Exercise 2

Danchen Zhao, Shan Qin, Candice Zuo

Flights at ABIA

```
library(ggplot2)
library(readr)
ABIA = read.csv("~/data/ABIA.csv")
```

The question we are interested in is to figure out the best day and the quickest airline for the top-10 most popular destination. There are two parts of our analysis. The first is the percentage of chance of one specific airline to arrive early on each weekday. And the second is the percentage of chance of one unique airline to arrive early on each weekday.

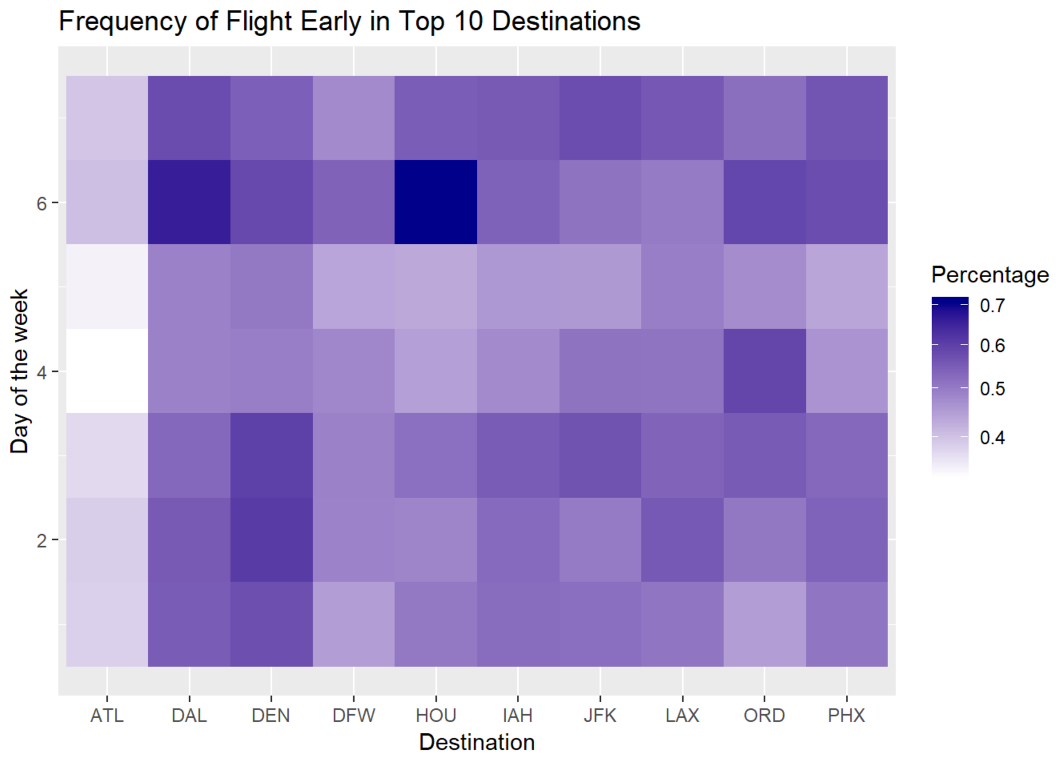
First we need to know which ten destination are the most popular.

```
depart = ABIA[which(ABIA$Origin=='AUS'),]
depart_dest_freq = data.frame(aggregate(depart$FlightNum~ depart$Dest,
                                         depart, length))
depart_dest_freq = depart_dest_freq[order(depart_dest_freq$depart.FlightNum,decreasing = FALSE),]
depart_dest_freq_most = tail(depart_dest_freq, 10)
depart_dest_freq_most
```

```
##    depart.Dest depart.FlightNum
## 23         JFK             1358
## 25         LAX             1733
## 2          ATL             2252
## 17         HOU             2319
## 38         ORD             2514
## 10         DEN             2673
## 41         PHX             2783
## 20         IAH             3691
## 11         DFW             5506
## 9          DAL             5573
```

Then we select out all the flights that have these destinations and have negative number on ArrDelay column. We pick three variables that is useful to us: time of arrive early, day of week and the destination airport. We calculate the percentage of flight that arrives early for each airline in each weekday.

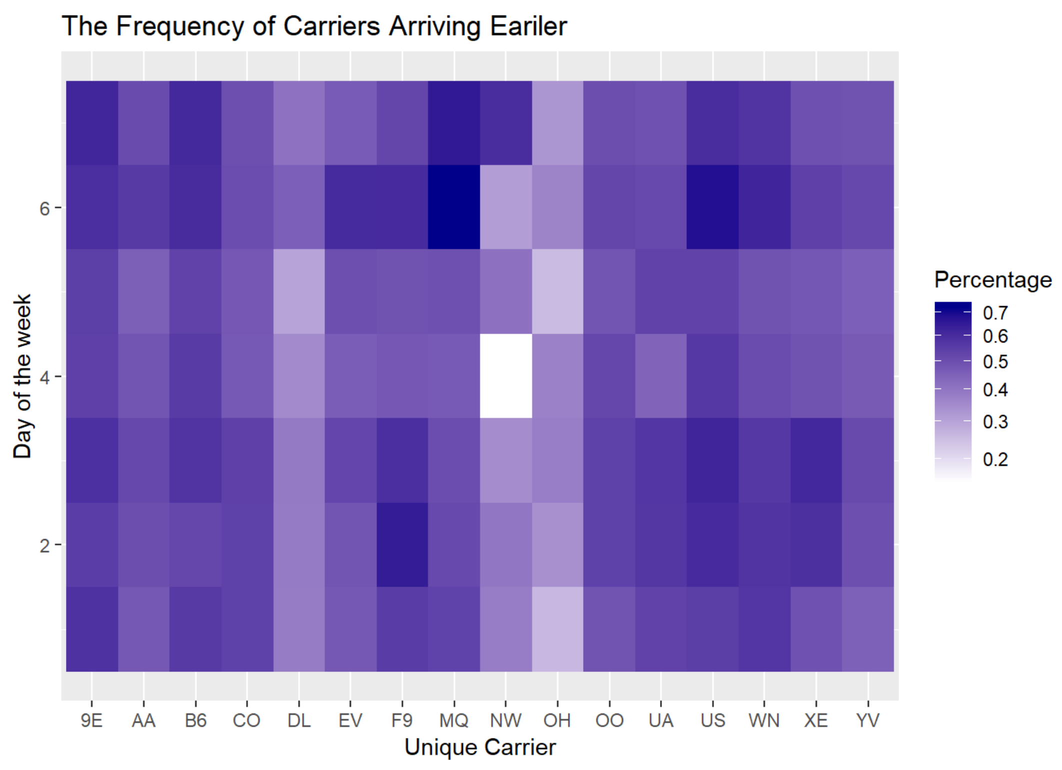
```
dest_most = ABIA[which(ABIA$Dest=='JFK'|ABIA$Dest=='LAX'|ABIA$Dest=='ATL'|ABIA$Dest=='HOU' |ABIA$Dest=='ORD'
|ABIA$Dest=='DEN'|ABIA$Dest=='PHX'|ABIA$Dest=='IAH'|ABIA$Dest=='DFW'|ABIA$Dest=='DAL'),]

dest_most_arrivetime_day <- na.omit(dest_most[,c("ArrDelay","DayOfWeek", "Dest")])
dest_most_arriveearly_day = dest_most_arrivetime_day[which(dest_most_arrivetime_day$ArrDelay<0),]

dest_most_arriveearly_day_freq = data.frame(aggregate(dest_most_arriveearly_day$ArrDelay~ dest_most_arriveea
rly_day$Dest+dest_most_arriveearly_day$DayOfWeek,
                                                      dest_most_arriveearly_day, length))

dest_most_freq = data.frame(aggregate(dest_most$FlightNum~ dest_most$Dest+dest_most$DayOfWeek,
                                      dest_most, length))

dest_most_early_percent = merge(dest_most_arriveearly_day_freq, dest_most_freq, by.x=c("dest_most_arriveearl
y_day.Dest","dest_most_arriveearly_day.DayOfWeek"), by.y=c("dest_most.Dest","dest_most.DayOfWeek"))
dest_most_early_percent = within(dest_most_early_percent, percent <- dest_most_arriveearly_day.ArrDelay/dest
_most.FlightNum)

dest_most_arriveearly_day_freq_plt <- ggplot(dest_most_early_percent, aes(dest_most_arriveearly_day.Dest, de
st_most_arriveearly_day.DayOfWeek, fill = percent))  + geom_tile() + ylab("Day of the week") + xlab("Destina
tion") + ggtitle("Frequency of Flight Early in Top 10 Destinations ") + scale_fill_gradient( trans="sqrt", l
ow = "white", high="dark blue",name = 'Percentage')
dest_most_arriveearly_day_freq_plt
```



As a result, we find that the flights going to Houston airport on Saturday have the highest early arrival rate: more than 70%. For ATL is Saturday; for DAL is Saturday; for DEN is Tuesday and Wednesday; for DFW is Saturday; for IAH is Wednesday; for JFK is Wednesday and Sunday; for LAX is Tuesday; for ORD is Thursday and Saturday; for PHX is Saturday and Sunday.

Next step is to early arrival rate of each airline.

```
arrive_early= ABIA[which(ABIA[,15] < 0),]

carrier_freq_early = data.frame(aggregate(arrive_early$ArrDelay~ arrive_early$UniqueCarrier+arrive_early$Day
OfWeek,
                                          arrive_early, length))
carrier_freq = data.frame(aggregate(ABIA$FlightNum~ ABIA$UniqueCarrier+ABIA$DayOfWeek,
                          ABIA, length))

carrier_early_percent = merge(carrier_freq_early, carrier_freq, by.x=c("arrive_early.UniqueCarrier","arrive_
early.DayOfWeek"), by.y=c("ABIA.UniqueCarrier","ABIA.DayOfWeek"))
carrier_early_percent = within(carrier_early_percent, percent <- arrive_early.ArrDelay/ABIA.FlightNum)

carrier_early_percent_plt <- ggplot(carrier_early_percent, aes(arrive_early.UniqueCarrier, arrive_early.DayO
fWeek, fill = percent))  + geom_tile() + ylab("Day of the week") + xlab("Unique Carrier") + ggtitle("The Fre
quency of Carriers Arriving Eariler ") + scale_fill_gradient( trans="sqrt", low = "white", high="dark blue",
name = 'Percentage')
carrier_early_percent_plt
```



From the graph, we can see that 9E does the best job on Monday; F9 does the best job on Tuesday; US and XE do the best job on Wednesday; US does a little better job than others on Thursday; 9E and B6 do a good job on Friday; MQ does a super good job on Saturday; 9E and MQ both do a good job on Sunday.

Conclusion

Combing the two pictures, we get the answer to our question.

Destination	**Airline**	**Arrival Airport**	**Day of the Week**
Atlanta	MQ	ATL	Saturday
Dallas	MQ	DAL	Saturday
Denver	F9	DEN	Tuesday
Houston	MQ	HOU	Saturday
New York	MK	JFK	Sunday
Los Angeles	F9	LAX	Tuesday
Chicago	US or MQ	ORD	Saturday
Chicago	US	ORD	Thursday
Phoenix	MQ	PHX	Saturday or Sunday

For example, if one goes to Chicago, he should take US or MQ on Saturday or US on Thursday.

Author attribution

Data Preparation

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

First, we got the two DTMs for the train set and the test set and built matrices for them.

```r
author_dirs = Sys.glob('~/data/ReutersC50/C50train/*')

file_list = c()
labels = c()
for(author in author_dirs) {
    author_name = substring(author, first=49)
    files_to_add = Sys.glob(paste0(author, '/*.txt'))
    file_list = append(file_list, files_to_add)
}
all_docs = lapply(file_list, readerPlain)
train_corpus = Corpus(VectorSource(all_docs))



train_corpus = tm_map(train_corpus, content_transformer(tolower)) # make everything lowercase
train_corpus = tm_map(train_corpus, content_transformer(removeNumbers)) # remove numbers
train_corpus = tm_map(train_corpus, content_transformer(removePunctuation)) # remove punctuation
train_corpus = tm_map(train_corpus, content_transformer(stripWhitespace)) ## remove excess white-space
train_corpus = tm_map(train_corpus, content_transformer(removeWords), stopwords("SMART"))

DTM_train = DocumentTermMatrix(train_corpus)
DTM_train = removeSparseTerms(DTM_train, 0.95)
```

```r
X_train = as.matrix(DTM_train)
```

```r
author_dirs = Sys.glob('~/data/ReutersC50/C50test/*')


file_list = c()
labels = c()
for(author in author_dirs) {
    author_name = substring(author, first=48)
    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
names(all_docs) = sub('.txt', '', names(all_docs))



test_corpus = Corpus(VectorSource(all_docs))



test_corpus = tm_map(test_corpus, content_transformer(tolower)) # make everything lowercase
test_corpus = tm_map(test_corpus, content_transformer(removeNumbers)) # remove numbers
test_corpus = tm_map(test_corpus, content_transformer(removePunctuation)) # remove punctuation
test_corpus = tm_map(test_corpus, content_transformer(stripWhitespace)) ## remove excess white-space
test_corpus = tm_map(test_corpus, content_transformer(removeWords), stopwords("SMART"))
```

```r
DTM_test_temp = DocumentTermMatrix(test_corpus)
DTM_test_temp = removeSparseTerms(DTM_test_temp, 0.95)


X_test = as.matrix(DTM_test_temp)
```

Then, for modeling purpose, we reconstructed the matrix for the test set so that its structure is same as that of the train set. Meanwhile, we added a pseudo word to both sets to represent the the words appear in the test set but not the train set.

```
train_words = colnames(X_train)

sum1 = rowSums(X_test)


test_words = colnames(X_test)

# get the words to be droped or added
add = vector(length=0)
drop = vector(length=0)

for (test_word in test_words) {
  if (!test_word %in% train_words) {
    drop <- c(drop, test_word)
  }
}

for (word in train_words) {
  if (!word %in% test_words) {
    add <- c(add, word)
  }
}


# build a dataframe for words that appear in train set but not test set
zeros <- matrix(0, nrow = nrow(X_train), ncol=length(add))

colnames(zeros) <- add

X_test_2 = cbind(X_test, zeros)

X_test_2 = X_test_2[,order(colnames(X_test_2))]


X_test_2 = X_test_2[,!colnames(X_test_2) %in% drop]
X_train = X_train[,order(colnames(X_train))]


# create the pseudo word and pseudo count
pseudo = rep(0, nrow(X_train))
X_train = cbind(X_train, pseudo)
X_train = X_train + 1/nrow(X_train)



sum2 = rowSums(X_test_2)
pseudo = sum1 - sum2

X_test_2 = cbind(X_test_2, pseudo)
```

The columns for two sets are the same now.

```
dim(X_train)
```

```
## [1] 2500  661
```

```
X_train[1:10,650:661]
```

```
##      work workers working  world worlds worldwide worth   yday   year
## 1  0.0004   4e-04  1.0004 1.0004  4e-04     4e-04 4e-04 1.0004 1.0004
## 2  0.0004   4e-04  0.0004 1.0004  4e-04     4e-04 4e-04 1.0004 1.0004
## 3  0.0004   4e-04  0.0004 0.0004  4e-04     4e-04 4e-04 1.0004 1.0004
## 4  1.0004   4e-04  0.0004 2.0004  4e-04     4e-04 4e-04 1.0004 2.0004
## 5  0.0004   4e-04  0.0004 2.0004  4e-04     4e-04 4e-04 1.0004 1.0004
## 6  0.0004   4e-04  1.0004 0.0004  4e-04     4e-04 4e-04 1.0004 1.0004
## 7  0.0004   4e-04  0.0004 0.0004  4e-04     4e-04 4e-04 1.0004 1.0004
## 8  0.0004   4e-04  0.0004 0.0004  4e-04     4e-04 4e-04 1.0004 3.0004
## 9  0.0004   4e-04  0.0004 0.0004  4e-04     4e-04 4e-04 1.0004 3.0004
## 10 0.0004   4e-04  0.0004 0.0004  4e-04     4e-04 4e-04 1.0004 4.0004
##     years  york pseudo
## 1  0.0004 4e-04  4e-04
## 2  0.0004 4e-04  4e-04
## 3  0.0004 4e-04  4e-04
## 4  0.0004 4e-04  4e-04
## 5  0.0004 4e-04  4e-04
## 6  0.0004 4e-04  4e-04
## 7  0.0004 4e-04  4e-04
## 8  0.0004 4e-04  4e-04
## 9  1.0004 4e-04  4e-04
## 10 0.0004 4e-04  4e-04
```

```
X_test_2[1:10,650:661]
```

```
##    work workers working world worlds worldwide worth yday year years york
## 1     0       0       1     0      0         0     0    1    1     0    0
## 2     0       0       0     0      0         0     0    1    1     1    0
## 3     1       0       0     5      0         0     0    1    1     1    1
## 4     0       0       0     0      0         0     0    1    1     0    0
## 5     1       0       0     1      0         0     0    1    1     1    0
## 6     0       0       0     0      0         0     0    1    1     0    0
## 7     0       0       0     0      0         0     0    1    7     2    1
## 8     1       0       1     0      0         0     0    1    3     2    0
## 9     0       0       0     0      0         0     0    1    3     2    0
## 10    0       0       1     0      0         1     0    1   11     0    0
##    pseudo
## 1       5
## 2       7
## 3      13
## 4       1
## 5      17
## 6       5
## 7       2
## 8       8
## 9       4
## 10      4
```

```
y_train = labels
y_test = labels
train = cbind(X_train, y_train)
```

Model 1: Naive Bayes

The first model we built is naive Bayes based on the word counts. We got a matrix of probabilities. Each row of this matrix is an author; each column for this matrix is the weights for a word for authors.

```
# get the matrix of probabilities
matrix_of_probabilities = aggregate(.~y_train, data=as.data.frame(X_train), FUN=sum)
row.names(matrix_of_probabilities) = matrix_of_probabilities$y_train
matrix_of_probabilities$y_train <- NULL
sums = rowSums(matrix_of_probabilities)
matrix_of_probabilities = matrix_of_probabilities / sums
```

Then, we can get the predictions by matrix multiplication and get the author with highest probability for each row.

```
matrix_of_probabilities_t = t(matrix_of_probabilities)
predictions = as.matrix(X_test_2) %*% log(matrix_of_probabilities_t)
```

```r
print("The out-of-sample error rate for the naive bayes model is")
```

```
## [1] "The out-of-sample error rate for the naive bayes model is"
```

```r
1 - sum(predictions == y_test)/length(y_test)
```

```
## [1] 0.442
```

The out-of-sample error rate for the naive Bayes model is 0.442.

Model 2: Random Forest

The second model we used is random forest. We first built the matrix for train and test set separately, then we kept the words that appeared in both data sets. Lastly, we transformed these two data sets into TFIDF matrix and used random forest to make predictions.

```r
X_train = as.data.frame(X_train)
X_test = as.data.frame(X_test)

# find the shared columns
share = intersect(colnames(X_train), colnames(X_test))
X_train_common = X_train[,share]
X_test_common = X_test[,share]
y_train = as.factor(y_train)
y_test = as.factor(y_test)
```

```r
N = nrow(X_train_common)
D = ncol(X_train_common)
# TF weights
TF_mat = X_train_common/rowSums(X_train_common)

# IDF weights
IDF_vec = log(1 + N/colSums(X_train_common > 0))


# TF-IDF weights:
# use sweep to multiply the columns (margin = 2) by the IDF weights
TFIDF_mat = sweep(TF_mat, MARGIN=2, STATS=IDF_vec, FUN="*")




TF_mat_test = X_test_common/rowSums(X_test_common)

# IDF weights
IDF_vec_test = log(1 + N/colSums(X_test_common > 0))


# TF-IDF weights:
# use sweep to multiply the columns (margin = 2) by the IDF weights
TFIDF_mat_test = sweep(TF_mat_test, MARGIN=2, STATS=IDF_vec_test, FUN="*")
```

```r
library(randomForest)
set.seed(20)
rf.model = randomForest(y_train~., data = TFIDF_mat, distribution = 'multinominal', ntree = 500)

pred = predict(rf.model, newdat = TFIDF_mat_test)
rf.pred = data.frame(pred)

print ("The out-of-sample error rate for the randomforest model is")
```

```
## [1] "The out-of-sample error rate for the randomforest model is"
```

```r
1 - sum(pred == as.factor(y_test))/length(y_test)
```

```
## [1] 0.5676
```

The out-of-sample error rate for this model is about 57%.

Conclusion

We chose Naive Bayes as our final model because it is easier to interpret and has lower error rate compare to the random forest model.

There are several combinations of authors that are difficult to distinguish.

The model classified Alexander Smith as Joe Ortiz for 18 times, David Lauder as Todd Nissen for 19 times, Jan Lopatka as John Mastrini for 22 times, Scott Hillis as Jane Macartney for 18 times.

It is interesting to see that these errors are not symmetrical. For example, the model only classifies Joe Ortiz as Alexander Smith for 8 times.

```
for (row in 1:nrow(confusion_matrix)){
  for (col in 1:ncol(confusion_matrix)){
    number = confusion_matrix[row, col]
    if (number > 15 & rownames(confusion_matrix)[row] != colnames(confusion_matrix)[col]){
      print("The real author is ")
      print(rownames(confusion_matrix)[row])
      print("The predicted author is")
      print(colnames(confusion_matrix)[col])
      print("The number of times it happens")
      print(confusion_matrix[row, col])
      cat("\n")
    }
  }
}
```

```
## [1] "The real author is "
## [1] "AlexanderSmith"
## [1] "The predicted author is"
## [1] "JoeOrtiz"
## [1] "The number of times it happens"
## [1] 18
##
## [1] "The real author is "
## [1] "DavidLawder"
## [1] "The predicted author is"
## [1] "ToddNissen"
## [1] "The number of times it happens"
## [1] 19
##
## [1] "The real author is "
## [1] "JanLopatka"
## [1] "The predicted author is"
## [1] "JohnMastrini"
## [1] "The number of times it happens"
## [1] 22
##
## [1] "The real author is "
## [1] "ScottHillis"
## [1] "The predicted author is"
## [1] "JaneMacartney"
## [1] "The number of times it happens"
## [1] 21
```

```
confusion_matrix["JoeOrtiz","AlexanderSmith"]
```

```
## [1] 8
```

Practice with Association Rule Mining

In this exercises, we identified association rules between grocery items using the apriori algorithm.

There are 9835 transactions and 169 unique grocery items in the dataset. Majority of the transactions has a length >=5.

```
dim(groceries)
```

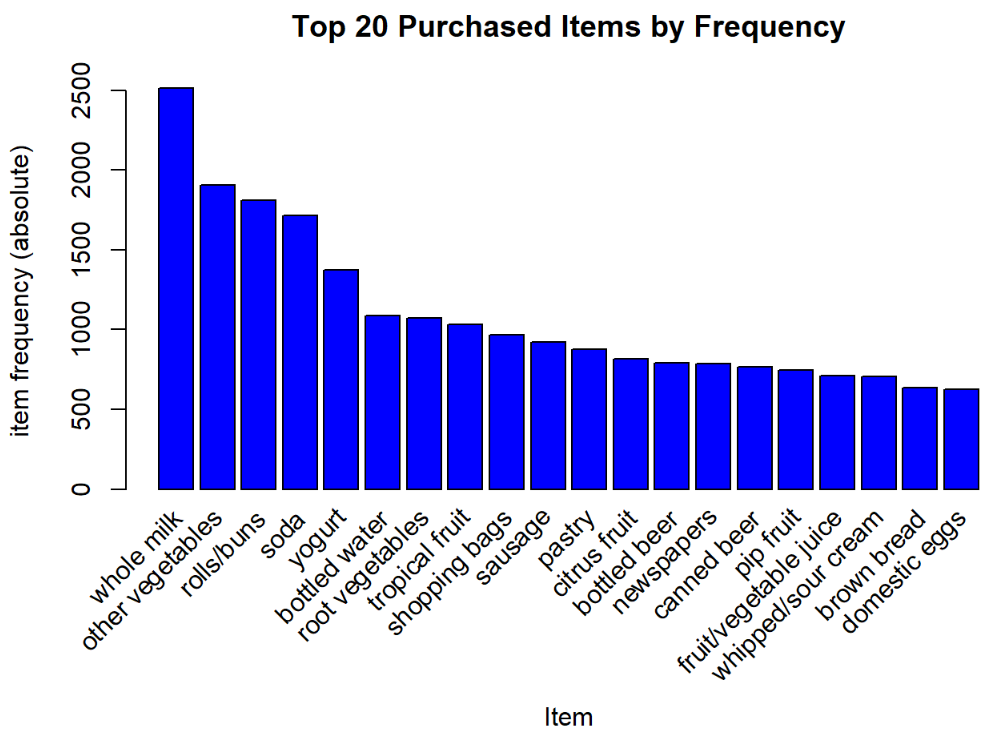
```
## [1] 9835  169
```

```
summary(groceries)
```

```
## transactions as itemMatrix in sparse format with
##  9835 rows (elements/itemsets/transactions) and
##  169 columns (items) and a density of 0.02609146
##
## most frequent items:
##       whole milk other vegetables       rolls/buns             soda
##             2513             1903             1809             1715
##           yogurt          (Other)
##             1372            34055
##
## element (itemset/transaction) length distribution:
## sizes
##    1    2    3    4    5    6    7    8    9   10   11   12   13   14   15
## 2159 1643 1299 1005  855  645  545  438  350  246  182  117   78   77   55
##   16   17   18   19   20   21   22   23   24   26   27   28   29   32
##   46   29   14   14    9   11    4    6    1    1    1    1    3    1
##
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   1.000   2.000   3.000   4.409   6.000  32.000
##
## includes extended item information - examples:
##             labels
## 1 abrasive cleaner
## 2 artif. sweetener
## 3   baby cosmetics
```

To get a basic idea of how frequently items are purchased, we ploted the top 20 most frequently appeared items in the dataset. The results are showed in the graph.

```
itemFrequencyPlot(groceries, topN=20, type = "absolute", col = 'blue', xlab = 'Item', main = 'Top 20 Purchas
ed Items by Frequency')
```



We ran the apriori algorithm to look at rules with support >.005 (0.5% of all transactions contain both X and Y), confidence = .1 (10% of the transactions that contain X also contains Y), and maxlen(# of grocery items) <= 5. There are 1582 rules under this setting.

```
rules = apriori(groceries,
    parameter=list(support=.005, confidence=.1, maxlen=5))
```

```
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##         0.1    0.1    1 none FALSE            TRUE       5   0.005      1
##  maxlen target   ext
##       5  rules FALSE
##
## Algorithmic control:
##  filter tree heap memopt load sort verbose
##     0.1 TRUE TRUE  FALSE TRUE    2    TRUE
##
## Absolute minimum support count: 49
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
## sorting and recoding items ... [120 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 3 4 done [0.00s].
## writing ... [1582 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
```

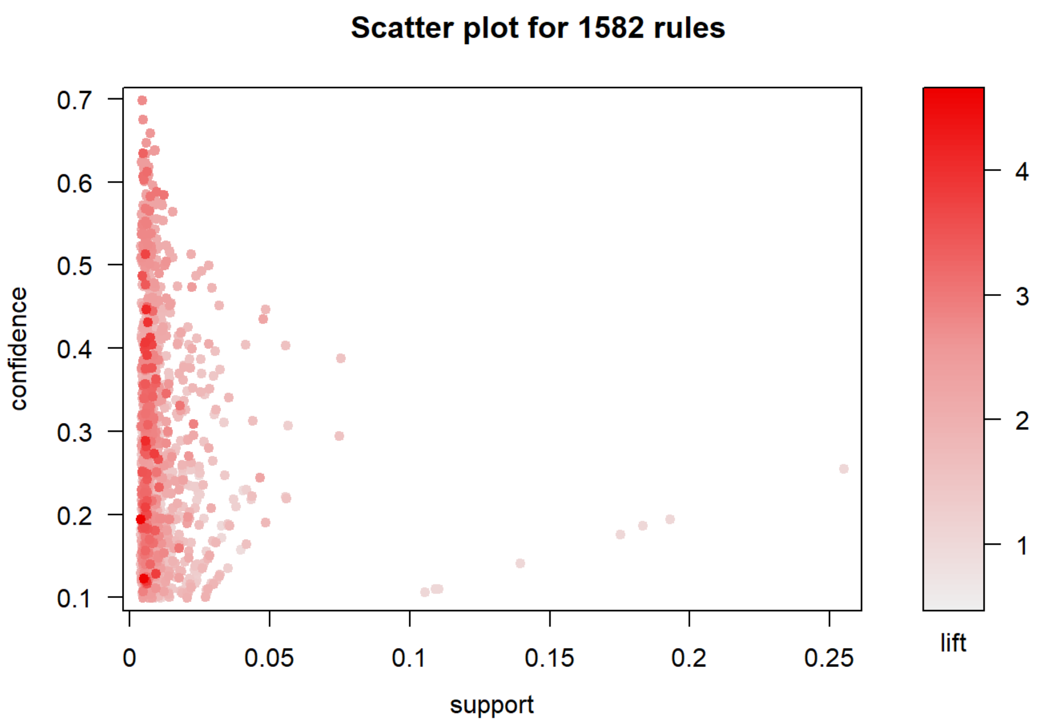
We plotted all the 1582 rules in (support, confidence) space.

Rules that have lift higher than 3 are in the confidence range between .1 and .5. Those rules also tend to have a low support (below 0.025).

We selected subset based on this plot to identify strong association rules.

```
plot(rules)
```

```
## To reduce overplotting, jitter is added! Use jitter = 0 to prevent jitter.
```



We can find the 85 rules with lift > 3. Customers are 3 or more times more likely to purchase left hand side items given that they purchased the right hand side items.

```
subset1 = subset(rules, subset=lift > 3)
```

Then we selected the 113 rules with confidence > 0.5. More than 50% of the transactions that contains the right hand side items also contains the left hand side items.

We found the right hand side item is either whole milk or other vegetables, while the left hand side has a lot of combinations of items. This suggest that whole milk and other vegetables are associated with many grocery items.

```
subset2 = subset(rules, subset=confidence > 0.5)
```

Next, we are interested to find out the strong association rules with lift >3 and confidence higher than 0.5.

The right hand side is dominated by other vegetables, suggesting that other vegetables has strong association with many grocery items such as onions, root vegetables, citrus fruit, and whole milk.

```
inspect(subset(rules, subset=lift > 3 & confidence > 0.5))
```

```
##     lhs                        rhs                    support confidence     lift count
## [1] {onions,                                                                           
##      root vegetables}    => {other vegetables} 0.005693950  0.6021505 3.112008    56
## [2] {curd,                                                                             
##      tropical fruit}     => {yogurt}           0.005287239  0.5148515 3.690645    52
## [3] {pip fruit,                                                                        
##      whipped/sour cream} => {other vegetables} 0.005592272  0.6043956 3.123610    55
## [4] {citrus fruit,                                                                     
##      root vegetables}    => {other vegetables} 0.010371124  0.5862069 3.029608   102
## [5] {root vegetables,                                                                  
##      tropical fruit}     => {other vegetables} 0.012302999  0.5845411 3.020999   121
## [6] {pip fruit,                                                                        
##      root vegetables,                                                                  
##      whole milk}         => {other vegetables} 0.005490595  0.6136364 3.171368    54
## [7] {citrus fruit,                                                                     
##      root vegetables,                                                                  
##      whole milk}         => {other vegetables} 0.005795628  0.6333333 3.273165    57
## [8] {root vegetables,                                                                  
##      tropical fruit,                                                                   
##      whole milk}         => {other vegetables} 0.007015760  0.5847458 3.022057    69
```

We selected another subset with confident > 0.5 and support 0.01. The right hand side is dominated by whole milk. Whole milk is strongly associated with various combinations of yogurt, whipped/sour cream, other vegetables, tropical fruit, root vegetables. Other vegetables is strongly associated with citrus fruit, root vegetables, tropical fruit, and rolls/buns.

```
## Choose a subset
inspect(subset(rules, subset=confidence > 0.5 & support > 0.01))
```

```
##      lhs                        rhs                   support confidence     lift count
## [1]  {curd,                                                                            
##       yogurt}              => {whole milk}       0.01006609  0.5823529 2.279125    99
## [2]  {butter,                                                                          
##       other vegetables}    => {whole milk}       0.01148958  0.5736041 2.244885   113
## [3]  {domestic eggs,                                                                   
##       other vegetables}    => {whole milk}       0.01230300  0.5525114 2.162336   121
## [4]  {whipped/sour cream,                                                              
##       yogurt}              => {whole milk}       0.01087951  0.5245098 2.052747   107
## [5]  {other vegetables,                                                                
##       whipped/sour cream} => {whole milk}       0.01464159  0.5070423 1.984385   144
## [6]  {other vegetables,                                                                
##       pip fruit}           => {whole milk}       0.01352313  0.5175097 2.025351   133
## [7]  {citrus fruit,                                                                    
##       root vegetables}     => {other vegetables} 0.01037112  0.5862069 3.029608   102
## [8]  {root vegetables,                                                                 
##       tropical fruit}      => {other vegetables} 0.01230300  0.5845411 3.020999   121
## [9]  {root vegetables,                                                                 
##       tropical fruit}      => {whole milk}       0.01199797  0.5700483 2.230969   118
## [10] {tropical fruit,                                                                  
##       yogurt}              => {whole milk}       0.01514997  0.5173611 2.024770   149
## [11] {root vegetables,                                                                 
##       yogurt}              => {whole milk}       0.01453991  0.5629921 2.203354   143
## [12] {rolls/buns,                                                                      
##       root vegetables}     => {other vegetables} 0.01220132  0.5020921 2.594890   120
## [13] {rolls/buns,                                                                      
##       root vegetables}     => {whole milk}       0.01270971  0.5230126 2.046888   125
## [14] {other vegetables,                                                                
##       yogurt}              => {whole milk}       0.02226741  0.5128806 2.007235   219
```

Conclusion

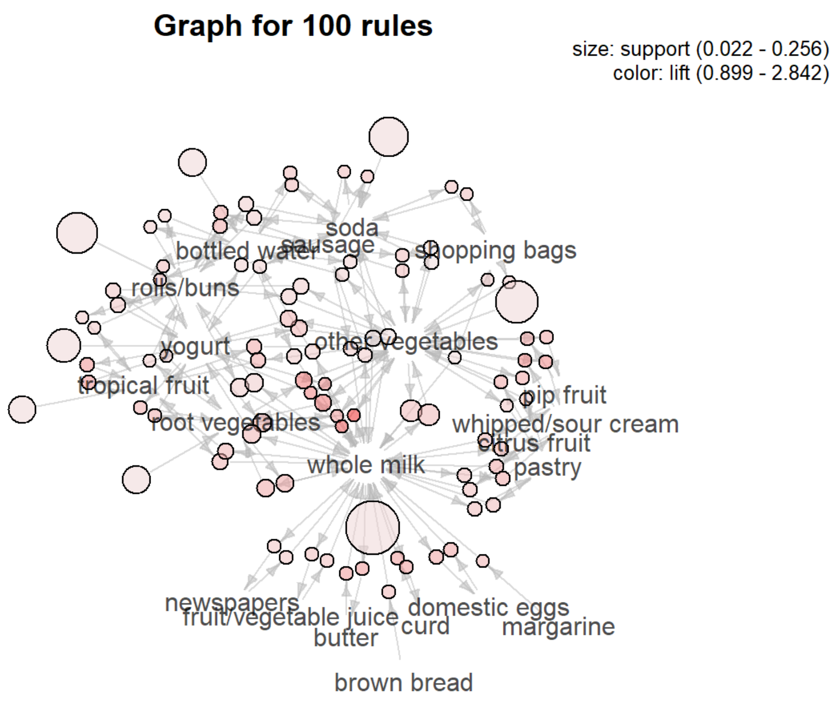
Based on the analysis above, we wanted to visualize the subset with confidence of 0.1 and support of 0.01. Not surprisingly, we see that many items cluster around whole milk and other vegetables. Rolls buns and soda are also surrounded by many items. These four cluster centers are also the top 4 most frequently purchased items. When customers came to the store to buy any of those four items, they are very likely to buy many combinations of other common grocery items too. It is a general grocery shopping pattern that people tend to buy many grocery items at once. To increase the chance of being sold for the food products with high profit margin, the grocery stores can place them around the cluster centers based on the category of the product. Moreover, the stores can scatter the products with high buying frequency since people will look for those items regardless of the placement very likely.

Additional, we found that the strong association rules exist mostly among food items. Household products did not appear in above analysis. This might due to the fact that the those products' buying frequency is much lower than that of food products.

```
# graph-based visualization
sub1 = subset(rules, subset= confidence > 0.1 & support > 0.01)
summary(sub1)
```

```
## set of 435 rules
##
## rule length distribution (lhs + rhs):sizes
##   1   2   3
##   8 331  96
##
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   1.000   2.000   2.000   2.202   2.000   3.000
##
## summary of quality measures:
##     support           confidence          lift            count
##  Min.   :0.01007   Min.   :0.1007   Min.   :0.7899   Min.   :  99.0
##  1st Qu.:0.01149   1st Qu.:0.1440   1st Qu.:1.3486   1st Qu.: 113.0
##  Median :0.01454   Median :0.2138   Median :1.6077   Median : 143.0
##  Mean   :0.02051   Mean   :0.2455   Mean   :1.6868   Mean   : 201.7
##  3rd Qu.:0.02115   3rd Qu.:0.3251   3rd Qu.:1.9415   3rd Qu.: 208.0
##  Max.   :0.25552   Max.   :0.5862   Max.   :3.3723   Max.   :2513.0
##
## mining info:
##       data ntransactions support confidence
##  groceries          9835   0.005        0.1
```

```
plot(sub1, method='graph')
```


Graph for 100 rules
size: support (0.022 - 0.256)
color: lift (0.899 - 2.842)
soda
bottled water
sausage
shopping bags
rolls/buns
yogurt
other vegetables
tropical fruit
pip fruit
root vegetables
whipped/sour cream
citrus fruit
whole milk
pastry
newspapers
fruit/vegetable juice
butter
domestic eggs
curd
margarine
brown bread