Predictive Modeling Exercises

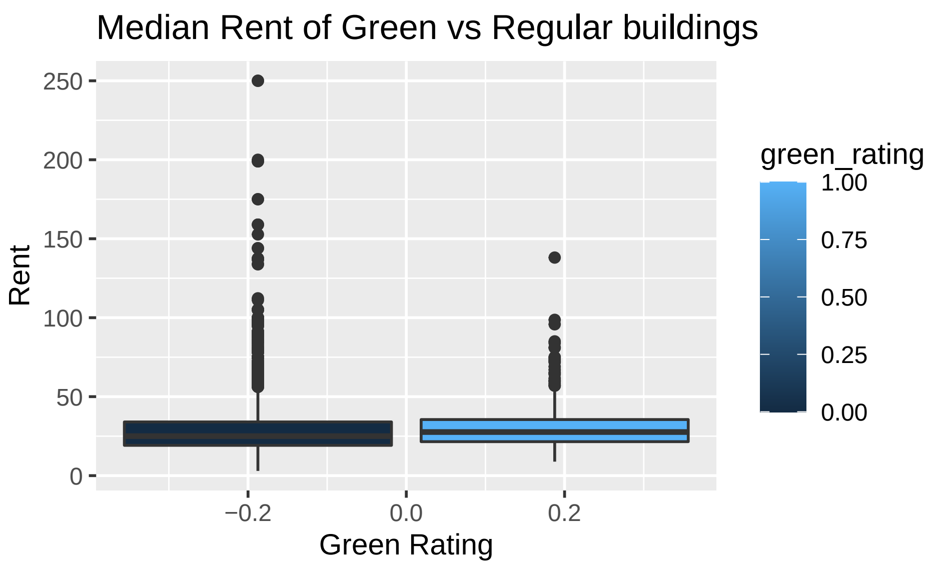
Shan Ali, Shifan Hu, Sitong Li

8/17/2020

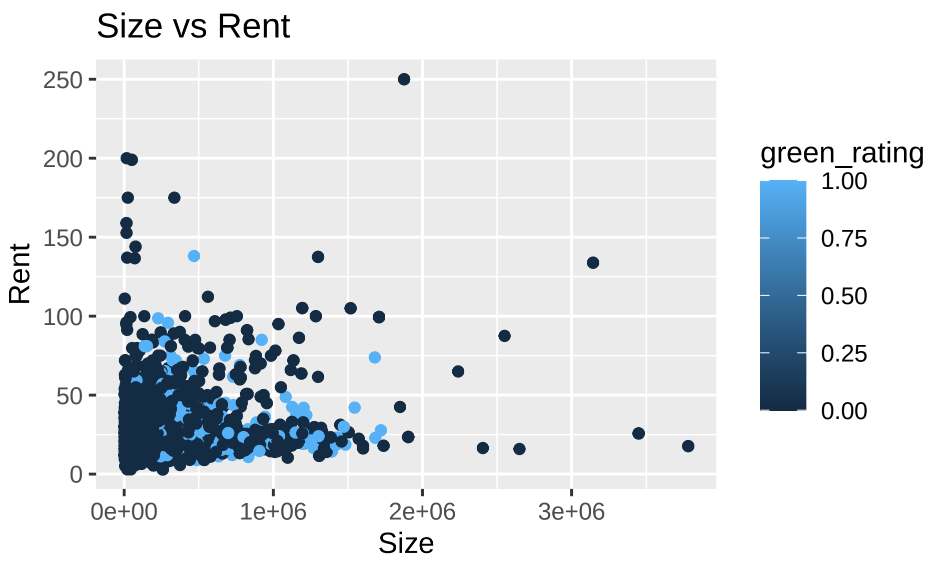
Visual Story Telling Part 1: Green Buildings

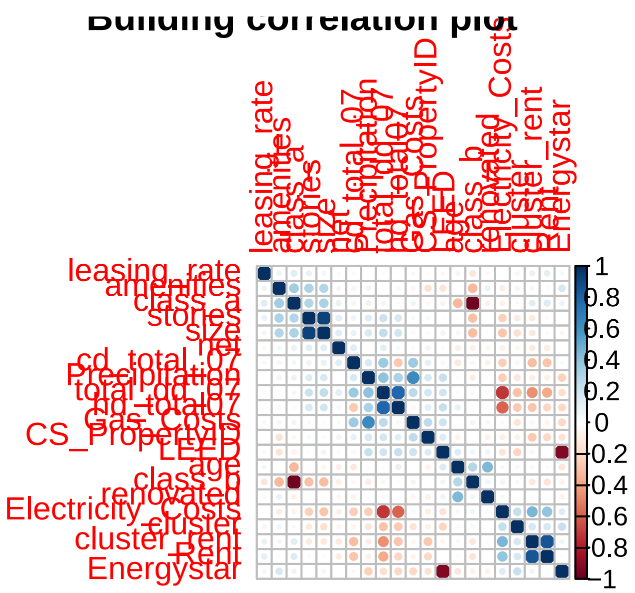
```r
# read file
gb = read.csv('../data/greenbuildings.csv')

# check for confounding variables
green = gb[gb$green_rating == 1,]
ngreen = gb[gb$green_rating == 0,]
```

From my exploration, I agree with the staff report. The data does support the conclusion that green buildings have higher median rent compared to that of regular buildings. However, regular buildings do display greater variability which could indicate confounding factors.

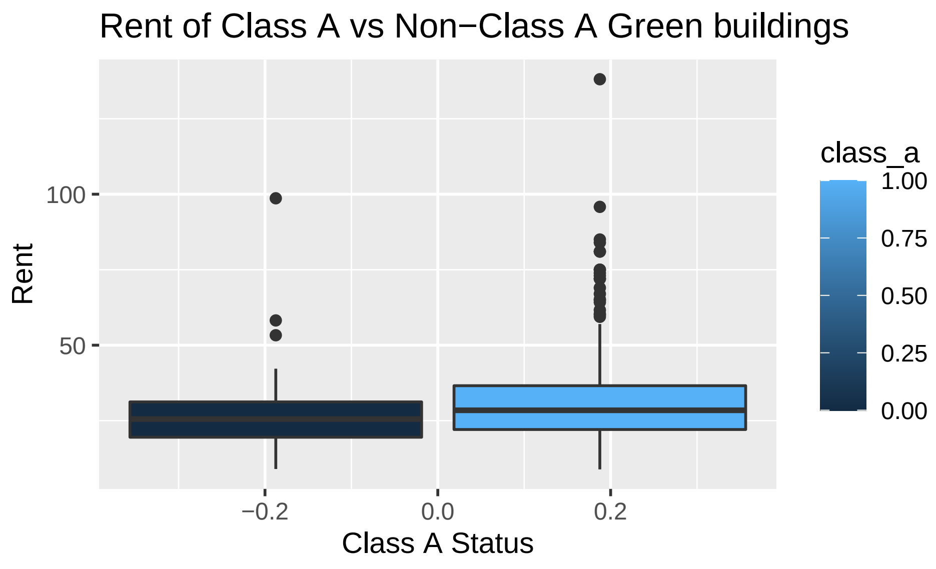


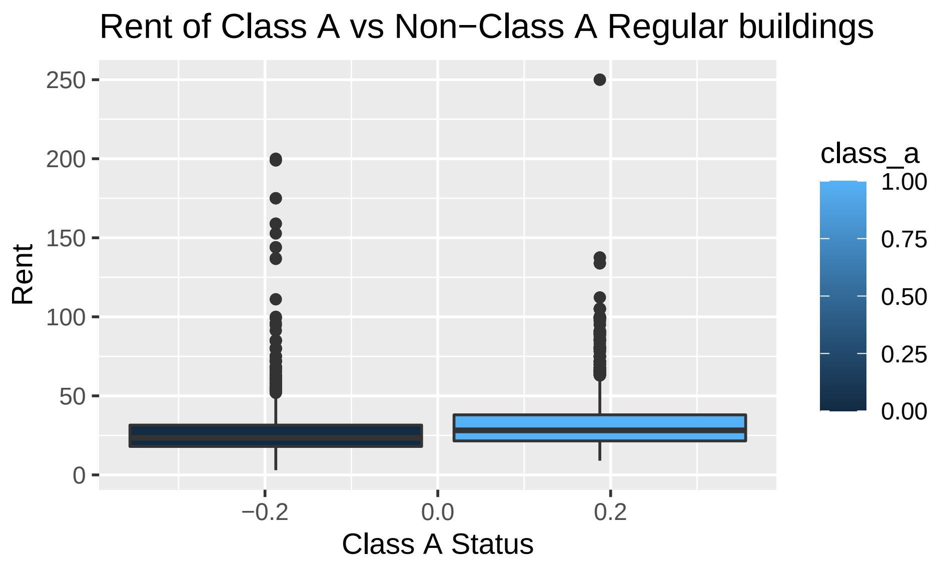
Thus, I explored for confounding variables that disrupted this position; exploring the data correlation by size, cluster, class, and precipitation. The exploration shows little correlation between rent prices and size, even accounting for cluster, class, and green rating.





Upon further analysis, the exploration does show evidence of confounding variables. For example, median rent for buildings in class A were slightly higher for green buildings, but had less of a spread than median rent not accounting for class. This trend also holds true for Class B buildings and supports the idea that class may also influence Rent; with green building status being less important for class a buildings.





This would then support the theory that the staff report was insufficient in its analysis, specifically in the profit projections. The presence of confounding factors necessitates the need to preform proper modeling to determine the potential lift in rent going green would make and then how it would price out. We recommend conducting a multi-variable regression or KNN analysis to determine most similar green and regular buildings and then compare the predicted rents to determine if going green is truly the better economic decision.

Visual story telling part 2: flights at ABIA

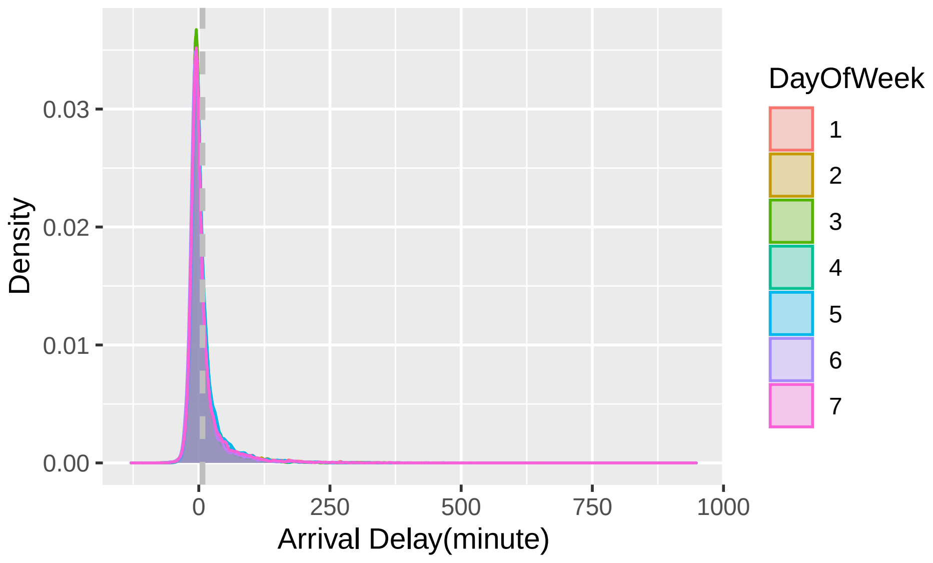
In this section, I would like to figure out the distribution of arrival delay (in minutes) in different day in week (between 1 to 7).

For this purpose, we plot a density distribution plot because we can review the density distribution of each attribute broken down by class value.

Like the scatter plot, the density plot by class can help see the separation of classes. It can also help to understand the overlap in class values for an attribute.

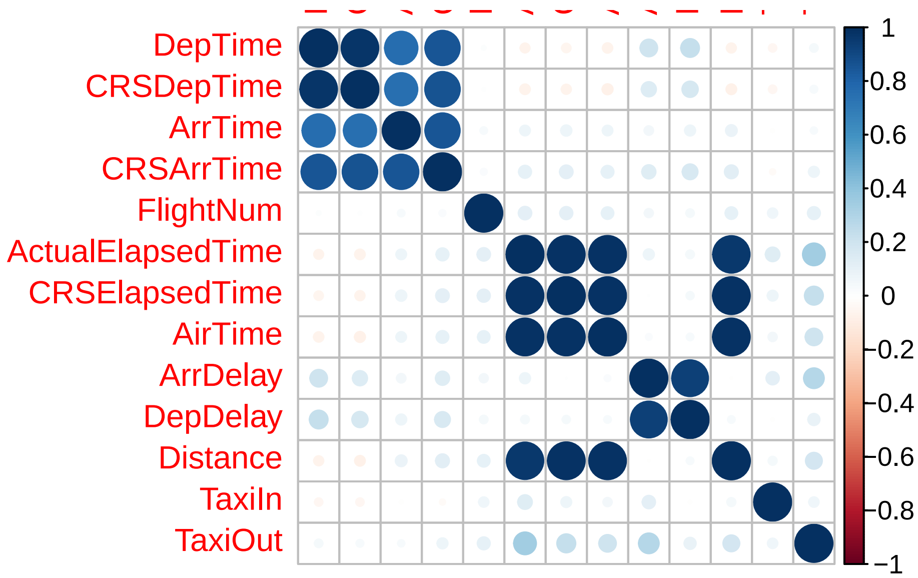
```
# read data
dat <- read.csv('ABIA.csv')
# choose the object variables to small set
subset1 <- subset(dat,
                  select = c('ArrDelay',
                             'DayOfWeek'))
# delete NA value
subset1 <- na.omit(subset1)

# check structure
str(subset1)
```

```
## 'data.frame':    97659 obs. of  2 variables:
##  $ ArrDelay : int  339 -9 -1 -23 -6 2 -15 -16 -6 -16 ...
##  $ DayOfWeek: int  2 2 2 2 2 2 2 2 2 2 ...
##  - attr(*, "na.action")= 'omit' Named int [1:1601] 250 251 252 253 254 255 552 553 554 555 ...
##   ..- attr(*, "names")= chr [1:1601] "250" "251" "252" "253" ...
```



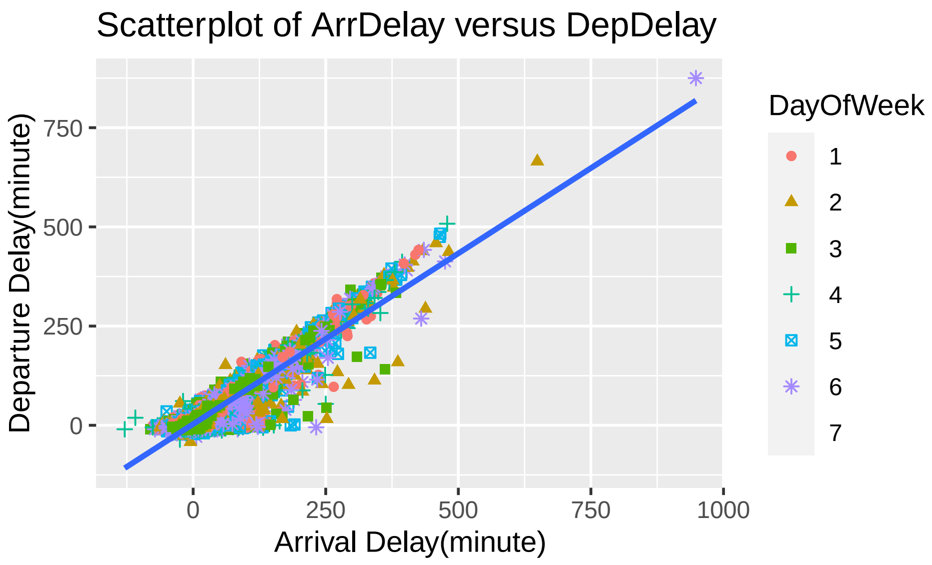
According to the density plot, we have reason to believe that the delay of arrival has no obvious relationship with the day of the week of the arrival date.

So further more, I generate a correlation plot to find out which variable is numerically correlated to arrive delay.



And it turns out to be the Departure delay. Now I use the scatter plot to show that:

```
## `geom_smooth()` using formula 'y ~ x'
```



It shows strong linear dependence between Arrival Delay and Departure Delay, but also prove that the delay of arrival has no obvious relationship with the day of the week of the arrival date.

Author Attribution

In this question, I am building the best predictive model for author attribution to the Reuters C50 corpus. This problem involved text processing and tokenization, then modeling.

First, I defined several text processing functions to simplify the text processing scripts since we need to process both the C50train and C50test data sets. The main function *textPipe* reads the files, trims and updates the file names, creates a raw text mining corpus, and finishes with tokenization and weighing. To tokenize, I made everything lowercase, removed numbers and punctuation, dropped all white space, and removed the stopwords (‘en’). This simplifies the text; reducing the information to the more relevant types and minimizes later computation requirements. I then converted the corpus into a doc-term-matrix and dropped the sparse terms. This reduces the terms significantly from about 32600 to 800 terms. Finally, *textPipe* applies TF-IDF weights to clean low and excessively high frequencies. I then applied the processing functions to the test and train directories, generating a X and Y train and test data sets.

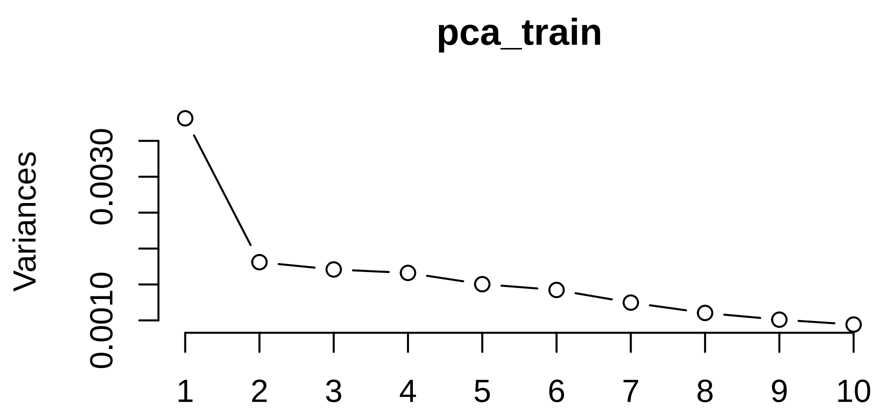
```
####################################
#####  Step 1: Function Prep  #####
####################################

# set up better read function to set id and language
readerPlain = function(fname){
                readPlain(elem=list(content=readLines(fname)),
                            id=fname, language='en') }

# main text processing function, for easy test and train processing
# returns td-idf table
textPipe = function(cdir){

  # read all files
  file_list = Sys.glob(cdir)
  C50 = lapply(file_list, readerPlain)

  # clean file names to reflect author and entry
  mynames = file_list %>%
    { strsplit(., '/', fixed=TRUE) } %>%
    { lapply(., tail, n=2) } %>%
    { lapply(., paste0, collapse = '') } %>%
      unlist

  # Rename the articles
  names(C50) = mynames

  # create text mining corpus
  documents_raw = Corpus(VectorSource(C50))

  # pre-processing for tokenization
  my_documents = documents_raw %>%
    tm_map(content_transformer(tolower))  %>%             # make everything lowercase
    tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
    tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
    tm_map(content_transformer(stripWhitespace))          # remove excess white-space

  # remove stopwords -> may or may not keep
  my_documents = tm_map(my_documents, content_transformer(removeWords), stopwords("en"))

  # create a doc-term-matrix from the corpus
  DTM_C50 = DocumentTermMatrix(my_documents)

  # remove sparse terms
  DTM_C50 = removeSparseTerms(DTM_C50, 0.95) # now ~ 800 terms (versus ~32600 before) -> big changes he
  # we want TF-IDF weights
  weightTfIdf(DTM_C50)
}

# function to get authors for Y
authorPipe = function(cdir){

  # read all files
  file_list = Sys.glob(cdir)
  C50 = lapply(file_list, readerPlain)

  # clean file names to reflect author and entry
  mynames = file_list %>%
    { strsplit(., '/', fixed=TRUE) } %>%
    { lapply(., head, n=5) } %>%
    { lapply(., tail, n=1) } %>%
      unlist

  mynames
}
```

```
####################################
#####  Step 2: Text Processing #####
####################################

train = '../data/ReutersC50/C50train/*/*.txt'
test = '../data/ReutersC50/C50test/*/*.txt'

X_train = textPipe(train)
X_test = textPipe(test)

Y_train = authorPipe(train)
Y_test = authorPipe(test)
```

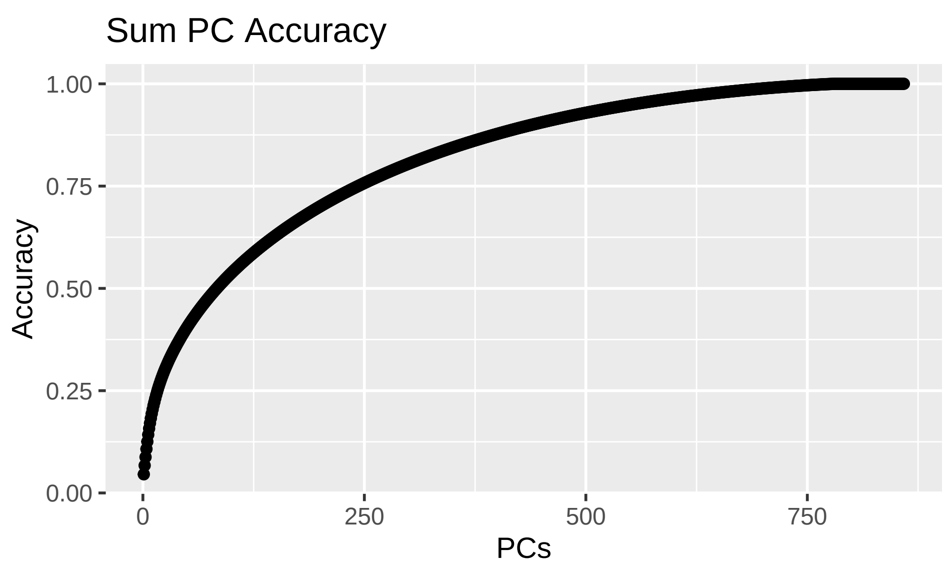
Next, I began reducing the dimensions to streamline the prediction computation required. First I combined the testing and training X data, converted it to a matrix, and then dropped all columns without any information. This trims the data set and removes and words (types/columns/terms) that are not in both data frames. I then ran a principal component analysis on the training data to compress the information. Upon exploration, approximately 70% of the variance is explained by the first 200 PCs. Reducing the features from >800 to 200 significantly reduces the feature requirement. Using this PC amount, I then set the test and train data sets to the first 200 PC for each set and added their true values (document authors).

```
########################################
#####  Step 3: Set Test and Train  #####
########################################

# get test and train as PCA
# process all documents for PCA, this allows scrubbing of non overlapping types
X = c(X_train,X_test)
X = as.matrix(X)
X = X[,which(colSums(X) != 0)]

# In case need to split before PCA
```

```r
X_train = X[1:2500,]
X_test = X[2501:5000,]

# split out to isolate train again
pca_train = prcomp(X_train)
pca_test = predict(pca_train ,newdata = X_test)

# explore num of PCs to use in model
plot(pca_train,type='line')
```



```r
var = apply(pca_train$x, 2, var)
prop = var / sum(var)
data = NULL
data['Acc'] = data.frame(cumsum(pca_train$sdev^2/sum(pca_train$sdev^2)))
data = data.frame(data)
data['# PCs'] = 1:859

# PC Accuracy Plot
ggplot(data = data) +
  geom_point(mapping = aes(x=`# PCs`, y=Acc)) +
  labs(title="Sum PC Accuracy",x="PCs", y = "Accuracy")
```



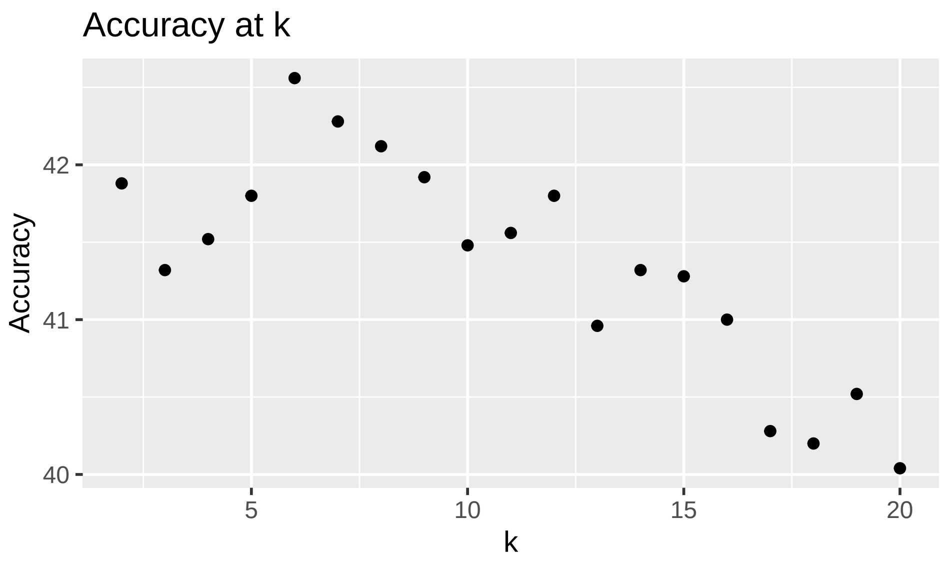
```
# isolate desired PCAs into train and test
npca = 200 # 200 PCs explain approx. 70% of variance
pca_train = data.frame(pca_train$x[,1:npca]) #first half is train
pca_test = data.frame(pca_test[,1:npca]) # 2nd is test

# add Y to get complete DF
pca_train['author'] = Y_train
pca_test['author'] = Y_test
```

To predict the authors attribution, I am exploring three model types: random forests, naive Bayes, and KNN. First, I explored random forest using a mtry = 14, which is the rounded squared root of p predictors. This resulted in a testing accuracy of 51.16%. Next, I explored a simpler Niave Bayes model. This resulted in a testing accuracy of 42.80%. Lastly, I explored the KNN model. I compared models of k = 2:20, and discovered the best k = 7. This model with k =7 resulted in a testing accuracy of around 41.04%

```
#################################
#####  Random Forest Model  #####
#################################

# set mtry
mtry = round(sqrt(npca)) # rounded sqrt of number of features

randForst = randomForest(as.factor(author)~., data=pca_train, mtry=mtry, importance=TRUE)

# accuracy check
testPred = predict(randForst, newdata=pca_test) # get predictions
temp = as.data.frame(cbind(testAct, testPred)) # codify results
temp$flag = ifelse(temp$testAct == temp$testPred, 1, 0) # calculate number correct
acc = c(acc, sum(temp$flag)*100/nrow(temp)) # store accuracy
```

```
###############################
#####  Naive Bayes Model  #####
###############################
```

```
naiBay = naiveBayes(as.factor(author)~., data=pca_train)

# accuracy check
testPred = predict(naiBay, newdata=pca_test) # get predictions
temp = as.data.frame(cbind(testAct, testPred)) # codify results
temp$flag = ifelse(temp$testAct == temp$testPred, 1, 0) # calculate number correct
acc = c(acc, sum(temp$flag)*100/nrow(temp)) # store accuracy
```



```
#######################
#####  Knn Model  #####
#######################

# best model (k = 7)
knnPred = knn(trainX, testX, trainAct, k=kk[best])

# accuracy check
temp = as.data.frame(cbind(testAct, knnPred)) # codify results
temp$flag = ifelse(temp$testAct == temp$knnPred, 1, 0) # calculate number correct
acc = c(acc, sum(temp$flag)*100/nrow(temp)) # store accuracy
```

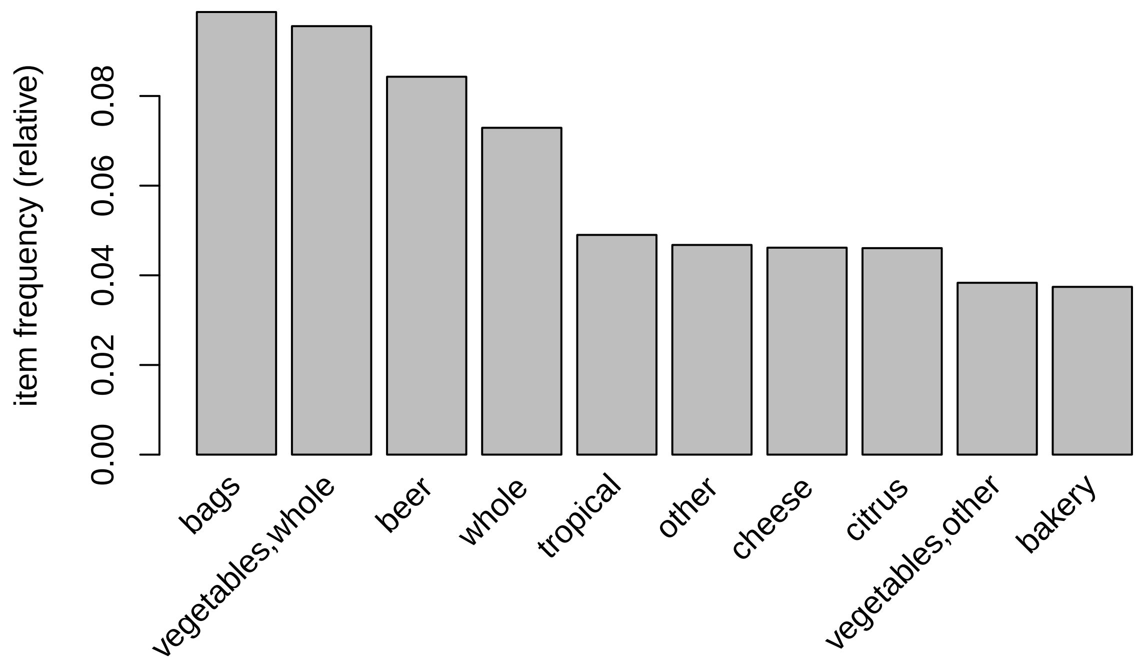
In conclusion, the Random Forest model was the best model generated to predict author attribution. Compare to a baseline accuracy of 2%, all of these models are a significant improvement from random guessing.

```
##           Model Test.Accuracy
## 1 Random Forest         51.16
## 2   Naive Bayes         42.80
## 3           KNN         41.04
```

Association rule mining

In this section, I will use the data on grocery purchases in `groceries.txt` and find some interesting association rules for these shopping baskets.

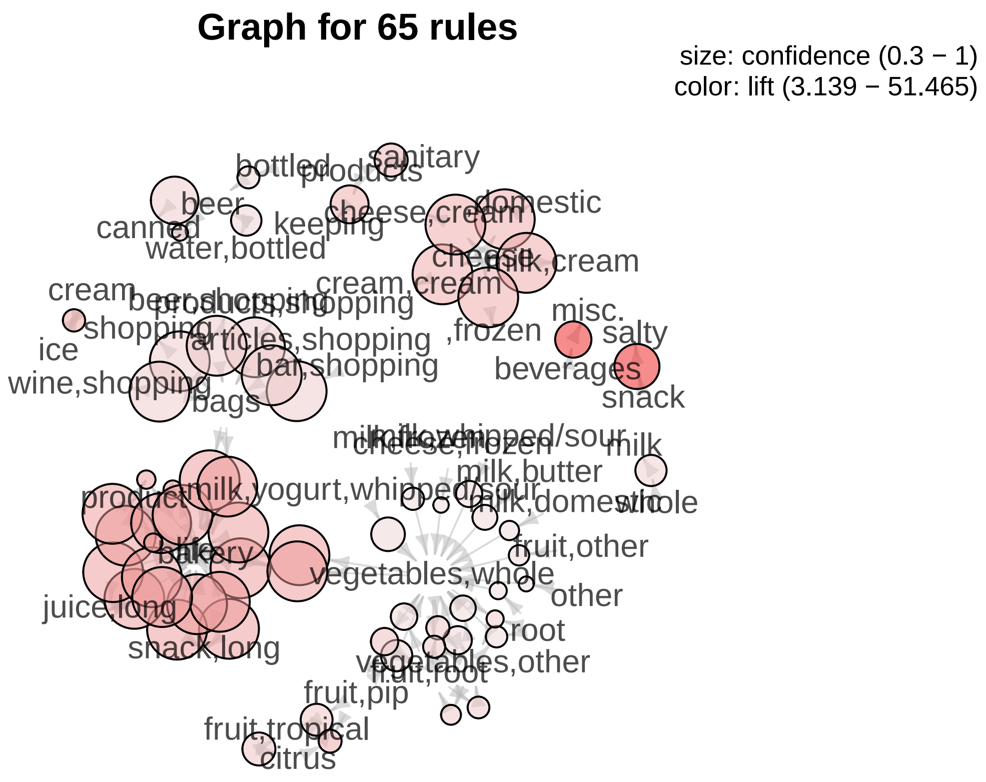
```r
# plot the most frequent items (top 10)
itemFrequencyPlot(tdata, topN = 10)
```



```r
# Visualising the rules
inspect(sort(associa_rules, by = 'lift')[1:10])
```

```
##      lhs              rhs          support     confidence coverage    lift
## [1]  {salty}       => {snack}      0.003050330 0.7692308  0.003965430 51.46520
## [2]  {misc.}       => {beverages}  0.003152008 0.6326531  0.004982206 51.42267
## [3]  {snack,long}  => {bakery}     0.003355363 1.0000000  0.003355363 26.72554
## [4]  {snack,long}  => {life}       0.003355363 1.0000000  0.003355363 26.72554
## [5]  {juice,long}  => {bakery}     0.004270463 1.0000000  0.004270463 26.72554
## [6]  {juice,long}  => {life}       0.004270463 1.0000000  0.004270463 26.72554
## [7]  {product}     => {bakery}     0.013218099 1.0000000  0.013218099 26.72554
## [8]  {bakery}      => {product}    0.013218099 0.3532609  0.037417387 26.72554
## [9]  {product}     => {life}       0.013218099 1.0000000  0.013218099 26.72554
## [10] {life}        => {product}    0.013218099 0.3532609  0.037417387 26.72554
##      count
## [1]   30
## [2]   31
## [3]   33
## [4]   33
## [5]   42
## [6]   42
## [7]  130
## [8]  130
## [9]  130
## [10] 130
```

```
plot(associa_rules, method = "graph",
     measure = "confidence", shading = "lift")
```


Graph for 65 rules
size: confidence (0.3 − 1)
color: lift (3.139 − 51.465)
bottled
products
sanitary
beer
canned
keeping
cheese,cream
domestic
water,bottled
milk,cream
cream
cream,cream
misc.
shopping
articles,shopping
,frozen
salty
ice
bar,shopping
beverages
wine,shopping
bags
snack
cheese,frozen
milk
milk,butter
milk,yogurt,whipped/sour
milk,domestic
whole
product
fruit,other
bakery
vegetables,whole
other
juice,long
root
snack,long
vegetables,other
fruit,root
fruit,pip
fruit,tropical
citrus