STA 380 HW 2 - Clay Mason

Clay Mason

8/20/2018

STA 380, Part 2: Exercises 2

Problem 1 ** Austin Airport **

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 example, you might consider one of the following questions:

What is the best time of day to fly to minimize delays? What is the best time of year to fly to minimize delays? How do patterns of flights to different destinations or parts of the country change over the course of the year? What are the bad airports to fly to? But anything interesting will fly. If you want to try your hand at mapping or looking at geography, you can cross-reference the airport codes here: https://github.com/datasets/airport-codes (https://github.com/datasets/airport-codes). Combine this with a mapping package like ggmap, and you should have lots of possibilities!

```
# Import library
library(ggplot2)
rm(list=ls())

ABIA = read.csv("/Users/claytonmason/GitHub/STA_380_Clay/Data/ABIA.csv")

#head(ABIA, 5)
```

I added a column for pulling the Hour of the Day, a Unique flight identifier for counting, and a binary delayed flight column. I used this to also create a delayed flight only dataframe to make some analytics a little easier.

I also created an outbound-only and inbound-only data frame, but I didn't find this to be as helpful as I wanted. Most of the statistics were summarized line by line, so there was just one line of noise that i was trying to avoid.

```
# Departure time - Extract Hour of Day
ABIA$Departure_hour = as.numeric(substr(ABIA$DepTime, 1, nchar(ABIA$DepTime)-2))
#ABIA$Departure_hour

#unique flights column
ABIA$unique_flights <- paste(ABIA$UniqueCarrier,ABIA$FlightNum, ABIA$Month, ABIA$Dayo
fMonth)



# Create delay column
ABIA$Delay <- ifelse(ABIA$ArrDelay > 0, 1, ifelse(ABIA$ArrDelay <= 0, "0", ifelse(ABI
A$ArrDelay <= NA, NA,NA)))
#ABIA$Delay
ABIA$Delay <- as.numeric(as.character(ABIA$Delay))

#view null data
#colSums(!is.na(ABIA))

No_delay = sum(!is.na(ABIA$Delay[ABIA$Delay==0]))
Delay2 = sum(!is.na(ABIA$Delay[ABIA$Delay==1]))
Delay_NA = sum(is.na(ABIA$Delay))
#No_delay
#Delay2
#Delay_NA

Total_Delay_Col = No_delay + Delay2 + Delay_NA
#Total_Delay_Col
```

42.7% of flights were delayed base on the stringent criteria of anything more than zero minutes past the anticipated time of arrival.

```
#42.7% of flights are delayed
Delay2 / Total_Delay_Col
```

```
## [1] 0.4267177
```

```
#new delay data frame
ABIA_delay_df = ABIA[ABIA$Delay==1,]
#ABIA_delay_df

#new Outbound df
ABIA_Outbound_df = ABIA[ABIA$Dest!="AUS",]
ABIA_Outbound_df = ABIA_Outbound_df[ABIA_Outbound_df$Delay!="NA",]
#ABIA_Outbound_df

#new Inbound df
ABIA_Inbound_df = ABIA[ABIA$Dest=="AUS",]
ABIA_Inbound_df = ABIA_Inbound_df[ABIA_Inbound_df$Delay!="NA",]
#ABIA_Inbound_df
```

I calculated the average delay time by carrier and plotted this into a bar chart

```
#average delay by carrier
Carrier_mean = aggregate(ABIA_delay_df$ArrDelay, by=list(ABIA_delay_df$UniqueCarrier)
, FUN=mean)
Carrier_mean
```

```
##    Group.1        x
## 1       9E 27.00534
## 2       AA 28.55008
## 3       B6 41.74502
## 4       CO 30.20733
## 5       DL 29.23239
## 6       EV 36.18883
## 7       F9 20.18879
## 8       MQ 29.57084
## 9       NW 27.94203
## 10      OH 30.34235
## 11      OO 29.66935
## 12      UA 31.82909
## 13      US 16.11490
## 14      WN 24.45246
## 15      XE 25.25124
## 16      YV 35.35556
```

```
library(dplyr)
```

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

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

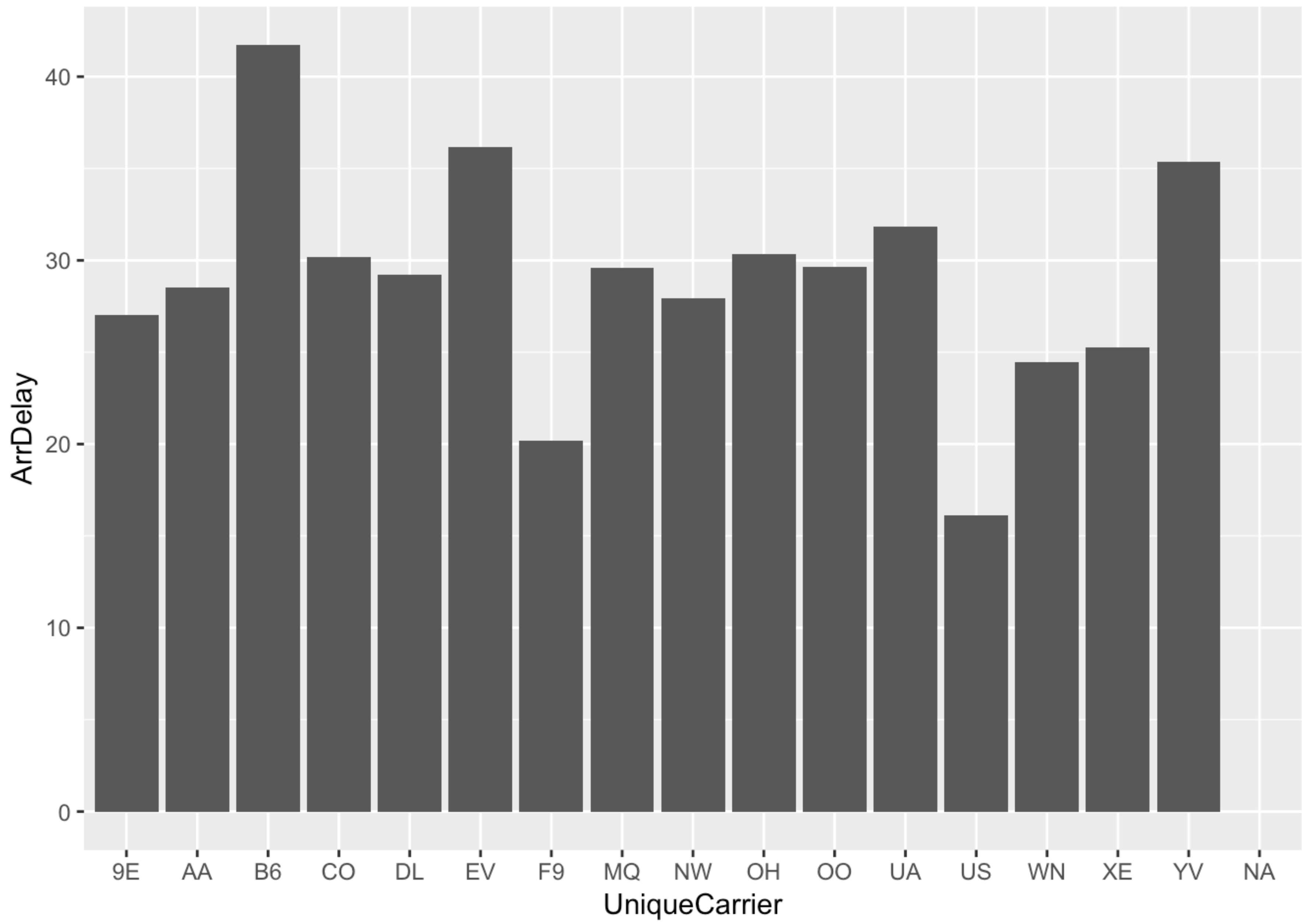
```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
ABIA %>%
  group_by(UniqueCarrier) %>%
  summarise(n_distinct(UniqueCarrier))
```

```
## # A tibble: 16 x 2
##    UniqueCarrier `n_distinct(UniqueCarrier)`
##    <fct>                               <int>
##  1 9E                                      1
##  2 AA                                      1
##  3 B6                                      1
##  4 CO                                      1
##  5 DL                                      1
##  6 EV                                      1
##  7 F9                                      1
##  8 MQ                                      1
##  9 NW                                      1
## 10 OH                                      1
## 11 OO                                      1
## 12 UA                                      1
## 13 US                                      1
## 14 WN                                      1
## 15 XE                                      1
## 16 YV                                      1
```

```
#average delay by carrier - plot
library(ggplot2)
ggplot(ABIA_delay_df) +
  stat_summary(aes(x = UniqueCarrier, y = ArrDelay),
               fun.y = function(x) mean(x),
               geom = "bar")
```

```
## Warning: Removed 1601 rows containing non-finite values (stat_summary).
```

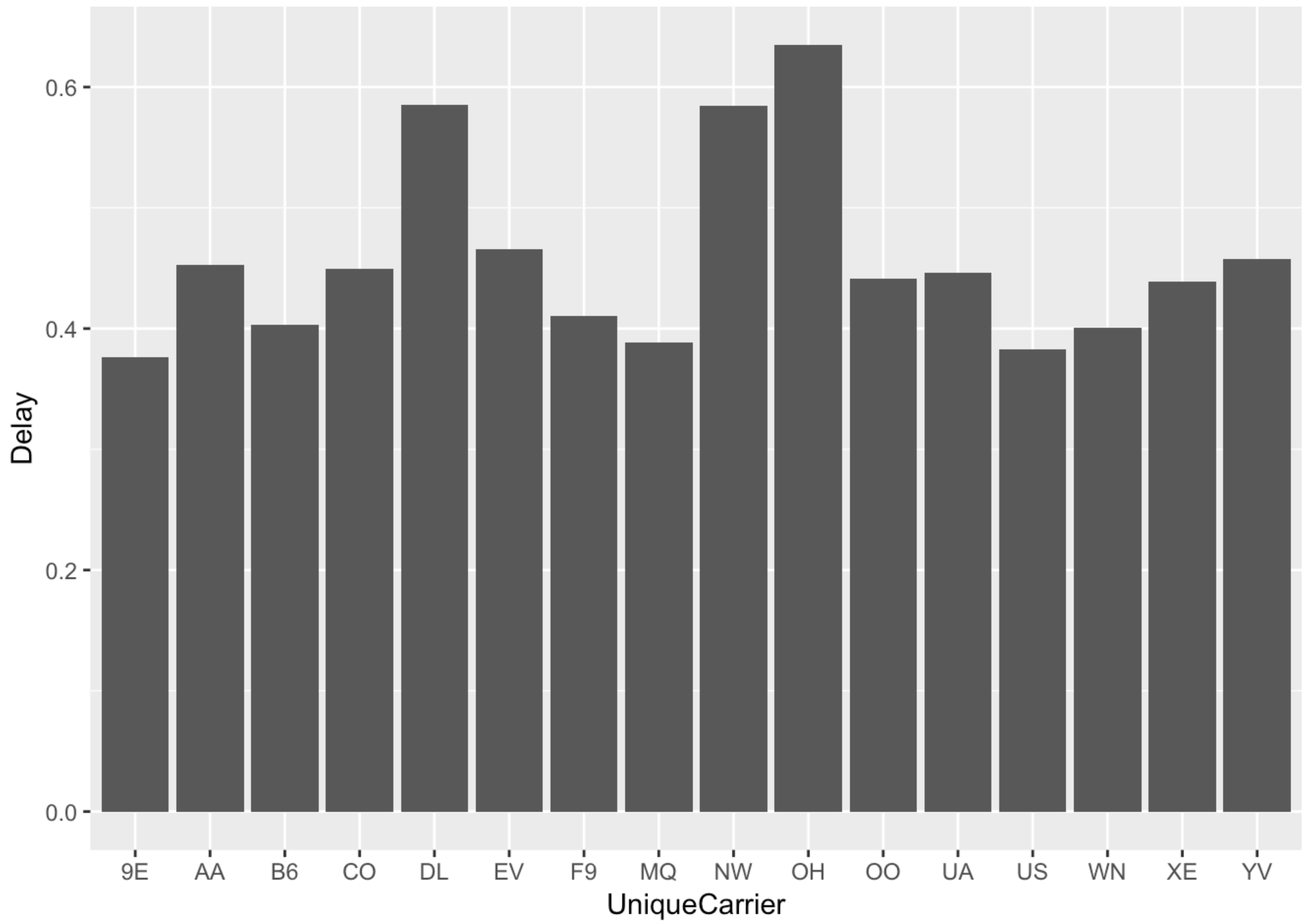


calculated the percentage of delays by carrier so that large carriers with more frequent flights would be more comparable to smaller carriers. This still isn't quite apples to apples as there would still be increased complexities with a larger carrier that could result in delays.

It would be preferable to identify when the greatest odds of delays are, which requires the transformation I mentioned.

```
#% of delays by carrier - plot
library(ggplot2)
ggplot(ABIA) +
  stat_summary(aes(x = UniqueCarrier, y = Delay),
               fun.y = function(x) sum(x)/length(x),
               geom = "bar")
```

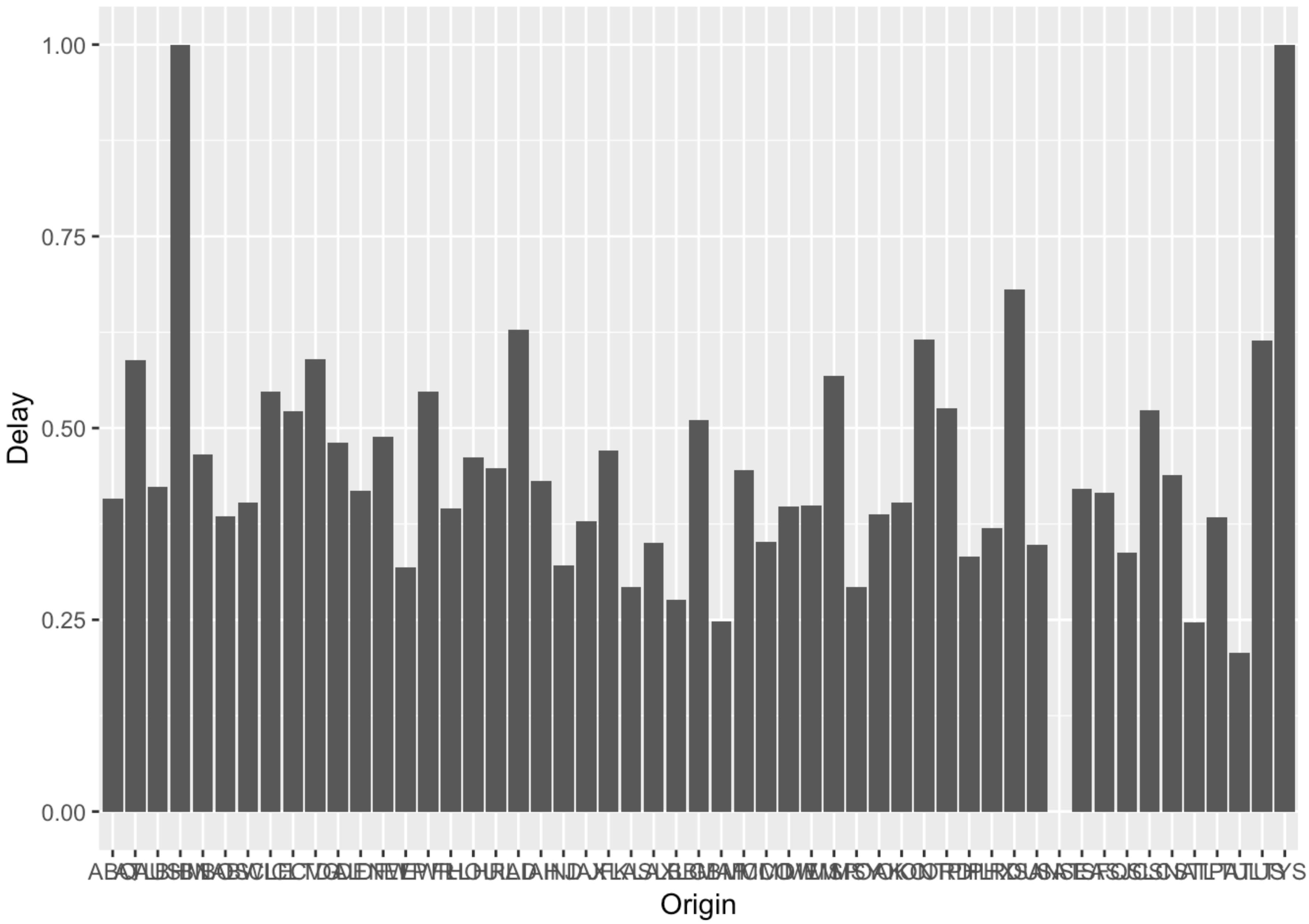
```
## Warning: Removed 1601 rows containing non-finite values (stat_summary).
```



calculated the percentage of delays by origin (where flights are coming from before they land in Austin) and plotted it into a chart that you cannot read. I then just published it to a table so that you can read it and gain some insight.

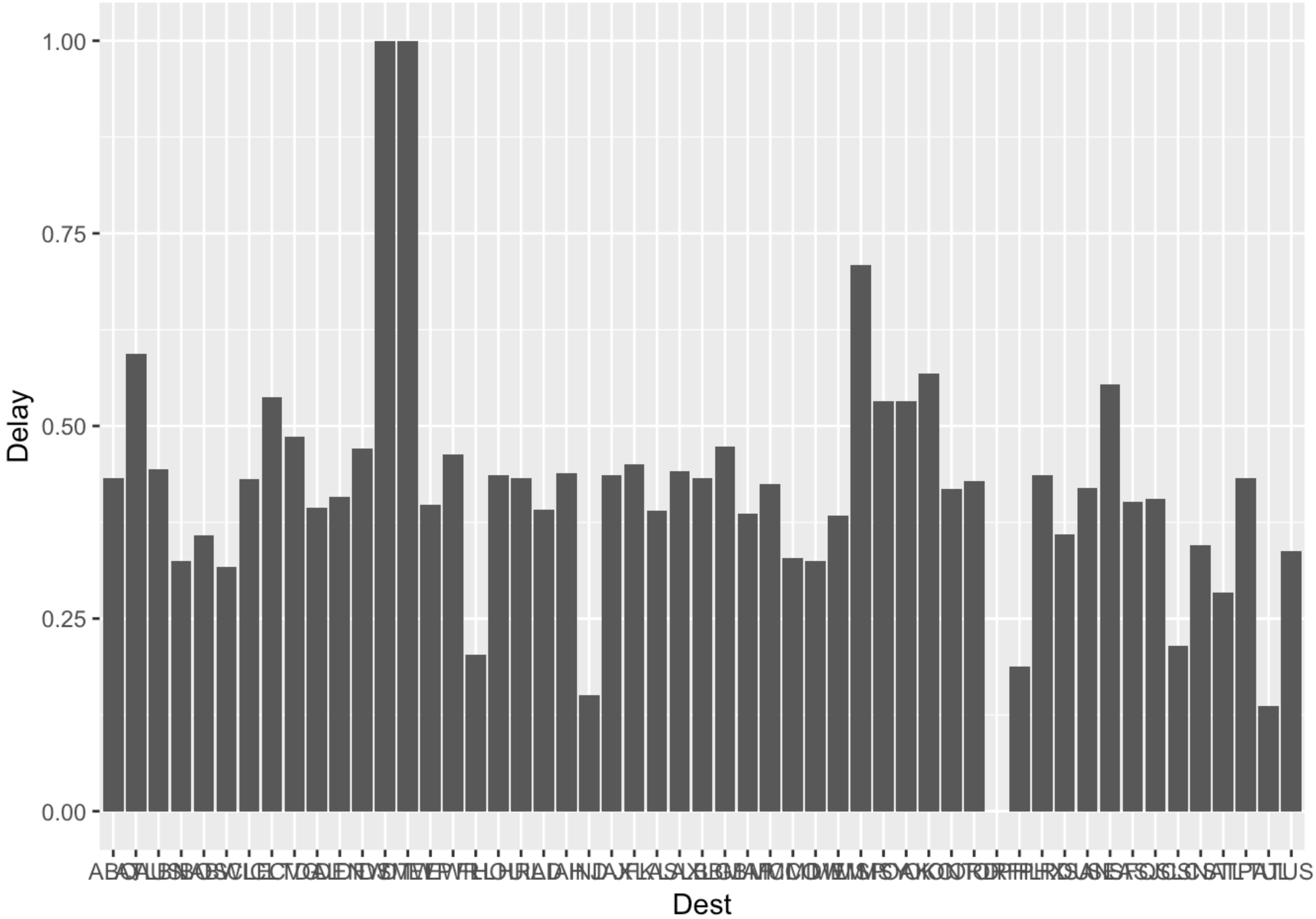
```
#% of delays by origin
library(ggplot2)
ggplot(ABIA) +
  stat_summary(aes(x = Origin, y = Delay),
               fun.y = function(x) sum(x)/length(x),
               geom = "bar")
```

```
## Warning: Removed 1601 rows containing non-finite values (stat_summary).
```




```
#% of delays by origin
library(ggplot2)
ggplot(ABIA) +
  stat_summary(aes(x = Dest, y = Delay),
               fun.y = function(x) sum(x)/length(x),
               geom = "bar")
```

```
## Warning: Removed 1601 rows containing non-finite values (stat_summary).
```



then did a similar method for Destination instead of Origin

```
#number of flights by Destination
#install.packages("data.table")
library(data.table)
```

```
##
## Attaching package: 'data.table'
```

```
## The following objects are masked from 'package:dplyr':
##
##     between, first, last
```

```
DT2 <- data.table(ABIA_Outbound_df)
DT2[, .(outbound_flights = length(unique(unique_flights)),delays = sum(Delay), delay_
percent = sum(Delay)/length(unique(unique_flights))), by = Dest]
```

```
##     Dest outbound_flights delays delay_percent
##  1:  ORD             2412   1034     0.4286899
##  2:  PHX             2765   1207     0.4365280
##  3:  MEM              807    310     0.3841388
```

```
## 4:  DFW            5347   2514     0.4701702
## 5:  MSP              55     39     0.7090909
## 6:  IAH            3636   1596     0.4389439
## 7:  JFK            1325    597     0.4505660
## 8:  MSY             443    236     0.5327314
## 9:  TUS             228     77     0.3377193
## 10: MDW             707    230     0.3253182
## 11: SFO             605    243     0.4016529
## 12: SNA             243     84     0.3456790
## 13: ONT             304    127     0.4177632
## 14: SLC             546    117     0.2142857
## 15: DEN            2653   1082     0.4078402
## 16: ATL            2213   1313     0.5933122
## 17: LAX            1718    759     0.4417928
## 18: LAS            1225    478     0.3902041
## 19: SAN             717    301     0.4198047
## 20: ABQ             432    187     0.4328704
## 21: BWI             730    231     0.3164384
## 22: MCI             445    189     0.4247191
## 23: CVG             642    312     0.4859813
## 24: DAL            5442   2142     0.3936053
## 25: HOU            2275    993     0.4364835
## 26: CLE             376    162     0.4308511
## 27: IAD             650    254     0.3907692
## 28: RDU             228     82     0.3596491
## 29: EWR             936    433     0.4626068
## 30: ELP            1349    536     0.3973314
## 31: HRL             359    155     0.4317549
## 32: MCO             630    207     0.3285714
## 33: BOS             363    130     0.3581267
## 34: OKC              88     50     0.5681818
## 35: TUL              88     12     0.1363636
## 36: TPA             366    158     0.4316940
## 37: SJC             933    378     0.4051447
## 38: MAF             466    180     0.3862661
## 39: STL              88     25     0.2840909
## 40: LBB             690    298     0.4318841
## 41: BNA             792    257     0.3244949
## 42: JAX             225     98     0.4355556
## 43: <NA>              1     NA            NA
## 44: PHL             288     54     0.1875000
## 45: DSM               1      1     1.0000000
## 46: SEA             148     82     0.5540541
## 47: FLL             478     97     0.2029289
## 48: LGB             241    114     0.4730290
## 49: IND             213     32     0.1502347
## 50: CLT             649    349     0.5377504
## 51: OAK             235    125     0.5319149
## 52: DTW               1      1     1.0000000
##     Dest outbound_flights delays delay_percent
```

```r
#number of flights by Origin
#install.packages("data.table")
library(data.table)
DT <- data.table(ABIA_Inbound_df)
DT[, .(inbound_flights = length(unique(unique_flights)),delays = sum(Delay), delay_pe
rcent = sum(Delay)/length(unique(unique_flights))), by = Origin]
```

```
##     Origin inbound_flights delays delay_percent
##  1:    MEM             816    326     0.3995098
##  2:    MCI             447    199     0.4451902
##  3:    LAX            1715    601     0.3504373
##  4:    ELP            1338    426     0.3183857
##  5:    JFK            1308    616     0.4709480
##  6:    ORD            2424   1276     0.5264026
##  7:    MSY             441    129     0.2925170
##  8:    SAN             715    249     0.3482517
##  9:    IAH            3651   1575     0.4313887
## 10:    ATL            2216   1304     0.5884477
## 11:    DFW            5344   2613     0.4889596
## 12:    IAD             616    387     0.6282468
## 13:    LAS            1225    359     0.2930612
## 14:    CLE             374    205     0.5481283
## 15:    BWI             728    293     0.4024725
## 16:    BOS             366    141     0.3852459
## 17:    ONT             304    187     0.6151316
## 18:    LBB             689    190     0.2757620
## 19:    TPA             367    141     0.3841962
## 20:    DAL            5464   2625     0.4804173
## 21:    PHX            2778   1028     0.3700504
## 22:    ABQ             427    174     0.4074941
## 23:    SFO             604    251     0.4155629
## 24:    JAX             227     86     0.3788546
## 25:    SJC             936    316     0.3376068
## 26:    MCO             629    221     0.3513514
## 27:    CVG             650    383     0.5892308
## 28:    SLC             547    286     0.5228519
## 29:    HOU            2270   1048     0.4616740
## 30:    DEN            2708   1132     0.4180207
## 31:    BNA             793    369     0.4653216
## 32:    STL              89     22     0.2471910
## 33:    HRL             357    160     0.4481793
## 34:    EWR             929    509     0.5479010
## 35:    OKC              87     35     0.4022989
## 36:    TUL              87     18     0.2068966
## 37:    SNA             246    108     0.4390244
## 38:    MAF             468    116     0.2478632
## 39:    TUS             228    140     0.6140351
## 40:    RDU             229    156     0.6812227
## 41:    MSP              51     29     0.5686275
```

```
## 42:    MDW            709    282     0.3977433
## 43:   <NA>              1     NA            NA
## 44:    TYS              3      3     1.0000000
## 45:    BHM              1      1     1.0000000
## 46:    PHL            289     96     0.3321799
## 47:    SEA            145     61     0.4206897
## 48:    LGB            243    124     0.5102881
## 49:    FLL            478    189     0.3953975
## 50:    IND            215     69     0.3209302
## 51:    CLT            657    343     0.5220700
## 52:    OAK            235     91     0.3872340
##      Origin inbound_flights delays delay_percent
```

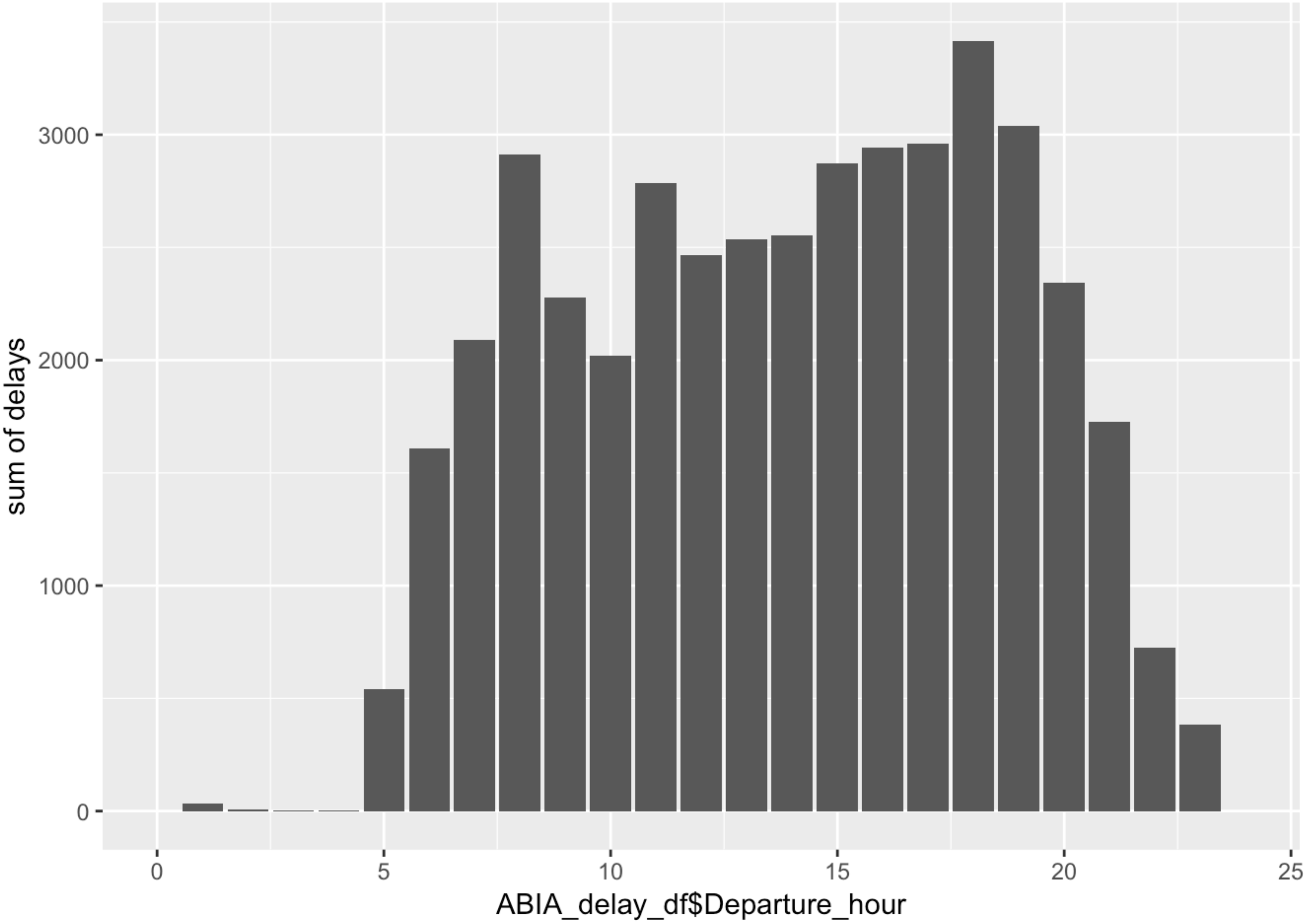
I plotted the number of delays by the hour of the day. I should have done this as a percentage to normalize peak flying hours with the rest of the day. Again, we are trying to find when the highest odds of delays are.

I also plotted the mean delay by the hour of the day, and it seems like there is a bias in the data. The later in the day (the longer the delay that's already occurred), the longer the mean delay time. I don't think this statistic is revealing.

I also plotted delays by month with December (holidays) having a large amount of delays.

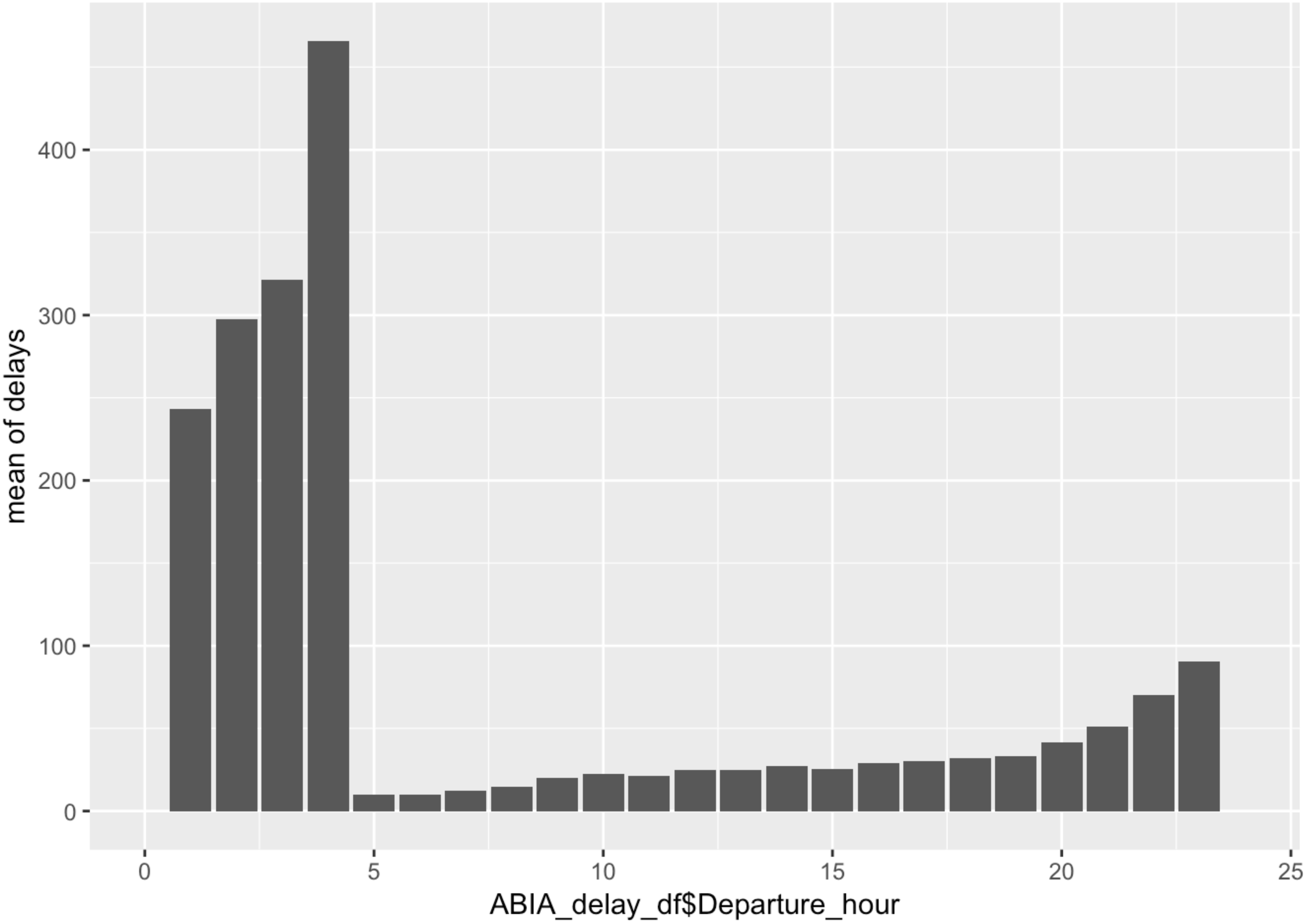
```
#count of delays by hour
ggplot(data = ABIA_delay_df, aes(ABIA_delay_df$Departure_hour, ABIA_delay_df$Delay<-1
)) + stat_summary(fun.y = sum, geom = "bar") + xlim(0,24) + scale_y_continuous("sum o
f delays")
```

```
## Warning: Removed 1715 rows containing non-finite values (stat_summary).
```



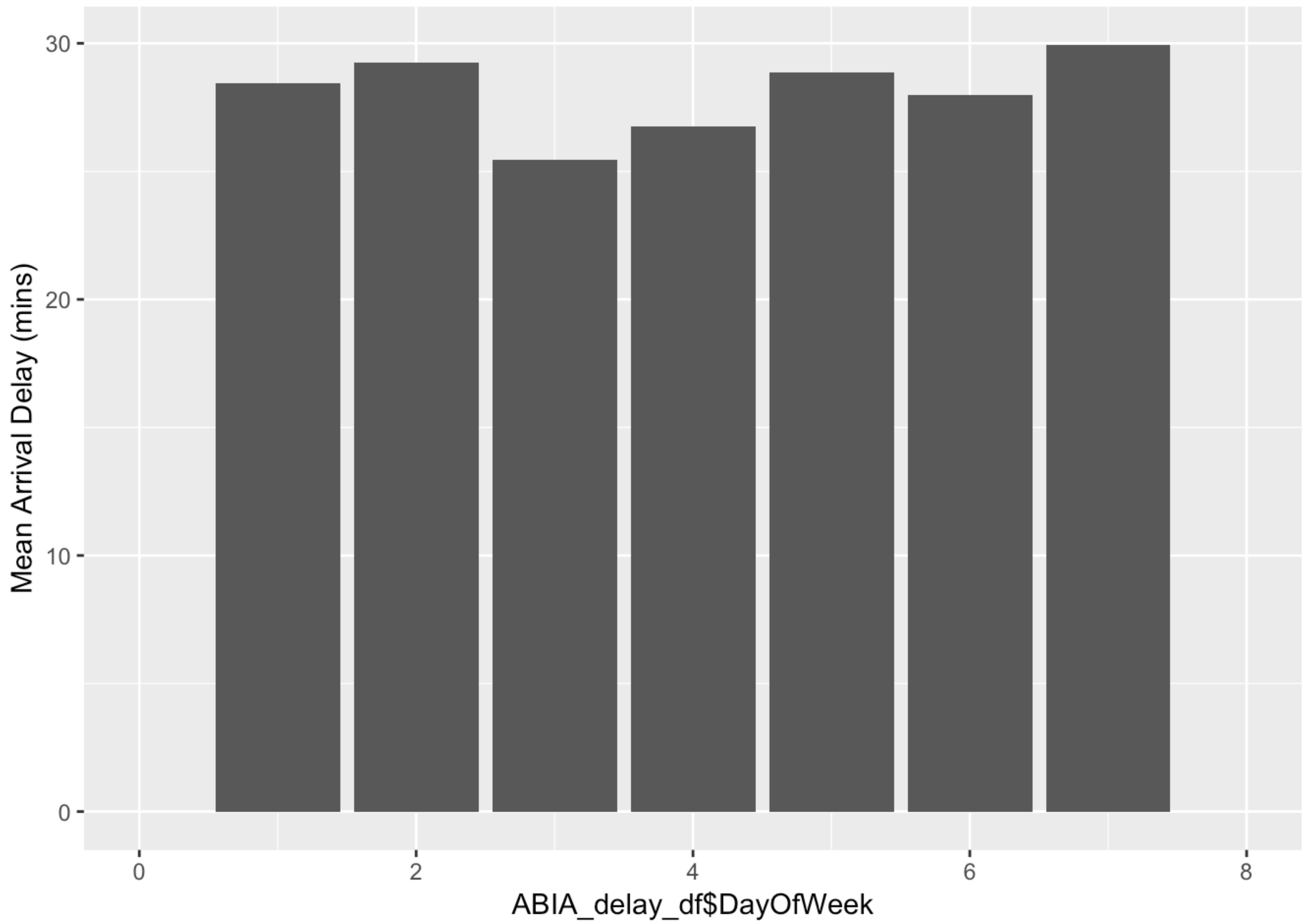

```
#mean arrival delays by hour of the day
ggplot(data = ABIA_delay_df, aes(ABIA_delay_df$Departure_hour, ABIA_delay_df$ArrDelay
)) + stat_summary(fun.y = mean, geom = "bar") + xlim(0,24) + scale_y_continuous("mean
of delays")
```

```
## Warning: Removed 1715 rows containing non-finite values (stat_summary).
```



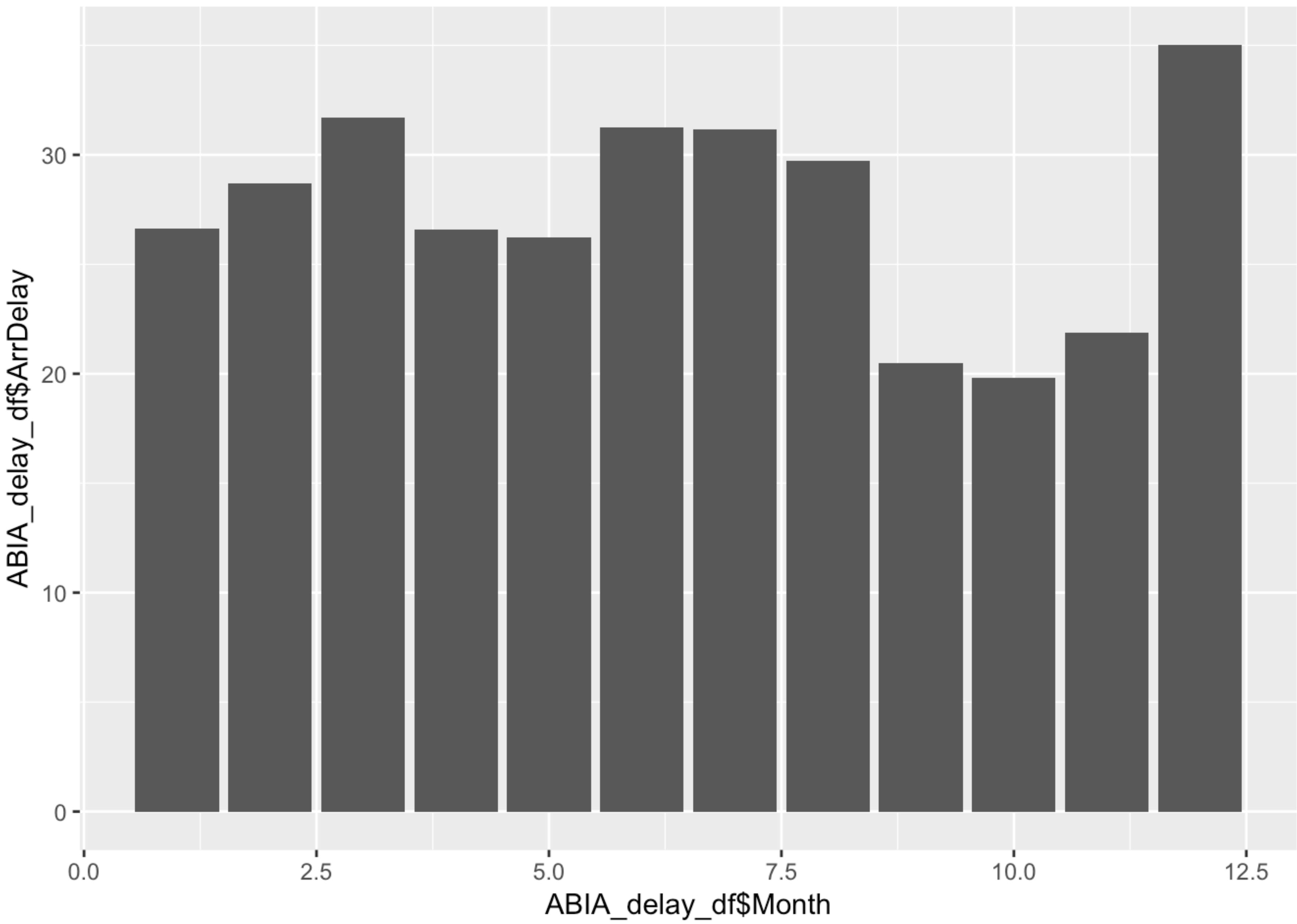

```
# Plot average arrival delays by day of week
ggplot(data = ABIA_delay_df, aes(ABIA_delay_df$DayOfWeek, ABIA_delay_df$ArrDelay)) +
stat_summary(fun.y = mean, geom = "bar")  + xlim(0,8) + scale_y_continuous("Mean Arri
val Delay (mins)")
```

```
## Warning: Removed 1601 rows containing non-finite values (stat_summary).
```




```
# Plot average arrival delays by month
ggplot(data = ABIA_delay_df, aes(ABIA_delay_df$Month, ABIA_delay_df$ArrDelay)) + stat
_summary(fun.y = mean, geom = "bar")
```

```
## Warning: Removed 1601 rows containing non-finite values (stat_summary).
```



Problem 2 ** Author Attribution **

Revisit the Reuters C50 corpus that we explored in class. Your task is to build two separate models (using any combination of tools you see fit) for predicting the author of an article on the basis of that article's textual content. Describe clearly what models you are using, how you constructed features, and so forth. Yes, this is a supervised learning task, but it potentially draws on a lot of what you know about unsupervised learning, since constructing features for a document might involve dimensionality reduction.

In the C50train directory, you have ~50 articles from each of 50 different authors (one author per directory). Use this training data (and this data alone) to build the two models. Then apply your model to the articles by the same authors in the C50test directory, which is about the same size as the training set. How well do your models do at predicting the author identities in this out-of-sample setting? Are there any sets of authors whose articles seem difficult to distinguish from one another? Which model do you prefer?

Note: you will need to figure out a way to deal with words in the test set that you never saw in the training set. This is a nontrivial aspect of the modeling exercise. You might, for example, consider adding a pseudo-word to the training set vocabulary, corresponding to "word not seen before," and add a pseudo-count to it so it doesn't look like these out-of-vocabulary words have zero probability on the testing set.

```r
rm(list=ls())
## The tm library and related plugins comprise R's most popular text-mining stack.
## See http://cran.r-project.org/web/packages/tm/vignettes/tm.pdf



## tm has many "reader" functions.  Each one has
## arguments elem, language, id
## (see ?readPlain, ?readPDF, ?readXML, etc)
## This wraps another function around readPlain to read
## plain text documents in English.
readerPlain = function(fname){
  readPlain(elem=list(content=readLines(fname)),
            id=fname, language='en') }

# Import libraries
library(tm)
```

```
## Loading required package: NLP
```

```
##
## Attaching package: 'NLP'
```

```
## The following object is masked from 'package:ggplot2':
##
##     annotate
```

```r
library(SnowballC)
library(plyr)
```

```
## ------------------------------------------------------------------------------
```

```
## You have loaded plyr after dplyr - this is likely to cause problems.
## If you need functions from both plyr and dplyr, please load plyr first, then dplyr
:
## library(plyr); library(dplyr)
```

```
## ------------------------------------------------------------------------------
```

```
##
## Attaching package: 'plyr'
```

```
## The following objects are masked from 'package:dplyr':
##
##     arrange, count, desc, failwith, id, mutate, rename, summarise,
##     summarize
```

```
library(tm)
library(magrittr)
library(slam)
```

```
##
## Attaching package: 'slam'
```

```
## The following object is masked from 'package:data.table':
##
##     rollup
```

```
library(proxy)
```

```
##
## Attaching package: 'proxy'
```

```
## The following objects are masked from 'package:stats':
##
##     as.dist, dist
```

```
## The following object is masked from 'package:base':
##
##     as.matrix
```

```
library(glmnet)
```

```
## Loading required package: Matrix
```

```
## Loading required package: foreach
```

```
## Loaded glmnet 2.0-16
```

```r
library(nnet)

# single corpus
## "globbing" = expanding wild cards in filename paths
setwd("/Users/claytonmason/GitHub/STA_380_Clay/Data")
file_list_import_train = Sys.glob('../data/ReutersC50/C50train/*')

# Loop through to get the files/authors
file_list_train = c()
labels_train = c()
for(author_train in file_list_import_train) {
  author_name_train = substring(author_train, first=29)
  files_to_add_train = Sys.glob(paste0(author_train, '/*.txt'))
  file_list_train = append(file_list_train, files_to_add_train)
  labels_train = append(labels_train, rep(author_name_train, length(files_to_add_trai
n)))
}
#file_list_train

# Remove .txt extension from the file name
all_docs_train = lapply(file_list_train, readerPlain)
names(all_docs_train) = file_list_train
names(all_docs_train) = sub('.txt', '', names(all_docs_train))

## once you have documents in a vector, you
## create a text mining 'corpus' with:
my_corpus_train = Corpus(VectorSource(all_docs_train))

#ugh - https://stackoverflow.com/questions/40462805/names-function-in-r-not-working-a
s-expected
#https://stackoverflow.com/questions/10566473/names-attribute-must-be-the-same-length
-as-the-vector
length(my_corpus_train)
```

```
## [1] 2500
```

```r
length(labels_train)
```

```
## [1] 2500
```

```r
#names(my_corpus_train) = labels_train
#labels_train
```

Naive Bayes model Data Import and Cleaning I pulled in the files with a For Loop, but i ran into an issue with the names function. This was the beginning of my problems. I was running into this same issue regardless of how I was trying to apply the names.

I calculated the length of the vector used for the naming convention at 2500 rows, but it kept saying i only had three items in the list. As previously mentioned, I found some other similar code on the internet and also the tutorial exercise, but I kept receiving the same error below.

“Error in names(my_corpus_train) = labels_train : ‘names’ attribute [2500] must be the same length as the vector [3]”

I made everything lowercase, removed numbers, removed punctuation, removed extra white spaces and used the “SMART” stop words tool . Everything was similar to the example exercise that was provided to us.

I then created a Doc term matrix + the sparse matrix with the clean data.

I looked through a few of the items and found some frequent terms for a sanity check.

I dropped sparse words based on a threshold of 94% (count of 0 in 94% of docs), and then I created a new matrix.

```
## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_corpus_train = tm_map(my_corpus_train, content_transformer(tolower)) # make everyt
hing lowercase
```

```
## Warning in tm_map.SimpleCorpus(my_corpus_train,
## content_transformer(tolower)): transformation drops documents
```

```
my_corpus_train = tm_map(my_corpus_train, content_transformer(removeNumbers)) # remov
e numbers
```

```
## Warning in tm_map.SimpleCorpus(my_corpus_train,
## content_transformer(removeNumbers)): transformation drops documents
```

```
my_corpus_train = tm_map(my_corpus_train, content_transformer(removePunctuation)) # r
emove punctuation
```

```
## Warning in tm_map.SimpleCorpus(my_corpus_train,
## content_transformer(removePunctuation)): transformation drops documents
```

```
my_corpus_train = tm_map(my_corpus_train, content_transformer(stripWhitespace)) # rem
ove excess white-space
```

```
## Warning in tm_map.SimpleCorpus(my_corpus_train,
## content_transformer(stripWhitespace)): transformation drops documents
```

```
## Remove stopwords.  Always be careful with this: one person's trash is another one'
s treasure.
#stopwords("en")
#stopwords("SMART")
#?stopwords
my_corpus_train = tm_map(my_corpus_train, content_transformer(removeWords), stopwords
("SMART")) # remove stop words
```

```
## Warning in tm_map.SimpleCorpus(my_corpus_train,
## content_transformer(removeWords), : transformation drops documents
```

```
my_corpus_train = tm_map(my_corpus_train, stemDocument) # combine stem words
```

```
## Warning in tm_map.SimpleCorpus(my_corpus_train, stemDocument):
## transformation drops documents
```

```
## create a doc-term-matrix
DTM_train = DocumentTermMatrix(my_corpus_train)
DTM_train # some basic summary statistics
```

```
## <<DocumentTermMatrix (documents: 2500, terms: 22324)>>
## Non-/sparse entries: 435143/55374857
## Sparsity           : 99%
## Maximal term length: 44
## Weighting          : term frequency (tf)
```

```
# a special kind of sparse matrix format
class(DTM_train)
```

```
## [1] "DocumentTermMatrix"    "simple_triplet_matrix"
```

```
## You can inspect its entries...
inspect(DTM_train[1:10,1:20])
```

```
## <<DocumentTermMatrix (documents: 10, terms: 20)>>
## Non-/sparse entries: 62/138
## Sparsity           : 69%
## Maximal term length: 9
## Weighting          : term frequency (tf)
## Sample             :
##     Terms
## Docs access agenc announc author board busi charact charg commiss comput
##   1       1     1       1      1     1    2       4     1       3      1
##   10      4     0       0      0     0    2       4     0       0      4
##   2       0     0       1      0     0    2       4     1       0      1
##   3       2     0       0      0     0    0       4     0       0      0
##   4       0     0       1      4     2    1       4     0       0      1
##   5       0     0       1      4     2    1       4     0       0      1
##   6       0     0       0      0     0    0       4     0       0      0
##   7       0     1       0      2     0    0       4     0       5      1
##   8       0     0       0      0     3    1       4     1       2      0
##   9       0     1       0      1     0    1       4     1       2      0
```

```
## ...find words with greater than a min count...
#findFreqTerms(DTM_train, 100)
```

I chose to inspect the word “fed” because I figured this would likely be tied to some other words. This wasn’t really necessary for the exercise, but more for practicing the function.

```
## ...or find words whose count correlates with a specified word.
findAssocs(DTM_train, "fed", .5)
```

```
## $fed
##        litan      larocca        rivlin         elat    underwrit
##         0.80         0.75          0.74         0.64         0.62
## glasssteagal         alic      comptrol
##         0.59         0.57          0.52
```

```
## Finally, drop those terms that only occur in one or two documents
## This is a common step: the noise of the "long tail" (rare terms)
##  can be huge, and there is nothing to learn if a term occured once.
## Below removes those terms that have count 0 in >94% of docs.
# Remove sparse terms
DTM_train = removeSparseTerms(DTM_train, 0.94)
#DTM_train





# Create a new matrix
X_train = as.matrix(DTM_train)
#X_train
```

I repeated the same steps for the test data.

As the instructions mentioned, I will need to compare the words between the two sets of data and account for missing words in the training data.

```
########## repeat steps for test data

file_list_import_test = Sys.glob('../data/ReutersC50/C50test/*')

#  get the files and the authors
file_list_test = c()
labels_test = c()
for(author_test in file_list_import_test) {
  author_name_test = substring(author_test, first=29)
  files_to_add_test = Sys.glob(paste0(author_test, '/*.txt'))
  file_list_test = append(file_list_test, files_to_add_test)
  labels_test = append(labels_test, rep(author_name_test, length(files_to_add_test)))
}
#file_list_test

# Read in file_list and remove .txt from the file name
all_docs_test = lapply(file_list_test, readerPlain)
names(all_docs_test) = file_list_test
names(all_docs_test) = sub('.txt', '', names(all_docs_test))
#all_docs_test

## once you have documents in a vector, you
## create a text mining 'corpus' with:
my_corpus_test = Corpus(VectorSource(all_docs_test))

#ugh - https://stackoverflow.com/questions/40462805/names-function-in-r-not-working-a
s-expected
#https://stackoverflow.com/questions/10566473/names-attribute-must-be-the-same-length
-as-the-vector
length(my_corpus_test)
```

```
## [1] 2500
```

```
length(labels_test)
```

```
## [1] 2500
```

```
#names(my_corpus_test) = labels_test



## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_corpus_test = tm_map(my_corpus_test, content_transformer(tolower)) # make everythi
ng lowercase
```

```
## Warning in tm_map.SimpleCorpus(my_corpus_test,
## content_transformer(tolower)): transformation drops documents
```

```
my_corpus_test = tm_map(my_corpus_test, content_transformer(removeNumbers)) # remove
numbers
```

```
## Warning in tm_map.SimpleCorpus(my_corpus_test,
## content_transformer(removeNumbers)): transformation drops documents
```

```
my_corpus_test = tm_map(my_corpus_test, content_transformer(removePunctuation)) # rem
ove punctuation
```

```
## Warning in tm_map.SimpleCorpus(my_corpus_test,
## content_transformer(removePunctuation)): transformation drops documents
```

```
my_corpus_test = tm_map(my_corpus_test, content_transformer(stripWhitespace)) # remov
e excess white-space
```

```
## Warning in tm_map.SimpleCorpus(my_corpus_test,
## content_transformer(stripWhitespace)): transformation drops documents
```

```
## Remove stopwords.  Always be careful with this: one person's trash is another one'
s treasure.
#stopwords("en")
#stopwords("SMART")
#?stopwords
my_corpus_test = tm_map(my_corpus_test, content_transformer(removeWords), stopwords("
SMART")) # remove stop words
```

```
## Warning in tm_map.SimpleCorpus(my_corpus_test,
## content_transformer(removeWords), : transformation drops documents
```

```
my_corpus_test = tm_map(my_corpus_test, stemDocument) # combine stem words
```

```
## Warning in tm_map.SimpleCorpus(my_corpus_test, stemDocument):
## transformation drops documents
```

```
## create a doc-term-matrix
DTM_test = DocumentTermMatrix(my_corpus_test)
DTM_test # some basic summary statistics
```

```
## <<DocumentTermMatrix (documents: 2500, terms: 22844)>>
## Non-/sparse entries: 440961/56669039
## Sparsity           : 99%
## Maximal term length: 45
## Weighting          : term frequency (tf)
```

```
# a special kind of sparse matrix format
class(DTM_test)
```

```
## [1] "DocumentTermMatrix"    "simple_triplet_matrix"
```

```
## You can inspect its entries...
inspect(DTM_test[1:10,1:20])
```

```
## <<DocumentTermMatrix (documents: 10, terms: 20)>>
## Non-/sparse entries: 50/150
## Sparsity           : 75%
## Maximal term length: 10
## Weighting          : term frequency (tf)
## Sample             :
##     Terms
## Docs aaron account address affect agenc allow approach bank base begin
##   1      1       4       2      1     1     1        1    1    1     1
##   10     0       0       1      1     2     0        0    9    0     0
##   2      0       0       1      0     0     0        0    6    0     0
##   3      0       0       0      0     0     1        0    0    2     3
##   4      0       0       0      0     0     0        0    0    0     0
##   5      1       1       1      0     0     0        0    0    0     1
##   6      0       0       0      0     0     1        3    8    3     0
##   7      0       1       1      1     0     0        0   10    0     1
##   8      1       0       0      0     2     3        1    0    0     0
##   9      1       0       0      0     1     3        0    0    0     0
```

```
## ...find words with greater than a min count...
#findFreqTerms(DTM_test, 50)

## ...or find words whose count correlates with a specified word.
findAssocs(DTM_test, "fed", .5)
```

```
## $fed
##        afoul baltimorebas      firewal       hyland      larocco
##         0.59         0.59         0.59         0.59         0.59
##  lifesupport glasssteagal
##         0.59         0.52
```

```
## Finally, drop those terms that only occur in one or two documents
## This is a common step: the noise of the "long tail" (rare terms)
##  can be huge, and there is nothing to learn if a term occured once.
## Below removes those terms that have count 0 in >95% of docs.
# Remove sparse terms
DTM_test = removeSparseTerms(DTM_test, 0.94)
#DTM_test



# Create a dense matrix
X_test = as.matrix(DTM_test)
#X_test
```

dealing with extra/missing words between the datasets In this next section, I grab all of the words not in the test set that are in the training set, and vice versa. I will drop the extra words from the test set and add missing words to the test set to ensure my matrices are the same size later for the multiplication.

```
# Pull training set words
X_words = colnames(X_train)
#X_words



# Pull test set words
X_test_words = colnames(X_test)
#X_test_words

# initialize the vectors that will add the words to the matrices
test_add = vector(length=0)
test_drop = vector(length=0)

# Add words not in the train to the vector test_drop
for (test_word in X_test_words) {
  if (!test_word %in% X_words) {
    test_drop <- c(test_drop, test_word)
  }
}
#test_drop




# Add words not in test set to the vector test_add
for (word in X_words) {
  if (!word %in% X_test_words) {
    test_add <- c(test_add, word)
  }
}
#test_add
```

```
# initialize the matrix insert with a bunch of zeroes.

zero <- matrix(0, nrow = nrow(X_train), ncol=length(test_add))

# Name the columns
colnames(zero) <- test_add

# Add the blank matrix insert
X2_test = cbind(X_test, zero)


# sort the columns to match the train index
X2_test = X2_test[,order(colnames(X2_test))]
#X2_test

# Drop the words from the test_drop vector so that the matrices will match.
X2_test = X2_test[,!colnames(X2_test) %in% test_drop]

#X2_test




# Create a dense matrix
X = as.matrix(DTM_train)
```

After matching the matrices up, I was ready to start applying values to the words. I used a smoothing factor, which is just the 1/2500 or the lengtho of the matrix.

My labels/name issue will come up again because I needed those values to properply apply values to each author.

We transformed the values with log and we transposed the matrix so that we could multiply with the probability vector. We added the prediction column based on the max values of this multiplication, but I couldn't figure out how to check the accuracy of the prediction after adding it to the column. My index for each row was just a number instead of the author names, so I didn't have the actual values to compare to my prediction column.

This resulted in a zero accuracy for each row.

Had I been able to return a 1 for correct predictions and 0 for incorrect predictions. I could have taken the average of this column to get the overall model accuracy, and then i could have taken it a step further to analyze the accuracy at the author level.

```
# Calculate the smoothing factor
smooth_count = 1/nrow(X)
#nrow(X)
#smooth_count

#colnames(X)
#labels_train


#ugh I believe this is where my issue lies
# Add the smoothing factor and aggregate the word counts + smoothing factor for each
author
by_word_wc = rowsum(X + smooth_count, labels_train)
#by_word_wc
#smooth_count
#X


# Sum the word counts + smoothing factor for each word for each author
total_wc = rowSums(by_word_wc)
#total_wc


#  multinomial probability vector
w = by_word_wc / total_wc
#w

# Log the vector for easier interpretability
w = log(w)
#w
# Set X2 equal to the multinomial probability vector w
X2 = w


# Transpose the multinomial probability vector for matrix multiplication
X2 = t(X2)
#X2

# Multiply the test matrix by X2
log_prob = X2_test %*% X2
colnames(log_prob)
```

```
##  [1] "AaronPressman"     "AlanCrosby"        "AlexanderSmith"
##  [4] "BenjaminKangLim"   "BernardHickey"     "BradDorfman"
##  [7] "DarrenSchuettler"  "DavidLawder"       "EdnaFernandes"
## [10] "EricAuchard"       "FumikoFujisaki"    "GrahamEarnshaw"
## [13] "HeatherScoffield"  "JaneMacartney"     "JanLopatka"
## [16] "JimGilchrist"      "JoeOrtiz"          "JohnMastrini"
## [19] "JonathanBirt"      "JoWinterbottom"    "KarlPenhaul"
## [22] "KeithWeir"         "KevinDrawbaugh"    "KevinMorrison"
## [25] "KirstinRidley"     "KouroshKarimkhany" "LydiaZajc"
## [28] "LynneO'Donnell"    "LynnleyBrowning"   "MarcelMichelson"
## [31] "MarkBendeich"      "MartinWolk"        "MatthewBunce"
## [34] "MichaelConnor"     "MureDickie"        "NickLouth"
## [37] "PatriciaCommins"   "PeterHumphrey"     "PierreTran"
## [40] "RobinSidel"        "RogerFillion"      "SamuelPerry"
## [43] "SarahDavison"      "ScottHillis"       "SimonCowell"
## [46] "TanEeLyn"          "TheresePoletti"    "TimFarrand"
## [49] "ToddNissen"        "WilliamKazer"
```

```
# Get the prediction by return the column name of the max value for each document
predict = colnames(log_prob)[max.col(log_prob)]
#predict

# Add the prediction the the matrix
log_prob = cbind(log_prob, predict)
#head(log_prob,10)
#rownames(log_prob) #these are numbers. I need
#colnames(log_prob)
#log_prob[,51]



### i cannot figure out how to check the accuracy of my predictions
# Create a column that checks the prediction against the actual

accurate = as.integer(rownames(log_prob) == log_prob[,51])
#accurate
```

PCA The Principal Component Analysis was less of a success compared to my attempt at Naive Bayes, which I felt like I got all the way until the very end, which unfortunately is where it matters most.

The PCA i was able to replicate the tutorial exercise, which only looked at the Simon author files.

I've had to start over too many times now after issues with non-conformable argurments when trying to start my regression.

Below is just what i was able to successfully perform. I cut my failed attempts. I had not started on this problem at the time of the office hours last Friday. I had only made progress on problem #1.

```

###PCA
```

```
# construct TF IDF weights
tfidf_train = weightTfIdf(DTM_train)

####
# Compare documents
####

inspect(tfidf_train[1,])
inspect(tfidf_train[2,])
inspect(tfidf_train[3,])

# could go back to the raw corpus
content(my_corpus_train[[1]])
content(my_corpus_train[[2]])
content(my_corpus_train[[3]])

# cosine similarity
i = 1
j = 3
sum(tfidf_train[i,] * (tfidf_train[j,]))/(sqrt(sum(tfidf_train[i,]^2)) * sqrt(sum(tfi
df_train[j,]^2)))


# the full set of cosine similarities
# two helper functions that use some linear algebra for the calculations
cosine_sim_docs = function(dtm) {
  crossprod_simple_triplet_matrix(t(dtm))/(sqrt(col_sums(t(dtm)^2) %*% t(col_sums(t(d
tm)^2))))
}

# use the function to compute pairwise cosine similarity for all documents
cosine_sim_mat = cosine_sim_docs(tfidf_train)
# Now consider a query document
content(my_corpus_train[[17]])
cosine_sim_mat[17,]

#
sort(cosine_sim_mat[18,], decreasing=TRUE)
content(my_corpus_train[[18]])
content(my_corpus_train[[19]])

#####
# Cluster documents
#####

# define the cosine distance
cosine_dist_mat = proxy::dist(as.matrix(tfidf_train), method='cosine')
tree_simon = hclust(cosine_dist_mat)
plot(tree_simon)
```

```
clust5 = cutree(tree_simon, k=5)

# inspect the clusters
which(clust5 == 1)
content(my_corpus_train[[1]])
content(my_corpus_train[[4]])
content(my_corpus_train[[5]])



####
# Dimensionality reduction
####

# Now PCA on term frequencies
X = as.matrix(tfidf_train)
summary(colSums(X))
scrub_cols = which(colSums(X) == 0)
X = X[,-scrub_cols]

pca_train = prcomp(X, scale=TRUE)
plot(pca_train)

# Look at the loadings
pca_train$rotation[order(abs(pca_train$rotation[,1]),decreasing=TRUE),1][1:25]
pca_train$rotation[order(abs(pca_train$rotation[,2]),decreasing=TRUE),2][1:25]


## Look at the first two PCs..
# We've now turned each document into a single pair of numbers -- massive dimensional
ity reduction
pca_train$x[,1:2]

plot(pca_train$x[,1:2], xlab="PCA 1 direction", ylab="PCA 2 direction", bty="n",
     type='n')
text(pca_train$x[,1:2], labels = 1:length(my_corpus_train), cex=0.7)



# Conclusion: even just these two-number summaries still preserve a lot of informatio
n


# Now look at the word view
# 5-dimensional word vectors
word_vectors = pca_train$rotation[,1:5]

word_vectors[982,]

d_mat = dist(word_vectors)
```

```
###################
# Now PCA on term frequencies
X_test2 = as.matrix(DTM_test)
X_test2 = X_test2/rowSums(X_test2)



pca_X_test2 = prcomp(X_test2, scale=TRUE)
plot(pca_X_test2)

# Look at the loadings
pca_X_test2$rotation[order(abs(pca_X_test2$rotation[,1]),decreasing=TRUE),1][1:25]
pca_X_test2$rotation[order(abs(pca_X_test2$rotation[,2]),decreasing=TRUE),2][1:25]


## Plot the first two PCs..
plot(pca_X_test2$x[,1:2], xlab="PCA 1 direction", ylab="PCA 2 direction", bty="n",
     type='n')
text(pca_X_test2$x[,1:2], labels = 1:length(all_docs_test), cex=0.7)
identify(pca_X_test2$x[,1:2], n=4)
```

Problem 3 ** Groceries **

Revisit the notes on association rule mining, and walk through the R example on music playlists: playlists.R and playlists.csv. Then use the data on grocery purchases in groceries.txt and find some interesting association rules for these shopping baskets. The data file is a list of baskets: one row per basket, with multiple items per row separated by commas – you'll have to cobble together a few utilities for processing this into the format expected by the "arules" package. Pick your own thresholds for lift and confidence; just be clear what these thresholds are and how you picked them. Do your discovered item sets make sense? Present your discoveries in an interesting and concise way.

```
rm(list=ls())

library(arulesViz)
```

```
## Loading required package: arules
```

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

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

```
## The following object is masked from 'package:dplyr':
##
##     recode
```

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

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

```
library(arules)
library(reshape2)
```

```
##
## Attaching package: 'reshape2'
```

```
## The following objects are masked from 'package:data.table':
##
##     dcast, melt
```

```
library(plyr)
#setwd("/Users/claytonmason/GitHub/STA_380_Clay/Data")
```

Association Rule Mining - find rules that will predict the occurrence of an item based on the occurrences of other items in the transaction goal of association rule mining is to find all rules having – support greater than a minsup threshold (s Fraction of transactions that contain both X and Y) – confidence greather than a minconf threshold (c = Measures how often items in Y appear in transactions that contain X)

This analysis is useful for understanding consumer behavior. If we understand what products consumers frequently buy together, then we could suggest appealing marketing proposals.

Below, I will present findings with various support and confidence levels to see if there are any interesting trends.

```

```

There are 9,835 rows spanning across 169 grocery store items.

```
groceries = read.transactions("/Users/claytonmason/GitHub/STA_380_Clay/Data/groceries
.txt", format="basket", sep=",")
dim(groceries)
```

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

```
#9,835 rows and 169 variables


# #Cast this variable as a special arules transactions class
groceries_transaction <- as(groceries, "transactions")
```

I first looked at the apriori algo settings from the music example. Support >= .01 Confidence >= .5 max length = 4

After sorting by support, you can see that whole milk is the most commonly associated item for 7 of the top 10 items. Milk spoils quickly and is frequently purchased by consumers. Grocery stores strategically place milk in the far back of the store to promote cross selling of products on the way to purchase this product.

The 2 highest confidence items contatin other veggies in the rhs column paired with citrus fruit/root vegetables and tropical fruit/root vegetables. The confidence level was about .58 which means in 58 % of the transactions that include these items, then other vegetables are purchased as well.

I don't think this setting is particuarlly useful because of the reason i mentioned about the frequency of purchase with regards to milk. This top confidence table just shows that milk is purchased frequently regardless of the other items in the sets.

Lift is capped out around 3 and it is for the same veggie sets i previously mentioned. This is not to interesting to me and just implies that if people shop in this area of the grocery store, they are likely to buy from other closeby sections.

```
# Now run the 'apriori' algorithm
# Look at rules with support > .01 & confidence >.5 & length(# of items) <= 4
groceries_rules1 <- apriori(groceries_transaction, parameter=list(support=.01, confid
ence=.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_transaction, parameter = list(support =
## 0.01, : 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].
```

```
#Top 10 Support
top.support <- sort(groceries_rules1, decreasing = TRUE, na.last = NA, by = "support"
)
inspect(sort(top.support)[1:10])
```

```
##      lhs                        rhs                    support confidence     lift cou
nt
## [1]  {other vegetables,
##       yogurt}              => {whole milk}       0.02226741  0.5128806 2.007235   2
19
## [2]  {tropical fruit,
##       yogurt}              => {whole milk}       0.01514997  0.5173611 2.024770   1
49
## [3]  {other vegetables,
##       whipped/sour cream} => {whole milk}       0.01464159  0.5070423 1.984385   1
44
## [4]  {root vegetables,
##       yogurt}              => {whole milk}       0.01453991  0.5629921 2.203354   1
43
## [5]  {other vegetables,
##       pip fruit}           => {whole milk}       0.01352313  0.5175097 2.025351   1
33
## [6]  {root vegetables,
##       yogurt}              => {other vegetables} 0.01291307  0.5000000 2.584078   1
27
## [7]  {rolls/buns,
##       root vegetables}     => {whole milk}       0.01270971  0.5230126 2.046888   1
25
## [8]  {domestic eggs,
##       other vegetables}    => {whole milk}       0.01230300  0.5525114 2.162336   1
21
## [9]  {root vegetables,
##       tropical fruit}      => {other vegetables} 0.01230300  0.5845411 3.020999   1
21
## [10] {rolls/buns,
##       root vegetables}     => {other vegetables} 0.01220132  0.5020921 2.594890   1
20
```

```
#Top 10 Confidence
top.confidence <- sort(groceries_rules1, decreasing = TRUE, na.last = NA, by = "confi
dence")
inspect(head(top.confidence, 10))
```

```
##      lhs                        rhs                    support confidence     lift cou
nt
## [1]  {citrus fruit,
##       root vegetables}    => {other vegetables} 0.01037112  0.5862069 3.029608   1
02
## [2]  {root vegetables,
##       tropical fruit}     => {other vegetables} 0.01230300  0.5845411 3.020999   1
21
## [3]  {curd,
##       yogurt}             => {whole milk}       0.01006609  0.5823529 2.279125    
99
## [4]  {butter,
##       other vegetables}   => {whole milk}       0.01148958  0.5736041 2.244885   1
13
## [5]  {root vegetables,
##       tropical fruit}     => {whole milk}       0.01199797  0.5700483 2.230969   1
18
## [6]  {root vegetables,
##       yogurt}             => {whole milk}       0.01453991  0.5629921 2.203354   1
43
## [7]  {domestic eggs,
##       other vegetables}   => {whole milk}       0.01230300  0.5525114 2.162336   1
21
## [8]  {whipped/sour cream,
##       yogurt}             => {whole milk}       0.01087951  0.5245098 2.052747   1
07
## [9]  {rolls/buns,
##       root vegetables}    => {whole milk}       0.01270971  0.5230126 2.046888   1
25
## [10] {other vegetables,
##       pip fruit}          => {whole milk}       0.01352313  0.5175097 2.025351   1
33
```

more frequent purchase observation

I next looked at more frequent purchases but lower confidence Support >= .02 Confidence >= .2 max length = 3

I messed with max length a few different ways, but didn't find any meaninfgul findings, so I kept it low. I increased the frequency requirement (support), but lowered the confidence interval.

This finding revealed that milk is often purchased by itself, which led to a lift of 1. Additionally, the support and confidence level were the same at .255.

If butter is bought, then milk is bought as well half of the time.

This modification from the first set of parameters didn't yield too much information. The eggs and milk combination also yielded a high confidence level, but this isn't surprising as it is common knowledge these items are replenished together.

```
# Look at rules with support > .02 & confidence >.2 & length(# of items) <= 3
groceries_rules2 <- apriori(groceries_transaction, parameter=list(support=.02, confid
ence=.2, maxlen=4))
```

```
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##         0.2    0.1    1 none FALSE            TRUE       5    0.02      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: 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 ... [73 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
```

```
#Top 10 Support
top.support <- sort(groceries_rules2, decreasing = TRUE, na.last = NA, by = "support"
)
inspect(sort(top.support)[1:10])
```

```
##      lhs                   rhs                  support    confidence
## [1]  {}                 => {whole milk}       0.25551601 0.2555160
## [2]  {other vegetables} => {whole milk}       0.07483477 0.3867578
## [3]  {whole milk}       => {other vegetables} 0.07483477 0.2928770
## [4]  {rolls/buns}       => {whole milk}       0.05663447 0.3079049
## [5]  {whole milk}       => {rolls/buns}       0.05663447 0.2216474
## [6]  {yogurt}           => {whole milk}       0.05602440 0.4016035
## [7]  {whole milk}       => {yogurt}           0.05602440 0.2192598
## [8]  {root vegetables}  => {whole milk}       0.04890696 0.4486940
## [9]  {root vegetables}  => {other vegetables} 0.04738180 0.4347015
## [10] {other vegetables} => {root vegetables}  0.04738180 0.2448765
##      lift     count
## [1]  1.000000 2513
## [2]  1.513634  736
## [3]  1.513634  736
## [4]  1.205032  557
## [5]  1.205032  557
## [6]  1.571735  551
## [7]  1.571735  551
## [8]  1.756031  481
## [9]  2.246605  466
## [10] 2.246605  466
```

```
#Top 10 Confidence
top.confidence <- sort(groceries_rules2, decreasing = TRUE, na.last = NA, by = "confi
dence")
inspect(head(top.confidence, 10))
```

```
##      lhs                                   rhs                  support
## [1]  {other vegetables,yogurt}          => {whole milk}         0.02226741
## [2]  {butter}                           => {whole milk}         0.02755465
## [3]  {curd}                             => {whole milk}         0.02613116
## [4]  {other vegetables,root vegetables} => {whole milk}         0.02318251
## [5]  {root vegetables,whole milk}       => {other vegetables} 0.02318251
## [6]  {domestic eggs}                    => {whole milk}         0.02999492
## [7]  {whipped/sour cream}               => {whole milk}         0.03223183
## [8]  {root vegetables}                  => {whole milk}         0.04890696
## [9]  {root vegetables}                  => {other vegetables} 0.04738180
## [10] {frozen vegetables}                => {whole milk}         0.02043721
##      confidence lift     count
## [1]  0.5128806  2.007235 219
## [2]  0.4972477  1.946053 271
## [3]  0.4904580  1.919481 257
## [4]  0.4892704  1.914833 228
## [5]  0.4740125  2.449770 228
## [6]  0.4727564  1.850203 295
## [7]  0.4496454  1.759754 317
## [8]  0.4486940  1.756031 481
## [9]  0.4347015  2.246605 466
## [10] 0.4249471  1.663094 201
```

infrequent purchase, but high confidence

I next looked at very infrequent purchases, but high confidence. Support >= .02 Confidence >= .2 max length = 3

This setting yielded unusable results. After sorting for high confidence, i found single transactions with 100% confidence of sound storage medium being purchased along with a single item on 9 different occasions.

This type of information wouldn't be useful unless we looked at the full universe of this storage medium. We would have to know that this item was always purchased with one other item to find anything useful.

After sorting by the low support threshold, I noted a similar observation as earlier. Butter/yogurt, and whole milk for a high confidence data set. These items are sold in similar areas of the grocery store, but it could also mean that the Milk marketing strategy that promotes cross-selling is working. These items are relatively inexpensive and small, which could mean they are items that are getting added to a "quick milk run".

```
# Look at rules with support > .0001 & confidence >.6 & length(# of items) <= 4
groceries_rules3 <- apriori(groceries_transaction, parameter=list(support=.0001, conf
idence=.6, maxlen=4))
```

```
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##         0.6    0.1    1 none FALSE            TRUE       5   1e-04      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: 0
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
## sorting and recoding items ... [169 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 3 4
```

```
## Warning in apriori(groceries_transaction, parameter = list(support =
## 1e-04, : Mining stopped (maxlen reached). Only patterns up to a length of 4
## returned!
```

```
##  done [0.08s].
## writing ... [1144432 rule(s)] done [0.14s].
## creating S4 object  ... done [0.24s].
```

```
#Top 10 Support
top.support <- sort(groceries_rules3, decreasing = TRUE, na.last = NA, by = "support"
)
inspect(sort(top.support)[1:10])
```

```
##      lhs                        rhs              support confidence     lift count
## [1]  {butter,
##       yogurt}               => {whole milk} 0.009354347  0.6388889 2.500387    92
## [2]  {butter,
##       root vegetables}      => {whole milk} 0.008235892  0.6377953 2.496107    81
## [3]  {other vegetables,
##       root vegetables,
##       yogurt}               => {whole milk} 0.007829181  0.6062992 2.372842    77
## [4]  {other vegetables,
##       tropical fruit,
##       yogurt}               => {whole milk} 0.007625826  0.6198347 2.425816    75
## [5]  {domestic eggs,
##       tropical fruit}       => {whole milk} 0.006914082  0.6071429 2.376144    68
## [6]  {butter,
##       whipped/sour cream} => {whole milk} 0.006710727  0.6600000 2.583008    66
## [7]  {curd,
##       tropical fruit}       => {whole milk} 0.006507372  0.6336634 2.479936    64
## [8]  {butter,
##       tropical fruit}       => {whole milk} 0.006202339  0.6224490 2.436047    61
## [9]  {butter,
##       domestic eggs}        => {whole milk} 0.005998983  0.6210526 2.430582    59
## [10] {pip fruit,
##       whipped/sour cream} => {whole milk} 0.005998983  0.6483516 2.537421    59
```

```
#Top 10 Confidence
top.confidence <- sort(groceries_rules3, decreasing = TRUE, na.last = NA, by = "confi
dence")
inspect(head(top.confidence, 10))
```

```
##      lhs                        rhs                       support
## [1]  {sound storage medium} => {frozen potato products} 0.0001016777
## [2]  {sound storage medium} => {cat food}               0.0001016777
## [3]  {sound storage medium} => {candy}                  0.0001016777
## [4]  {sound storage medium} => {ham}                    0.0001016777
## [5]  {sound storage medium} => {white bread}            0.0001016777
## [6]  {sound storage medium} => {pastry}                 0.0001016777
## [7]  {sound storage medium} => {shopping bags}          0.0001016777
## [8]  {sound storage medium} => {bottled water}          0.0001016777
## [9]  {sound storage medium} => {soda}                   0.0001016777
## [10] {baby food}            => {finished products}      0.0001016777
##      confidence lift       count
## [1]  1          118.493976 1
## [2]  1           42.947598 1
## [3]  1           33.452381 1
## [4]  1           38.417969 1
## [5]  1           23.756039 1
## [6]  1           11.240000 1
## [7]  1           10.149639 1
## [8]  1            9.047838 1
## [9]  1            5.734694 1
## [10] 1          153.671875 1
```