Code

Iris Data Clustering, Outlier analysis, and Change Detection

Mike Huang

May 28, 2017

Part 1

Prompt

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.

Response

There are various types of physicians that use electronic medical records(EMR) and varying levels of adoption. We need to be able to see which physicians belong in a similar bucket so that we can compare how efficient they are to their peers. To do so, clustering would be a great way. The features we would look at include the following:

- provider's credentials
    - Provider's training can determine what type of work they do.
    - examples: MD, DO, APN, NP, etc
- provider's specialty
    - Different specialties will have different work they need to complete in the EMR.
    - examples: Family med, cardio, neurology, etc.
- provider's organization type
    - Different organizations have different focuses. This can result in different EMR system builds that can differ from one to another
    - examples: pediatric, safety-net, non-profit, etc.
- provider's organization revenue
    - more money means more project which means more staff to handle them and better functionalities will be implemented.

Part 2

Prompt

The iris data set 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.

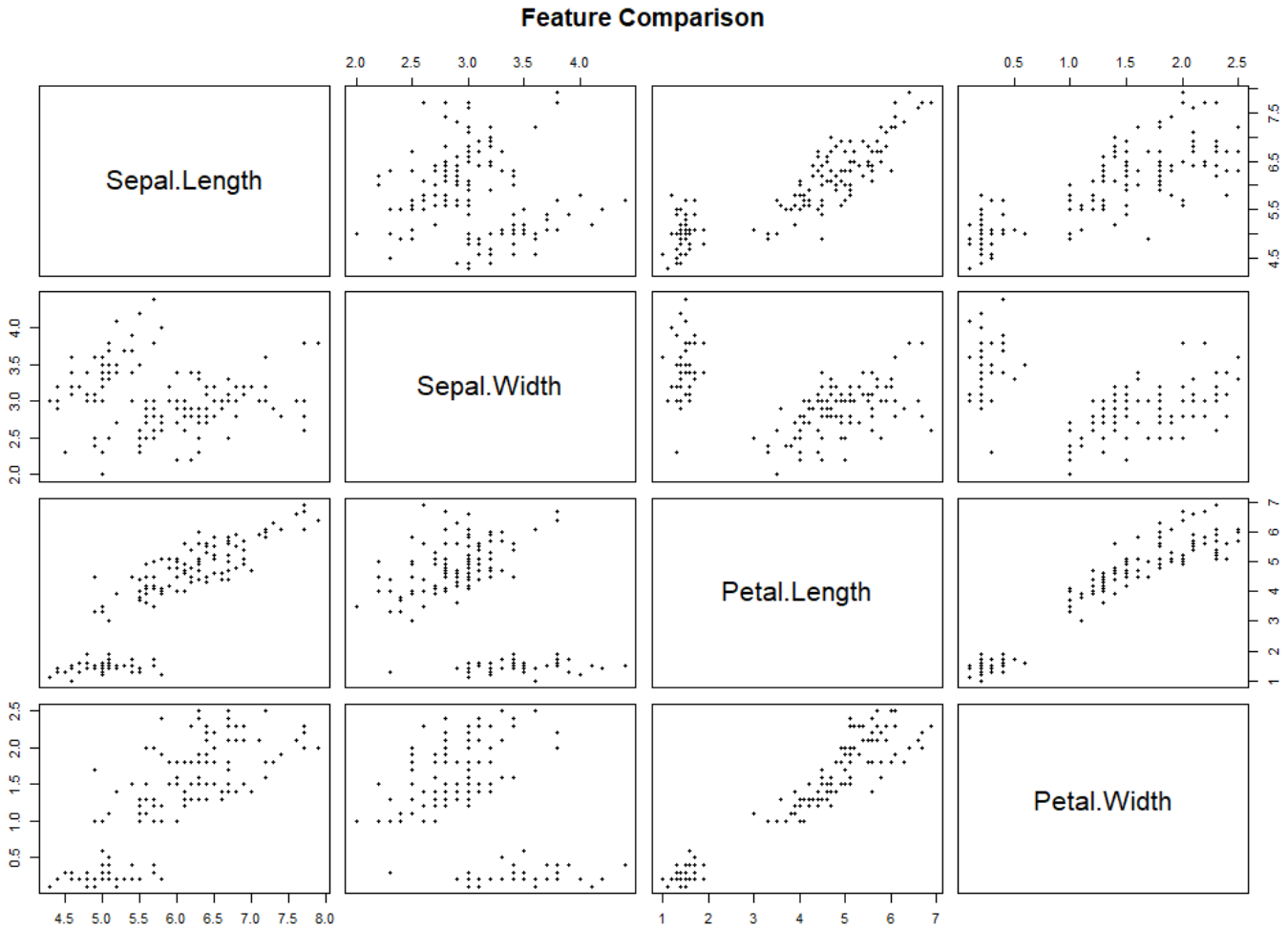
Response

Viewing the Data

Before doing anything, I want to see what the data actually looks like and whether we'll be able to clearly discern certain categories within the data without actually seeing the labels. So I'm just doing a basic plot of plot out the comparison between each feature.

Hide

```
#Load Libraries
library(cluster)  #using kmeans from this package
library(ggplot2)
library(GGally)
library(grid)
library(gridExtra)
#Load and separate data
irisTbl <- read.table('iris.txt')
irisFeat <- irisTbl[,1:4]
irisLab <- irisTbl[,5]
irisLabNum <- factor(irisLab,labels=c(2,3,1),levels=c("setosa","versicolor","virginica"))
#View the features
plot(irisFeat,pch=20,main="Feature Comparison")
```

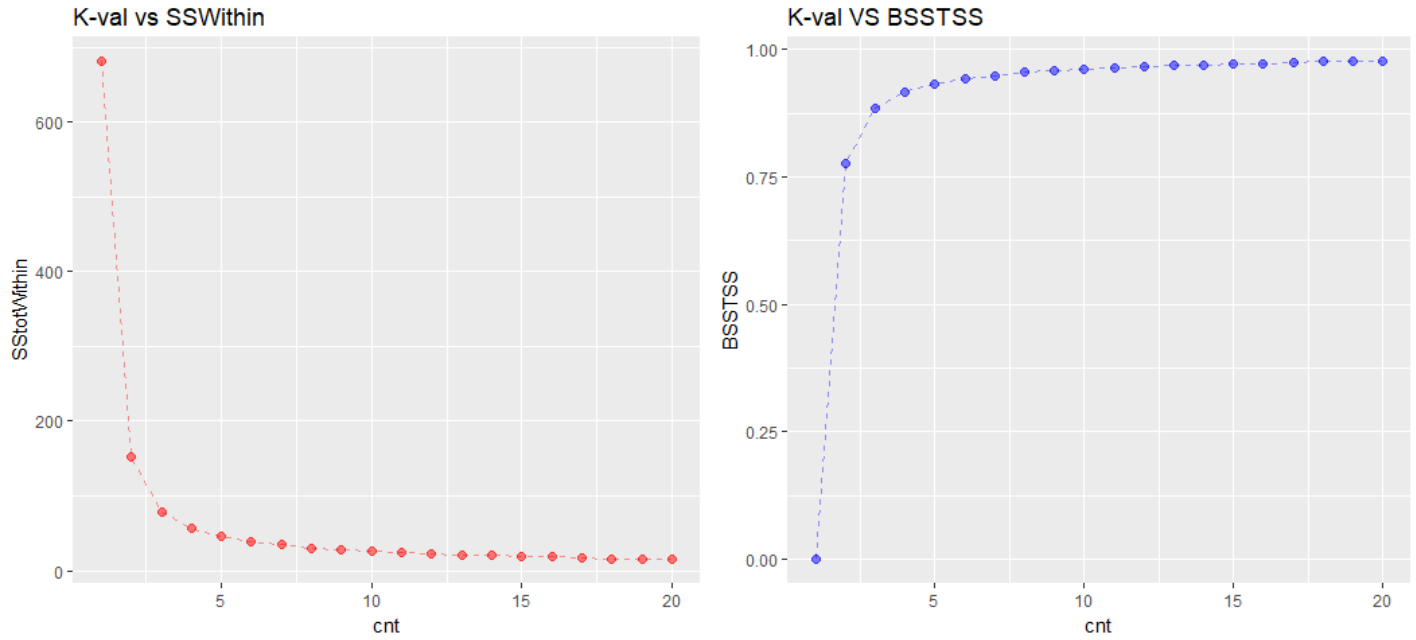


Determine optimal K value

Most of what I really see is two different features between all of this data. It seems like the Petal Length and Petal Width probably helps separate the data out the best with Sepal Length and Width being a bit more mixed. Let's pretend that we do not have the labels so we will need to decide for ourselves how many clusters there are. Additionally, according to the question, we weren't given any requirements around the number of flowers types provided to us so that leaves us with the elbow joint method of determining the most optimal number of clusters.

Hide

```
SStotWithin <- vector()
BSSTSS <- vector()
#Loop through k values of 1 to 20
for (i in 1:20)
{
  model=kmeans(irisTbl[,1:4],centers=i,iter.max=100,nstart=10)
  #Store off total sum of square within
  SStotWithin[i]=model$tot.withinss
  #Store off ratio of sum of squares between over total sum of squares
  BSSTSS[i]=model$betweenss/model$totss
}
#setup plot
cnt=1:20
KSSwithin=data.frame(cbind(cnt,SStotWithin))
KBTSS=data.frame(cbind(cnt,BSSTSS))
p1 <-
ggplot(KSSwithin,aes(cnt,SStotWithin))+geom_point(colour="red",size=2,alpha=.5)+labs(title="K-va
l vs SSWithin")+geom_line(alpha=.4,color="red",linetype=2)
p2 <- ggplot(KBTSS,aes(cnt,BSSTSS))+geom_point(colour="blue",size=2,alpha=.5)+labs(title="K-val
 VS BSSTSS")+geom_line(alpha=.4,color='blue',linetype=2)
grid.arrange(p1,p2,ncol=2)
```



In the above graph, I took a look at two different values:

- **Total Sum of Square(SS) within**: This is calculated by determining the sum of square for each cluster and summing it up
- **Ratio of SS Between and SS Total**: This ratio helps us determine how much of the variance comes from the separation of the clusters vs how well the grouping is in each cluster. As the number of clusters get closer to the number of points, this value goes to 1

Both of these graphs can be used to determine the best location for the elbow joint. Just eyeballing this graph, I get the feeling that the joint is probably 3. So this is my guess of most optimal number of clusters to use.

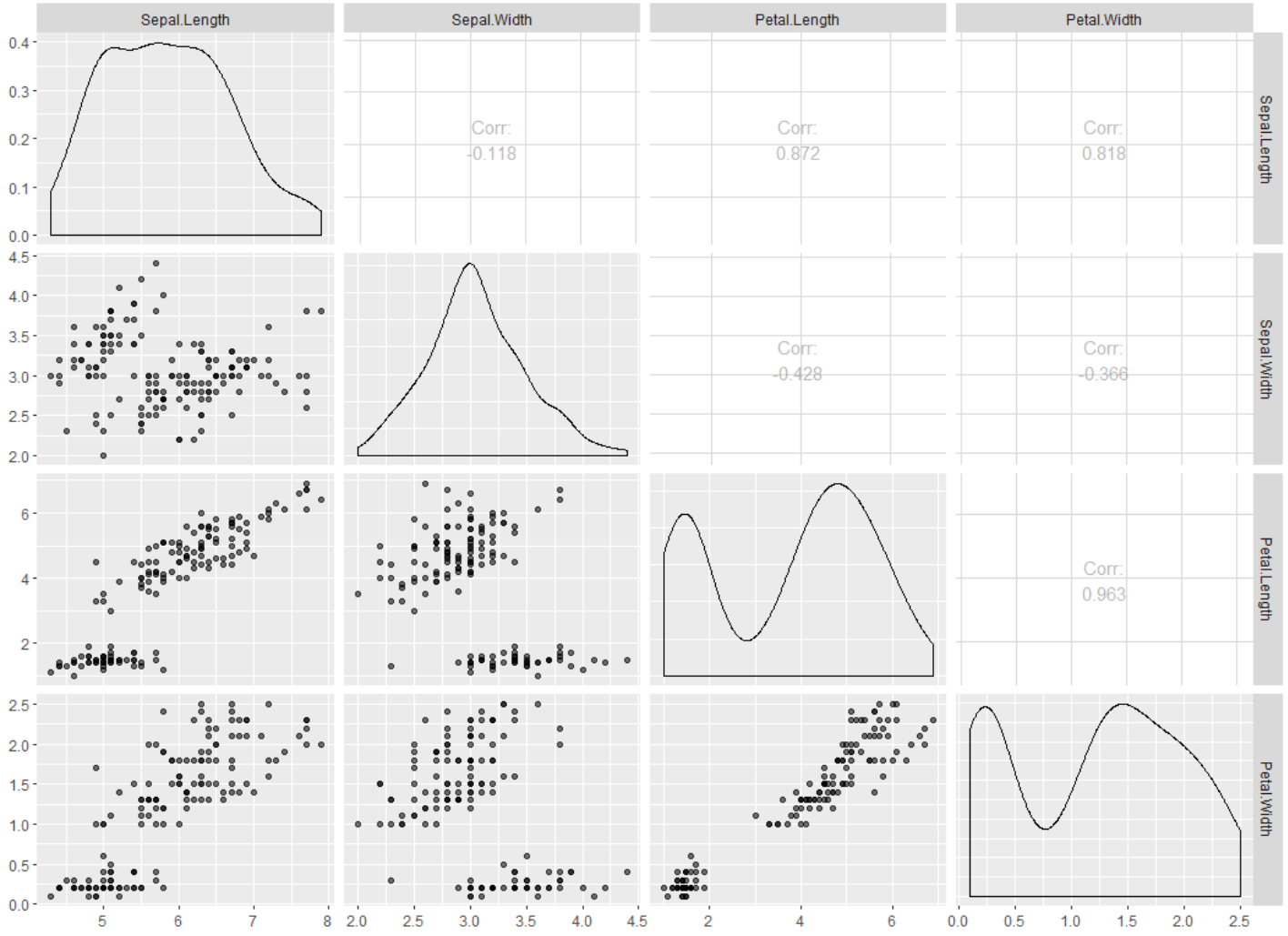
To get better understanding of how these are calculated, check out the youtube video on Total Sum of Squares (https://www.youtube.com/watch?v=EFdlFoHI_0I) and Sum of Squares Within and Between (https://www.youtube.com/watch?v=EFdlFoHI_0I).

Feature selection

Now we need to look at each individual feature to see whether they are helpful in separating out data to an individual class. I'm using ggpairs from GGally library to plot out each column with respect to each other and itself to see how well the data separates.

Hide

```
ggp=ggpairs(irisFeat,aes(alpha=0.8))
print(ggp,progress = F)
```



From these graphs, you can really see a huge difference. between the different features. Petal Length and Petal Width separates out two distinct classes out really well while Sepal Length and Sepal Width is not very distinct. Visually, I would use Petal Length and Petal Width as my features for clustering.

Accuracy

My guess is around using just the Petal Length and Width being the most accurate predictors. For this test, I'm going to sample a few combinations to see how they do but most of them are focused on using Petal Lenth and Width.

Hide

```
#Petal Length and Width
model1=kmeans(irisTbl[,3:4],centers=3,iter.max=100,nstart=10)
#Petal Length and Width, Sepal Length
model2=kmeans(irisTbl[,c(1,3,4)],centers=3,iter.max=100,nstart=10)
#Petal Length and Width, Sepal Width
model3=kmeans(irisTbl[,c(2,3,4)],centers=3,iter.max=100,nstart=10)
#Petal Length and Width, Sepal Length and Width
model4=kmeans(irisTbl[,c(1,2,3,4)],centers=3,iter.max=100,nstart=10)
#Just for kicks, Sepal Length and Width.
model5=kmeans(irisTbl[,c(1,2)],centers=3,iter.max=100,nstart=10)
PetLW=irisLab
PetLW_SepL=irisLab
PetLW_SepW=irisLab
PetLW_SepLW=irisLab
SepLW=irisLab
table(PetLW,model1$cluster)
```

```
PetLW         1  2  3
  setosa     50  0  0
  versicolor  0 48  2
  virginica   0  4 46
```

Hide

```
table(PetLW_SepL,model2$cluster)
```

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

Hide

```
table(PetLW_SepW,model3$cluster)
```

```
PetLW_SepW    1  2  3
  setosa      0  0 50
  versicolor  2 48  0
  virginica  45  5  0
```

Hide

```
table(PetLW_SepLW,model4$cluster)
```

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

Hide

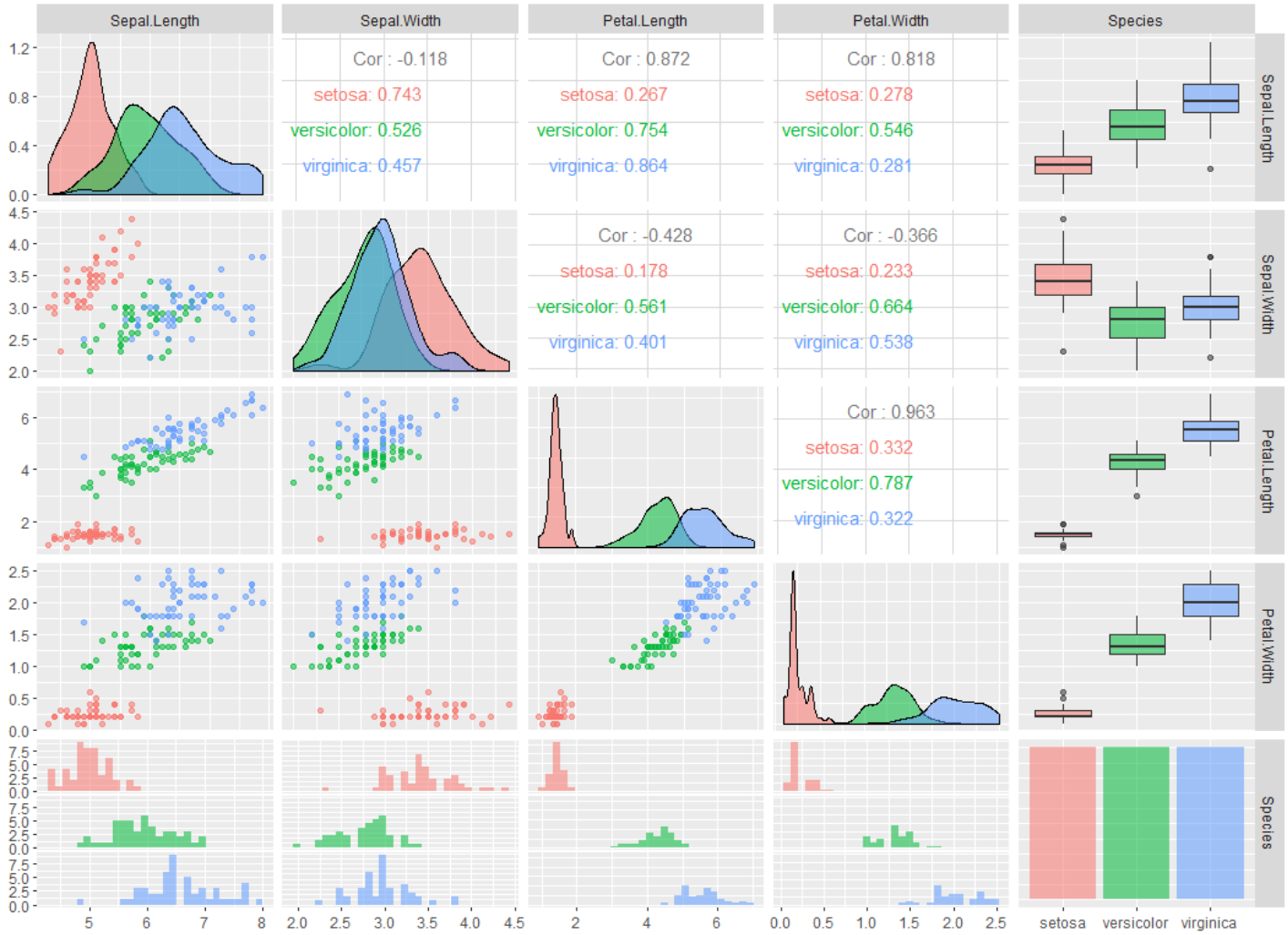
```
table(SepLW,model5$cluster)
```

```
SepLW         1  2  3
  setosa     50  0  0
  versicolor  0 12 38
  virginica   0 35 15
```

From the tables above, we can really see that Petal Length and Width really are the best classifiers with the additional of Sepal Width also performing quite well. As for the other combinations, classifying Virginica correctly has been the most difficult thing to do. So to get a better graphical view of the data, we can use ggpairs with the full dataset including the labels to really discern which features separate our data the best!

Hide

```
ggp=ggpairs(irisTbl,aes(color=Species,alpha=0.3))
print(ggp,progress = F)
```



With this graphical tool, we can really see that with each feature setosa is extremely easy to separate out while versicolor and virignica are much harder to separate. Looking at the table information we have above, we could see that Sepal Width really doesn't add anything to the data but also doesn't really add bad noise. Meanwhile, Sepal Length does actually contribute to the noise and hurts the classifier in separating the datasets.

Hide

```
#cleanup environment
rm(list=ls(all=TRUE))
```

Part 3

Prompt

Using crime data from statsci (http://www.statsci.org/data/general/uscrime.txt), test to see whether there is an outlier in the last
column (number of crimes per 100,000 people). Is the lowest-crime city an outlier? Is the highest-crime city an outlier? Use the grubbs.test function in the outliers package in R. check out uscrimehtml (http://www.statsci.org/data/general/uscrime.html) for description of the data.

Response

Loading the data

Hide

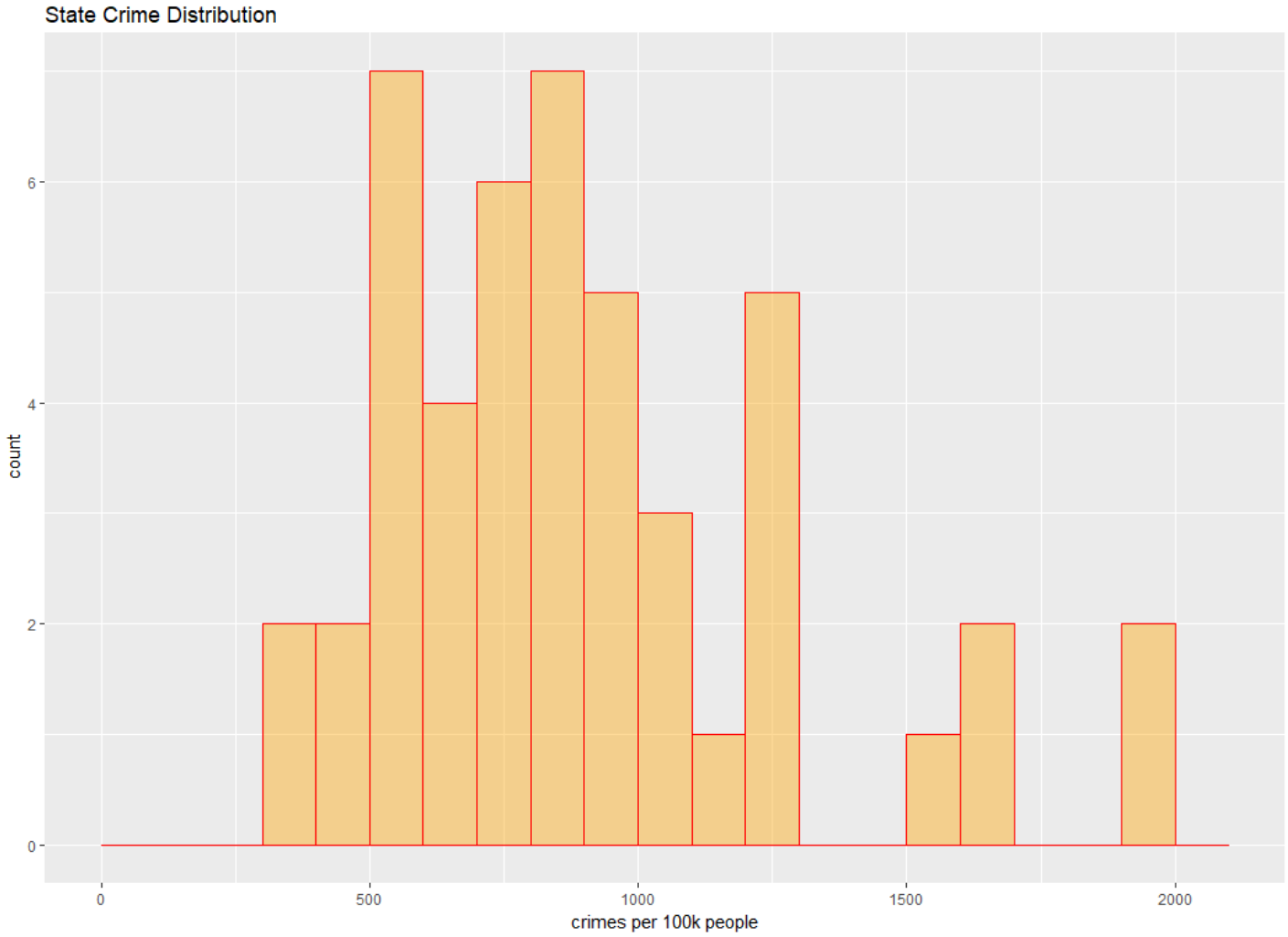
```
library(outliers)  #using grubbs.test from this package
library(ggplot2)
#Load and separate data
crimeTbl <- read.table('uscrime.txt',header=TRUE)
crimeDF <- data.frame(crimeTbl['Crime'])
crimeVec <- as.vector(crimeTbl['Crime'])
```

Viewing Data

Grubbs' test has an assumption of normality so before we even apply Grubbs' test, we need to make sure that our data can be approximated by a normal distrubtion.

Hide

```
ggplot(data=crimeDF,aes(Crime)) + geom_histogram(breaks=seq(0,2100,by=100),col="red",fill="orang
e",alpha=.4) + labs(x="crimes per 100k people",title="State Crime Distribution")
```



Visually, we can already see some of the data that seems to be further away from what our actual distribution should be. As for our distribution, it looks gaussian enough that we can probably run the grubbs test on it.

Grubbs' test

Quick background on Grubbs' test. This is a test that checks whether your maximum value or minimum value is an outlier or not. It does not determine which points are outliers and which ones are not. If you suspect you have multiple outliers, you will want to use generalized extreme studentized deviate test or Tietjen-Moore test instead. Grubbs test uses the following formulas for a single tailed test:

$$G_{test} = \frac{\overline{Y} - Y_{min}}{\delta}$$

$$G_{test} = \frac{Y_{max} - \overline{Y}}{\delta}$$

Legend

- $\overline{Y}$: mean value of the dataset
- Y_{max}: max value of the dataset
- Y_{min}: min value of the dataset
- δ: standard deviation of the dataset
- G_{test}: G test statistic

Once you've found the G test statistic, you need to compare it to a G critical value to see whether or not we reject the fact that the datapoint is an outlier.

- $G_{test} < G_{critical}$: keep the point in the data set; it is not an outlier
- $G_{test} > G_{critical}$: reject the point as an outlier

You can find more information on grubbs-test site (http://www.statisticshowto.com/grubbs-test/) and the G critical values site (http://www.sediment.uni-goettingen.de/staff/dunkl/software/pep-grubbs.pdf).

Is max value an outlier?

Here we perform the test with the type set to 10. This is the first test which is used to detect if the sample dataset contains one outlier, statistically different than the other values.

Hide

```
gtHigh=grubbs.test(as.matrix(crimeVec),type=10)
print(gtHigh)
```

```
    Grubbs test for one outlier

data:  as.matrix(crimeVec)
G = 2.81290, U = 0.82426, p-value = 0.07887
alternative hypothesis: highest value 1993 is an outlier
```

Looking at the chart from the G critical values site above, we can see that a p value of 0.075 and a N value of 48 gives us a value of 0. Since our G value was 2.81290, it is larger than the critical value so the max value in our dataset is not considered an outlier.

Is min value an outlier?

Here we perform the test with the opposite value set to true so that we can check whether the lowest value is considered an outlier.

Hide

```
gtLow=grubbs.test(as.matrix(crimeVec),type=10,opposite=TRUE)
print(gtLow)
```

```
    Grubbs test for one outlier

data:  as.matrix(crimeVec)
G = 1.45590, U = 0.95292, p-value = 1
alternative hypothesis: lowest value 342 is an outlier
```

We can see that p-value is a solid 1 which basically means we cannot reject our null hypothesis which is the lowest value in our dataset is not an outlier.

Hide

```
#cleanup environment
rm(list=ls(all=TRUE))
```

Part 4

Prompt

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?

Response

CUSUM technique can be applied to climate change. We can break apart the year into 4 quarters and apply CUSUM separately to the average of each quarter. Using historical data(pre-2000), we can set our critical value based on 2 standard deviations away from the mean. Next, we'll have to plot our CUSUM graph and set a threshold just inside 1.5 standard deviations from our CUSUM data. I'm using this value because I believe we should be on the watch for large temperature fluctuations and actively monitor this. At a threshold of 2 standard deviations, I think it would be getting to the point of being to late. But we might be there already.

Part 5

Prompt

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. That involves finding a good critical value and threshold to use across all years. You can get the data that you need online, for example at iweathernet (http://www.iweathernet.com/atlanta-weather-records) or wunderground (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.
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).

Response

For this segment, I will be using the getWeatherForDate API to pull data from wunderground. I've pulled a period of June 1998 to November 1998 to just take a look at the data.

Load Data

Hide

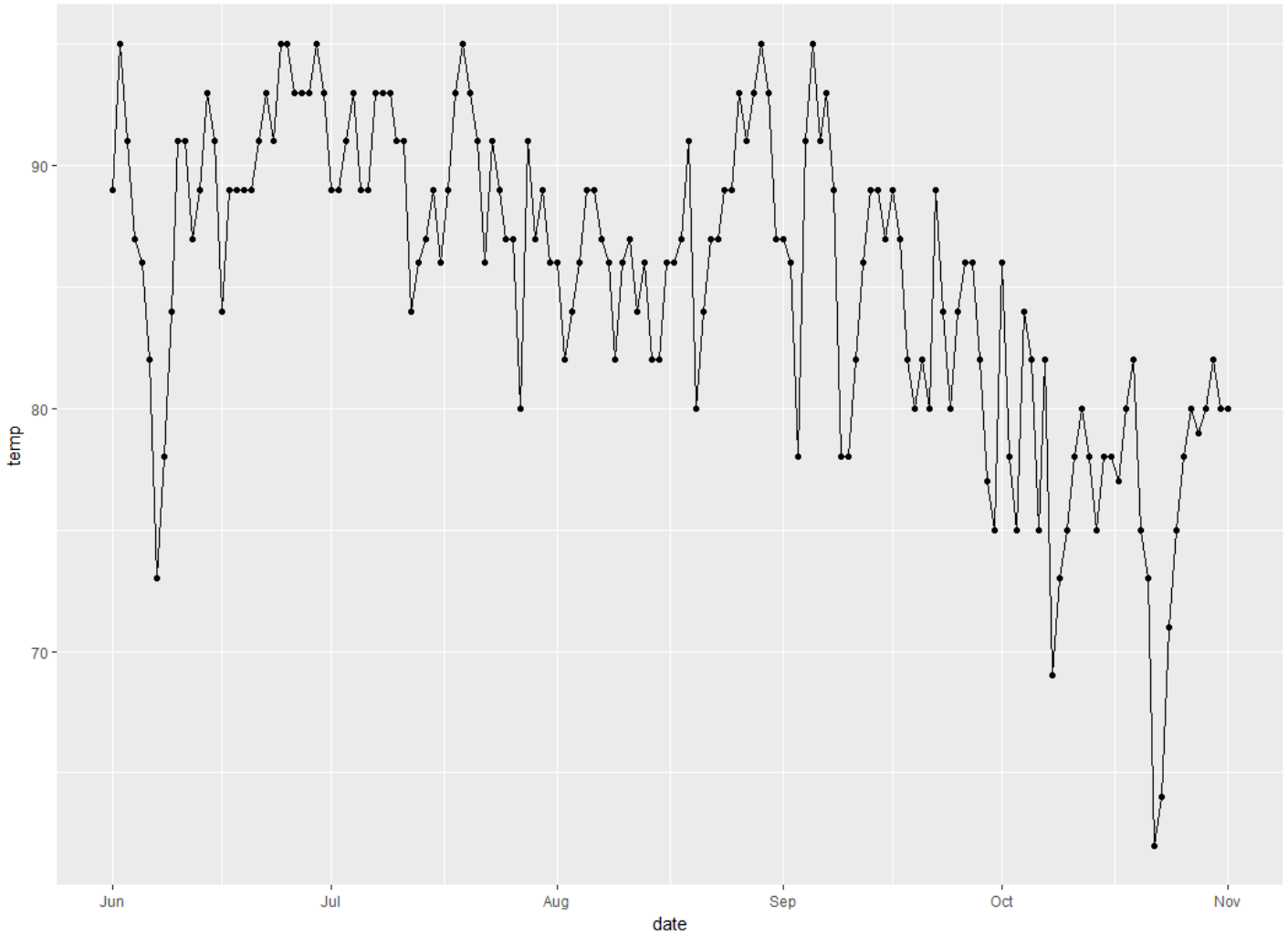
```
library(weatherData)
library(ggplot2)
library(reshape2)
#Load data and suppress some outputs
weatherTbl <- getWeatherForDate("KFTY","1998-06-01",end_date="1998-11-1")
#format data and plot
weatherDF <- data.frame(weatherTbl)
date <- as.Date(as.matrix(weatherDF["Date"]))
temp <- as.vector(weatherDF["Max_TemperatureF"])
```

View Sample

Hide

```
qplot(date,temp,aes()) + geom_line()
```

```
Ignoring unknown parameters: NA
```



Looking at this graph, I would guess the temperatures are stable between middle of June to middle of September. But to play it safe, we'll designate the "norm" to be July 1st and Sept 1st.

Generate table containing all data

I'm going to collect all the data into a single matrix where the first column is the date and the rest of the columns are the highest temperature during every year. I've skipped 1996 and 2000 as there have been some corrupt data.

Hide

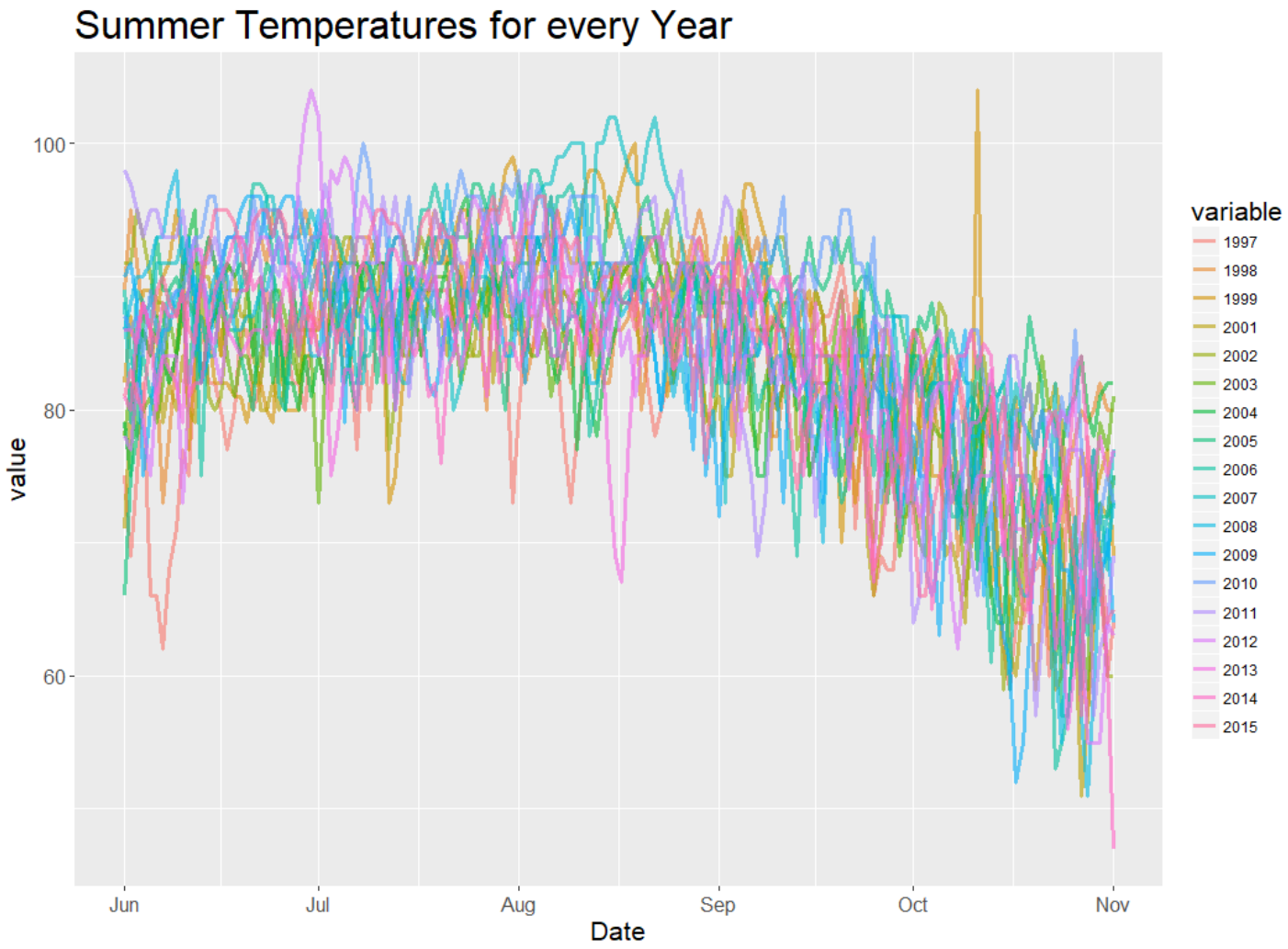
```
yrAry <- 1997:2015
for (i in 1:length(yrAry))
{
  #set the year for this loop
  yr <- yrAry[i]

  #skipping year 2000 due to corrupt data
  if (yr==2000)
  {next}

  #concat year with dates for the loop
  startDate <- paste0(yr,"-06-01")
  endDate <- paste0(yr,"-11-1")

  #pull data from wunderground
  weatherTbl <- getWeatherForDate("KFTY",startDate,end_date=endDate)
  wDF <- data.frame(weatherTbl)

  #if it's the first value, setup the dataframe
  if (i == 1)
  {
    weatherDF <- wDF[,c('Date','Max_TemperatureF')]
    names(weatherDF)[names(weatherDF)=='Max_TemperatureF'] <- yr
  }

  #if it's not the first value, bind the columns
  else
  {
    weatherDF <- cbind(weatherDF,wDF['Max_TemperatureF'])
    names(weatherDF)[names(weatherDF)=='Max_TemperatureF'] <- yr
  }
}
```

Visualize the temperature for every year

Here I can melt the cusumDF and plot the cusum vector for each year.

Hide

```
weatherDF$Date <- as.Date(weatherDF$Date)
ggplot(melt(weatherDF,id.vars="Date"),aes(Date,value,color=variable))+geom_line(alpha=.6,size=1.
3)+labs(title="Summer Temperatures for every
Year")+theme(title=element_text(size=20),axis.text=element_text(size=12),axis.title=element_text(s
ize=16),legend.title=element_text(size=16))
```



CUSUM function

Here I wrote my CUSUM function to calculate a vector a cusum values from an input vector. This also takes in a C values that you can tune.

Hide

```
cusum <- function(x,C)
{
  x <- unlist(x)
  ary <- vector()
  for (i in 1:length(x))
  {
    if (i==1)
    {
      ary[i] <- min(0,x[i])
    }
    else
    {
      ary[i] <- min(0,ary[i-1]+x[i]+C)
    }

  }
  return(ary)
}
```

Applying the CUSUM

There's multiple steps taken here. First, I create a matrix with only summer dates and then found the mean and standard deviation for those dates. Next I calculated the temperature deviation from the mean for each year and then applied my custom cusum formula to it. Finally, I formatted this data frame for viewing.

Hide

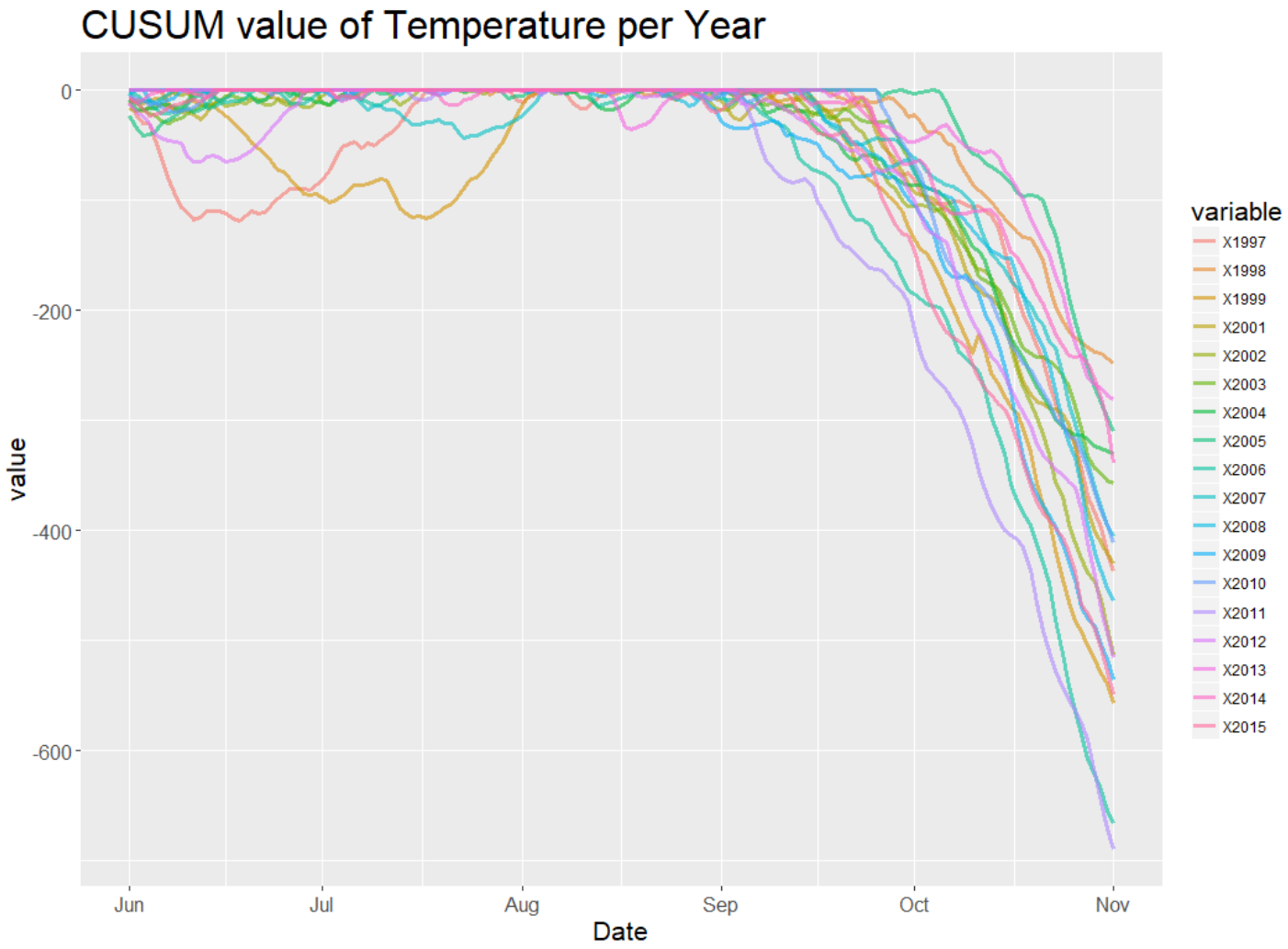
```
#Create a matrix with only the "summer dates"
summerDF <- weatherDF[(weatherDF['Date']>"1997-07-01")&(weatherDF['Date']<"1997-09-01"),]
#Find the mean and standard deviation temperature out of all of those dates.
summerMean <- colMeans(summerDF[2:19])
summerStDev <- apply(summerDF[2:19],2,sd)
#Calculate how much the Temperature deviated from its mean every year
weatherDeviation <- sweep(weatherDF[,2:19], 2, summerMean)
#Apply the cusum to the deviation matrix to obtain a cusum matrix
weatherCusum <- apply(weatherDeviation,2,function(x) cusum(x,C=mean(summerStDev)))
#Attach the Dates to our matrix
Date <-  as.Date(as.matrix(format(weatherDF[,1],format="%m-%d")),format="%m-%d")
cusumDF <- cbind(Date,data.frame(weatherCusum))
cusumDF$Date <- as.Date(cusumDF$Date)
```

Visualize CUSUM for every year

Here I can melt the cusumDF and plot the cusum vector for each year.

Hide

```
ggplot(melt(cusumDF,id.vars="Date"),aes(Date,value,color=variable))+geom_line(alpha=.6,size=1.3)+l
abs(title="CUSUM value of Temperature per Year")+theme(title=element_text(size=20),axis.text=ele
ment_text(size=12),axis.title=element_text(size=16),legend.title=element_text(size=16))
```



From the graph above, I can see that June and July was quite cold during 1997, 1999, and 2012 but that would probably be considered outlier years. If we want to make sure that our algorithm doesn't catch those values, we can set our threshold to -120. But if we want to catch the end of summer early, we can also set the value to -50 assuming the algorithm doesn't start running until Sept 1st, after we've collected our summer averages.

Calculate End of Summer Dates for Each Year

Now we get to using the threshold to determine what is the end of summer for each year. I've cut off the data so it will use data after Sept 1st with a lower threshold of -50.

Hide

```
#still practicing with date formats,forgive the 2017 in the beginning
test <- cusumDF[cusumDF$Date>="2017-09-01",]
threshold <- -50
#loop through matrix to identify the end of summer for each year
EndOfSummer <- vector()
for (i in 2:length(test))
{
  indexAry <- which(test[i]<threshold)
  minIndex <- min(indexAry)
  endSummerDate <- test[minIndex,1]
  EndOfSummer[i-1] <- format(endSummerDate)
  #EndOfSummer[i-1] <- format(test[min(which(test[i]<threshold)),1],format="%m-%d")
}
year=colnames(test[2:length(test)])
print(cbind(year,EndOfSummer))
```

```
      year    EndOfSummer
 [1,] "X1997" "2017-09-27"
 [2,] "X1998" "2017-10-07"
 [3,] "X1999" "2017-09-21"
 [4,] "X2001" "2017-09-26"
 [5,] "X2002" "2017-09-24"
 [6,] "X2003" "2017-09-29"
 [7,] "X2004" "2017-09-20"
 [8,] "X2005" "2017-10-09"
 [9,] "X2006" "2017-09-13"
[10,] "X2007" "2017-09-29"
[11,] "X2008" "2017-09-23"
[12,] "X2009" "2017-09-17"
[13,] "X2010" "2017-09-30"
[14,] "X2011" "2017-09-08"
[15,] "X2012" "2017-09-20"
[16,] "X2013" "2017-10-09"
[17,] "X2014" "2017-09-28"
[18,] "X2015" "2017-09-22"
```