Exercise 2

Lio, Li, Zhu, Thelakkat

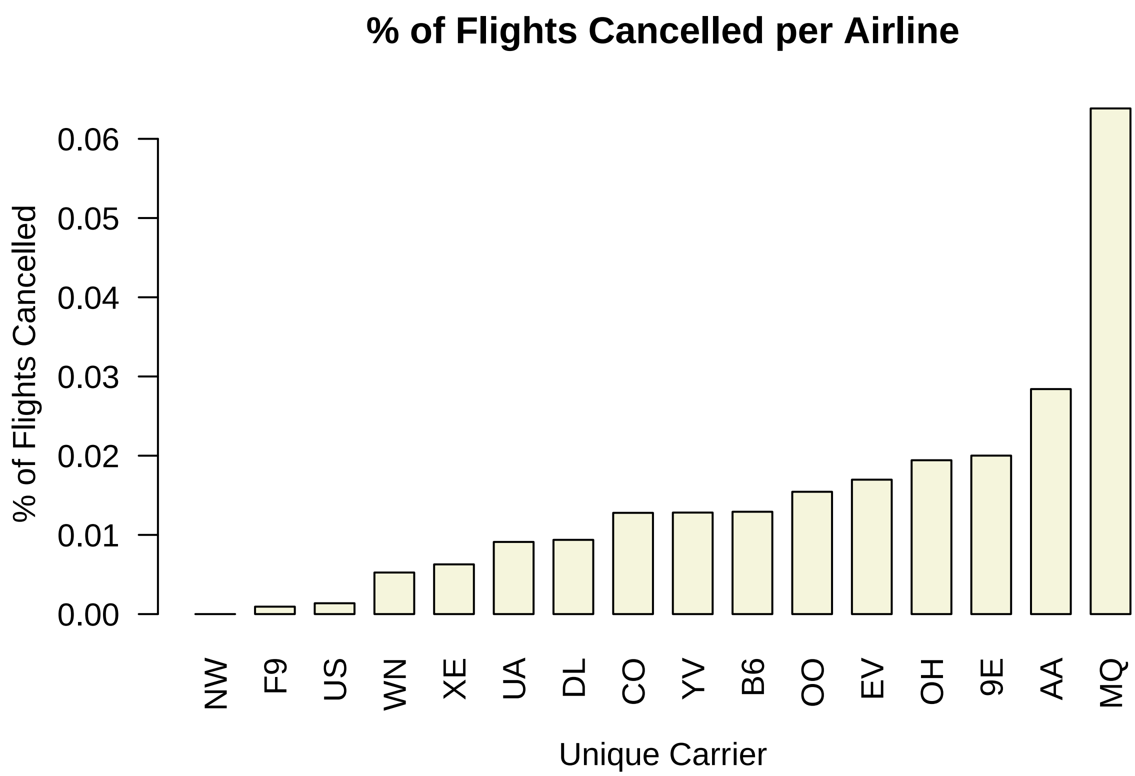
August 13, 2017

Flights at ABIA

"Your task is to create a figure, or set of related figures, that tell an interesting story about flights into and out of Austin. You can annotate the figure and briefly describe it, but strive to make it as stand-alone as possible. It shouldn't need many, many paragraphs to convey its meaning. Rather, the figure should speak for itself as far as possible."

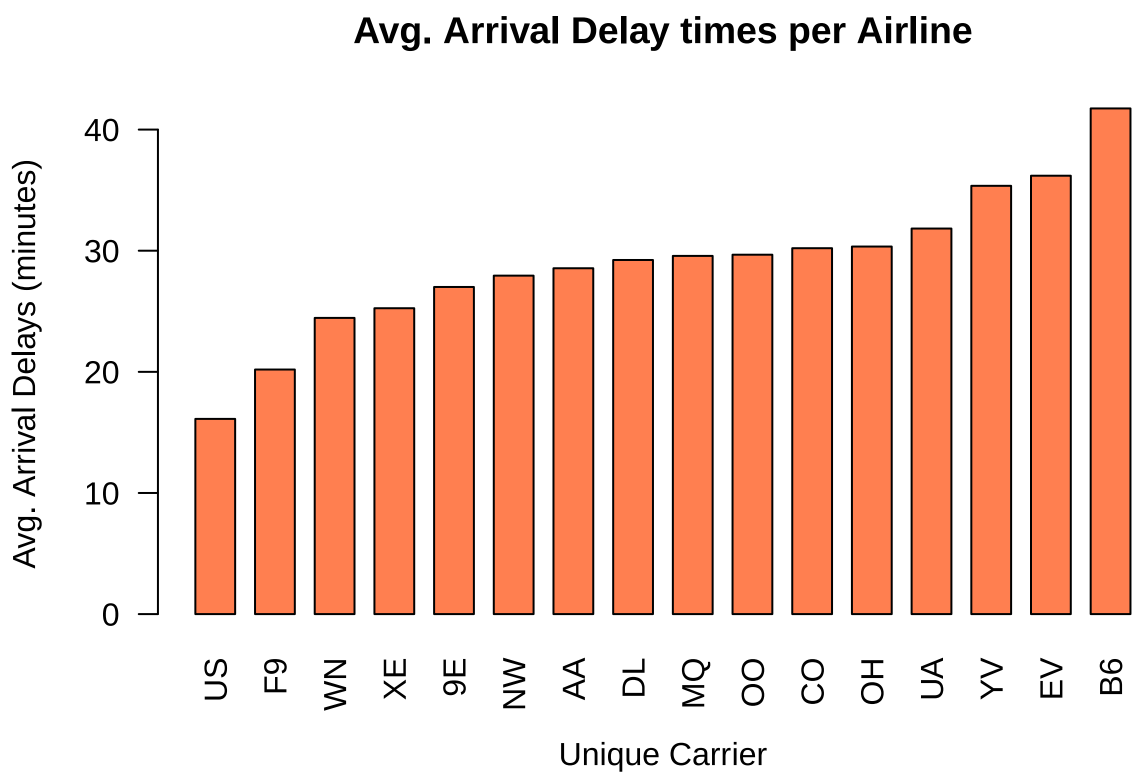
For our first section of exploratory data analysis, we decided to focus on airlines to see if we could draw any insights about which Airlines were more reliable in terms of delays and cancellations. We then looked into average arrival and average departure delay times (in minutes) each airline had when flying into or out of Austin.

```
airlinedata = read.csv("ABIA.csv")
#calculating percentage of flights cancelled for each airline

dfcancel = data.frame(aggregate(airlinedata$Cancelled~ airlinedata$UniqueCarrier,
                                airlinedata ,sum))

df = data.frame(aggregate(airlinedata$FlightNum~ airlinedata$UniqueCarrier,
                          airlinedata, length))

finaldf = merge(dfcancel, df)
finaldf = within(finaldf, percent <- airlinedata.Cancelled/airlinedata.FlightNum)
finaldf = finaldf[order(finaldf$percent),]

barplot(finaldf$percent, names = finaldf$arrivaldelays.UniqueCarrier,
        xlab = "Unique Carrier", ylab = "% of Flights Cancelled",
        main = "% of Flights Cancelled per Airline", las=2, space=.5, col='beige',
        names.arg=c("NW", "F9", "US",'WN', "XE", "UA", "DL","CO", "YV",
                    "B6","OO", "EV", "OH", "9E", "AA", "MQ"))
```

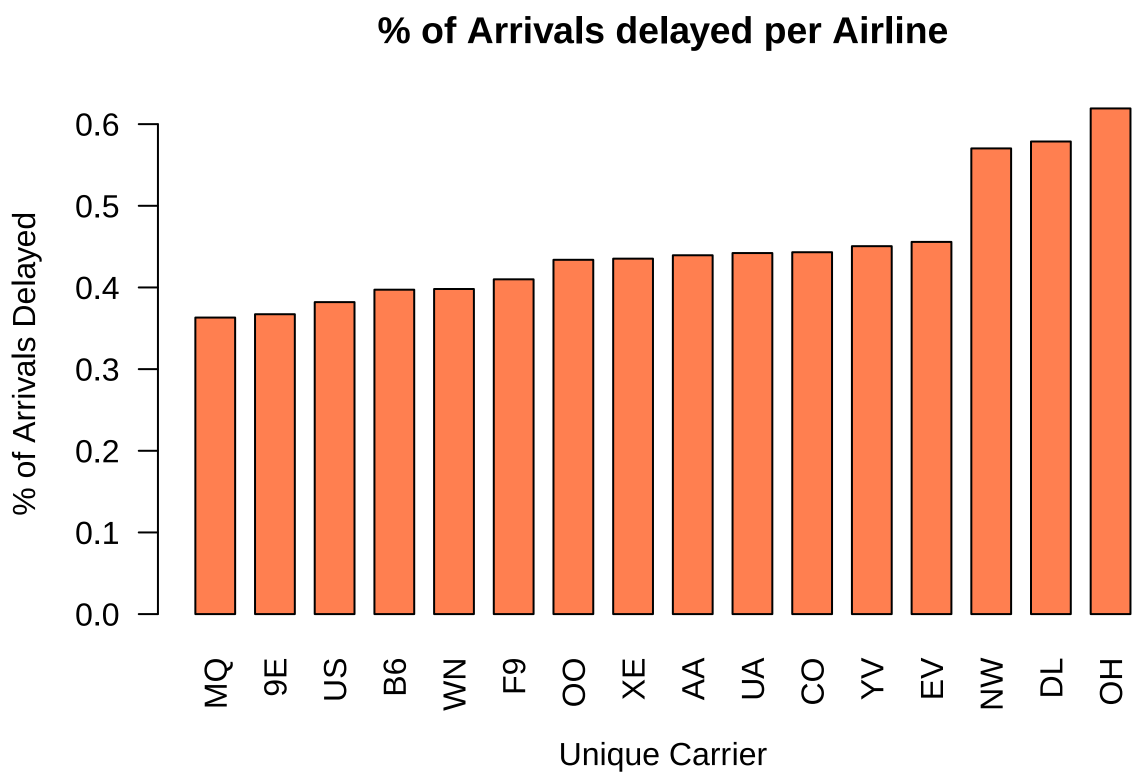


We took the percentage of flights cancelled for each airline, and the airline with the most cancellations was MQ-American Eagle, followed by American Airlines. Even though MQ was the highest, the percent of cancellation was still small, only being aroun 6% of the time.

```
arrivaldelays = airlinedata[which(airlinedata[,15]>0),]
df3 = data.frame(aggregate(arrivaldelays$ArrDelay~ arrivaldelays$UniqueCarrier,
                           arrivaldelays, mean))
df4 = df3[order(df3$arrivaldelays.ArrDelay),]
fix(df4)

barplot(df4$arrivaldelays.ArrDelay, names = df4$arrivaldelays.UniqueCarrier,
        xlab = "Unique Carrier", ylab = "Avg. Arrival Delays (minutes)",
        main = "Avg. Arrival Delay times per Airline", las=2, space=.5, col='coral')
```

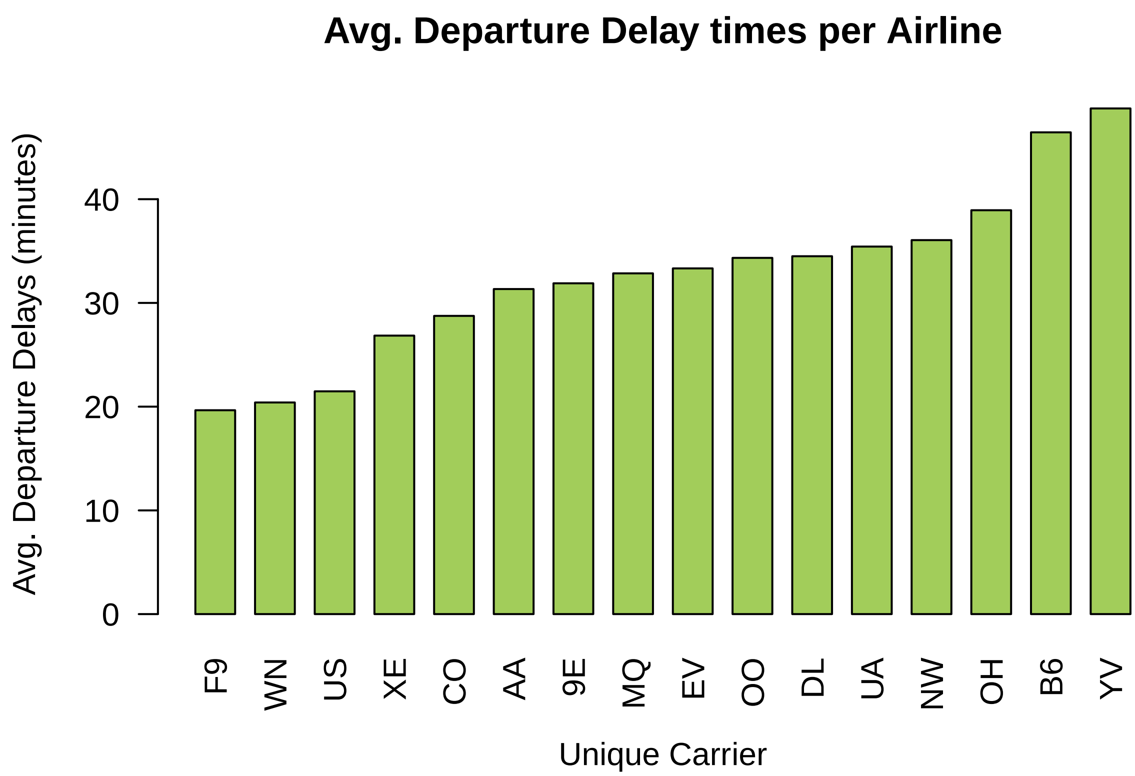



```r
#calculating percentage of flights delayed for each airline

df33 = data.frame(aggregate(arrivaldelays$ArrDelay~ arrivaldelays$UniqueCarrier,
                           arrivaldelays, length))
df = data.frame(aggregate(airlinedata$FlightNum~ airlinedata$UniqueCarrier,
                          airlinedata, length))

finaldf = merge(df33, df, by.x="arrivaldelays.UniqueCarrier", by.y="airlinedata.UniqueCarrier")
finaldf = within(finaldf, percent <- arrivaldelays.ArrDelay/airlinedata.FlightNum)
finaldf = finaldf[order(finaldf$percent),]
barplot(finaldf$percent, names = finaldf$arrivaldelays.UniqueCarrier,
        xlab = "Unique Carrier", ylab = "% of Arrivals Delayed",
        main = "% of Arrivals delayed per Airline", las=2, space=.5, col='coral')
```

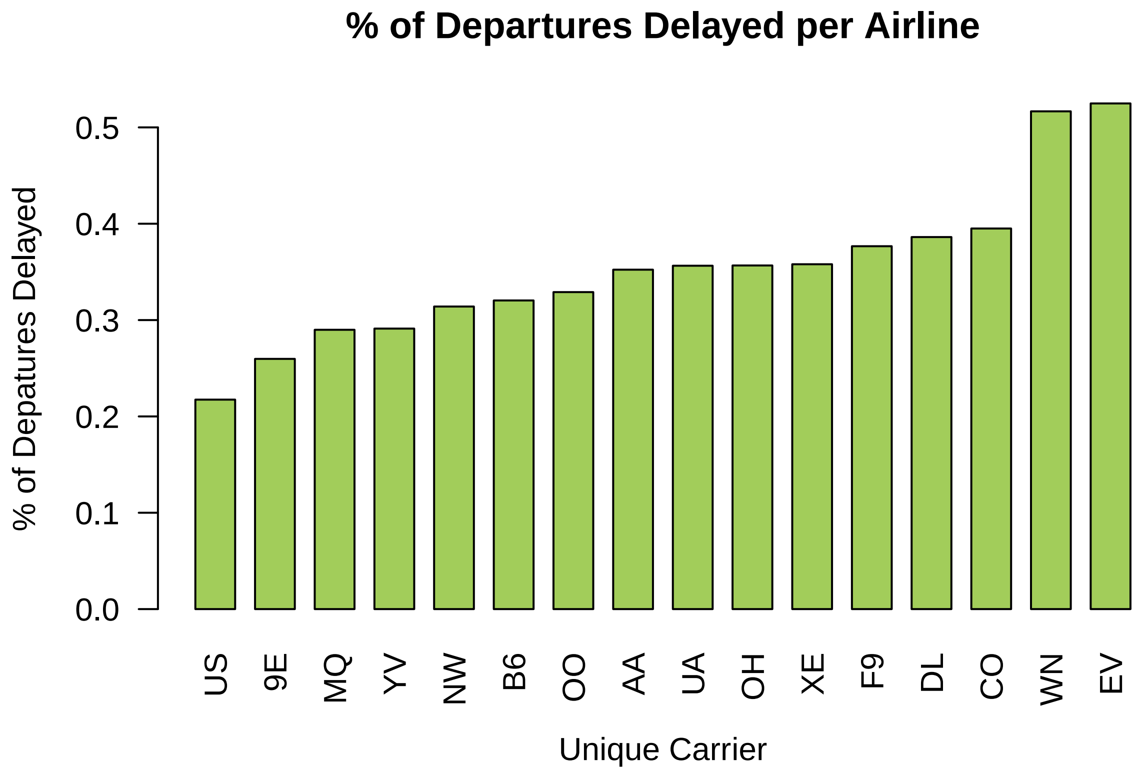


The top 3 airlines who had the longest arrival delay times were EV- Express Jet, B6-Jet Blue airways, and YV-Mesa Airlines. The top 3 airlines who had the highest percentage of arrival delays were NW-Northwest, DL-Delta, and OH-PSA airlines. You can see that these top 3 categories differ, so although one airline may have long arrival delay times on average, it does not mean they necessariy mean they are delayed the most.

```
departdelays = airlinedata[which(airlinedata[,16]>0),]
df5 = data.frame(aggregate(departdelays$DepDelay~ departdelays$UniqueCarrier,
                           departdelays, mean))

df6 = df5[order(df5$departdelays.DepDelay),]

barplot(df6$departdelays.DepDelay, names = df6$departdelays.UniqueCarrier,
        xlab = "Unique Carrier", ylab = "Avg. Departure Delays (minutes)",
        main = "Avg. Departure Delay times per Airline", las=2, space=.5,
        col='darkolivegreen3')
```



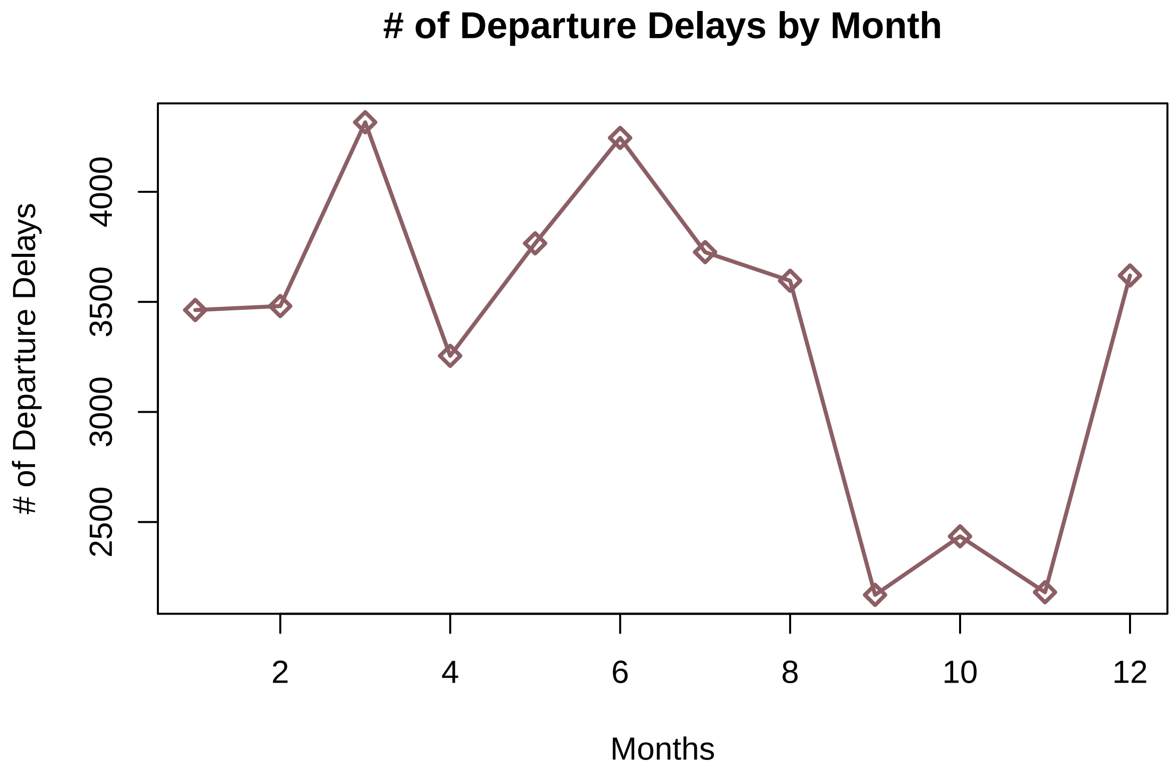

```
#calculating percentage of flights delayed for each airline

df55 = data.frame(aggregate(departdelays$DepDelay~ departdelays$UniqueCarrier,
                            departdelays, length))

df = data.frame(aggregate(airlinedata$FlightNum~ airlinedata$UniqueCarrier, airlinedata, length))

finaldf = merge(df55, df, by.x="departdelays.UniqueCarrier", by.y="airlinedata.UniqueCarrier")
finaldf = within(finaldf, percent <- departdelays.DepDelay/airlinedata.FlightNum)
finaldf = finaldf[order(finaldf$percent),]
barplot(finaldf$percent, names = finaldf$departdelays.UniqueCarrier,
        xlab = "Unique Carrier", ylab = "% of Depatures Delayed",
        main = "% of Departures Delayed per Airline", las=2, space=.5, col='darkolivegreen3')
```



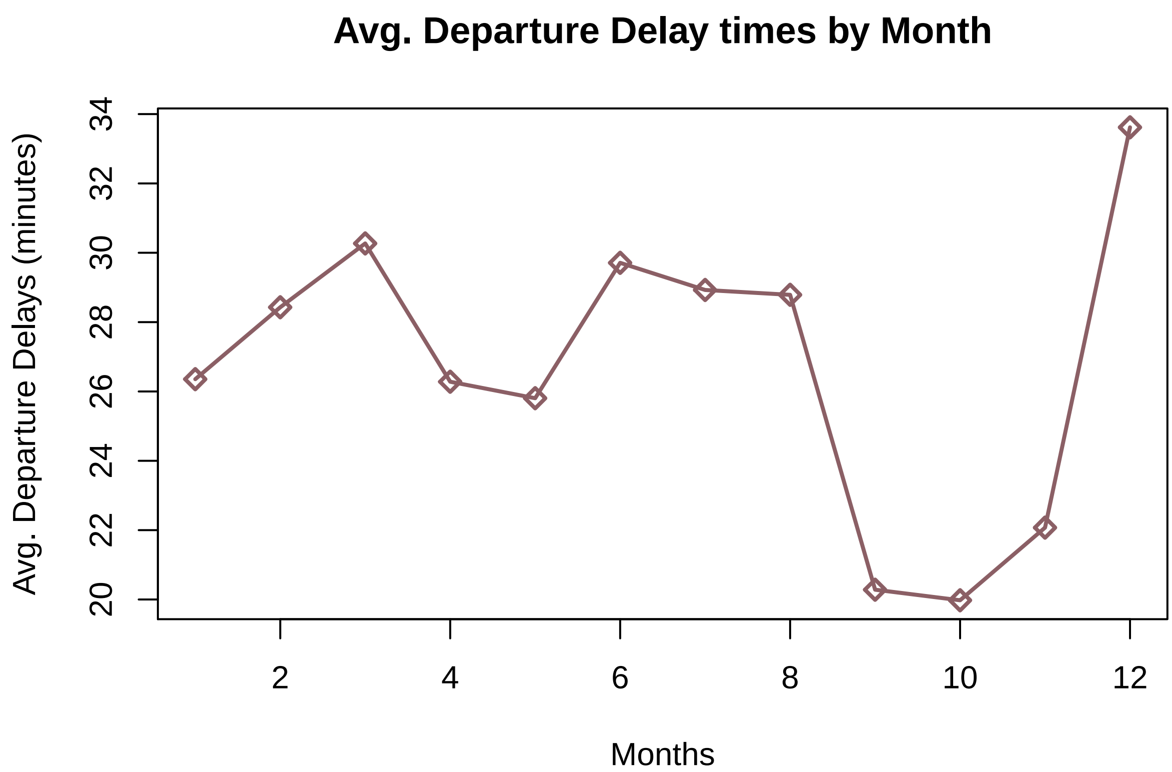
The top 3 airlines who had the longest departure delay times were OH- PSA Airlines, B6-Jet Blue airways, and YV-Mesa Airlines. The top 3 airlines who had the highest percentage of departure delays were EV-Express Jet, WN-Southwest Airlines , and CO-Continental Airlines. You can see that these top 3 categories differ, so although one airline may have long departure delay times on average, it does not mean they necessariy mean they are delayed the most. The top 3 categories for the arrival and departure average delay times are similar, but for the % of delays for both departure and arrivals, these top 3 categories differ.

For our next part of the analysis we focused more on which dates (time, days, months) of the year were the most reliable to fly on. We used a subset of the data, using only the rows where Departure Delay was greater than 0 (i.e. showing a departure delay took place). We did this becuase the departure delay variable focused on people in Austin, who would be flying out of Austin.

```
delays = airlinedata[which(airlinedata[,16]>0),]

dfm = data.frame(aggregate(delays$DepDelay~ delays$Month, delays, length))

plot(dfm$delays.Month, dfm$delays.DepDelay,
        xlab = "Months", ylab = "# of Departure Delays",
        main = "# of Departure Delays by Month", type='o',
        col='lightpink4', lwd=2, pch=5)
```

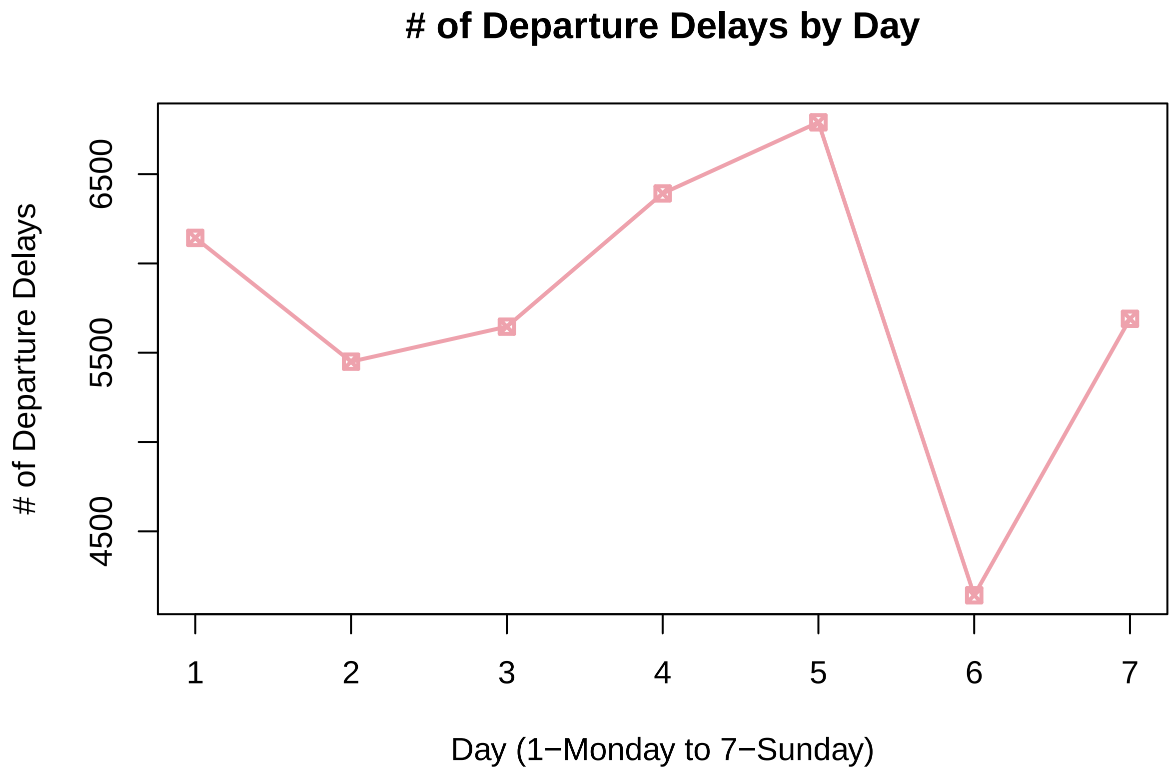



```
dfmm = data.frame(aggregate(delays$DepDelay~ delays$Month, delays, mean))


plot(dfmm$delays.Month, dfmm$delays.DepDelay,
        xlab = "Months", ylab = "Avg. Departure Delays (minutes)",
        main = "Avg. Departure Delay times by Month", type='o',
        col='lightpink4', lwd=2, pch=5)
```

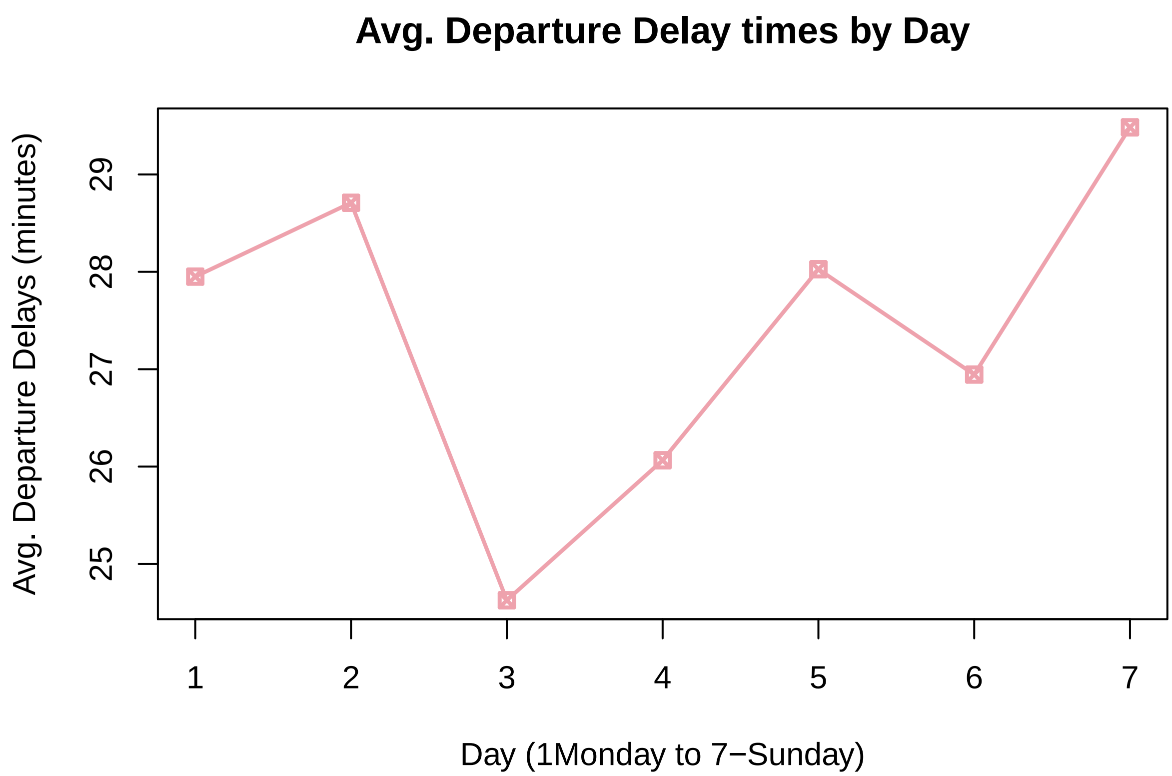


The highest amount of departure delays happened in March and June, but as you can tell from the plots above, the month with the longest departure delays (on average) is in Decemember.

```r
delays = airlinedata[which(airlinedata[,16]>0),]

dfday = data.frame(aggregate(delays$DepDelay~ delays$DayOfWeek, delays, length))

plot(dfday$delays.DayOfWeek, dfday$delays.DepDelay,
        xlab = "Day (1-Monday to 7-Sunday)", ylab = "# of Departure Delays",
        main = "# of Departure Delays by Day", type='o',
        col='lightpink2', lwd=2, pch=7)
```

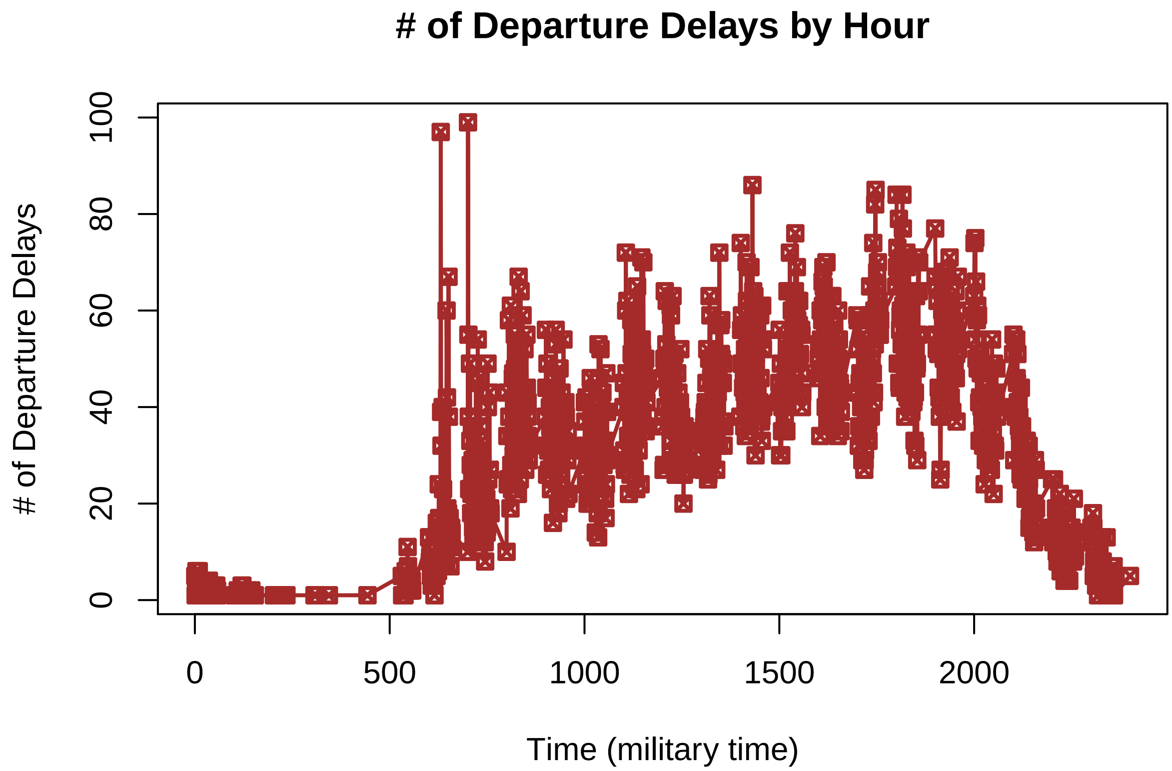



```
dfmdays = data.frame(aggregate(delays$DepDelay~ delays$DayOfWeek, delays, mean))


plot(dfmdays$delays.DayOfWeek, dfmdays$delays.DepDelay,
        xlab = "Day (1Monday to 7-Sunday)", ylab = "Avg. Departure Delays (minutes)",
        main = "Avg. Departure Delay times by Day", type='o',
        col='lightpink2', lwd=2, pch=7)
```

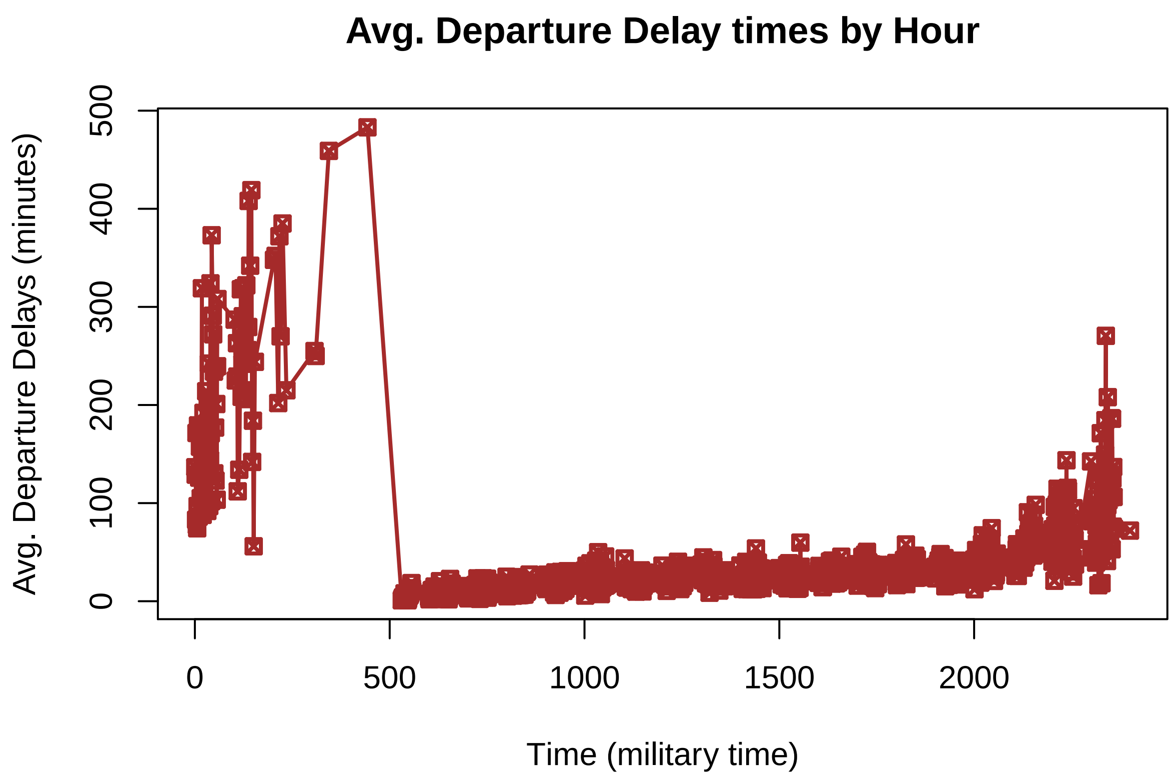


Although Friday had the highest amount of delays, the delays were not necessarily the longest. The longest delays, on average, happened on Sunday.

```
delays = airlinedata[which(airlinedata[,16]>0),]

dfhour = data.frame(aggregate(delays$DepDelay~ delays$DepTime, delays, length))


plot(dfhour$delays.DepTime, dfhour$delays.DepDelay,
        xlab = "Time (military time)", ylab = "# of Departure Delays",
        main = "# of Departure Delays by Hour", type='o',
        col='brown', lwd=2, pch=7)
```




```
dfmhours = data.frame(aggregate(delays$DepDelay~ delays$DepTime, delays, mean))


plot(dfmhours$delays.DepTime, dfmhours$delays.DepDelay,
        xlab = "Time (military time)", ylab = "Avg. Departure Delays (minutes)",
        main = "Avg. Departure Delay times by Hour", type='o',
        col='brown', lwd=2, pch=7)
```



Lastly, it seems like the most delays happen in the middle of the day between 05:00 and 20:00. In contrast, barely any delays occur betwen 0:00 and 5:00, but when they do, they are very long.

Author attribution

```
rm(list=ls())
library(tm)
library(magrittr)
```

First, we need to prepare for the training data, by reading from the train folder.

```
readerPlain = function(fname){
                readPlain(elem=list(content=readLines(fname)),
                            id=fname, language='en') }

## get a list of author directories
author_dirs = Sys.glob('data/ReutersC50/C50train/*')

file_list = NULL
#loop through all the documents for each authors
for(author in author_dirs) {
    author_name = substring(author, first=21)
    files_to_add = Sys.glob(paste0(author, '/*.txt'))
    file_list = append(file_list, files_to_add)
}
all_docs = lapply(file_list, readerPlain)

#name all the documents
```

```
mynames = file_list %>%
    { strsplit(., '/', fixed=TRUE) } %>%
    { lapply(., tail, n=2) } %>%
    { lapply(., paste0, collapse = '') } %>%
    unlist
authorname = file_list %>%
    { strsplit(., '/', fixed=TRUE) } %>%
    { lapply(.,extract,4)} %>%
    unlist
names(all_docs)= mynames
my_documents = Corpus(VectorSource(all_docs))

#preprocess/tokenize the corpus
# make everything lowercase
my_documents = tm_map(my_documents, content_transformer(tolower))
# remove numbers
my_documents = tm_map(my_documents, content_transformer(removeNumbers))
# remove punctuation
my_documents = tm_map(my_documents, content_transformer(removePunctuation))
# remove excess white-space
my_documents = tm_map(my_documents, content_transformer(stripWhitespace))
#remove stopwords
my_documents = tm_map(my_documents, content_transformer(removeWords), stopwords("en"))

## create a doc-term-matrix
DTM_all_doc = DocumentTermMatrix(my_documents,control = list(weighting = weightTfIdf))
# inspect(DTM_all_doc[1:10,1:20])
DTM_all_doc = removeSparseTerms(DTM_all_doc, 0.95)

X_train_df = as.data.frame(as.matrix(DTM_all_doc))
author_train = factor(authorname)
```

Then, let's repeat the same step for testing data.

```
#read in test data

## get a list of author directories
author_dirs_test = Sys.glob('data/ReutersC50/C50test/*')

file_list_test = NULL

#loop through all the documents for each authors
for(author in author_dirs_test) {
    author_name = substring(author, first=21)
    files_to_add = Sys.glob(paste0(author, '/*.txt'))
    file_list_test = append(file_list_test, files_to_add)
}
all_docs_test = lapply(file_list_test, readerPlain)

#name all the documents
mynames_test = file_list_test %>%
    { strsplit(., '/', fixed=TRUE) } %>%
    { lapply(., tail, n=2) } %>%
    { lapply(., paste0, collapse = '') } %>%
```

```
    unlist
authorname_test = file_list_test %>%
    { strsplit(., '/', fixed=TRUE) } %>%
    { lapply(.,extract,4)} %>%
    unlist
names(all_docs_test)= mynames_test
my_documents_test = Corpus(VectorSource(all_docs_test))
#preprocess/tokenize the corpus
# make everything lowercase
my_documents_test = tm_map(my_documents_test, content_transformer(tolower))
# remove numbers
my_documents_test = tm_map(my_documents_test, content_transformer(removeNumbers))
# remove punctuation
my_documents_test = tm_map(my_documents_test, content_transformer(removePunctuation))
# remove stopwords
my_documents_test = tm_map(my_documents_test, content_transformer(removeWords), stopwords("en"))
# remove excess white-space
my_documents_test = tm_map(my_documents_test, content_transformer(stripWhitespace))
## create a doc-term-matrix
DTM_all_doc_test = DocumentTermMatrix(my_documents_test,control = list(weighting = weightTfIdf))
# Remove sparse terms
DTM_all_doc_test = removeSparseTerms(DTM_all_doc_test, 0.95)
#put testing data in dataframe form
X_test_df = as.data.frame(as.matrix(DTM_all_doc_test))
author_test = factor(authorname_test)
```

After preparing the data, we would like to fit a naivebayes model and a random forest model to the data. We would use the naivebayes library to fit the first model. Then, we will compare the percentage of correctness of the two models.

```
#Naive Bayes
library(naivebayes)
nB.model = naive_bayes(author_train~.,data=X_train_df)
nB.pred = data.frame(predict(nB.model,X_test_df))
nB.result = cbind(nB.pred,author_test)
nB.result$correct = (nB.result[,1] == nB.result[,2])
mean(nB.result[,3])
```

```
## [1] 0.4416
```

The Naive Bayes function gives us a out-of-sample accuracy of 44.16%. This means that about 44% of the time the naive bayes classfier will be able to attribute the articles to the correct author.

```
#whose articles are difficult to distinguish?
nB.False = nB.result[nB.result[,3]==FALSE,]
nB.False <- table(nB.False[,2])

print('Frequent mistakes:')
```

```
## [1] "Frequent mistakes:"
```

```
print(sort(nB.False,decreasing = TRUE)/50)
```

```
##
##       DavidLawder     JaneMacartney   BenjaminKangLim     EdnaFernandes
##              0.90              0.88              0.82              0.80
##   DarrenSchuettler      MarkBendeich        JanLopatka        MureDickie
```

```
##             0.78              0.72              0.70              0.70
##       ScottHillis        AlanCrosby  HeatherScoffield KouroshKarimkhany
##             0.70              0.68              0.68              0.68
##        MartinWolk   PatriciaCommins       WilliamKazer    KevinDrawbaugh
##             0.66              0.66              0.66              0.64
##        PierreTran       SamuelPerry       JohnMastrini   MarcelMichelson
##             0.64              0.64              0.62              0.62
##        ToddNissen     BernardHickey     KevinMorrison          TanEeLyn
##             0.62              0.60              0.60              0.60
##    TheresePoletti    AlexanderSmith      MichaelConnor        TimFarrand
##             0.58              0.56              0.56              0.54
##      SarahDavison       EricAuchard        KarlPenhaul         KeithWeir
##             0.52              0.50              0.50              0.50
##     PeterHumphrey          JoeOrtiz       JonathanBirt       SimonCowell
##             0.50              0.48              0.48              0.48
##    GrahamEarnshaw     KirstinRidley     JoWinterbottom         NickLouth
##             0.44              0.44              0.42              0.42
##       BradDorfman       JimGilchrist         RobinSidel      RogerFillion
##             0.40              0.40              0.40              0.40
##         LydiaZajc     AaronPressman     LynneO'Donnell   LynnleyBrowning
##             0.38              0.32              0.32              0.32
##    FumikoFujisaki      MatthewBunce
##             0.26              0.20
```

As we can tell from the table above, each of these authors have 50 articles included in the training set, David Lawder has the highest misclassfy rate by using Naive Bayes classifier, followed by Jane Macartney.

Let's try randomForest model to compare the result.

```
library(randomForest)
```

```
## randomForest 4.6-12
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
#prepare data for random forest
share = intersect(names(X_train_df),names(X_test_df))
X_train_df <- X_train_df[,share]
X_test_df <- X_test_df[,share]

#take care of invalid type for variable next
names(X_train_df) = paste(names(X_train_df),'0',sep='')
names(X_test_df) = paste(names(X_test_df),'0',sep='')
X_train_df$author = author_train
X_test_df$author = author_test

rf.model = randomForest(author_train ~.,data=X_train_df,distribution = 'multinomial',ntree=500)
# newdata=cbind(X_test_df,author_test)
pred = predict(rf.model,newdat=X_test_df)
rf.pred = data.frame(pred)
rf.result = cbind(rf.pred,author_test)
rf.result$correct = (rf.result[,1] == rf.result[,2])
mean(rf.result[,3])
```

```
## [1] 0.5956
```

The random forest model returns a better accuracy rate than naive bayes classifier with a tree number of 500.

```r
rf.False = rf.result[rf.result[,3]==FALSE,]
rf.False <- table(rf.False[,2])

print('Frequent mistakes:')
```

```
## [1] "Frequent mistakes:"
```

```r
print(sort(rf.False,decreasing = TRUE)/50)
```

```
##
##       ScottHillis       EricAuchard     EdnaFernandes       DavidLawder
##              0.96              0.88              0.82              0.80
##      WilliamKazer   BenjaminKangLim        MartinWolk  DarrenSchuettler
##              0.80              0.74              0.74              0.72
##        ToddNissen     KevinMorrison HeatherScoffield       SamuelPerry
##              0.70              0.66              0.62              0.62
##          JoeOrtiz          TanEeLyn        AlanCrosby    AlexanderSmith
##              0.60              0.60              0.54              0.48
##     JaneMacartney      SarahDavison    TheresePoletti      JonathanBirt
##              0.46              0.46              0.46              0.44
##     KevinDrawbaugh      MarkBendeich        PierreTran     BernardHickey
##              0.44              0.44              0.40              0.38
##        BradDorfman    KirstinRidley        MureDickie         LydiaZajc
##              0.38              0.38              0.38              0.34
##     MichaelConnor          KeithWeir        JanLopatka        TimFarrand
##              0.34              0.32              0.30              0.28
##    MarcelMichelson   PatriciaCommins       JohnMastrini         NickLouth
##              0.26              0.26              0.24              0.22
##     JoWinterbottom    GrahamEarnshaw    LynneO'Donnell       KarlPenhaul
##              0.20              0.18              0.18              0.16
## KouroshKarimkhany      MatthewBunce       SimonCowell     PeterHumphrey
##              0.16              0.14              0.14              0.12
##        RobinSidel      RogerFillion     AaronPressman   LynnleyBrowning
##              0.12              0.12              0.10              0.10
##    FumikoFujisaki      JimGilchrist
##              0.04              0.00
```

The most frequent mistake that Random Forest classifier makes is the author, Scott Hillis with a percentage of 94%.

Practice with association rule mining

The groceries text file has a wide range of products including food products like whole milk, liver loaf and household goods like cleaner and detergent and other items. We apply market basket analysis here and use the Apriori algorithm to find patterns of user behaviour.

We first read in the groceries text file by using read.transactions.

```r
rm(list=ls())
library(arules)
```

```
## Loading required package: Matrix

##
## Attaching package: 'arules'
```

```
## The following object is masked from 'package:tm':
##
##     inspect
```

```
## The following objects are masked from 'package:base':
##
##     abbreviate, write
```

```r
library(reshape2)
library(plyr)
library(arulesViz)
```

```
## Loading required package: grid
```

```r
groceries = read.transactions(file = "groceries.txt", rm.duplicates = TRUE,
                              format = "basket", sep = ',')
```

The read.transactions() reads a data file and creates a transactions object. The rm.duplicates in the above equation removes the duplicates just like lapply(groceries_list, unique) does.

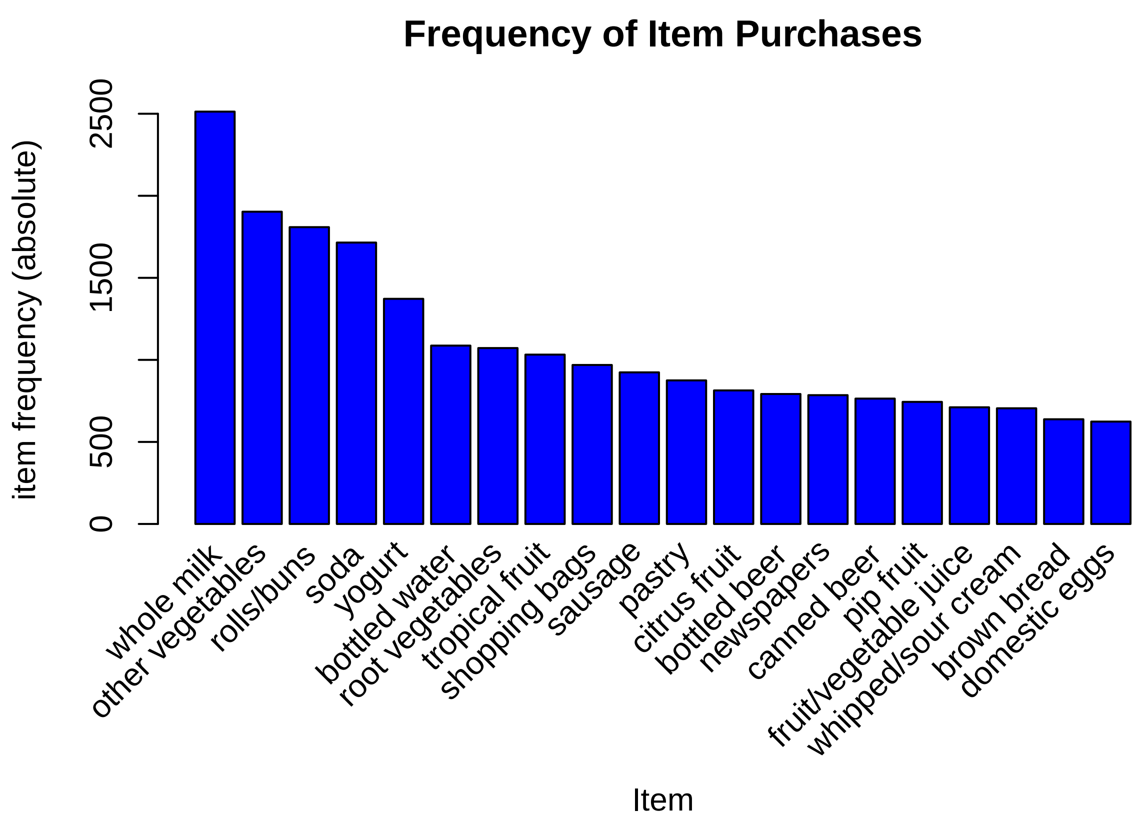
```r
dim(groceries)
```

```
## [1] 9835  169
```

The groceries data has 9835 rows where each row is an associated list of items in the transaction. There are 169 unique grocery store items.

We next plot the top 20 items by frequency

```r
itemFrequencyPlot(groceries,topN=20,type = "absolute", col = 'blue', xlab = 'Item',
                  main = 'Frequency of Item Purchases')
```



We see that whole milk, other vegetables and buns are some of the most likely to be purchased items based on various itemsets The frequencies of items will become more relevant when results for different iterations of the Apriori algorithm are generated.

We now apply the Apriori algorithm which is basically a bottoms-up approach used to identify frequent items and extend them to larger and large item sets as long as those item sets appear sufficiently often in the database.

We juggle with different values of the three main parameters , namely Support (indication of how frequently the itemset appears in the dataset), Confidence (indication of how often the rule has been found to be true) and Lift (ratio of the observed support to that expected if X and Y were independent)

Attempt 1 :

```
grocrules1 <- apriori(groceries, parameter=list(support=.01, confidence=.5, maxlen=4))
```

```
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##         0.5    0.1    1 none FALSE            TRUE       5    0.01      1
##  maxlen target   ext
##       4  rules FALSE
##
## Algorithmic control:
##  filter tree heap memopt load sort verbose
##     0.1 TRUE TRUE  FALSE TRUE    2    TRUE
##
## Absolute minimum support count: 98
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
## sorting and recoding items ... [88 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 3 4
```

```
## Warning in apriori(groceries, parameter = list(support = 0.01, confidence
## = 0.5, : Mining stopped (maxlen reached). Only patterns up to a length of 4
## returned!
```

```
##  done [0.00s].
## writing ... [15 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
```

```
inspect(grocrules1)
```

```
##      lhs                        rhs                   support confidence     lift
## [1]  {curd,
##       yogurt}               => {whole milk}       0.01006609  0.5823529 2.279125
## [2]  {butter,
##       other vegetables}     => {whole milk}       0.01148958  0.5736041 2.244885
## [3]  {domestic eggs,
##       other vegetables}     => {whole milk}       0.01230300  0.5525114 2.162336
## [4]  {whipped/sour cream,
##       yogurt}               => {whole milk}       0.01087951  0.5245098 2.052747
## [5]  {other vegetables,
##       whipped/sour cream} => {whole milk}       0.01464159  0.5070423 1.984385
## [6]  {other vegetables,
```

```
##       pip fruit}          => {whole milk}       0.01352313  0.5175097 2.025351
## [7]  {citrus fruit,
##       root vegetables}    => {other vegetables} 0.01037112  0.5862069 3.029608
## [8]  {root vegetables,
##       tropical fruit}     => {other vegetables} 0.01230300  0.5845411 3.020999
## [9]  {root vegetables,
##       tropical fruit}     => {whole milk}       0.01199797  0.5700483 2.230969
## [10] {tropical fruit,
##       yogurt}             => {whole milk}       0.01514997  0.5173611 2.024770
## [11] {root vegetables,
##       yogurt}             => {other vegetables} 0.01291307  0.5000000 2.584078
## [12] {root vegetables,
##       yogurt}             => {whole milk}       0.01453991  0.5629921 2.203354
## [13] {rolls/buns,
##       root vegetables}    => {other vegetables} 0.01220132  0.5020921 2.594890
## [14] {rolls/buns,
##       root vegetables}    => {whole milk}       0.01270971  0.5230126 2.046888
## [15] {other vegetables,
##       yogurt}             => {whole milk}       0.02226741  0.5128806 2.007235
```

Here, we have 15 rules .

The right hand side of all rules is either whole milk or other vegetables, and the items on the left are different combinations of other items that increase the likelihood of finding either milk or veggies in the same transaction.{Citrus fruit,root vegetables} and {root vegetables,tropical fruit} are three times more likely because of their high lift values.

```r
inspect(subset(grocrules1,subset=lift > 3))
```

```
##     lhs                  rhs                    support confidence     lift
## [1] {citrus fruit,
##      root vegetables} => {other vegetables} 0.01037112  0.5862069 3.029608
## [2] {root vegetables,
##      tropical fruit}  => {other vegetables} 0.01230300  0.5845411 3.020999
```

```r
inspect(subset(grocrules1, subset=confidence > 0.5))
```

```
##      lhs                      rhs                    support confidence     lift
## [1]  {curd,
##       yogurt}             => {whole milk}       0.01006609  0.5823529 2.279125
## [2]  {butter,
##       other vegetables}   => {whole milk}       0.01148958  0.5736041 2.244885
## [3]  {domestic eggs,
##       other vegetables}   => {whole milk}       0.01230300  0.5525114 2.162336
## [4]  {whipped/sour cream,
##       yogurt}             => {whole milk}       0.01087951  0.5245098 2.052747
## [5]  {other vegetables,
##       whipped/sour cream} => {whole milk}       0.01464159  0.5070423 1.984385
## [6]  {other vegetables,
##       pip fruit}          => {whole milk}       0.01352313  0.5175097 2.025351
## [7]  {citrus fruit,
##       root vegetables}    => {other vegetables} 0.01037112  0.5862069 3.029608
## [8]  {root vegetables,
##       tropical fruit}     => {other vegetables} 0.01230300  0.5845411 3.020999
## [9]  {root vegetables,
##       tropical fruit}     => {whole milk}       0.01199797  0.5700483 2.230969
```

```
## [10] {tropical fruit,
##       yogurt}            => {whole milk}       0.01514997  0.5173611 2.024770
## [11] {root vegetables,
##       yogurt}            => {whole milk}       0.01453991  0.5629921 2.203354
## [12] {rolls/buns,
##       root vegetables}   => {other vegetables} 0.01220132  0.5020921 2.594890
## [13] {rolls/buns,
##       root vegetables}   => {whole milk}       0.01270971  0.5230126 2.046888
## [14] {other vegetables,
##       yogurt}            => {whole milk}       0.02226741  0.5128806 2.007235
```

```
inspect(subset(grocrules1, subset = support > .01 & confidence > 0.3))
```

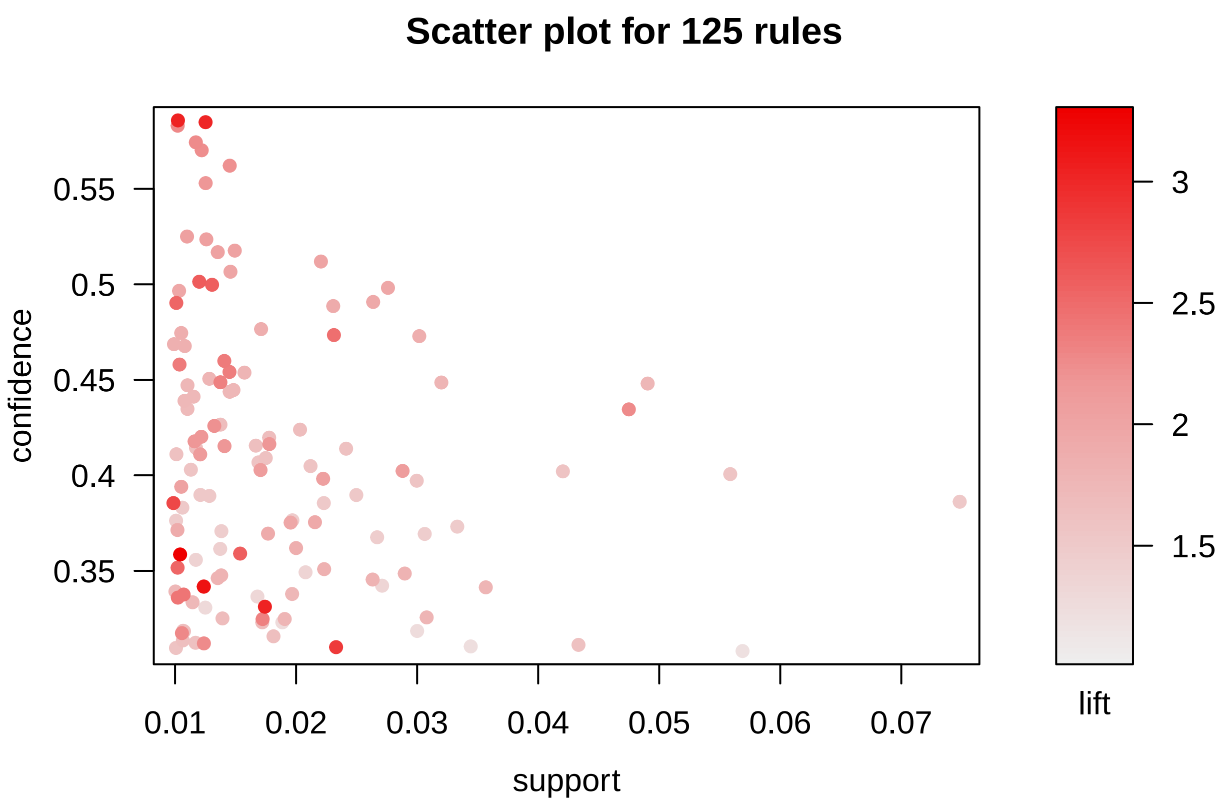
```
##      lhs                        rhs                   support confidence     lift
## [1]  {curd,
##       yogurt}            => {whole milk}       0.01006609  0.5823529 2.279125
## [2]  {butter,
##       other vegetables}  => {whole milk}       0.01148958  0.5736041 2.244885
## [3]  {domestic eggs,
##       other vegetables}  => {whole milk}       0.01230300  0.5525114 2.162336
## [4]  {whipped/sour cream,
##       yogurt}            => {whole milk}       0.01087951  0.5245098 2.052747
## [5]  {other vegetables,
##       whipped/sour cream} => {whole milk}       0.01464159  0.5070423 1.984385
## [6]  {other vegetables,
##       pip fruit}         => {whole milk}       0.01352313  0.5175097 2.025351
## [7]  {citrus fruit,
##       root vegetables}   => {other vegetables} 0.01037112  0.5862069 3.029608
## [8]  {root vegetables,
##       tropical fruit}    => {other vegetables} 0.01230300  0.5845411 3.020999
## [9]  {root vegetables,
##       tropical fruit}    => {whole milk}       0.01199797  0.5700483 2.230969
## [10] {tropical fruit,
##       yogurt}            => {whole milk}       0.01514997  0.5173611 2.024770
## [11] {root vegetables,
##       yogurt}            => {other vegetables} 0.01291307  0.5000000 2.584078
## [12] {root vegetables,
##       yogurt}            => {whole milk}       0.01453991  0.5629921 2.203354
## [13] {rolls/buns,
##       root vegetables}   => {other vegetables} 0.01220132  0.5020921 2.594890
## [14] {rolls/buns,
##       root vegetables}   => {whole milk}       0.01270971  0.5230126 2.046888
## [15] {other vegetables,
##       yogurt}            => {whole milk}       0.02226741  0.5128806 2.007235
```

```
rules = apriori(groceries, parameter = list(support=.01, confidence=.3, target='rules'))
```

```
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##         0.3    0.1    1 none FALSE            TRUE       5    0.01      1
##  maxlen target   ext
##      10  rules FALSE
##
```

```
## Algorithmic control:
##  filter tree heap memopt load sort verbose
##     0.1 TRUE TRUE  FALSE TRUE    2    TRUE
##
## Absolute minimum support count: 98
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
## sorting and recoding items ... [88 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 3 4 done [0.00s].
## writing ... [125 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
```

```
plot(rules)
```



The plot shows that rules with high lift typically have slightly low support.

We moved on to further iterations.

Attempt 2 :

```
grocrules2 <- apriori(groceries,parameter=list(support=.02, confidence=.4, maxlen=6))
```

```
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##         0.4    0.1    1 none FALSE            TRUE       5    0.02      1
##  maxlen target   ext
```

```
##       6  rules FALSE
##
## Algorithmic control:
##  filter tree heap memopt load sort verbose
##     0.1 TRUE TRUE  FALSE TRUE    2    TRUE
##
## Absolute minimum support count: 196
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
## sorting and recoding items ... [59 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 3 done [0.00s].
## writing ... [15 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
```

```r
inspect(grocrules2)
```

```
##      lhs                                    rhs                support
## [1]  {frozen vegetables}                 => {whole milk}       0.02043721
## [2]  {beef}                              => {whole milk}       0.02125064
## [3]  {curd}                              => {whole milk}       0.02613116
## [4]  {margarine}                         => {whole milk}       0.02419929
## [5]  {butter}                            => {whole milk}       0.02755465
## [6]  {domestic eggs}                     => {whole milk}       0.02999492
## [7]  {whipped/sour cream}                => {other vegetables} 0.02887646
## [8]  {whipped/sour cream}                => {whole milk}       0.03223183
## [9]  {tropical fruit}                    => {whole milk}       0.04229792
## [10] {root vegetables}                   => {other vegetables} 0.04738180
## [11] {root vegetables}                   => {whole milk}       0.04890696
## [12] {yogurt}                            => {whole milk}       0.05602440
## [13] {other vegetables,root vegetables} => {whole milk}       0.02318251
## [14] {root vegetables,whole milk}       => {other vegetables} 0.02318251
## [15] {other vegetables,yogurt}          => {whole milk}       0.02226741
##      confidence lift
## [1]  0.4249471  1.663094
## [2]  0.4050388  1.585180
## [3]  0.4904580  1.919481
## [4]  0.4131944  1.617098
## [5]  0.4972477  1.946053
## [6]  0.4727564  1.850203
## [7]  0.4028369  2.081924
## [8]  0.4496454  1.759754
## [9]  0.4031008  1.577595
## [10] 0.4347015  2.246605
## [11] 0.4486940  1.756031
## [12] 0.4016035  1.571735
## [13] 0.4892704  1.914833
## [14] 0.4740125  2.449770
## [15] 0.5128806  2.007235
```

After trying different other combinations, we took support threshold as .02, so that only rules that are relevant to 2% of transactions or more are included. We took confidence as 0.4. Additionally, the maximum size was increased to 6 items, however this made no difference in practice as all rules containd two or less items.

Here in rhs, we have predominantly whole milk and other vegetables, which are the frequently bought items.

These are just showing us grocery patterns of users . Lift values have decreased from previous iterations.

In conclusion,

From a marketing perspective, iterations of the algorithm that allow for small cuts of data but require very strong associations produce the most actionable results.

Looking at the patterns of rules across all attempts, it is interesting to note that the strongest rules in every case were exclusively among food items. Other items such as garbage bags, cleaning products, etc. did not show up with much frequency. Anecdotally, it is likely that consumers simply buy these items when they run out, as opposed to on a weekly basis or in conjunction with other items, so their appearance is effectively random.

Many of the rules across all iterations were simply combinations of commonly-bought items.

If the grocery items have high support, confidence and lift values, then we can place them together in the grocery store. This is especially important where one item in a pair is very popular, and the other item is very high margin.

The results can be used to drive targeted marketing campaigns. For each user, we pick a handful of products based on products they have bought to date which have both a high uplift and a high margin, and send them a e.g. personalized email or display ads etc.