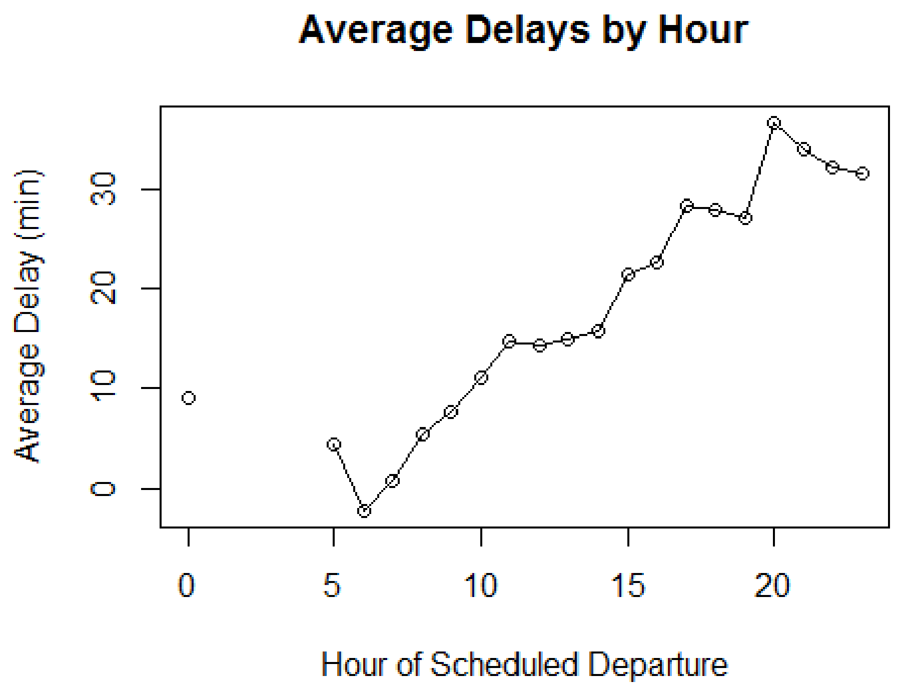
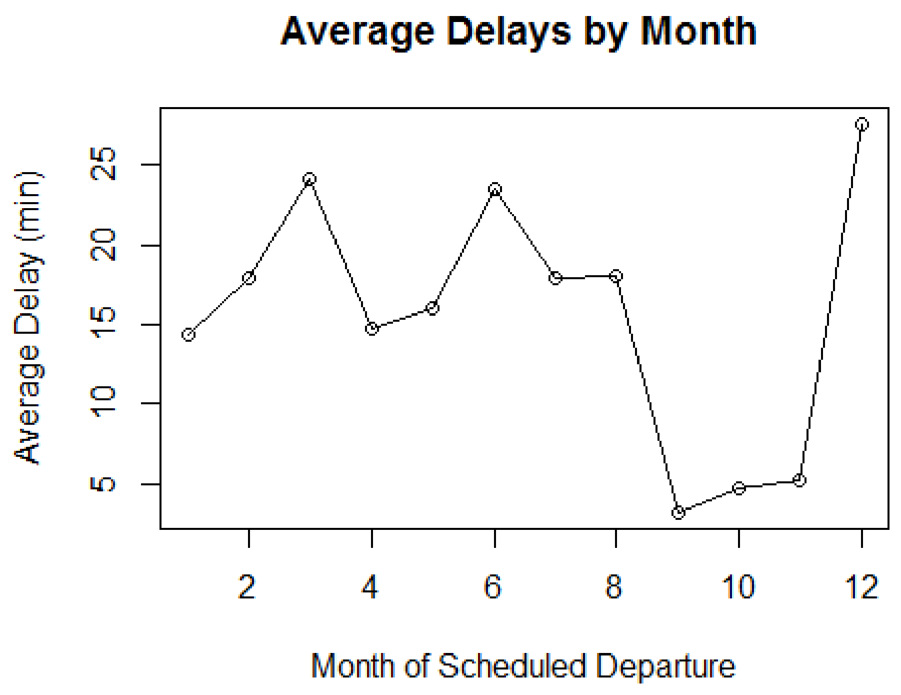
## Flights

What is the best time of day to minimize delays?



You should book flights for 6AM. This is unsurprising. Early in the day, mistakes and delays haven't piled up beyond the buffers that airlines have that would cause a delay. Also many people avoid flying early so it's less crowded. Still it is curious that there are more delays at 5 AM than 6 or 7 AM. These could be attributed to pilots and passengers arriving late to their job/flight because they overslept.

What is the best time of the year to minimize delays?



The best months are September, October, and November. Significantly so. It is likely that the fall has the least flights. While the chart is normalized, more flights probably lead to gridlock on the runway. When there is a lower volume, this probably happens less. While you may think Thanksgiving would have more flights, it is a single holiday with an unusually short break. Compared to Christmas and other vacation holidays and seasons, most people simply stay home.

## Author Attribution

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

## Loading required package: NLP

library(e1071)
source('textutils.R')
library(randomForest)

## randomForest 4.6-12

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

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

```
author_dirs=Sys.glob('C:/Users/vince/Desktop/R Dir/ReutersC50/C50train/*')
file_list=NULL
labels=NULL


for(author in author_dirs){
  author_name=substring(author, first=20)
  files_to_add= Sys.glob(paste0(author,'/*.txt'))
  file_list=append(file_list, files_to_add)
  labels=append(labels, rep(author_name, length(files_to_add)))
}

all_docs=lapply(file_list, readerPlain)
names(all_docs)=file_list

my_corpus=Corpus(VectorSource(all_docs))
names(my_corpus)=file_list

my_corpus= tm_map(my_corpus, content_transformer(tolower))
my_corpus= tm_map(my_corpus, content_transformer(removeNumbers))
my_corpus= tm_map(my_corpus, content_transformer(removePunctuation))
my_corpus= tm_map(my_corpus, content_transformer(stripWhitespace))
my_corpus= tm_map(my_corpus, content_transformer(removeWords),
stopwords("SMART"))

DTM=DocumentTermMatrix(my_corpus)
DTM=removeSparseTerms(DTM,0.975)

x=as.matrix(DTM)

x=x/rowSums(x)
x=idf.weight(x)


##### test set ####

author_test=Sys.glob('C:/Users/vince/Desktop/R Dir/ReutersC50/C50test/*')
filelist=NULL
labels2=NULL


for(author in author_test){
  authorname=substring(author, first=20)
  filestoadd= Sys.glob(paste0(author,'/*.txt'))
  filelist=append(filelist, filestoadd)
  labels2=append(labels2, rep(authorname, length(filestoadd)))
}

alldocs=lapply(filelist, readerPlain)
```

```
names(alldocs)=filelist

mycorpus=Corpus(VectorSource(alldocs))
names(mycorpus)=filelist

mycorpus= tm_map(mycorpus, content_transformer(tolower))
mycorpus= tm_map(mycorpus, content_transformer(removeNumbers))
mycorpus= tm_map(mycorpus, content_transformer(removePunctuation))
mycorpus= tm_map(mycorpus, content_transformer(stripWhitespace))
mycorpus= tm_map(mycorpus, content_transformer(removeWords),
stopwords("SMART"))

DTM2=DocumentTermMatrix(mycorpus)

DTM2=removeSparseTerms(DTM2,0.975)

x2=as.matrix(DTM2)

x2=x2/rowSums(x2)
x2=idf.weight(x2)
```

We read in all the data and in the document term matrices used TF-IDF weights.

```
words=colnames(x)
words2=colnames(x2)


W=words[!(words %in% words2)] #if train is in test, return FALSE #creating
this mask to only contain unique
W2=words2[!(words2 %in% words)] #if test is in train, return False




words_matrix=matrix(0,nrow=nrow(x2), ncol=length(W)) #empty matrix
colnames(words_matrix)=W

words_matrix2=matrix(0,nrow=nrow(x), ncol=length(W2)) #empty matrix
colnames(words_matrix2)=W2

train_matrix= cbind(x,words_matrix2)
test_matrix=cbind(x2,words_matrix)
```

We pulled out the words that weren't in the training set (from the test set) and added them to the training set with a matrix of 0's and sorted it so there wouldn't be any words in one matrix that were not in the other. We did the same for the test matrix. This was so that both the test and training sets are built with the same words, even if many words don't appear in the entire training set or test set.

We used Naive Bayes to try to predict the authors.

```
#Naive Bayes Model

test_matrix=as.data.frame(test_matrix)
train_matrix=as.data.frame(train_matrix)

NB=naiveBayes(x=train_matrix,y=as.factor(labels),laplace=1)
predNB=predict(NB,test_matrix)

actual <- rep(1:50,each=50)

ToriTable <- table(predNB,actual)
correct = 0
for (i in seq(1,50)){
  correct = correct + ToriTable[i,i]
}
correct/2500

## [1] 0.4004
```

This ended up being 40% accurate. Much better than randomly guessing (2%)

Joe Ortiz was frequently misidentified as Jo Winterbottom. John Mastrini was frequently mididentified as Joe Ortiz. And Jan Lopatka was misidentified as Jane Macartney. I am not really sure why so many authors from this part of the alphabet were misidentified this much.

We used a random forest because it is likely that several words are very significant to identifying authors and would make very good nodes.

```
### Random Forest

rand <- randomForest(y=as.factor(labels), x= train_matrix, ntrees= 100)
pr <- predict(rand, test_matrix, type = "response")

vinnyTable <- table(pr, actual)

correct2 = 0
for (i in seq(1,50)){
  correct2 = correct2 + vinnyTable[i,i]
}
correct2

## [1] 1360

correct2/2500

## [1] 0.544
```

This ended up being 54% accurate! Better than Naive Bayes.

Pierre Tran is frequently mistaken for Marcel Michelson, but she is not mistaken for Pierre. Once again Jane Macartney is difficult to ID. She is identified as Jan Lopatka and John Mastrini. And once again Joe Ortiz is mistaken for Jo Winterbottom. Johnathan Birt is misIDed as John Mastrini. There seems to be a few people such as Jane Macartney who are just extremely difficult to ID.

```
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

#manipulating data
max(count.fields("groceries.txt", sep = ','))

## [1] 32

groceries_raw <- read.table("groceries.txt",sep = ",",col.names =
paste0("V",seq_len(32)),header=FALSE,fill=TRUE,na.strings = c("","NA"))

groceries_raw$index=seq(9835)

library(reshape2)
melt=melt(groceries_raw, id.var = 'index')

## Warning: attributes are not identical across measure variables; they will
## be dropped

melt=melt[order(melt$index),]

groceries <- split(x=melt$value, f=melt$index)
groceries <- lapply(groceries, unique)
groceries <- na.omit(groceries)

groceries_trans <- as(groceries, "transactions")

rules <- apriori(groceries_trans,
                      parameter=list(support=.01, confidence=.5, maxlen=4))

## Apriori
##
## Parameter specification:
```

```
##  confidence minval smax arem  aval originalSupport support minlen maxlen
##         0.5    0.1    1 none FALSE            TRUE    0.01      1      4
##  target   ext
##   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 [0.01s].
## writing ... [15 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
```

```
inspect(rules)
```

```
##    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
```

```
#inspect(subset(rules, subset=lift > 2.5))
#inspect(subset(rules, subset=confidence > 0.5))
inspect(subset(rules, subset=lift > 2.5 & confidence > 0.5))
```

```
##   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
## 3 {rolls/buns,
##    root vegetables} => {other vegetables} 0.01220132  0.5020921 2.594890
```

I picked lift threshold 2.5 and confidence threshold 0.5. Confidence is conditional probability that customer buy product A will also buy product B. It measures how often items in B appear in transactions that contain A. So higher confidence is better and I chose 0.5 since this is a high confidence level for our subsets. But Confidence does not measure if the association between A and B is random or not. Whereas, Lift measures the strength of association between two items. Lift shows if someone buys Product A, what % of chance of buying product B would increase. Greater lift values indicate stronger associations so I picked lift=2.5 here since this is among the highest lift level.

The results I got showed some reasonable patterns. Citrus fruit, root vegetables are assocaited with 'other vegetables'; root vegetables, tropical fruit are associated with 'other vegetables'; rolls/buns,root vegetables are associated with other vegetables. These item sets make sense since most of them are often purchased together in the grocery store.