STA380 Exercises

2020/8/7

Visual story telling part 1: green buildings

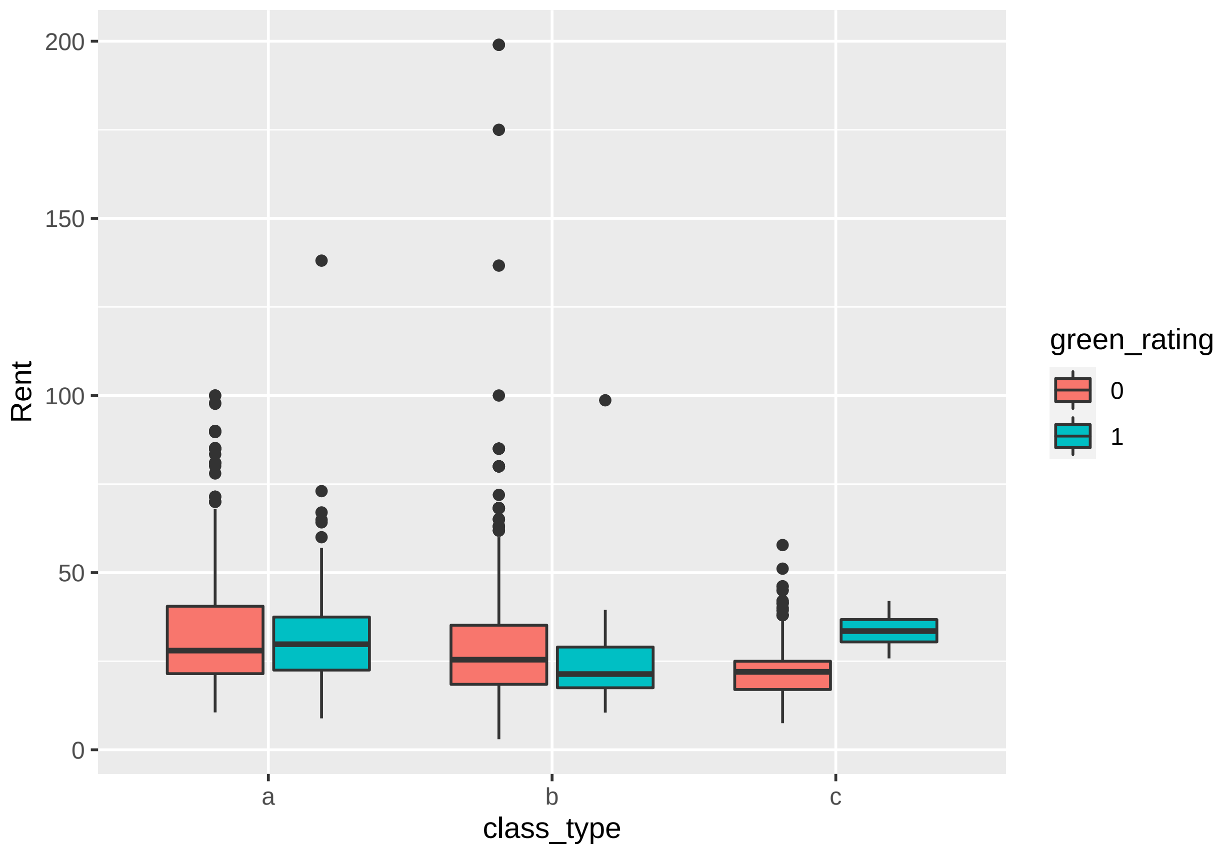
```
library(ggplot2)
library(tidyverse)
greenbuildings_df<-read.csv("data/greenbuildings.csv",stringsAsFactors = F)

greenbuildings_df<-greenbuildings_df %>%
  mutate(class_type=case_when(class_a == 1 ~ "a",
                              class_b == 1 ~ "b",
                              TRUE ~ "c"),green_rating=as.factor(green_rating))

summary(greenbuildings_df$stories)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    1.00    4.00   10.00   13.58   19.00  110.00
```

```
greenbuildings_sel<-greenbuildings_df %>%
  filter(leasing_rate>10,stories>=10,stories<=30)

greenbuildings_sel %>%
  group_by(class_type,green_rating) %>%
  summarise(median_rent=median(Rent))
```

```
## # A tibble: 6 x 3
## # Groups:   class_type [3]
##   class_type green_rating median_rent
##   <chr>      <fct>              <dbl>
## 1 a          0                   28
## 2 a          1                   29.8
## 3 b          0                   25.4
## 4 b          1                   21.4
## 5 c          0                   22
## 6 c          1                   33.5
```

```
ggplot(data = greenbuildings_sel,aes(x = class_type, y = Rent,fill = green_rating)) +
  geom_boxplot()
```



Her on-staff considered removing a few buildings with very low occupancy rates, which I think makes sense. The floors in the data range from 1 to 110 floors, and floors have a certain impact on rent. In order to reduce this impact, I chose the data of 10 to 30 floors for analysis.

Building quality is divided into 3 categories in total, and the rents corresponding to these three types of buildings are calculated separately. Class A buildings are generally the highest-quality properties in a given market. The median market rent in the non-green buildings was $$$28.00 per square foot per year, while the median market rent in the green buildings was $$$29.79 per square foot per year: about $1.79 more per square foot. It is estimated that these costs can be recovered in 5000000/(250000*1.79)=11.2 years.

Class B buildings are a notch down, but still of reasonable quality. The median market rent in the non-green buildings was $$$25.44 per square foot per year, while the median market rent in the green buildings was $$$21.37 per square foot per year. On the contrary, the rent of the building is relatively low, and the construction of green buildings will not obtain additional benefits.

Class C buildings are the least desirable properties in a given market.The median market rent in the non-green buildings was $$$22.5 per square foot per year, while the median market rent in the green buildings was $$$28.9 per square foot per year: about $6.4 more per square foot. It is estimated that these costs can be recovered in 5000000/(250000*6.4)=3.1 years.

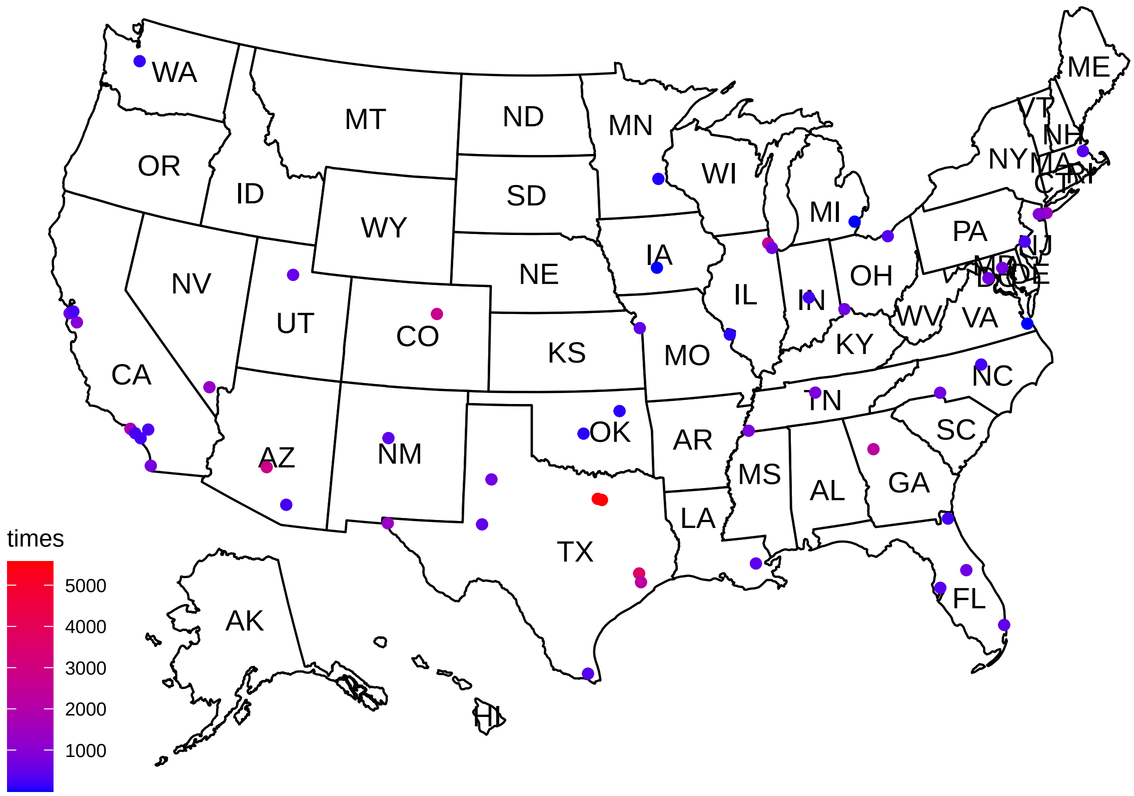
So for Class A or Class C buildings,building a green building seems like a good financial move to build the green building.But it is not suitable for Class B.

Visual story telling part 2: flights at ABIA

```
library(usmap)
library(tidyr)
```

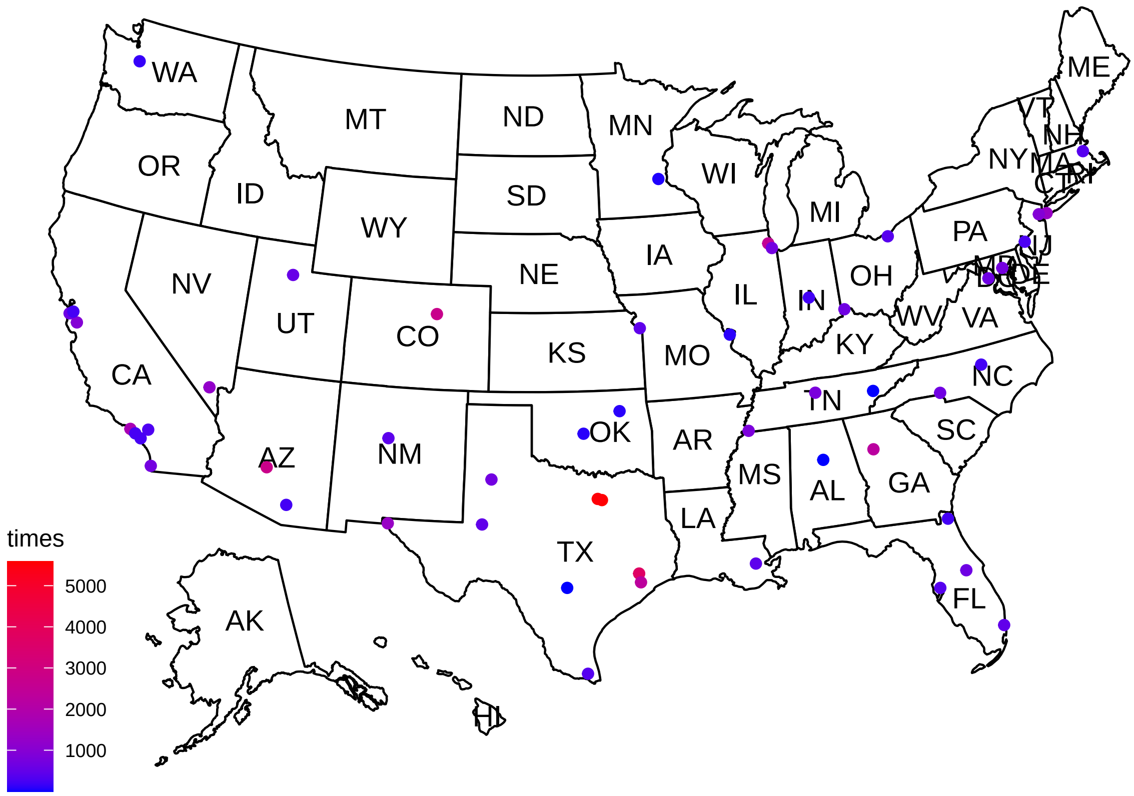
```r
abia_df<-read.csv("data/ABIA.csv",stringsAsFactors = F)
airport_codes<-read.csv("data/airport-codes.csv",stringsAsFactors = F)
```

```r
departed_df<-abia_df %>%
  filter(Origin=="AUS") %>%
  group_by(Dest) %>%
  summarise(times=n()) %>%
  rename(iata_code=Dest) %>%
  arrange(desc(times))

departed_location<-departed_df %>%
  left_join(airport_codes,by="iata_code") %>%
  separate(coordinates,c("lat","lon"),sep = ", ") %>%
  select(iata_code,lon,lat,times,iso_region) %>%
  mutate(lon=as.numeric(lon),lat=as.numeric(lat))


# Make a map.
# First, project project the lon, lat coordinates
# to the same coordinate system used by usmap
departed_station_map <- departed_location %>%
  select(lon, lat) %>%
  usmap_transform

departed_location <- merge(departed_location, departed_station_map, by=c('lat', 'lon'))

departed_location <- departed_location %>%
  arrange(desc(times))

# plot the coordinates of departed location
plot_usmap(labels = T) +
  scale_color_gradient(low = 'blue', high='red') +
  geom_point(data=departed_location, aes(x=lon.1, y=lat.1, color=times))
```




```r
landed_df<-abia_df %>%
  filter(Dest=="AUS") %>%
  group_by(Origin) %>%
  summarise(times=n()) %>%
  rename(iata_code=Origin) %>%
  arrange(desc(times))

landed_location<-landed_df %>%
  left_join(airport_codes,by="iata_code") %>%
  separate(coordinates,c("lat","lon"),sep = ", ") %>%
  select(iata_code,lon,lat,times,iso_region) %>%
  mutate(lon=as.numeric(lon),lat=as.numeric(lat))


# Make a map.
# First, project project the lon, lat coordinates
# to the same coordinate system used by usmap
landed_station_map <- landed_location %>%
  select(lon, lat) %>%
  usmap_transform

landed_location <- merge(landed_location, landed_station_map, by=c('lat', 'lon'))

landed_location <- landed_location %>%
  arrange(desc(times))
```

```r
# plot the coordinates of landed location
plot_usmap(labels = T) +
  scale_color_gradient(low = 'blue', high='red') +
  geom_point(data=landed_location, aes(x=lon.1, y=lat.1, color=times))
```



Austin-Bergstrom Interational Airport is in Texas, the United States. The three airports with the most flights into and out of Austin are DAL, DFW, and IAH. It can be seen from the figure that these four airports are in Texas State

Portfolio modeling

```r
library(mosaic)
library(quantmod)
library(foreach)

calc_VaR<-function(all_returns){
  set.seed(1234)
  # Now simulate many different possible futures
  # just repeating the above block thousands of times
  initial_wealth = 100000
  sim1 = foreach(i=1:5000, .combine='rbind') %do% {
    total_wealth = initial_wealth
    weights = rep(1/ncol(all_returns),ncol(all_returns))
    holdings = weights * total_wealth
```

```
    n_days = 20
    wealthtracker = rep(0, n_days)
    for(today in 1:n_days) {
        return.today = resample(all_returns, 1, orig.ids=FALSE)
        holdings = holdings + holdings*return.today
        total_wealth = sum(holdings)
        wealthtracker[today] = total_wealth
    }
    wealthtracker
  }
  # 5% value at risk:
  quantile(sim1[,n_days]- initial_wealth, prob=0.05)
}
```

- Government Bonds ETFs offer investors exposure to fixed income securities issued by government agencies. Bonds featured in these ETFs include U.S. Treasuries of varying maturities, floating rate Treasury bonds, and TIPS.Select 5 ETFs from Government Bonds ETFs for analysis as follows.

```
etfs1 = c("SHY", "SHV", "IEF", "TLT", "GOVT")
myprices = getSymbols(etfs1, from = "2014-01-01")

# A chunk of code for adjusting all stocks
# creates a new object adding 'a' to the end
# For example, WMT becomes WMTa, etc
for(ticker in etfs1) {
    expr = paste0(ticker, "a = adjustOHLC(", ticker, ")")
    eval(parse(text=expr))
}

# Combine all the returns in a matrix
all_returns = cbind(    ClCl(SHYa),
                                ClCl(SHVa),
                                ClCl(IEFa),
                                ClCl(TLTa),
                                ClCl(GOVTa))
all_returns = as.matrix(na.omit(all_returns))
print(calc_VaR(all_returns))
```

```
##        5%
## -1889.072
```

- China equities ETFs are funds that invest in China-based corporations. The funds in this category include index funds as well as category specific funds.Choose 5 ETFs from China equities ETFs for analysis as follows.

```
etfs2 = c("PGJ", "FXI", "CXSE", "KURE", "FCA")
myprices = getSymbols(etfs2, from = "2014-01-01")

# A chunk of code for adjusting all stocks
# creates a new object adding 'a' to the end
# For example, WMT becomes WMTa, etc
for(ticker in etfs2) {
```

```r
    expr = paste0(ticker, "a = adjustOHLC(", ticker, ")")
    eval(parse(text=expr))
}

# Combine all the returns in a matrix
all_returns = cbind(    ClCl(PGJa),
                                ClCl(FXIa),
                                ClCl(CXSEa),
                                ClCl(KUREa),
                                ClCl(FCAa))
all_returns = as.matrix(na.omit(all_returns))
print(calc_VaR(all_returns))
```

```
##        5%
## -11344.47
```

- Choose 3 ETFs from Government Bonds ETFs and 2 ETFs from China equities ETFs. Together, construct the investment portfolio as follows.

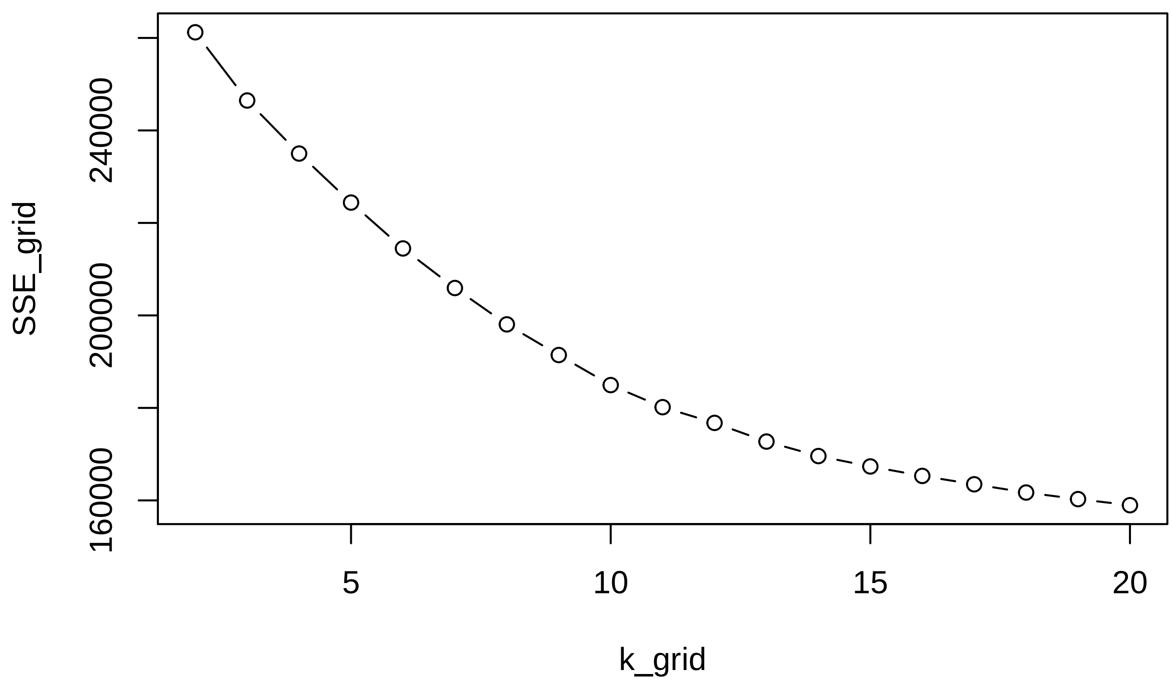
```r
etfs3 = c("SHY", "SHV", "IEF","PGJ", "FXI")
myprices = getSymbols(etfs3, from = "2014-01-01")

# A chunk of code for adjusting all stocks
# creates a new object adding 'a' to the end
# For example, WMT becomes WMTa, etc
for(ticker in etfs3) {
    expr = paste0(ticker, "a = adjustOHLC(", ticker, ")")
    eval(parse(text=expr))
}

# Combine all the returns in a matrix
all_returns = cbind(    ClCl(SHYa),
                                ClCl(SHVa),
                                ClCl(IEFa),
                                ClCl(PGJa),
                                ClCl(FXIa))
all_returns = as.matrix(na.omit(all_returns))
print(calc_VaR(all_returns))
```

```
##        5%
## -3635.952
```

From the above results, it can be seen that the risk of Bonds is relatively small, and the VaR value of the portfolio is -1889.072. Equities risk is relatively large first, and the VaR value of the constituted portfolio is -11344.47. The VaR value of the portfolio composed of Bonds and equity is -3635.952, and the risk is also in the middle of the three.

Market segmentation

```r
library(foreach)
market_df<-read.csv("data/social_marketing.csv")
market_scale<-scale(market_df[,-1])
k_grid <- seq(2,20,by=1)
SSE_grid <- foreach(k=k_grid,.combine='c') %do% {
  cluster_k=kmeans(market_scale, k, nstart=50)
  cluster_k$tot.withinss
}
plot(k_grid, SSE_grid,type="b")
```



The sharp decreases from one to three clusters (with little decrease after) suggests a three-cluster solution.

```r
set.seed(1234)
cluster_final=kmeans(market_scale, 3, nstart=50)
market_df$cluster<-cluster_final$cluster
mean_market_df<-market_df %>%
  group_by(cluster) %>%
  summarise(chatter_mean=mean(chatter),
            current_events_mean=mean(current_events),
            travel_mean=mean(travel),
            photo_sharing_mean=mean(photo_sharing),
            uncategorized_mean=mean(uncategorized),
            tv_film_mean=mean(tv_film),
            sports_fandom_mean=mean(sports_fandom),
            politics_mean=mean(politics),
```

```
            food_mean=mean(food),
            family_mean=mean(family),
            home_and_garden_mean=mean(home_and_garden),
            music_mean=mean(music),news_mean=mean(news),
            online_gaming_mean=mean(online_gaming),
            shopping_mean=mean(shopping),
            health_nutrition_mean=mean(health_nutrition),
            college_uni_mean=mean(college_uni),
            sports_playing_mean=mean(sports_playing),
            cooking_mean=mean(cooking),
            eco_mean=mean(eco),
            computers_mean=mean(computers),
            business_mean=mean(business),
            outdoors_mean=mean(outdoors),
            crafts_mean=mean(crafts),
            automotive_mean=mean(automotive),
            art_mean=mean(art),
            religion_mean=mean(religion),
            beauty_mean=mean(beauty),
            parenting_mean=mean(parenting),
            dating_mean=mean(dating),
            school_mean=mean(school),
            personal_fitness_mean=mean(personal_fitness),
            fashion_mean=mean(fashion),
            small_business_mean=mean(small_business),
            spam_mean=mean(spam),
            adult_mean=mean(adult),
            cluster_mean=mean(cluster))

t(round(mean_market_df,2))
```

```
##                       [,1] [,2] [,3]
## cluster               1.00 2.00 3.00
## chatter_mean          3.59 4.07 6.36
## current_events_mean   1.36 1.70 1.85
## travel_mean           1.24 1.54 2.39
## photo_sharing_mean    1.85 2.49 4.71
## uncategorized_mean    0.67 0.74 1.18
## tv_film_mean          0.81 1.04 1.67
## sports_fandom_mean    0.99 5.88 1.37
## politics_mean         1.34 1.50 2.94
## food_mean             0.82 4.58 1.52
## family_mean           0.57 2.47 0.94
## home_and_garden_mean  0.41 0.67 0.73
## music_mean            0.48 0.72 1.12
## news_mean             0.93 1.20 1.83
## online_gaming_mean    0.91 1.27 1.86
## shopping_mean         0.94 1.37 2.42
## health_nutrition_mean 1.64 2.18 4.84
## college_uni_mean      1.12 1.45 2.56
## sports_playing_mean   0.46 0.76 1.01
## cooking_mean          0.99 1.72 4.41
## eco_mean              0.35 0.65 0.83
```

```
## computers_mean        0.43 0.83 1.08
## business_mean         0.29 0.51 0.69
## outdoors_mean         0.53 0.76 1.37
## crafts_mean           0.32 1.07 0.76
## automotive_mean       0.63 1.07 1.19
## art_mean              0.47 0.87 1.25
## religion_mean         0.52 5.31 0.84
## beauty_mean           0.34 1.10 1.40
## parenting_mean        0.46 4.03 0.80
## dating_mean           0.44 0.67 1.35
## school_mean           0.40 2.68 0.88
## personal_fitness_mean 0.92 1.39 2.72
## fashion_mean          0.48 1.03 2.16
## small_business_mean   0.23 0.39 0.57
## spam_mean             0.00 0.01 0.01
## adult_mean            0.35 0.40 0.52
## cluster_mean          1.00 2.00 3.00
```

It can be seen from the results that the first group of family, food, music scores are the highest, the second group of religion, food scores are the highest, and the third group of chatter, photo_sharing scores are the highest.

Author attribution

```r
library(tm)
library(randomForest)
library(stringr)
readerPlain = function(fname){
                readPlain(elem=list(content=readLines(fname)),
                            id=fname, language='en') }

#
readC50Data<-function(c50_dir){
  file_list = NULL
  labels = NULL
  for(author in c50_dir) {
    author_name = str_split(author,"/")[[1]][4]
    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)))
  }

  file_content<-lapply(file_list,readerPlain)

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

  ## Some pre-processing/tokenization steps.
  ## tm_map just maps some function to every document in the corpus
  my_documents = documents_raw %>%
    tm_map(content_transformer(tolower))  %>%             # make everything lowercase
```

```r
    tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
    tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
    tm_map(content_transformer(stripWhitespace))          # remove excess white-space

  # let's just use the "basic English" stop words
  my_documents = tm_map(my_documents, content_transformer(removeWords), stopwords("en"))


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

  DTM_simon = removeSparseTerms(DTM_simon, 0.95)


  # construct TF IDF weights -- might be useful if we wanted to use these
  # as features in a predictive model
  tfidf_simon = weightTfIdf(DTM_simon)

  # Now PCA on term frequencies
  X = as.data.frame(as.matrix(tfidf_simon))

  ret<-cbind(X,labels)
  ret$labels<-as.factor(ret$labels)
  ret
}

#Read the files in the C50train directory as training set data
author_train_dirs = Sys.glob('data/ReutersC50/C50train/*')
train_data<-readC50Data(author_train_dirs)
names(train_data)[which(names(train_data) == "next")]<-"next_"
# Build a random forest model
rf_fit<-randomForest(labels~.,train_data)

#Read the files in the C50test directory as test set data
author_test_dirs = Sys.glob('data/ReutersC50/C50test/*')
test_data<-readC50Data(author_test_dirs)
terms_not_in_test<-setdiff(names(train_data),names(test_data))
for(term in terms_not_in_test){
  test_data[term]<-0
}
#Calculate model accuracy
mean(predict(rf_fit,test_data)==test_data$labels)
```

```
## [1] 0.5876
```

Association rule mining

Importing the required libraries

```r
library(arules)
```

Getting the data and preprocessing it.We can read each row of data in a loop, and then use strsplit to convert each row of data into a vector, and all rows of data are formed into a list object. Then use the as method to convert to transactions object.

```
conn <- file("data/groceries.txt",open="r")
lines <-readLines(conn)
line.max <-length(lines)
groceries.list<-list()
for(i in 1:line.max){
    line<-lines[i]
    groceries.list[[i]]<-unlist(strsplit(line, ","))
}
close(conn)

## coerce into transactions
groceries_trans_1 <- as(groceries.list, "transactions")
```

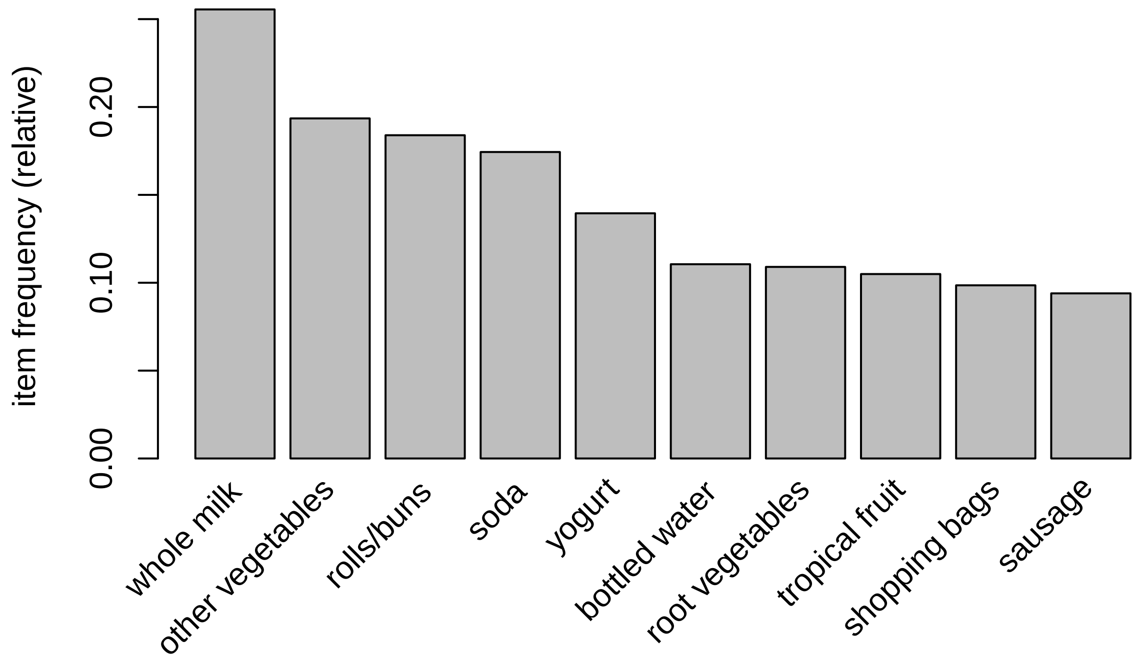
Package arules provides a more concise way, using the read.transactions method to directly convert the file into the required transaction format.

```
groceries_trans <- read.transactions("data/groceries.txt", sep = ",")
summary(groceries_trans)
```

```
## 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   16
## 2159 1643 1299 1005  855  645  545  438  350  246  182  117   78   77   55   46
##   17   18   19   20   21   22   23   24   26   27   28   29   32
##   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
```

The following graphic shows the 10 items with the highest frequency.

```
itemFrequencyPlot(groceries_trans,topN=10)
```



Below we use the apriori method, the default setting of the method is: (1) supp = 0.1, which is the minimum support for the rule; (2) conf = 0.8, which is the minimum confidence of the rule; (3) maxlen = 10, This is the maximum length of the rule. It can be seen from the results that no association rules have been filtered out.

```
apriori(groceries_trans)
```

```
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##         0.8    0.1    1 none FALSE            TRUE       5     0.1      1
##  maxlen target  ext
##      10  rules TRUE
##
## Algorithmic control:
##  filter tree heap memopt load sort verbose
##     0.1 TRUE TRUE  FALSE TRUE    2    TRUE
##
## Absolute minimum support count: 983
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.01s].
## sorting and recoding items ... [8 item(s)] done [0.00s].
## creating transaction tree ... done [0.01s].
## checking subsets of size 1 2 done [0.00s].
```

```
## writing ... [0 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].

## set of 0 rules
```

```r
rule1 <- apriori(groceries_trans, parameter = list(support =0.005, confidence = 0.1, minlen = 2))
```

```
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##         0.1    0.1    1 none FALSE            TRUE       5   0.005      2
##  maxlen target  ext
##      10  rules TRUE
##
## 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 [0.02s].
## sorting and recoding items ... [120 item(s)] done [0.00s].
## creating transaction tree ... done [0.01s].
## checking subsets of size 1 2 3 4 done [0.01s].
## writing ... [1574 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
```

```r
inspect(sort(rule1, by = "lift")[1:10])
```

```
##      lhs                        rhs                    support confidence   coverage     lift count
## [1]  {ham}                   => {white bread}      0.005083884  0.1953125 0.02602949 4.639851    50
## [2]  {white bread}           => {ham}              0.005083884  0.1207729 0.04209456 4.639851    50
## [3]  {citrus fruit,
##       other vegetables,
##       whole milk}            => {root vegetables}  0.005795628  0.4453125 0.01301474 4.085493    57
## [4]  {butter,
##       other vegetables}      => {whipped/sour cream} 0.005795628  0.2893401 0.02003050 4.036397    57
## [5]  {herbs}                 => {root vegetables}  0.007015760  0.4312500 0.01626843 3.956477    69
## [6]  {other vegetables,
##       root vegetables}       => {onions}           0.005693950  0.1201717 0.04738180 3.875044    56
## [7]  {citrus fruit,
##       pip fruit}             => {tropical fruit}   0.005592272  0.4044118 0.01382816 3.854060    55
## [8]  {berries}               => {whipped/sour cream} 0.009049314  0.2721713 0.03324860 3.796886    89
## [9]  {whipped/sour cream} => {berries}             0.009049314  0.1262411 0.07168277 3.796886    89
## [10] {other vegetables,
##       tropical fruit,
##       whole milk}            => {root vegetables}  0.007015760  0.4107143 0.01708185 3.768074    69
```

It can be seen from the results that white bread and ham have the highest correlation.

```r
options(digits = 5)

 rule2 <-apriori(groceries_trans, parameter = list(supp = 0.001, conf = 0.1,maxlen=5))
```

```
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##         0.1    0.1    1 none FALSE            TRUE       5   0.001      1
##  maxlen target  ext
##       5  rules TRUE
##
## Algorithmic control:
##  filter tree heap memopt load sort verbose
##     0.1 TRUE TRUE  FALSE TRUE    2    TRUE
##
## Absolute minimum support count: 9
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.01s].
## sorting and recoding items ... [157 item(s)] done [0.00s].
## creating transaction tree ... done [0.01s].
## checking subsets of size 1 2 3 4 5 done [0.04s].
## writing ... [32731 rule(s)] done [0.02s].
## creating S4 object  ... done [0.03s].
```

```r
inspect(sort(rule2, by = "lift", decreasing=TRUE)[1:10])
```

```
##      lhs                           rhs                       support confidence  coverage   lift count
## [1]  {bottled beer,
##       red/blush wine}         => {liquor}                  0.0019319    0.39583 0.0048805 35.716    19
## [2]  {hamburger meat,
##       soda}                   => {Instant food products} 0.0012201    0.21053 0.0057956 26.209    12
## [3]  {ham,
##       white bread}            => {processed cheese}      0.0019319    0.38000 0.0050839 22.928    19
## [4]  {other vegetables,
##       root vegetables,
##       whole milk,
##       yogurt}                 => {rice}                  0.0013218    0.16883 0.0078292 22.139    13
## [5]  {bottled beer,
##       liquor}                 => {red/blush wine}        0.0019319    0.41304 0.0046772 21.494    19
## [6]  {Instant food products,
##       soda}                   => {hamburger meat}        0.0012201    0.63158 0.0019319 18.996    12
## [7]  {curd,
##       sugar}                  => {flour}                 0.0011185    0.32353 0.0034570 18.608    11
## [8]  {salty snack,
##       soda}                   => {popcorn}               0.0012201    0.13043 0.0093543 18.068    12
## [9]  {baking powder,
##       sugar}                  => {flour}                 0.0010168    0.31250 0.0032537 17.973    10
## [10] {processed cheese,
##       white bread}            => {ham}                   0.0019319    0.46341 0.0041688 17.803    19
```

It can be seen from the results that bottled beer, red/blush wine and liquor have the highest correlation.