Exercises 2

Andrew Reuben

Problem 1

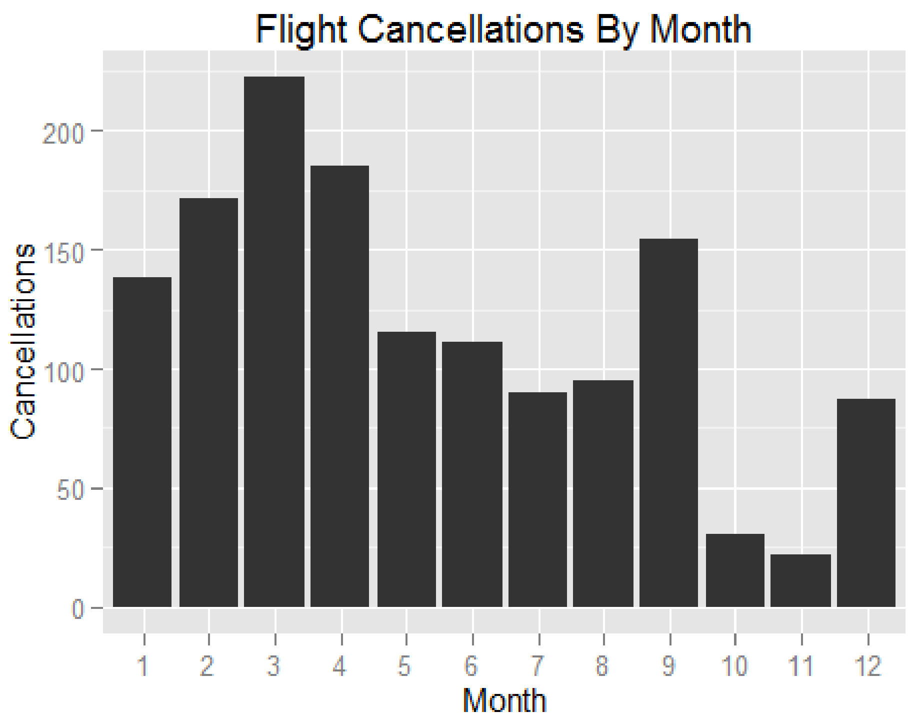
```
airport=read.csv("https://raw.githubusercontent.com/jgscott/STA380/master/dat
a/ABIA.csv")
library(ggplot2)
attach(airport)
```

I have chosen to look at the seasonality of cancellations. While it may seem like cancellations are unavoidable, passengers may benefit from knowing when cancellations occur most. To get the broadest sense of when cancellations occur cancellations by month is graphed below.

```
month=airport[c(2,22)]
monthdelay=tapply(airport$Cancelled, airport$Month, sum)
months=levels(factor(airport$Month))

cut=factor(levels(airport$month), levels = levels(airport$month))

qplot(months,monthdelay,geom="bar", stat="identity",xlab="Month",
ylab="Cancellations")+scale_x_discrete(limits=c(1,2,3,4,5,6,7,8,9,10,11,12))+
ggtitle("Flight Cancellations By Month")
```

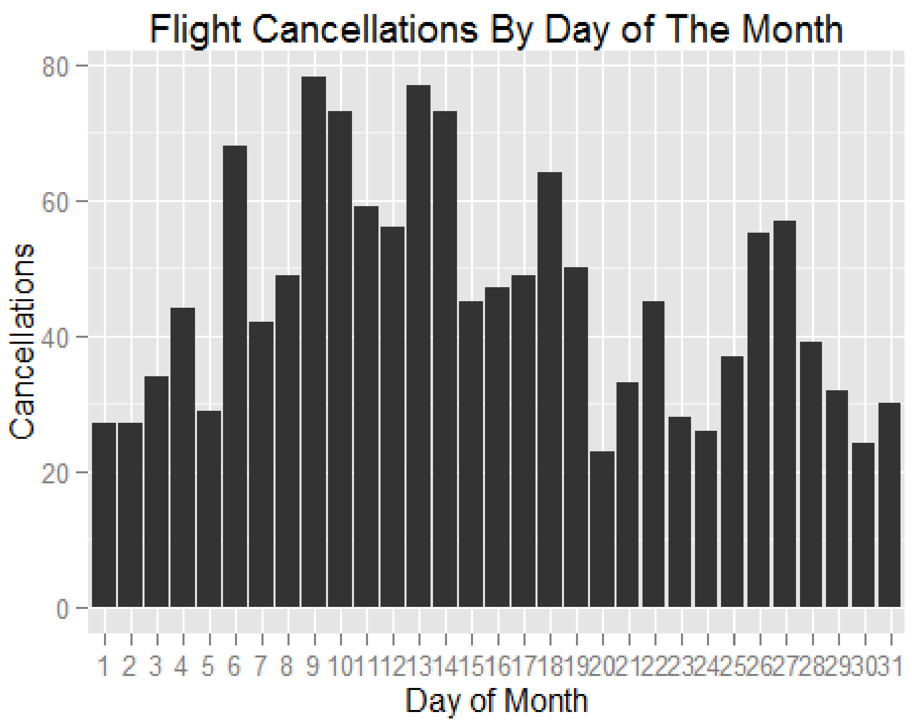


It may also be helpful to look at how cancellations vary between times of the month.

```
domdelay=tapply(airport$Cancelled, airport$DayofMonth, sum)
dom=levels(factor(airport$DayofMonth))



qplot(dom,domdelay,geom="bar", stat="identity",xlab="Day of Month",
ylab="Cancellations")+scale_x_discrete(limits=c(1,2,3,4,5,6,7,8,9,10,11,12,13
,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31))+ggtitle("Flight
Cancellations By Day of The Month")
```

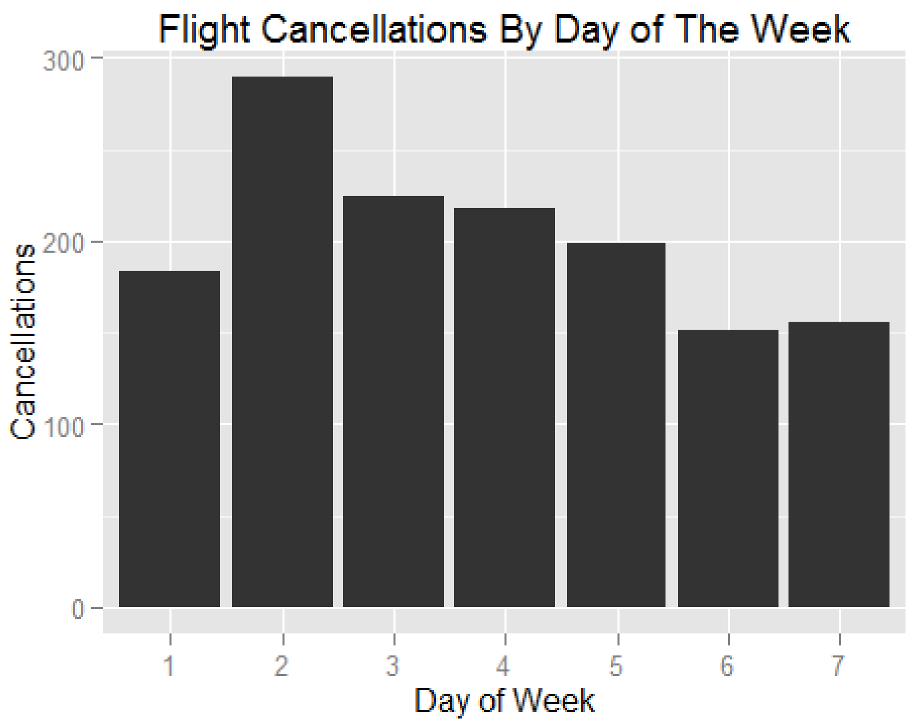


Finally, cancellations by day of the week can be seen below.

```
dowdelay=tapply(airport$Cancelled, airport$DayOfWeek, sum)
dow=levels(factor(airport$DayOfWeek))




qplot(dow,dowdelay,geom="bar", stat="identity",xlab="Day of Week",
ylab="Cancellations")+scale_x_discrete(limits=c(1,2,3,4,5,6,7))+ggtitle("Flig
ht Cancellations By Day of The Week")
```



Problem 2

The libraries neccesary to analyze the corpus, as well as the readerPlain function, are loaded.

```r
library(tm)
library(randomForest)
library(e1071)

## Warning: package 'e1071' was built under R version 3.2.2

library(rpart)
library(ggplot2)
library(caret)

## Warning: package 'caret' was built under R version 3.2.2

library(plyr)

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

The file names in the corpus are placed in the file list and the author for each article is placed in the train labels list.

```r
author_dirs = Sys.glob('./ReutersC50/C50train/*')
file_list = NULL
```

```r
train_labels = NULL
for(author in author_dirs) {
  author_name = substring(author, first=23)
  files_to_add = Sys.glob(paste0(author, '/*.txt'))
  file_list = append(file_list, files_to_add)
  train_labels = append(train_labels, rep(author_name, length(files_to_add)))
}
```

File names are cleaned in a way that removes the .txt from the end of the filename.

```r
all_docs = lapply(file_list, readerPlain)
names(all_docs) = file_list
names(all_docs) = sub('.txt', '', names(all_docs))
```

Initializing training corpus.

```r
train_corpus = Corpus(VectorSource(all_docs))
#names(train_corpus) = file_list ***This may have to be put back in
```

The articles are tokenized and stemmed. This process includes making all characters lower case and removing numbers, punctuation, whitespace and all stopwords included in the SMART library. The tokenization process makes the articles easier to analyze by making the articles easier to work with and removing words that are not meaningful.

```r
library(SnowballC)
train_corpus = tm_map(train_corpus, content_transformer(tolower))
train_corpus = tm_map(train_corpus, content_transformer(removeNumbers))
train_corpus = tm_map(train_corpus, content_transformer(removePunctuation))
train_corpus = tm_map(train_corpus, content_transformer(stripWhitespace))
train_corpus = tm_map(train_corpus, content_transformer(removeWords),
stopwords("SMART"))
train_corpus=tm_map(train_corpus, stemDocument)
```

A document term matrix is made using the tokenized and stemmed training corpus. During this process a sparsity level of 99% is used. This means that only words that are used in more than one percent of the documents will be used. This removes words that may add noise to the model, but isn't such a low sparsity level that terms that could be used in identifying an author are removed.

```r
DTM_train = DocumentTermMatrix(train_corpus)
DTM_train = removeSparseTerms(DTM_train, 0.99)
```

The train DTM is transformed into a data frame to be used during Naive Bayes.

```r
DTM_traindf = as.data.frame(inspect(DTM_train))
```

A test matrix is made using the same parameters as the training matrix.

```r
author_dirs = Sys.glob('./ReutersC50/C50test/*')
file_list = NULL
test_labels = NULL
for(author in author_dirs) {
  author_name = substring(author, first=22)
  files_to_add = Sys.glob(paste0(author, '/*.txt'))
  file_list = append(file_list, files_to_add)
  test_labels = append(test_labels, rep(author_name, length(files_to_add)))
}
```

Test_corpus initialized.

```
test_corpus = Corpus(VectorSource(all_docs))
```

Tokenization and stemming.

```
library(SnowballC)
test_corpus = tm_map(test_corpus, content_transformer(tolower))
test_corpus = tm_map(test_corpus, content_transformer(removeNumbers))
test_corpus = tm_map(test_corpus, content_transformer(removePunctuation))
test_corpus = tm_map(test_corpus, content_transformer(stripWhitespace))
test_corpus = tm_map(test_corpus, content_transformer(removeWords),
stopwords("SMART"))
test_corpus=tm_map(test_corpus, stemDocument)
```

Create Document Term Matrix

```
DTM_test = DocumentTermMatrix(test_corpus)
DTM_test = removeSparseTerms(DTM_test, 0.99)
```

Find all words used in the training data matrix. Only these words will be used when performing Naive Bayes. This will give more accurate results.

```
train_dict = NULL
train_dict = dimnames(DTM_train)[[2]]
```

Test data matrix is made and sparse terms are removed at the same level as the training data matrix.

```
DTM_test = DocumentTermMatrix(test_corpus, list(dictionary=train_dict))
DTM_test = removeSparseTerms(DTM_test, 0.99)
```

A data frame is formed using the test DTM.

```
DTM_testdf = as.data.frame(inspect(DTM_test))
```

A Naive Bayes model is made using the training data matrix and labels.

```
factorlabels=as.factor(train_labels)
model_NB = naiveBayes(x=DTM_traindf, y=as.factor(train_labels), laplace=1)
```

The Naive Bayes model is fit to the test matrix and used to predict the author of each article in the test matrix.

```
pred_NB = predict(model_NB, DTM_testdf)
```

A confusion Matrix is formed to show how accurate the Naive Bayes model was in prediction the authors of the articles in the test matrix. The overall accuracy of the model was 67.41%.

```
conf_NB = confusionMatrix(table(pred_NB,test_labels))
conf_NB_df = as.data.frame(conf_NB$byClass)
accmean=colMeans(conf_NB_df)[8]

accmean

## Balanced Accuracy
##         0.6740816
```

However, the model did not perform uniformly well on all authors. The table below displays the accuracy for each author in the corpus. This shows that the model was very succesful in predicting some other, but not as well at predicting others. The model was able to predict the author of articles by Tim Farrand, Kourosh Karimkhany, Lynne O'Donnell, Jo Winterbottom, JimGilchrist, Fumiko Fujisaki, David Lawder and Alan Crosby at rates higher than 90%. On the other hand articles by Jane Macartney, William Kazer, Tane Eelyn, Scott Hillis, Sarah Davison, Mure Dickie and Kirstin Ridley were only predicted correctly half the time. It is likely that these low performing classes likely include articles with words that are not distinct resulting in misclassifications.

```
accuracy_by_author=conf_NB_df[8]
accuracy_by_author

##                            Balanced Accuracy
## Class: AaronPressman               0.7995918
## Class: AlanCrosby                  0.9463265
## Class: AlexanderSmith              0.5200000
## Class: BenjaminKangLim             0.6287755
## Class: BernardHickey               0.8789796
## Class: BradDorfman                 0.5300000
## Class: DarrenSchuettler            0.5300000
## Class: DavidLawder                 0.9444898
## Class: EdnaFernandes               0.6000000
## Class: EricAuchard                 0.5100000
## Class: FumikoFujisaki              0.9540816
## Class: GrahamEarnshaw              0.6600000
## Class: HeatherScoffield            0.7900000
## Class: JaneMacartney               0.5000000
## Class: JanLopatka                  0.7179592
## Class: JimGilchrist                0.9338776
## Class: JoeOrtiz                    0.5700000
## Class: JohnMastrini                0.5400000
## Class: JonathanBirt                0.5500000
## Class: JoWinterbottom              0.9412245
## Class: KarlPenhaul                 0.5000000
## Class: KeithWeir                   0.5500000
## Class: KevinDrawbaugh              0.5100000
## Class: KevinMorrison              0.6600000
## Class: KirstinRidley              0.5000000
## Class: KouroshKarimkhany          0.9565306
## Class: LydiaZajc                  0.8961224
## Class: LynneO'Donnell             0.9481633
## Class: LynnleyBrowning            0.5200000
## Class: MarcelMichelson            0.8897959
## Class: MarkBendeich               0.6500000
## Class: MartinWolk                 0.5600000
## Class: MatthewBunce               0.8200000
## Class: MichaelConnor              0.5400000
## Class: MureDickie                 0.5000000
## Class: NickLouth                  0.7593878
## Class: PatriciaCommins            0.6300000
## Class: PeterHumphrey              0.6000000
## Class: PierreTran                 0.6400000
## Class: RobinSidel                 0.8487755
## Class: RogerFillion               0.8800000
## Class: SamuelPerry                0.5200000
## Class: SarahDavison               0.5000000
## Class: ScottHillis                0.5000000
## Class: SimonCowell                0.5800000
## Class: TanEeLyn                   0.5000000
## Class: TheresePoletti             0.7393878
## Class: TimFarrand                 0.9206122
## Class: ToddNissen                 0.5400000
## Class: WilliamKazer               0.5000000
```

After exploring a number of different possible models, I settled on the random forest model.

```
set.seed(34)
model_RF = randomForest(x=DTM_traindf, y=as.factor(train_labels), mtry=4,
ntree=200)
pred_RF = predict(model_RF, data=DTM_test)
```

Overall, the random forest model performed almost identically well as the Naive Bayes model with an accuracy score of 72%.

```
conf_RF = confusionMatrix(table(pred_RF,test_labels))
conf_RF$overall[1]
```

```
## Accuracy
##   0.7152
```

There was also variability between authors in terms of accuracy with the random forest model. However, this variability varied to a much greater extent than the Naive Bayes model. This could be a result of the random variability associated with how the random forest model works. Some splits may use accurate predictors, while others may use less accurate predictors. The varied levels of accuracy between authors can be seen below.

```
Accuracy_Author = as.data.frame(conf_RF$byClass)[1]
Accuracy_Author
```

```
##                             Sensitivity
## Class: AaronPressman               0.82
## Class: AlanCrosby                  0.82
## Class: AlexanderSmith              0.50
## Class: BenjaminKangLim             0.70
## Class: BernardHickey               0.70
## Class: BradDorfman                 0.38
## Class: DarrenSchuettler            0.84
## Class: DavidLawder                 0.90
## Class: EdnaFernandes               0.58
## Class: EricAuchard                 0.50
## Class: FumikoFujisaki              0.98
## Class: GrahamEarnshaw              0.82
## Class: HeatherScoffield            0.94
## Class: JaneMacartney               0.40
## Class: JanLopatka                  0.74
## Class: JimGilchrist                1.00
## Class: JoeOrtiz                    0.64
## Class: JohnMastrini                0.68
## Class: JonathanBirt                0.62
## Class: JoWinterbottom              0.88
## Class: KarlPenhaul                 0.90
## Class: KeithWeir                   0.68
## Class: KevinDrawbaugh              0.66
## Class: KevinMorrison               0.54
## Class: KirstinRidley               0.56
## Class: KouroshKarimkhany           0.92
## Class: LydiaZajc                   0.92
## Class: LynneO'Donnell              0.96
## Class: LynnleyBrowning             1.00
## Class: MarcelMichelson             0.86
## Class: MarkBendeich                0.84
## Class: MartinWolk                  0.66
## Class: MatthewBunce                0.94
## Class: MichaelConnor               0.68
## Class: MureDickie                  0.38
## Class: NickLouth                   0.82
## Class: PatriciaCommins             0.60
## Class: PeterHumphrey               0.82
## Class: PierreTran                  0.78
## Class: RobinSidel                  0.82
## Class: RogerFillion                0.94
## Class: SamuelPerry                 0.56
## Class: SarahDavison                0.52
## Class: ScottHillis                 0.30
## Class: SimonCowell                 0.72
## Class: TanEeLyn                    0.54
## Class: TheresePoletti           0.60
## Class: TimFarrand               0.78
## Class: ToddNissen               0.74
## Class: WilliamKazer             0.28
```

Problem 3

Data is imported as list.

```
library(arules)

## Warning: package 'arules' was built under R version 3.2.2

fc <-
file("https://raw.githubusercontent.com/jgscott/STA380/master/data/groceries.
txt")
groceries <- strsplit(readLines(fc), ",")
close(fc)
```

Duplicates are dropped

```
groceries <- lapply(groceries, unique)
```

The list is split into transactions.

```
groctrans <- as(groceries, "transactions")
```

Apriori algorithm is run in order to develop rules. While running the apriori algorithm with different levels of support and confidence, a common theme emerged. The most common item to be associated with the rules being developed was far and away whole milk. The only other item that appeared with any regularity was other vegetables. This reveals that whole milk, and to a lesser extent other vegetables, are purchased by a wide variety of shoppers.

```
grocrules <- apriori(groctrans,
    parameter=list(support=.01, confidence=.5, maxlen=5))

##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport support minlen maxlen
##         0.5    0.1    1 none FALSE            TRUE    0.01      1      5
##  target   ext
##   rules FALSE
##
## Algorithmic control:
##  filter tree heap memopt load sort verbose
##     0.1 TRUE TRUE  FALSE TRUE    2    TRUE
##
## apriori - find association rules with the apriori algorithm
## version 4.21 (2004.05.09)        (c) 1996-2004   Christian Borgelt
## 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 ... [15 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
```

```
inspect(grocrules)
```

```
##    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
```

While most support and confidence levels revealed whole milk to be popular among shoppers, one other interesting rule was discoverd with a support level of .001 and a confidence level of .8. As can be seen below, shoppers who buy liquor and red/blush wine also buy bottle beer. At a confidence of .9048 this is a very strong rules. This confidence level means that 90% of shoppers that bought liquor and red/blush wine also purchased beer.

```
grocrulesbeer <- apriori(groctrans,
    parameter=list(support=.001, confidence=.8, maxlen=4))

##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport support minlen maxlen
##         0.8    0.1    1 none FALSE            TRUE   0.001      1      4
##  target   ext
##   rules FALSE
##
## Algorithmic control:
##  filter tree heap memopt load sort verbose
##     0.1 TRUE TRUE  FALSE TRUE    2    TRUE
##
## apriori - find association rules with the apriori algorithm
## version 4.21 (2004.05.09)        (c) 1996-2004   Christian Borgelt
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.01s].
## sorting and recoding items ... [157 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 3 4 done [0.00s].
## writing ... [258 rule(s)] done [0.02s].
## creating S4 object  ... done [0.00s].

inspect(head(grocrulesbeer))

##   lhs                        rhs                  support confidence
lift
## 1 {liquor,
##    red/blush wine}        => {bottled beer} 0.001931876  0.9047619
11.235269
## 2 {cereals,
##    curd}                  => {whole milk}   0.001016777  0.9090909
3.557863
## 3 {cereals,
##    yogurt}                => {whole milk}   0.001728521  0.8095238
3.168192
```

```
## 4 {butter,
##    jam}                      => {whole milk}   0.001016777  0.8333333
3.261374
## 5 {bottled beer,
##    soups}                    => {whole milk}   0.001118454  0.9166667
3.587512
## 6 {house keeping products,
##    napkins}                  => {whole milk}   0.001321810  0.8125000
3.179840
```