     Prob
## 1 15.1  1  9.1  5.8  5.6 0.510  95.0  33 30.1 0.108 4.1   3940 26.1 0.084602
## 2 14.3  0 11.3 10.3  9.5 0.583 101.2  13 10.2 0.096 3.6   5570 19.4 0.029599
## 3 14.2  1  8.9  4.5  4.4 0.533  96.9  18 21.9 0.094 3.3   3180 25.0 0.083401
## 4 13.6  0 12.1 14.9 14.1 0.577  99.4 157  8.0 0.102 3.9   6730 16.7 0.015801
## 5 14.1  0 12.1 10.9 10.1 0.591  98.5  18  3.0 0.091 2.0   5780 17.4 0.041399
##      Time Crime
## 1 26.2011   791
## 2 25.2999  1635
## 3 24.3006   578
## 4 29.9012  1969
## 5 21.2998  1234
```

```r
# shapiro-wilk test
shapiro.test(crime2$Crime)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  crime2$Crime
## W = 0.93207, p-value = 0.01001
```

With a p-value of < 0.05, this dataset is definitely not normally distributed even with one outlier removed. We can say with confidence that the Grubbs' test will not produce accurate results and under normal circumstances, should not be used on this dataset.

However, for the purpose of this exercise, let's continue to try the Grubbs' test to identify one outlier on the *left* tail.

```r
# identify one outlier (left tail)
grubbs.test(crime[,16], type=10, opposite=T)
```

```
##
##  Grubbs test for one outlier
##
## data:  crime[, 16]
## G = 1.45589, U = 0.95292, p-value = 1
## alternative hypothesis: lowest value 342 is an outlier
```

The test has identified the data point 342 as an outlier. This is accompanied by a p-value of 1. This p-value is even larger than before, and greater than 0.05. Again, we fail to reject the null hypothesis that this data point is an outlier as we do not have enough evidence to say that 342 is an outlier.

The Grubbs' test for one outlier on either tail is called with the parameter `type=11` which should return both 1993 and 342:

```
# identify one outlier on each tail (two tails)
grubbs.test(crime[,16], type=11)
```

```
##
##  Grubbs test for two opposite outliers
##
## data:  crime[, 16]
## G = 4.26877, U = 0.78103, p-value = 1
## alternative hypothesis: 342 and 1993 are outliers
```

Finally, we try `type=20` to identify two outliers on the right tail.

```
# identify two outliers on one tail (right tail)
#grubbs.test(crime[,16], type=20)
```

Unexpectedly, this code chunk has thrown an error (I have commented it out, so it doesn't error during the knitting of this document). Here is the error message:

`Error in qgrubbs(q, n, type, rev = TRUE) : n must be in range 3-30`

It appears that the `grubbs.test` function for 2 outliers on one tail only works on datasets of length 3-30. As our dataset has 47 rows, we are not able to run this code.

As expected, using `grubbs.test` on a non-uniform distribution, we are unable to accept a single outlier at the min or max of our crime rate data, and we are unable to test for two outliers at the max due to the size of our dataset.

To identify outliers for a non-uniformly distributed dataset, it is instead recommended to use the Tietjen-Moore test, or the generalized extreme studentized deviate test.

Question 6.1 Describe a situation or problem from your job, everyday life, current events, etc., for which a Change Detection model would be appropriate. Applying the CUSUM technique, how would you choose the critical value and the threshold?

An asset management company can use CUSUM to monitor portfolios under their management.

Subtle changes in a portfolio's control chart can indicate that a portfolio is at risk, or that a portfolio is benefiting positively from a change in investment strategy.

I think both the critical and threshold value would be affected by the client's risk appetite, where a higher risk appetite means a higher critical value/threshold.

Question 6.2.1 Using July through October daily-high-temperature data for Atlanta for 1996 through 2015, use a CUSUM approach to identify when unofficial summer ends (i.e., when the weather starts cooling off) each year. You can get the data that you need from the file `temps.txt` or online, for example at http://www.iweathernet.com/atlanta-weather-records or https://www.wunderground.com/history/airport/KFTY/2015/7/1/CustomHistory.html. You can use R if you'd like, but it's straightforward enough that an Excel spreadsheet can easily do the job too.

Ref: https://www.rdocumentation.org/packages/qcc/versions/2.6/topics/cusum

We are going to use the `qcc` package for to carry out the CUSUM portion of question, which help from functions in the `reshape2` and `dplyr` packages.

```
library(qcc)
```

```
## Package 'qcc' version 2.7

## Type 'citation("qcc")' for citing this R package in publications.
```

```
library(reshape2)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
temps <- read.delim("../week 2 data-summer/temps.txt")
head(temps,5)
```

```
##     DAY X1996 X1997 X1998 X1999 X2000 X2001 X2002 X2003 X2004 X2005 X2006 X2007
## 1 1-Jul    98    86    91    84    89    84    90    73    82    91    93    95
## 2 2-Jul    97    90    88    82    91    87    90    81    81    89    93    85
## 3 3-Jul    97    93    91    87    93    87    87    87    86    86    93    82
## 4 4-Jul    90    91    91    88    95    84    89    86    88    86    91    86
## 5 5-Jul    89    84    91    90    96    86    93    80    90    89    90    88
##   X2008 X2009 X2010 X2011 X2012 X2013 X2014 X2015
## 1    85    95    87    92   105    82    90    85
## 2    87    90    84    94    93    85    93    87
## 3    91    89    83    95    99    76    87    79
## 4    90    91    85    92    98    77    84    85
## 5    88    80    88    90   100    83    86    84
```

For convenience, we will want to modify the table a bit and do some data exploration.

```
# wide to long
temps2 <- melt(temps, id.vars = c("DAY"),variable.name = "YEAR",value.name="TEMP")
```

Let's get some basic descriptive statistics, such as the mean and standard deviation temperature of every year (`mean_yearly` and `sd_yearly`), which we will be using in the next section.

```r
# yearly average temperature
mean_yearly<-aggregate(TEMP~YEAR, data=temps2, mean)
# yearly standard deviation
sd_yearly<-aggregate(TEMP~YEAR, data=temps2, sd)
```

The `cusum()` function only works with numerical or boolean data, so we have to drop the first date column via `temp[,2:21`. Save the new, entirely numerical data in a new dataframe `temps3`. We will map the indices back to their corresponding dates later.

The `decision.interval` parameter represents the threshold T in terms of the number of standard deviations from the mean. In this case, we have chosen the threshold to be 2 standard deviations. 2 standard deviations from the mean encompasses 95% of the entire distribution, and should be sufficient to detect any significant changes in temperature.

The `se.shift` parameter represents the critical value c in terms of the amount of standard error. We will set this at 3.

We will create a CUSUM model for every year in the data. From there, we can extract all instance of threshold breaches for the year with `models[[i]]$violations$lower`, which we will save to the `breaches` vector.

Set `plot=F` to prevent the `cusum()` function from generating a plot for every year (20 plots in total!). If you are interested to see how the CUSUM plot looks like in R, there is a single individual example in Q6.2.2.

```r
# remove date column
temps3 <- temps[,2:21]

# define variables
models <- vector(mode="list", length=ncol(temps3))
breaches <- vector(mode="list", length=ncol(temps3))

DI = 2 # flag breaches at 2 std dev
Shift = 3

# loop over years
for (i in 1:ncol(temps3)) {
  models[[i]] <- cusum(temps3[,i],
                       center=mean_yearly$TEMP[i],
                       std.dev=sd_yearly$TEMP[i],
                       decision.interval=DI,
                       se.shift=Shift,
                       plot=F) #no graph plotted

  # store any breach datapoints in vector
  breaches[[i]] <- models[[i]]$violations$lower
}
```

Each element of `breaches` is a *list* of the row indices of data points that have breached the threshold. We can demonstrate this below with a preview:

```r
breaches[1:5]
```

```
## [[1]]
## [1]  99 101 103 104 105
##
```

```
## [[2]]
##  [1] 110 111 114 115 116 117 118 119 120 121 122 123
##
## [[3]]
## [1] 114 115 116 117 118 119 120
##
## [[4]]
## [1] 115 116 117 118 119 120 121
##
## [[5]]
## [1] 101 102 103 104 105 106
```

```r
# count the number of years that had flagged datapoints
counter = 0
year_index = vector("numeric")

for (i in 1:length(breaches)){
  if (lengths(breaches[i]) != 0){
    counter = counter+1
    year_index <- c(year_index, i)
  }
}
counter
```

```
## [1] 20
```

```r
year_index
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
```

However, we do not need *all* the breaches, just the first breach of every year. Let's get the row index of the first breach of every year and store it in the vector `date_index`.

```r
# looking for the first breach in each year and when it occurs
date_index_vec <- vector(mode="list", length=counter)

# loop through year index (index 1 = 1996, index 2 = 1997, etc.)
for (i in year_index){
  # first breach of the year
  date_index_vec[i] <- breaches[[i]][1]
}

date_index <- unlist(date_index_vec)
date_index # index of the date when the first breach occurs
```

```
##  [1]  99 110 114 115 101 110 109 120 106 116 116 117 116 109  96 113 121 116 123
## [20] 119
```

Then, we can map the row index of each breach back to the corresponding dates from the original `temps` dataframe.

```
# mapping row/date indices to the actual dates
lowtemp_c <- date_index

df_reconnect <- temps %>%
                mutate(Low.expense.Index = row_number()) %>%
                select("DAY","Low.expense.Index")
df_lowtemp_days<- slice(df_reconnect, lowtemp_c)

# dates of the first breach (each row = 1 year)
df_lowtemp_days
```

```
##       DAY Low.expense.Index
## 1   7-Oct                99
## 2  18-Oct               110
## 3  22-Oct               114
## 4  23-Oct               115
## 5   9-Oct               101
## 6  18-Oct               110
## 7  17-Oct               109
## 8  28-Oct               120
## 9  14-Oct               106
## 10 24-Oct               116
## 11 24-Oct               116
## 12 25-Oct               117
## 13 24-Oct               116
## 14 17-Oct               109
## 15  4-Oct                96
## 16 21-Oct               113
## 17 29-Oct               121
## 18 24-Oct               116
## 19 31-Oct               123
## 20 27-Oct               119
```

Taking the median of the dates of all these breaches, we get a value of 114.5, which corresponds to a date of 22-23 Oct as the end of summer.

```
median(date_index)
```

```
## [1] 114.5
```

Question 6.2.2 Use a CUSUM approach to make a judgment of whether Atlanta's summer climate has gotten warmer in that time (and if so, when).

The advice from officer hours was to measure this metric by the length of summer. This is because warmer summers mean a longer time for the temperature to drop, which results in a longer time taken for a breach in the lower bound of temperatures.

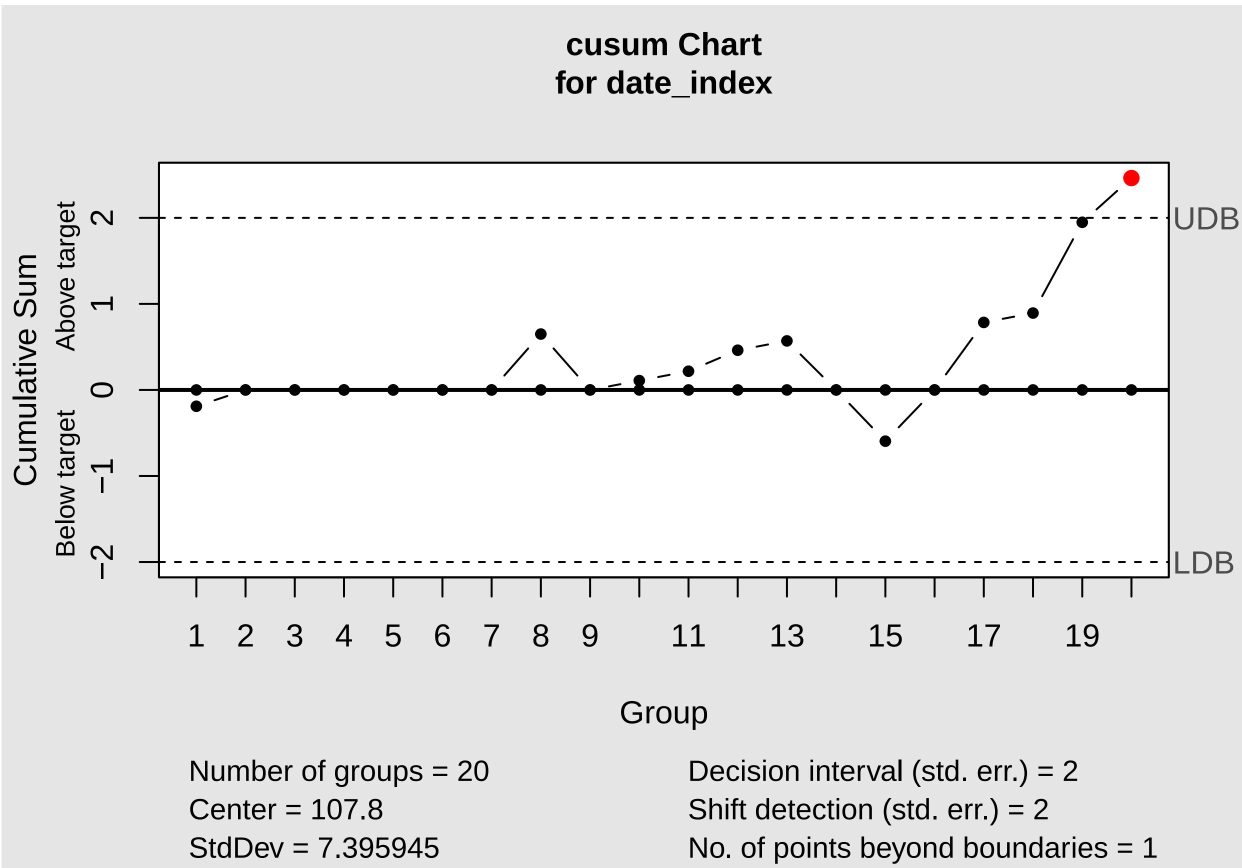
From Q6.2.1, we can take the first breach each year as the last day of summer. These dates are stored in `date_index` as row indices.

Remember that this row index is basically equivalent to the number of days since 1st July, which corresponds to the number of days in our definition of summer.

I have chosen to keep `decision.interval` at 2 standard deviations according to my reasonings above. However, `Shift` has been reduced to 2 standard errors. The number of days in summer is a smaller range than the range of temperatures in summer, thus I believe this merits a more sensitive model and thus a lower `Shift` value.

Additionally, we should not construct the CUSUM model based on the mean of the entire data in the event that summer indeed has gotten warmer. Instead, we will take the mean and standard deviation of the first 5 years as the norm and measure subsequent summers against this standard.

```
# take first 5 years of data as 'normal' - use this as the mean and standard deviation
cusum(date_index,
      center=mean(date_index[1:5]),
      std.dev=sd(date_index[1:5]),
      decision.interval=2,
      se.shift=2)
```




```
## List of 14
##  $ call             : language cusum(data = date_index, center = mean(date_index[1:5]), std.dev = sd
##  $ type             : chr "cusum"
##  $ data.name        : chr "date_index"
##  $ data             : int [1:20, 1] 99 110 114 115 101 110 109 120 106 116 ...
##   ..- attr(*, "dimnames")=List of 2
##  $ statistics       : Named int [1:20] 99 110 114 115 101 110 109 120 106 116 ...
##   ..- attr(*, "names")= chr [1:20] "1" "2" "3" "4" ...
##  $ sizes            : int [1:20] 1 1 1 1 1 1 1 1 1 1 ...
##  $ center           : num 108
##  $ std.dev          : num 7.4
##  $ pos              : num [1:20] 0 0 0 0 0 ...
```

```
##  $ neg             : num [1:20] -0.19 0 0 0 0 ...
##  $ head.start      : num 0
##  $ decision.interval: num 2
##  $ se.shift        : num 2
##  $ violations      :List of 2
##  - attr(*, "class")= chr "cusum.qcc"
```

Based on the critical value and threshold that we have chosen, we can safely say that Atlanta's summer climate has gotten warmer within the time range of 1996-2015.