STA 380 Exercises

Grant Zhong

8/18/2019

```
rm(list=ls())
setwd("~/Desktop/MSBA/Predictive Modeling/STA380 HW")
```

This is the R Markdown file for STA 380 Exercises for Grant Zhong, Abhinav Singh, Arjun Rao, and Thiru Vinayagam.

```
rm(list=ls())
```

Green Buildings

###First we take a look at data production of green houses to determine how to proceed with data.

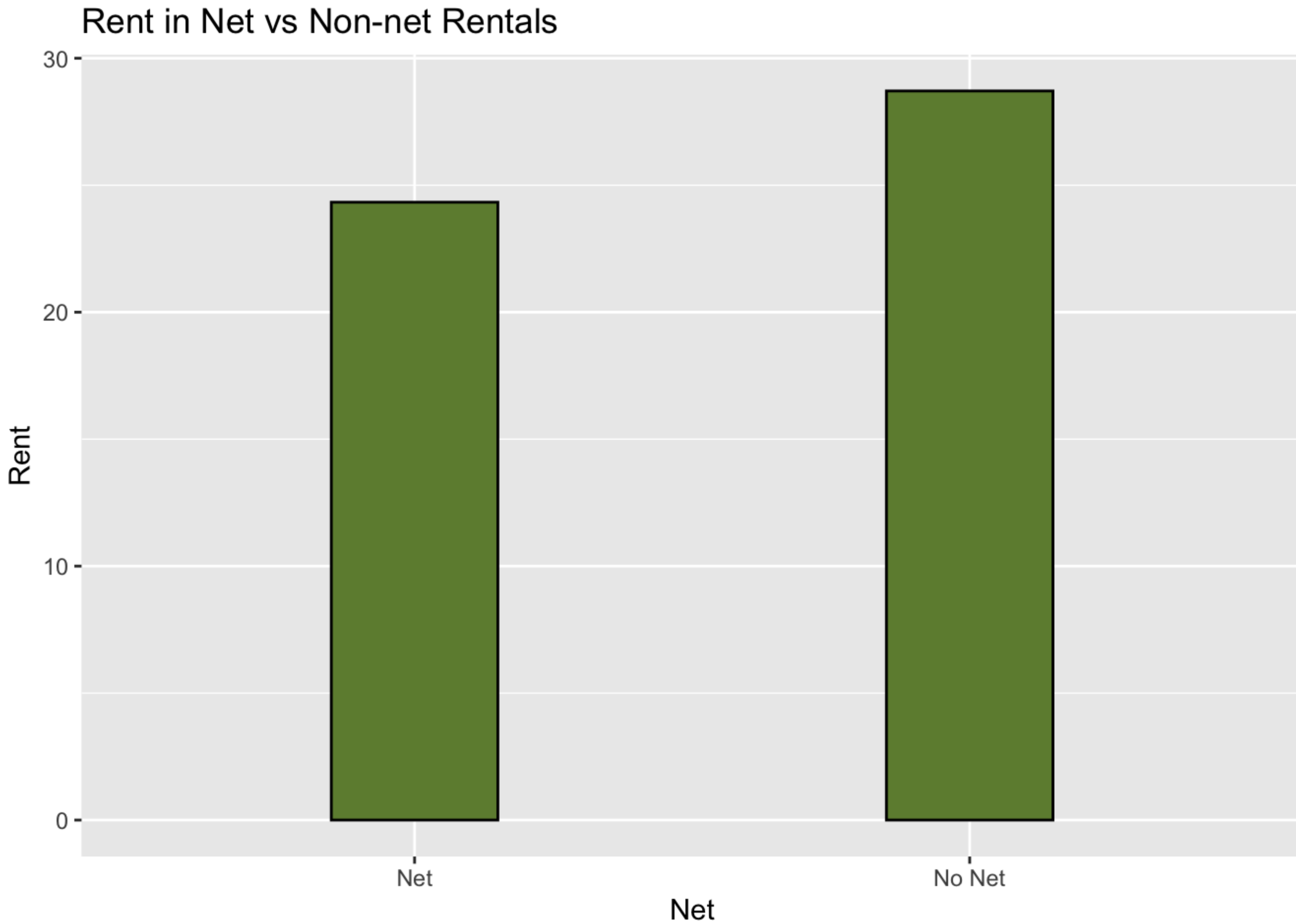
The developer is right, she should have second thoughts. The moment you start thinking about housing in real life, you realize random data from the whole country in not representative of a single city, so this analysis is useless, and the realtor has a poor business sense to use it to direct her decision. That being said, we get started by graphing the density of green and non-green building production over time.

We see that the market really started its green house production about 30 years ago, and in general, green houses follow the market, with the decline around 25 years ago, but more recently, it seem people have been producing green houses. We will proceed by looking only looking at homes produced in the past 30 years. Technology changes with time and older buildings don't really represent our target time period.

Also, we agree with the excel guru that buildings with <10% occupancy are outliers, so we removed those. Finally, our building is a 15 story multiplex, so it is not helpful to compare it to small apartments or giant skyscrapers, as there are differences in building design and energy uses, so we limited our comparisons to 15 +/- 2 stories.

###Green and non-green buildings follow similar trends. Shows that green and not green buildings tend to follow similar market trends. Before splitting into green and not green, we wanted to see if there was a difference in they were in the Net or No-net group. It would silly to compare models for people who pay their utilities on their own vs through rent without taking that into consideration.

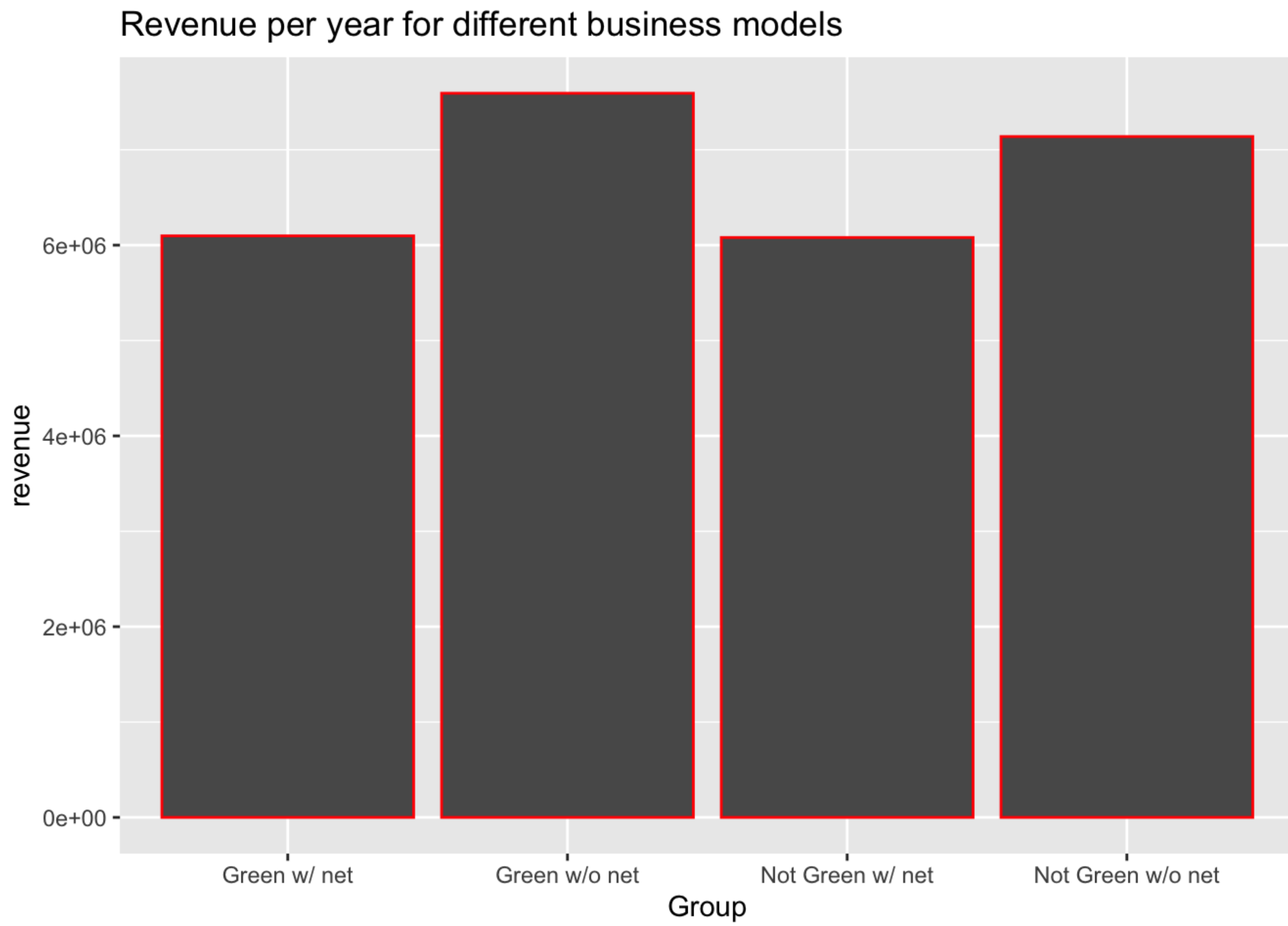
```
#checking net vs no net ==> $8/sq feet
rep_set_net= subset(rep_set, net==1 )
rep_set_nonet =subset(rep_set, net==0 )
#summary(rep_set_net$Rent)
#summary(rep_set_nonet$Rent)
q= mean(rep_set_net$Rent)
w= mean(rep_set_nonet$Rent)
renters <- data.frame("Net" = c("Net", "No Net"),
                      "Rent" = c(q,w))
p <-ggplot(data=renters, aes(x=Net, y=Rent), label = Rent) +
  geom_bar(stat="identity", color= "black", fill= "darkolivegreen4", width= .3)+ labs(ti
tle="Rent in Net vs Non-net Rentals")
p
```



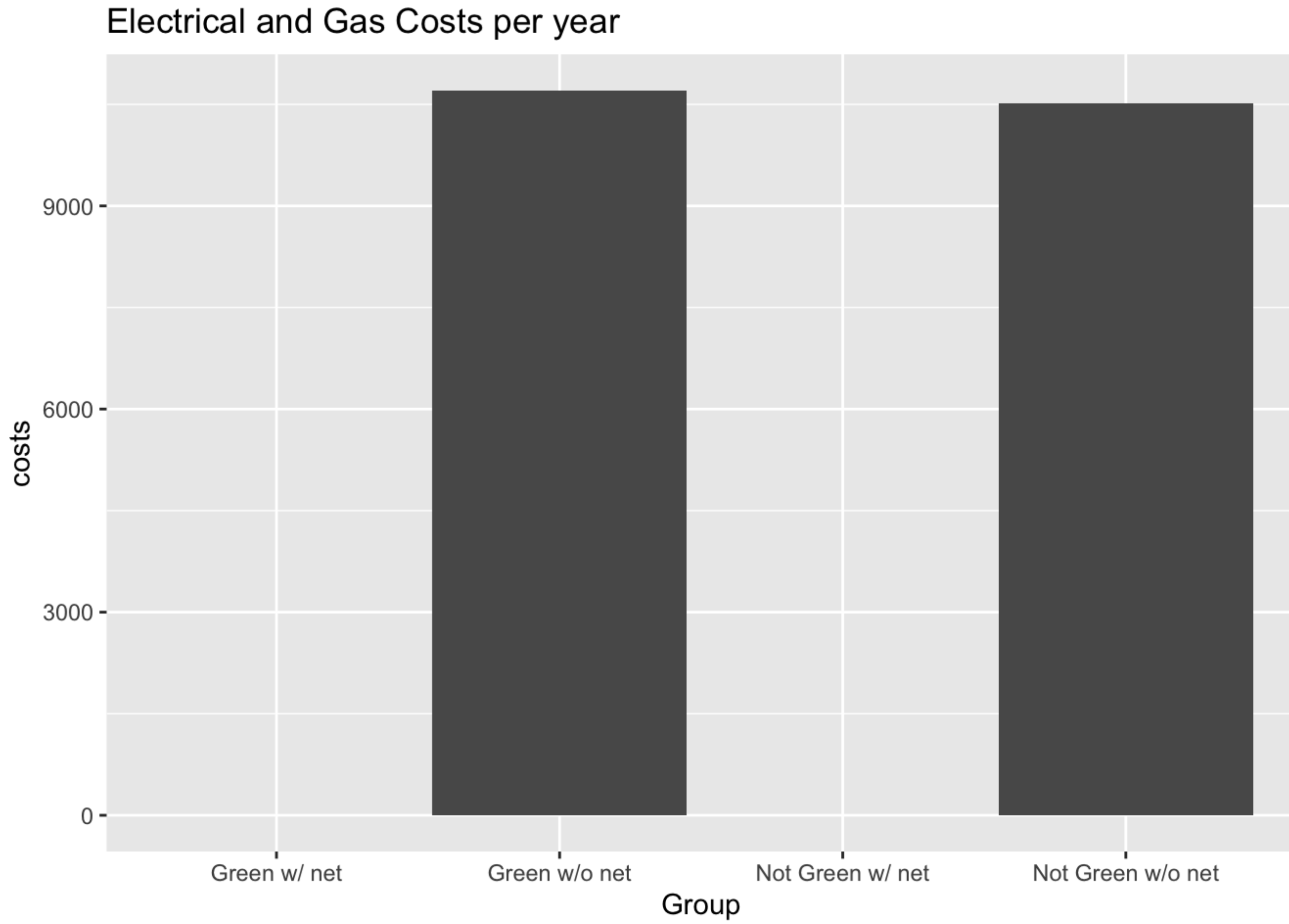
###Just as expected As expected, seems that there is a difference in means and medians for net vs no net prices. Now that we have identified our sample, we proceeded to split into green and non-green buildings.

Before, the excel guru used .9 occupancy, but we decided to calculate the occupancy based on the similar housing opetions to explore the preference for green and non-green houses.

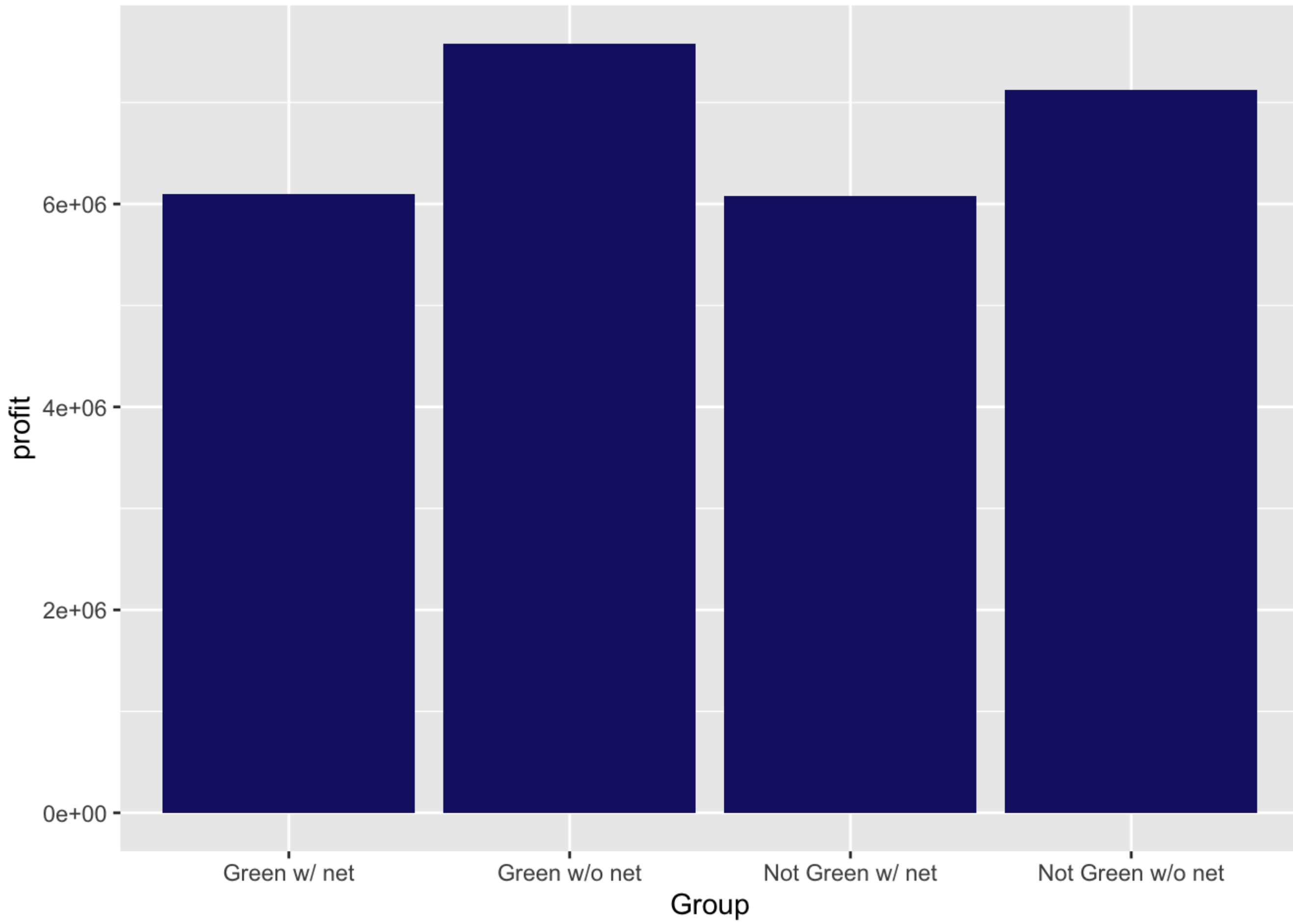
###Occupancy rates range from 83.7 to 89.5%. While similar, it can 6% can make a difference. Now we will understand the revenue.



From this we see that green properties that offer to pay the rent for the client can make ~$450000 more per year in revenue. But rembember, revenue is not profit, we have to look at the additional costs of actually paying the electricity. Now lets look at cost of electricity and gas.



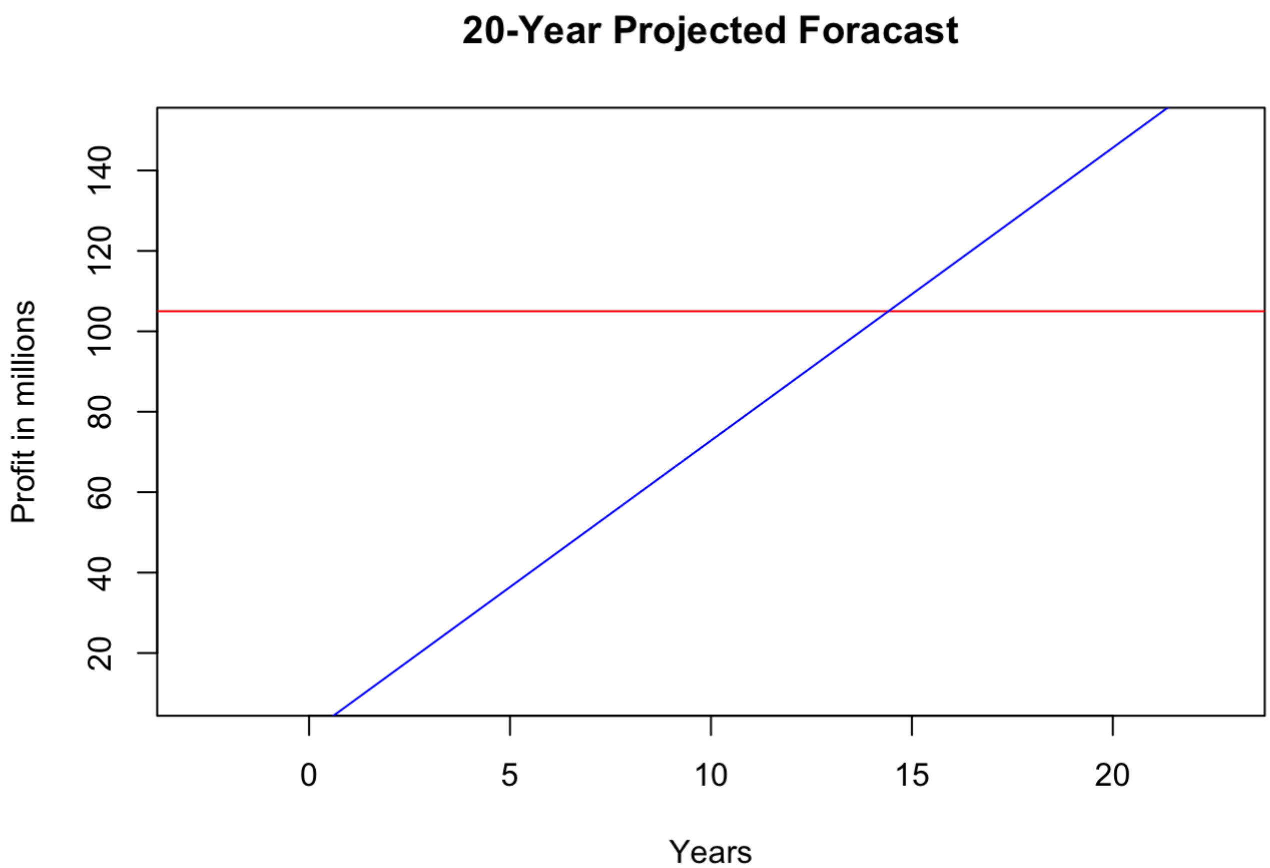
Haha, the cost of electricity is barelely $10,000 per year for a 250000 sqft building. Thats around a thousanth of the rent cost, but customers are willing to pay. That means the profits are barely changed:



Based on this we'd suggest going green but offering a no net policy, because rich people are willing to pay.

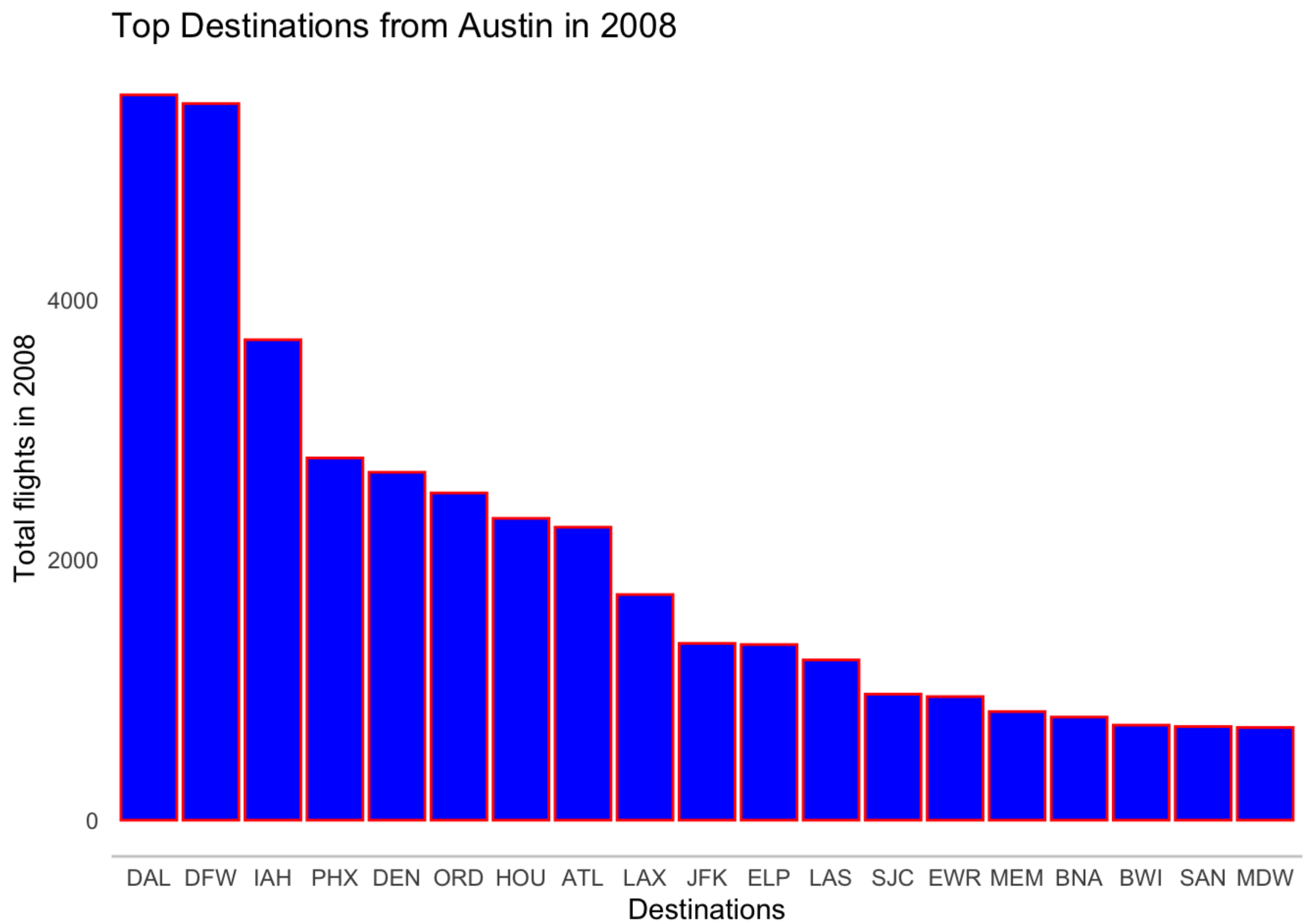
By investing in the green, no net policy business model, we would make 8003738 dollars per year, it would take us 14.4 years to repay the entire 105 million amount, or 6.5 years to repay the 5 million dollar using just the bonus.

With the next best model non-green, no net, it would take 15.6 years to pay the 100 million investment.

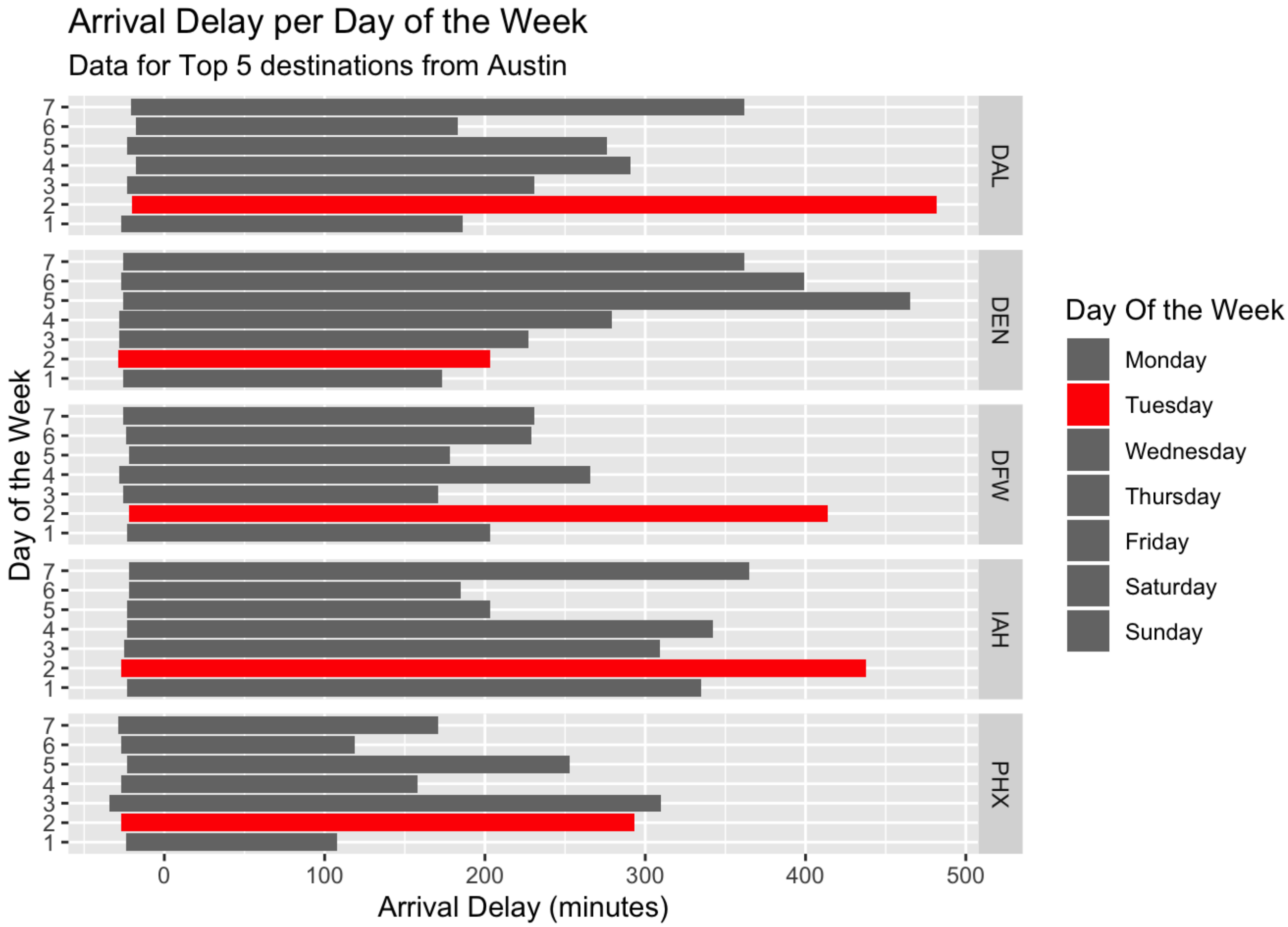


Flights at ABIA

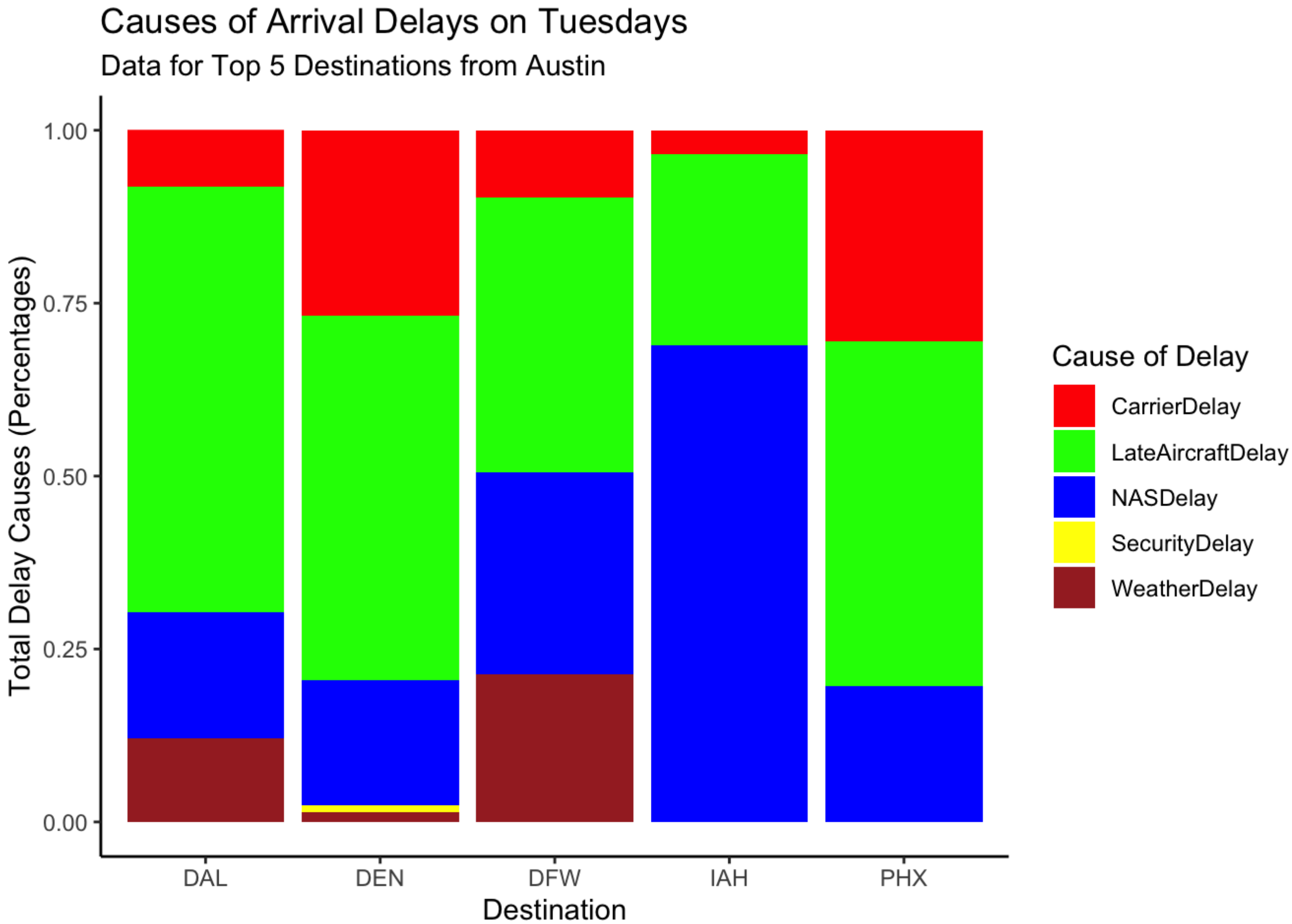
We are going to try to find the day with the most arrival delays for flights from Austin Airport. First we will find the Top 5 Destinations from Austin for 2008.



For each of these 5 destinations we will find the arrival delay per day for each different destination.



The majority of delays are on Tuesday for 3/5 destinations. Since this is a surprising insight lets find out why Tuesdays have the most delays for all 5 of the destinations.



Most delays at these airports are because of late aircraft. If you are flying from Austin to DAL, DFW, or IAH (all in Texas) don't fly on a Tuesday if you don't want to risk being late.

Portfolio Modeling

```
library(mosaic)
```

```
## Loading required package: lattice
```

```
## Loading required package: ggformula
```

```
## Loading required package: ggstance
```

```
##
## Attaching package: 'ggstance'
```

```
## The following objects are masked from 'package:ggplot2':
##
##     geom_errorbarh, GeomErrorbarh
```

```
##
## New to ggformula?  Try the tutorials:
##  learnr::run_tutorial("introduction", package = "ggformula")
##  learnr::run_tutorial("refining", package = "ggformula")
```

```
## Loading required package: mosaicData
```

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

```
##
## Attaching package: 'Matrix'
```

```
## The following object is masked from 'package:tidyr':
##
##     expand
```

```
## Registered S3 method overwritten by 'mosaic':
##   method                           from
##   fortify.SpatialPolygonsDataFrame ggplot2
```

```
##
## The 'mosaic' package masks several functions from core packages in order to add
## additional features.  The original behavior of these functions should not be affected
by this.
##
## Note: If you use the Matrix package, be sure to load it BEFORE loading mosaic.
```

```
##
## Attaching package: 'mosaic'
```

```
## The following object is masked from 'package:Matrix':
##
##     mean
```

```
## The following object is masked from 'package:plyr':
##
##     count
```

```
## The following object is masked from 'package:tseries':
##
##     value
```

```
## The following objects are masked from 'package:dplyr':
##
##     count, do, tally
```

```
## The following object is masked from 'package:purrr':
##
##     cross
```

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

```
## The following objects are masked from 'package:stats':
##
##     binom.test, cor, cor.test, cov, fivenum, IQR, median,
##     prop.test, quantile, sd, t.test, var
```

```
## The following objects are masked from 'package:base':
##
##     max, mean, min, prod, range, sample, sum
```

```
library(quantmod)
```

```
## Loading required package: TTR
```

```
## Version 0.4-0 included new data defaults. See ?getSymbols.
```

```
library(foreach)
```

```
##
## Attaching package: 'foreach'
```

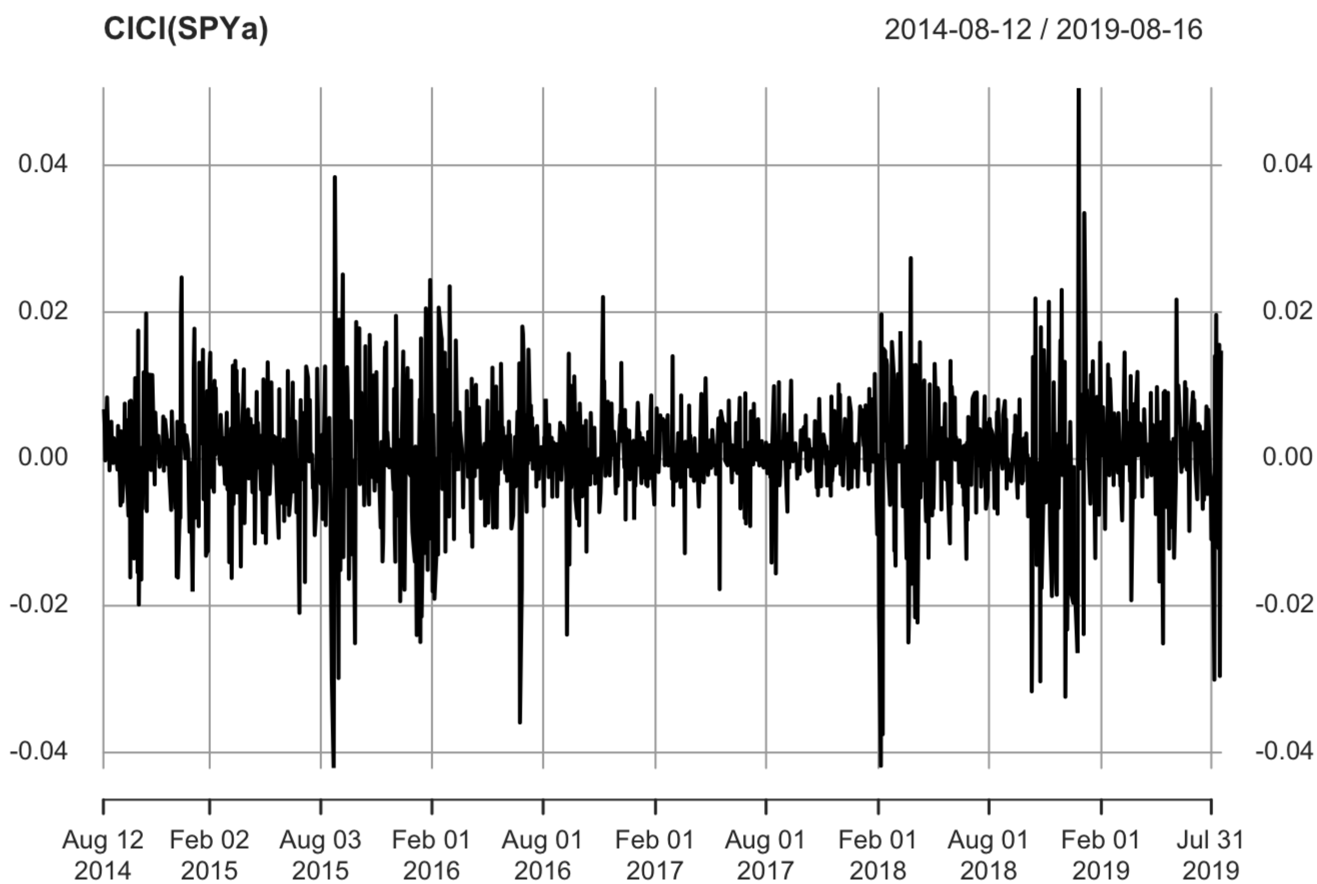
```
## The following objects are masked from 'package:purrr':
##
##     accumulate, when
```

```
myETFs = c('SPY','TQQQ','FSLR')
getSymbols(myETFs, from="2014-08-12")
```

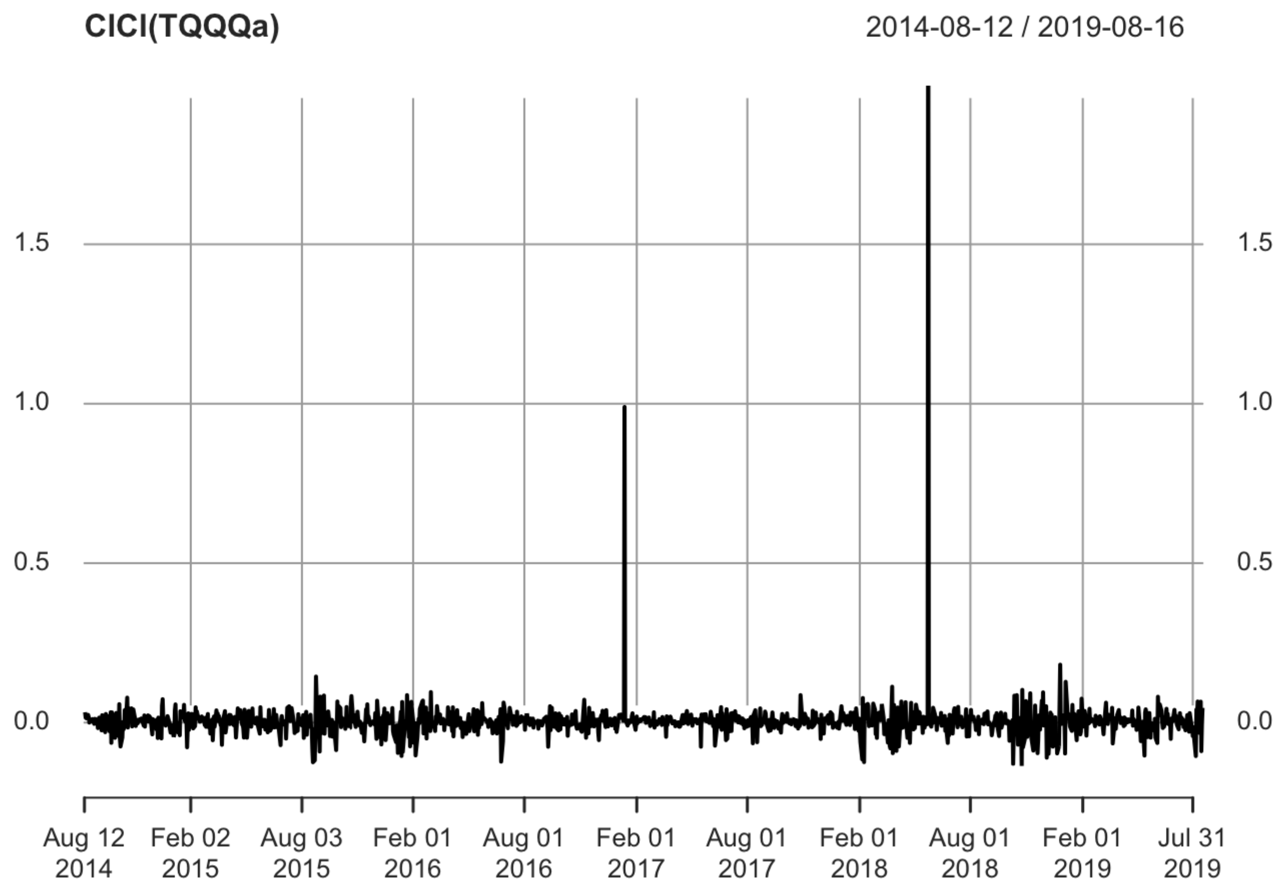
```
## 'getSymbols' currently uses auto.assign=TRUE by default, but will
## use auto.assign=FALSE in 0.5-0. You will still be able to use
## 'loadSymbols' to automatically load data. getOption("getSymbols.env")
## and getOption("getSymbols.auto.assign") will still be checked for
## alternate defaults.
##
## This message is shown once per session and may be disabled by setting
## options("getSymbols.warning4.0"=FALSE). See ?getSymbols for details.
```

```
## [1] "SPY"  "TQQQ" "FSLR"
```

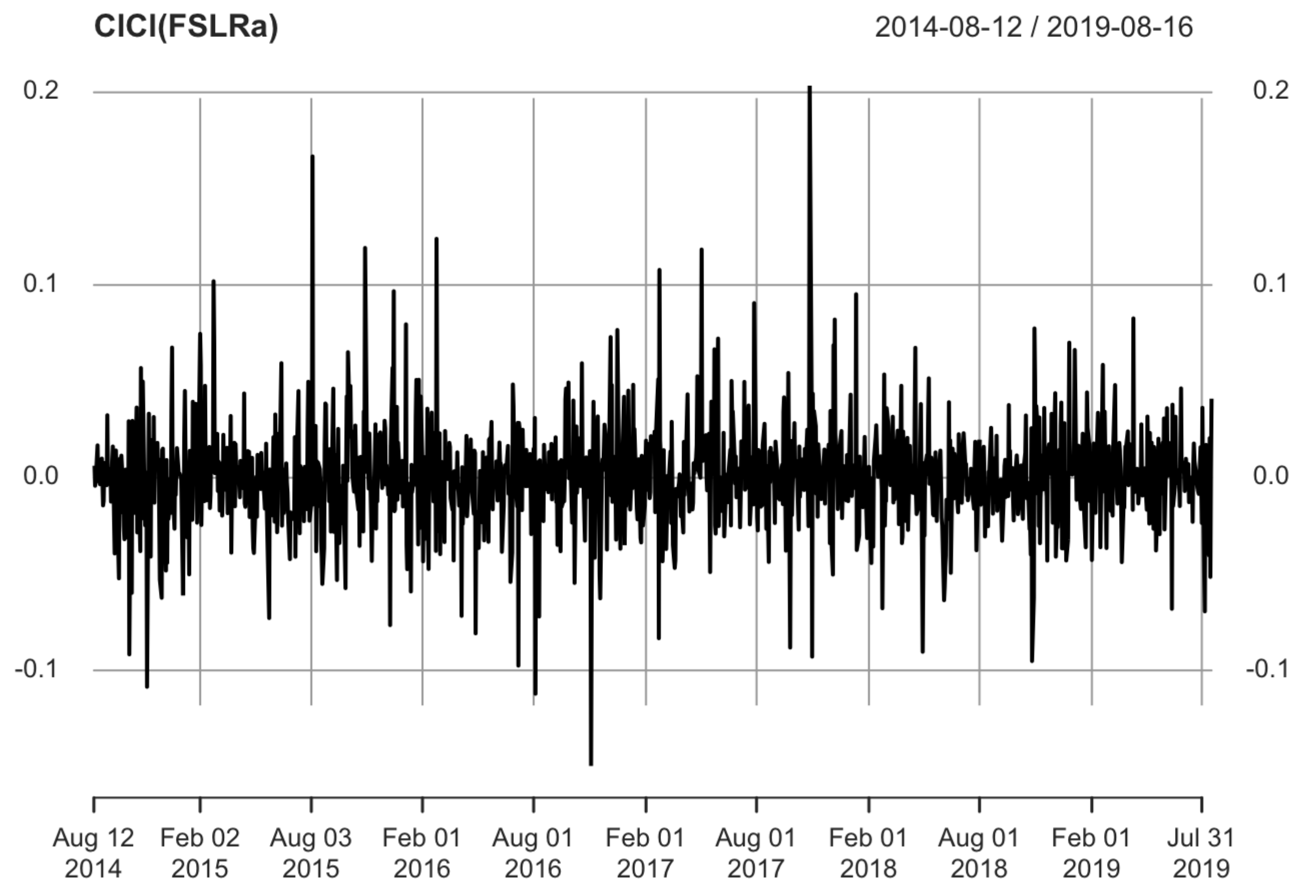
```
# Adjust for splits and dividends
SPYa = adjustOHLC(SPY)
TQQQa = adjustOHLC(TQQQ)
FSLRa = adjustOHLC(FSLR)


# Look at close-to-close changes
plot(ClCl(SPYa))
```




```
plot(ClCl(TQQQa))
```




```
plot(ClCl(FSLRa))
```




```
# Combine close to close changes in a single matrix
all_returns = cbind(ClCl(SPYa),ClCl(TQQQa),ClCl(FSLRa))
head(all_returns)
```

```
##                ClCl.SPYa ClCl.TQQQa   ClCl.FSLRa
## 2014-08-12            NA         NA           NA
## 2014-08-13  0.0067689609 0.03249432  0.006225568
## 2014-08-14  0.0047218180 0.01542475 -0.004172676
## 2014-08-15 -0.0002043012 0.01359794 -0.001300332
## 2014-08-18  0.0083793173 0.02417208  0.016782349
## 2014-08-19  0.0052188792 0.01722922 -0.001707385
```

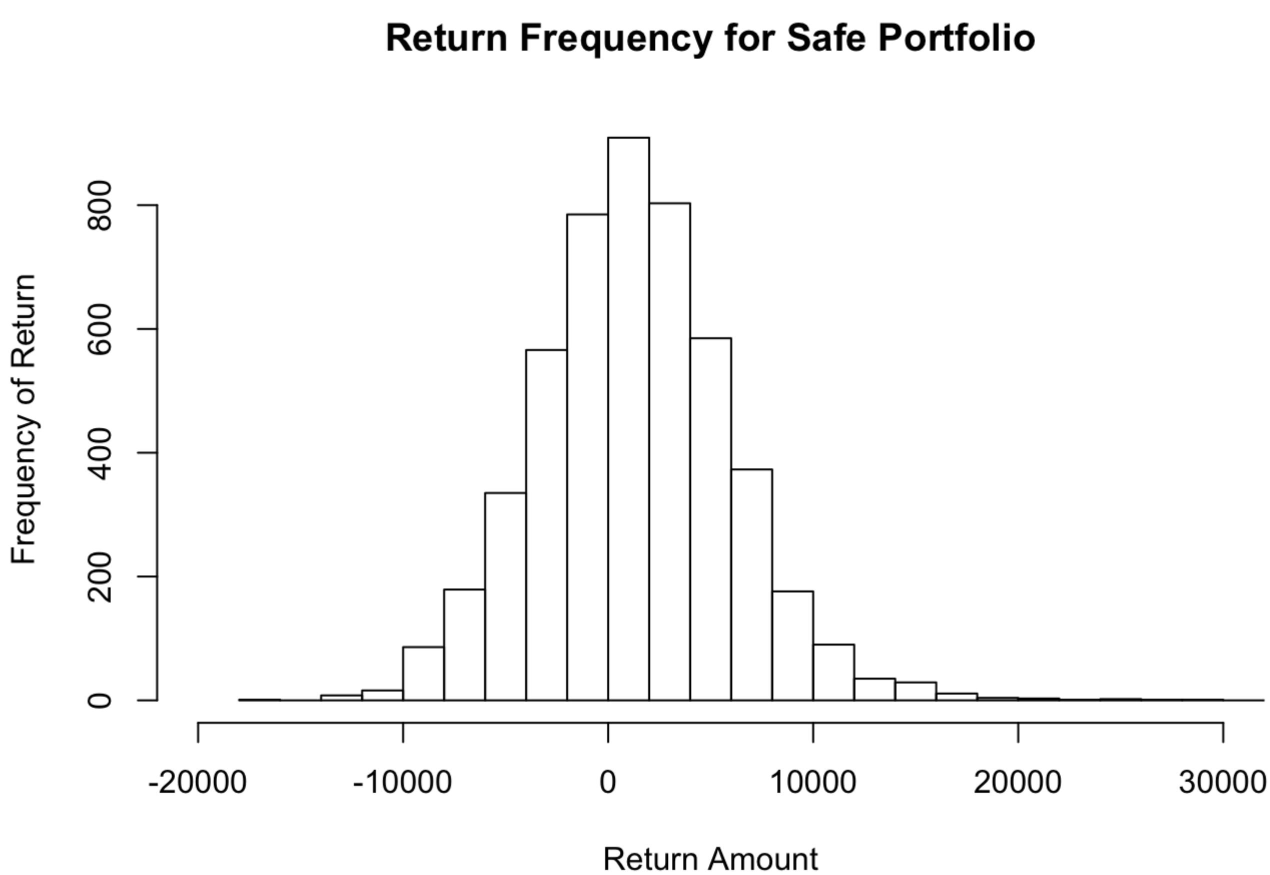
```
all_returns = as.matrix(na.omit(all_returns))
N = nrow(all_returns)

#Calculate volatility of ETFs
sigma_SPY = sd(all_returns[,1])
sigma_TQQQ = sd(all_returns[,2])
sigma_FSLR = sd(all_returns[,3])
```

The standard deviation for SPY is the lowest at 0.00848 so SPY is considered our safe ETF. The standard deviation for TQQQ and FSLR are 0.0705 and 0.02747 respectively which will make these two our more volatile and aggressive ETFs.

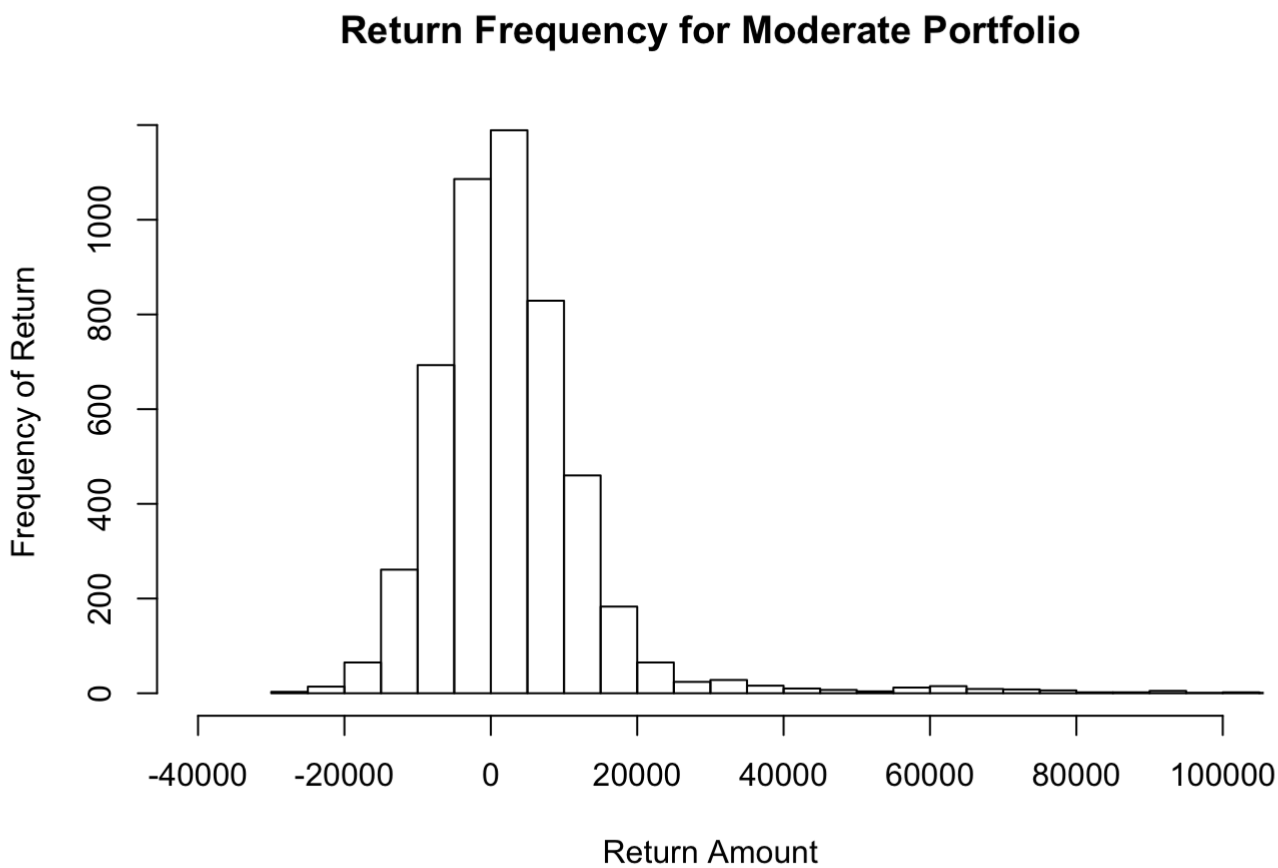
```
# Now simulate 3 scenarios over four trading weeks

#Sim1 is safe, most in SPY

initial_wealth = 100000

sim1 = foreach(i=1:5000, .combine='rbind') %do% {
    total_wealth = initial_wealth
    weights = c(0.90,0.05,0.05)
    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
}

#Sim2 is moderate, split between SPY and the high risk ETFs

initial_wealth = 100000

sim2 = foreach(i=1:5000, .combine='rbind') %do% {
    total_wealth = initial_wealth
    weights = c(0.40, 0.30, 0.30)
    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
}

#Sim3 is aggressive, almost none in SPY and all in the high risk ETFs

initial_wealth = 100000
sim3 = foreach(i=1:5000, .combine='rbind') %do% {
    total_wealth = initial_wealth
    weights = c(0.10,0.30,0.60)
    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
}
```

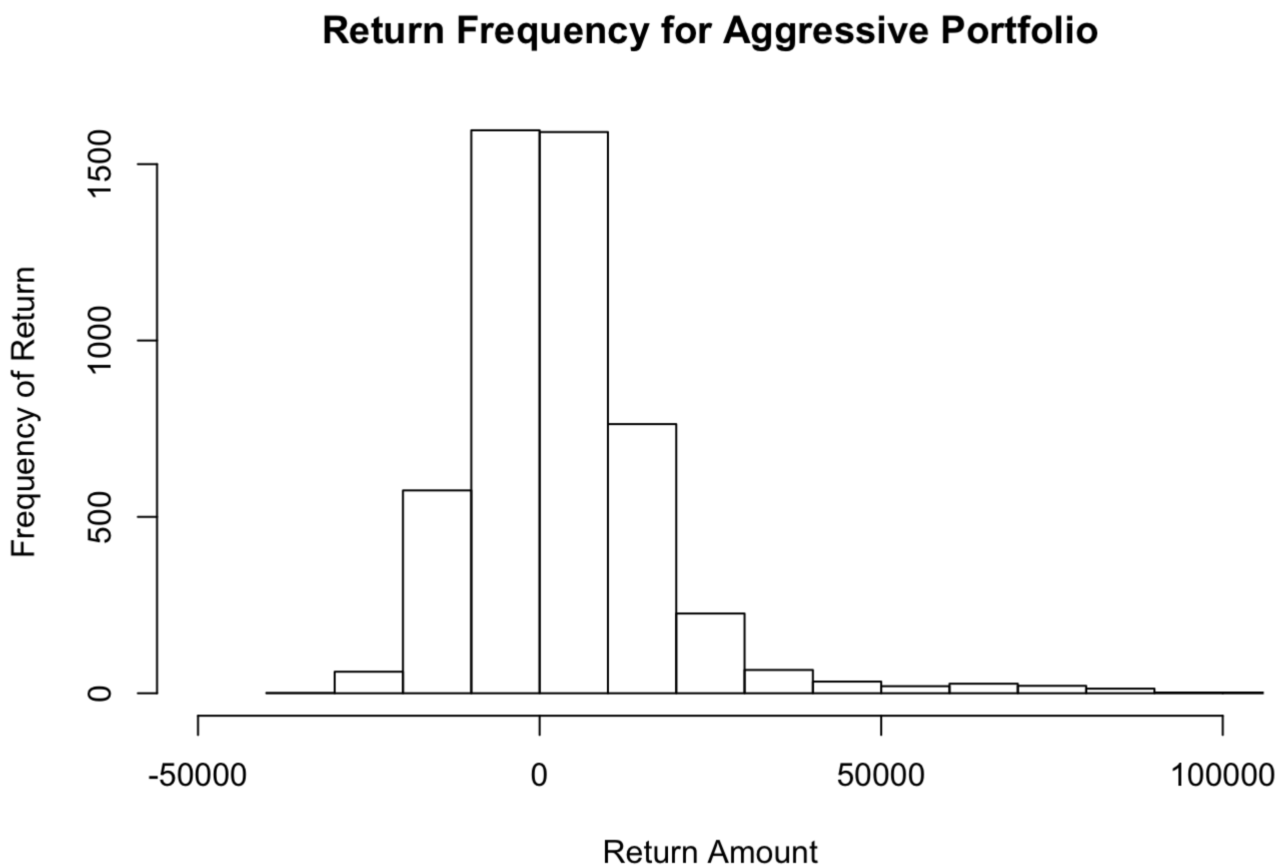
```
# Average Returns
avg_value_sim1 = mean(sim1[,n_days])
avg_value_sim2 = mean(sim2[,n_days])
avg_value_sim3 = mean(sim3[,n_days])

avg_return_sim1 = (avg_value_sim1-100000)/1000 #percentage
avg_return_sim2 = (avg_value_sim2-100000)/1000 #percentage
avg_return_sim3 = (avg_value_sim3-100000)/1000 #percentage

returns_sim1 = sim1[,n_days]-100000
returns_sim2 = sim2[,n_days]-100000
returns_sim3 = sim3[,n_days]-100000

#Plot
hist(returns_sim1, breaks=30,xlim = c(-20000,30000),
     xlab='Return Amount',ylab='Frequency of Return', main="Return Frequency for Safe Po
rtfolio")
```




```
hist(returns_sim2, breaks=30,xlim = c(-40000,100000), xlab='Return Amount', ylab='Freque
ncy of Return', main="Return Frequency for Moderate Portfolio")
```




```
hist(returns_sim3, breaks=30,xlim = c(-50000,100000), xlab='Return Amount', ylab='Freque
ncy of Return', main="Return Frequency for Aggressive Portfolio")
```



From the plots of our returns from different portfolios, I can see that our safe portfolio has a more normal distribution of returns. Our moderate portfolio has more frequent days of high negative returns but also more days of high positive returns. Our aggressive portfolio has similar patterns with our moderate portfolio (frequent, negative day but very high return days as well).

Our safe portfolio does not have any days with returns higher than 30000 while our moderate and aggressive portfolios both have returns of almost 100000.

```
#Calculate VaR

sim1_P= quantile(sim1[,n_days], 0.05)
sim2_P = quantile(sim2[,n_days], 0.05)
sim3_P = quantile(sim3[,n_days], 0.05)

p0 = 100000

var_sim1 = p0-sim1_P
var_sim2 = p0-sim2_P
var_sim3 = p0-sim3_P
cat('The VaR for our safe portfolio is',var_sim1,'\n')
```

```
## The VaR for our safe portfolio is 6214.373
```

```
cat('The VaR for our moderate portfolio is',var_sim2,'\n')
```

```
## The VaR for our moderate portfolio is 11153.22
```

```
cat('The VaR for our aggressive portfolio is',var_sim3,'\n')
```

```
## The VaR for our aggressive portfolio is 14665.64
```

For our safe portfolio with a 5% confidence, the Value at Risk (VaR) is 6092. For our moderate portfolio, the VaR is 11517. Finally, for our aggressive portfolio, the VaR is 14832.

Market Segmentation

```
library(tidyverse)
library(cluster)
library(factoextra)
```

```
## Welcome! Related Books: `Practical Guide To Cluster Analysis in R` at https://goo.gl/
13EFCZ
```

```
library(NbClust)
library(corrplot)
```

```
## corrplot 0.84 loaded
```

```
library(gridExtra)
```

```
social_marketing <- read_csv("social_marketing.csv")
```

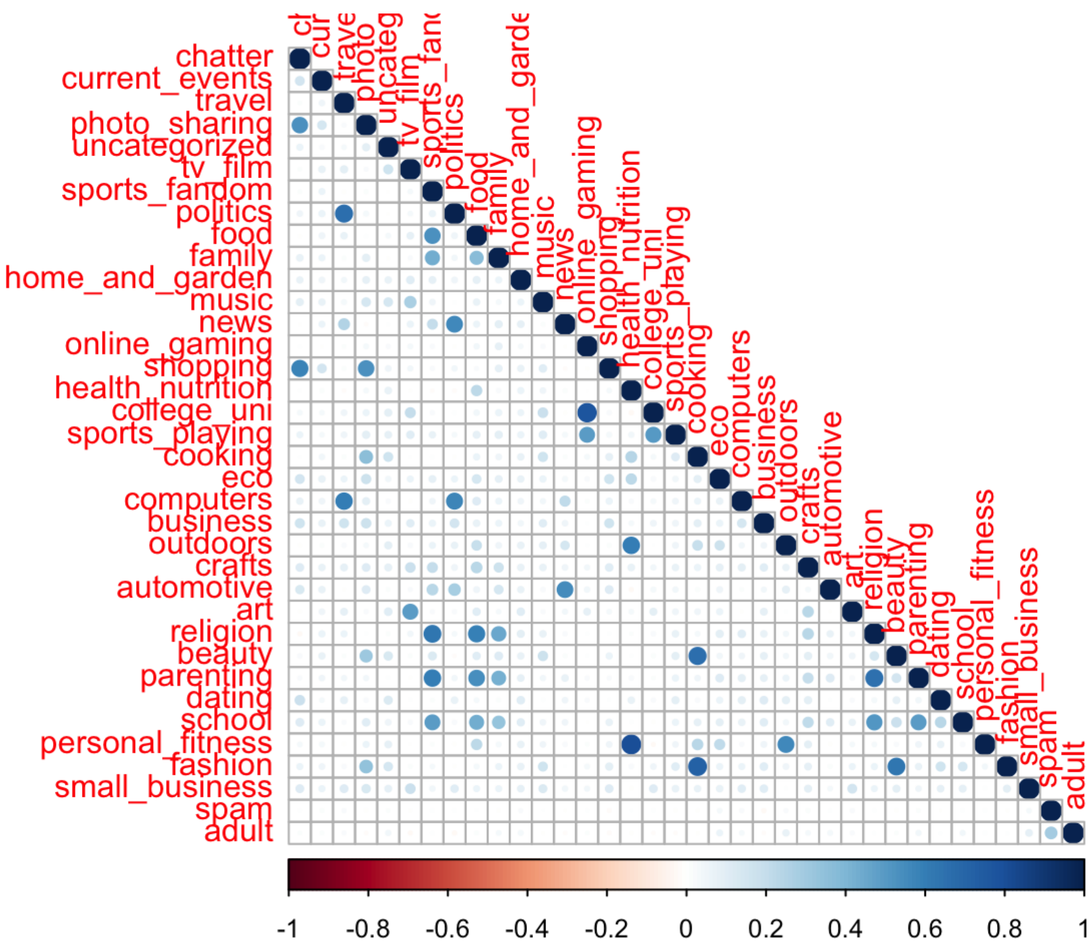
```
## Parsed with column specification:
## cols(
##   .default = col_double(),
##   X1 = col_character()
## )
```

```
## See spec(...) for full column specifications.
```

```
head(social_marketing)
```

```
## # A tibble: 6 x 37
##   X1     chatter current_events travel photo_sharing uncategorized tv_film
##   <chr>    <dbl>          <dbl>  <dbl>         <dbl>         <dbl>   <dbl>
## 1 hmjo…        2              0      2             2             2       1
## 2 clk1…        3              3      2             1             1       1
## 3 jcso…        6              3      4             3             1       5
## 4 3oeb…        1              5      2             2             0       1
## 5 fd75…        5              2      0             6             1       0
## 6 h6nv…        6              4      2             7             0       1
## # … with 30 more variables: sports_fandom <dbl>, politics <dbl>,
## #   food <dbl>, family <dbl>, home_and_garden <dbl>, music <dbl>,
## #   news <dbl>, online_gaming <dbl>, shopping <dbl>,
## #   health_nutrition <dbl>, college_uni <dbl>, sports_playing <dbl>,
## #   cooking <dbl>, eco <dbl>, computers <dbl>, business <dbl>,
## #   outdoors <dbl>, crafts <dbl>, automotive <dbl>, art <dbl>,
## #   religion <dbl>, beauty <dbl>, parenting <dbl>, dating <dbl>,
## #   school <dbl>, personal_fitness <dbl>, fashion <dbl>,
## #   small_business <dbl>, spam <dbl>, adult <dbl>
```

```
cormat <- cor(social_marketing[c(2:37)])
corrplot(cormat, method = 'circle', type = 'lower')
```



By

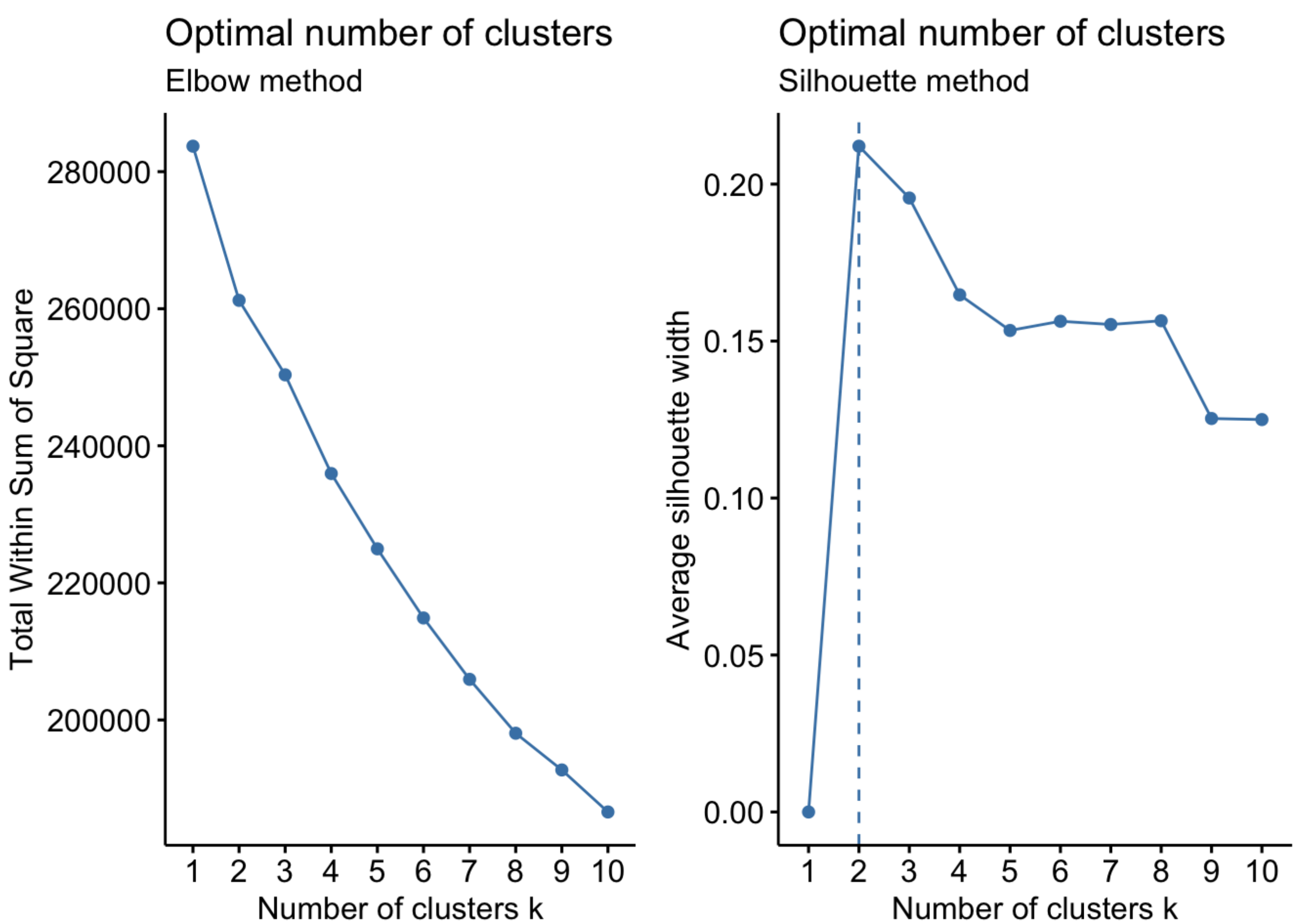
analyzing relationships between quantitative variables, we see that photo_sharing & chatter, chatter & shopping, politics & travel, computers & travel, personal fitness & health/nutrition, and more have very strong correlations.

I will use clustering to answer this question.

```
scaled_data <- scale(social_marketing[,2:37], center=TRUE, scale=TRUE)

# Extract the centers and scales from the rescaled data (which are named attributes)
cent = attr(scaled_data,"scaled:center")
scale = attr(scaled_data,"scaled:scale")
```

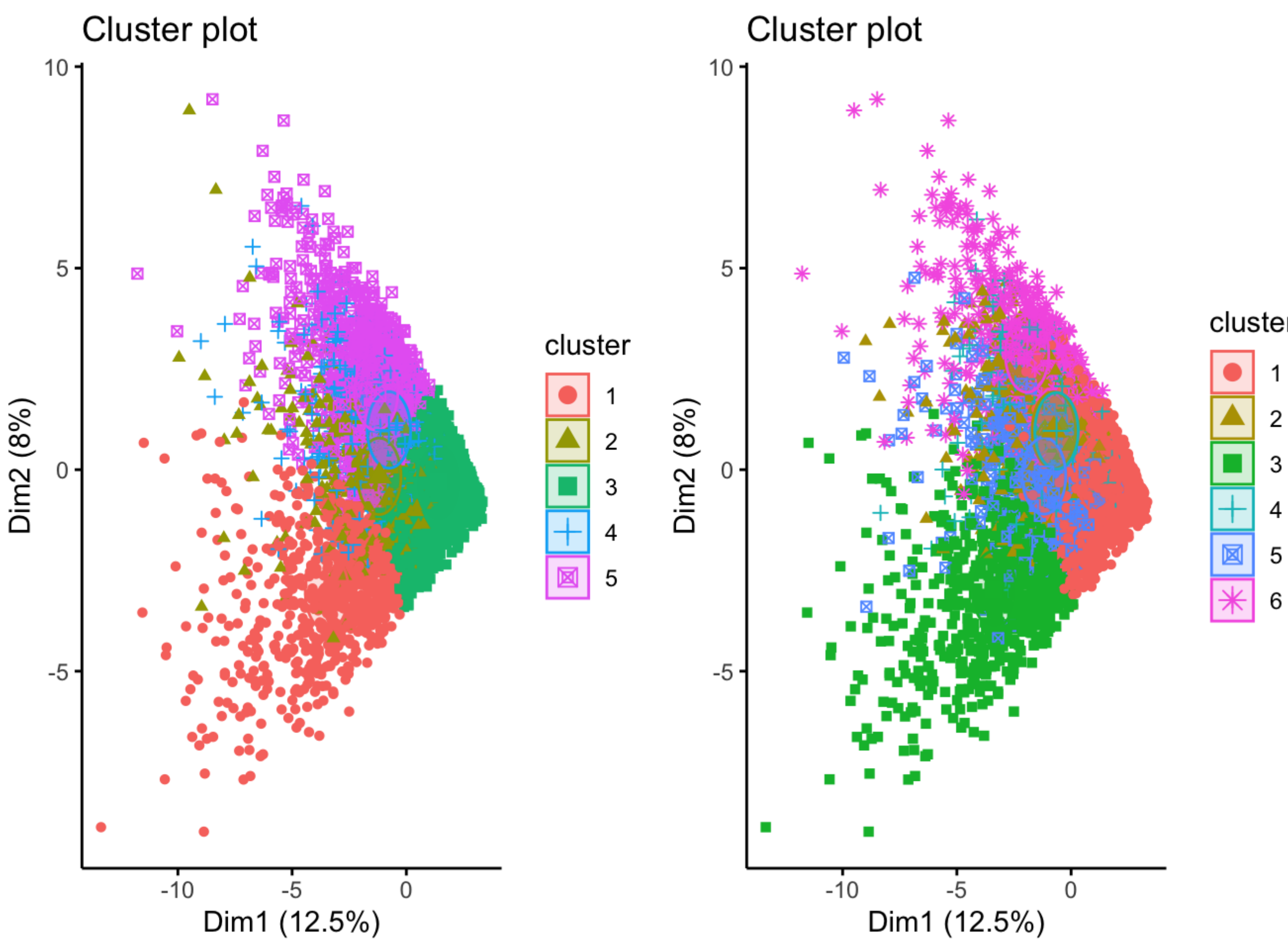
Now that the values have been standardized and scaled, we can do a elbow and silhouette plot to determine optimal number of clusters

```
# Elbow curve
wss = fviz_nbclust(scaled_data, kmeans, method = "wss") +
  labs(subtitle = "Elbow method")

# Silhouette curve
sil = fviz_nbclust(scaled_data, kmeans, method = "silhouette")+
  labs(subtitle = "Silhouette method")

grid.arrange(wss,sil,ncol = 2)
```



Let's try 5 clusters as it's in the middle between too few and too many.

```
set.seed(1)
clust5 = kmeans(scaled_data, 5, nstart=25)

# Visualizing the clusters on a dimensionally reduced plot
clus5plot = fviz_cluster(clust5, data = scaled_data,
             ellipse.type = "euclid", # Concentration ellipse
             ggtheme = theme_classic(),geom = c("point")
             )


# 6 clusters
set.seed(1)
clust6 = kmeans(scaled_data, 6, nstart=25)
# Visualizing the clusters on a dimensionally reduced plot
clus6plot = fviz_cluster(clust6, data = scaled_data,
             ellipse.type = "euclid", # Concentration ellipse
             ggtheme = theme_classic(),geom = c("point")
             )
grid.arrange(clus5plot,clus6plot,ncol = 2)
```



I will use 5 clusters as it strikes a balance between complexity & interpretability.

```
res = aggregate(scaled_data, by=list(cluster=clust5$cluster), mean)
res
```

```
##   cluster     chatter current_events      travel photo_sharing
## 1       1 -0.08580333     0.11668616 -0.09868363   -0.06461891
## 2       2 -0.00641441     0.10280954  1.84812814   -0.10919601
## 3       3 -0.21208342    -0.12931452 -0.23425089   -0.31370653
## 4       4 -0.08693446    -0.01073276 -0.15487410   -0.05184131
## 5       5  0.71904694     0.27525001 -0.03719217    1.03137406
##   uncategorized      tv_film sports_fandom   politics        food
## 1   -0.07741713 -0.004897907     2.0495745 -0.1991008  1.81964863
## 2   -0.04577894  0.030509586     0.1906405  2.4332797  0.03503888
## 3   -0.16642873 -0.149302014    -0.2976276 -0.2725943 -0.36615240
## 4    0.14824172 -0.104043474    -0.1924418 -0.1809244  0.43514619
## 5    0.45687479  0.487125072    -0.1645367 -0.1316750 -0.15454924
##        family home_and_garden       music        news online_gaming
## 1  1.49474377       0.1700386  0.04899008 -0.06695928    0.02537588
## 2  0.04903569       0.1238786 -0.04942746  1.93197154   -0.07159622
## 3 -0.27111964      -0.1709065 -0.20150490 -0.24798779   -0.12675943
## 4 -0.07567456       0.1404359  0.02124261 -0.03108450   -0.06273584
## 5  0.04054805       0.2671438  0.57276999 -0.12812762    0.42839988
##        shopping health_nutrition college_uni sports_playing    cooking
## 1 -0.0008460327       -0.1547148 -0.03247132     0.14546463 -0.0873412
## 2 -0.0560072470       -0.2028445 -0.02686707     0.02868869 -0.1926886
## 3 -0.2412300828       -0.3294894 -0.15731291    -0.20680092 -0.3403695
## 4 -0.0061958076        2.1570207 -0.15911271     0.01875251  0.4334867
## 5  0.7342329914       -0.1759325  0.58515190     0.50321224  0.8647706
##           eco   computers    business   outdoors      crafts  automotive
## 1  0.17186410  0.07640434  0.12061881 -0.0694683  0.70519032  0.17201565
## 2  0.09647687  1.64017370  0.32211525  0.1056471  0.10117989  1.06737413
## 3 -0.23646021 -0.25259442 -0.21005609 -0.3155585 -0.25768553 -0.20654128
## 4  0.53055728 -0.08156345  0.02785455  1.6603254  0.08330701 -0.12871383
## 5  0.23294281 -0.02144859  0.38254925 -0.1030419  0.28878012  0.09177554
##           art   religion     beauty   parenting     dating      school
## 1  0.10667355  2.2557923  0.3126127  2.12259872 -0.0114005  1.65800466
## 2 -0.05736018 -0.0303389 -0.1533685  0.02137552  0.2049316 -0.04566825
## 3 -0.15781693 -0.3041857 -0.2907738 -0.31173772 -0.1702148 -0.30577179
## 4 -0.01663923 -0.1668792 -0.1692076 -0.10510437  0.1615793 -0.17498576
## 5  0.44222466 -0.1641339  0.8613583 -0.13498888  0.3081778  0.16195230
##   personal_fitness     fashion small_business         spam        adult
## 1      -0.09828099  0.02774146     0.09424725 -0.012787316  0.017541974
## 2      -0.19645125 -0.15893806     0.22613309  0.011913412 -0.091545186
## 3      -0.34398083 -0.29466026    -0.18077167 -0.005740413 -0.005507525
## 4       2.11941253 -0.07670733    -0.13130140  0.005116119 -0.013875991
## 5      -0.14297245  0.96644602     0.45255909  0.014670356  0.058324281
```

```
#we dont want spam or clutter
results1 = res[,-c(2,36)]
results1 = as.data.frame(results1)
results1
```

```
##   cluster current_events      travel photo_sharing uncategorized
## 1       1     0.11668616 -0.09868363   -0.06461891   -0.07741713
## 2       2     0.10280954  1.84812814   -0.10919601   -0.04577894
## 3       3    -0.12931452 -0.23425089   -0.31370653   -0.16642873
## 4       4    -0.01073276 -0.15487410   -0.05184131    0.14824172
## 5       5     0.27525001 -0.03719217    1.03137406    0.45687479
##        tv_film sports_fandom   politics        food      family
## 1 -0.004897907     2.0495745 -0.1991008  1.81964863  1.49474377
## 2  0.030509586     0.1906405  2.4332797  0.03503888  0.04903569
## 3 -0.149302014    -0.2976276 -0.2725943 -0.36615240 -0.27111964
## 4 -0.104043474    -0.1924418 -0.1809244  0.43514619 -0.07567456
## 5  0.487125072    -0.1645367 -0.1316750 -0.15454924  0.04054805
##   home_and_garden       music        news online_gaming      shopping
## 1       0.1700386  0.04899008 -0.06695928    0.02537588 -0.0008460327
## 2       0.1238786 -0.04942746  1.93197154   -0.07159622 -0.0560072470
## 3      -0.1709065 -0.20150490 -0.24798779   -0.12675943 -0.2412300828
## 4       0.1404359  0.02124261 -0.03108450   -0.06273584 -0.0061958076
## 5       0.2671438  0.57276999 -0.12812762    0.42839988  0.7342329914
##   health_nutrition college_uni sports_playing    cooking         eco
## 1       -0.1547148 -0.03247132     0.14546463 -0.0873412  0.17186410
## 2       -0.2028445 -0.02686707     0.02868869 -0.1926886  0.09647687
## 3       -0.3294894 -0.15731291    -0.20680092 -0.3403695 -0.23646021
## 4        2.1570207 -0.15911271     0.01875251  0.4334867  0.53055728
## 5       -0.1759325  0.58515190     0.50321224  0.8647706  0.23294281
##     computers    business   outdoors      crafts  automotive         art
## 1  0.07640434  0.12061881 -0.0694683  0.70519032  0.17201565  0.10667355
## 2  1.64017370  0.32211525  0.1056471  0.10117989  1.06737413 -0.05736018
## 3 -0.25259442 -0.21005609 -0.3155585 -0.25768553 -0.20654128 -0.15781693
## 4 -0.08156345  0.02785455  1.6603254  0.08330701 -0.12871383 -0.01663923
## 5 -0.02144859  0.38254925 -0.1030419  0.28878012  0.09177554  0.44222466
##     religion     beauty   parenting     dating      school
## 1  2.2557923  0.3126127  2.12259872 -0.0114005  1.65800466
## 2 -0.0303389 -0.1533685  0.02137552  0.2049316 -0.04566825
## 3 -0.3041857 -0.2907738 -0.31173772 -0.1702148 -0.30577179
## 4 -0.1668792 -0.1692076 -0.10510437  0.1615793 -0.17498576
## 5 -0.1641339  0.8613583 -0.13498888  0.3081778  0.16195230
##   personal_fitness     fashion small_business        adult
## 1      -0.09828099  0.02774146     0.09424725  0.017541974
## 2      -0.19645125 -0.15893806     0.22613309 -0.091545186
## 3      -0.34398083 -0.29466026    -0.18077167 -0.005507525
## 4       2.11941253 -0.07670733    -0.13130140 -0.013875991
## 5      -0.14297245  0.96644602     0.45255909  0.058324281
```

```
transposed <- t(results1)

# get row and colnames in order
colnames(transposed) <- rownames(results1)
rownames(transposed) <- colnames(results1)

# REmoving cluster names
t_results2 = transposed[-1,]

k = colnames(t_results2)[apply(t_results2,1,which.max)]
clus_features = cbind(rownames(t_results2),k)

clus_features
```

```
##                          k
##  [1,] "current_events"   "5"
##  [2,] "travel"           "2"
##  [3,] "photo_sharing"    "5"
##  [4,] "uncategorized"    "5"
##  [5,] "tv_film"          "5"
##  [6,] "sports_fandom"    "1"
##  [7,] "politics"         "2"
##  [8,] "food"             "1"
##  [9,] "family"           "1"
## [10,] "home_and_garden"  "5"
## [11,] "music"            "5"
## [12,] "news"             "2"
## [13,] "online_gaming"    "5"
## [14,] "shopping"         "5"
## [15,] "health_nutrition" "4"
## [16,] "college_uni"      "5"
## [17,] "sports_playing"   "5"
## [18,] "cooking"          "5"
## [19,] "eco"              "4"
## [20,] "computers"        "2"
## [21,] "business"         "5"
## [22,] "outdoors"         "4"
## [23,] "crafts"           "1"
## [24,] "automotive"       "2"
## [25,] "art"              "5"
## [26,] "religion"         "1"
## [27,] "beauty"           "5"
## [28,] "parenting"        "1"
## [29,] "dating"           "5"
## [30,] "school"           "1"
## [31,] "personal_fitness" "4"
## [32,] "fashion"          "5"
## [33,] "small_business"   "5"
## [34,] "adult"            "5"
```

From the output, we can see that cluster 2 entails people who are interested in travel, news, politics, and automotive. They can be grouped as businessmen. Cluster 3 is those interested in health, eco, outdoors, personal fitness. They can be grouped as athletes. Cluster 4 is those in sports, food, family, crafts, religion, parenting, and school. They can be grouped as family-oriented. Cluster 5 is photosharing, tv, home, music, gaming, shopping, sports, cooking, business, art, beauty, dating, fashion, small business, and adult. They can be grouped as single or young adults.

Cluster 1 seems to not have any distinctive characteristics.

Author Attribution

```
library(tm)
```

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

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

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

```
##
## Attaching package: 'tm'
```

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

```
library(SnowballC)
library(plyr)

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

author_dirs = Sys.glob('C50train/*')
author_dirs_test = Sys.glob('C50test/*')
```

```r
author_list = c()
labels = c()
for(author in author_dirs) {
    author_name = substring(author,10)
    files_to_add = Sys.glob(paste0(author, '/*.txt'))
    author_list = append(author_list, files_to_add)
    labels = append(labels, rep(author_name, length(files_to_add)))
}

Trainingtext = lapply(author_list, readerPlain)
names(Trainingtext)=author_list
names(Trainingtext)=sub('.txt','',names(author_list))

my_corpus = VCorpus(VectorSource(Trainingtext))
names(my_corpus) = labels
```

```r
author_list_test = c()
labels_test = c()
for(author in author_dirs_test) {
    author_name_test = substring(author,9)
    files_to_add_test = Sys.glob(paste0(author, '/*.txt'))
    author_list_test = append(author_list_test, files_to_add_test)
    labels_test = append(labels_test, rep(author_name_test, length(files_to_add_test)))
}

Trainingtext_test = lapply(author_list_test, readerPlain)
names(Trainingtext_test)=author_list_test
names(Trainingtext_test)=sub('.txt','',names(author_list_test))

my_corpus_test = VCorpus(VectorSource(Trainingtext_test))
names(my_corpus_test) = labels_test
```

```r
my_corpus = tm_map(my_corpus, content_transformer(tolower))
my_corpus = tm_map(my_corpus, content_transformer(removeNumbers))
my_corpus = tm_map(my_corpus, content_transformer(removePunctuation))
my_corpus = tm_map(my_corpus, content_transformer(stripWhitespace))
my_corpus = tm_map(my_corpus, content_transformer(removeWords), stopwords("en"))
my_corpus = tm_map(my_corpus, stemDocument)

my_corpus_test = tm_map(my_corpus_test, content_transformer(tolower))
my_corpus_test = tm_map(my_corpus_test, content_transformer(removeNumbers))
my_corpus_test = tm_map(my_corpus_test, content_transformer(removePunctuation))
my_corpus_test = tm_map(my_corpus_test, content_transformer(stripWhitespace))
my_corpus_test = tm_map(my_corpus_test, content_transformer(removeWords), stopwords("en"
))
my_corpus_test = tm_map(my_corpus_test, stemDocument)
```

```r
DTM = DocumentTermMatrix(my_corpus)
DTM = removeSparseTerms(DTM, 0.95)
DTM
```

```
## <<DocumentTermMatrix (documents: 2500, terms: 812)>>
## Non-/sparse entries: 275749/1754251
## Sparsity           : 86%
## Maximal term length: 10
## Weighting          : term frequency (tf)
```

```
DTM_test = DocumentTermMatrix(my_corpus_test)
DTM_test = removeSparseTerms(DTM_test, 0.95)
DTM_test
```

```
## <<DocumentTermMatrix (documents: 2500, terms: 830)>>
## Non-/sparse entries: 280980/1794020
## Sparsity           : 86%
## Maximal term length: 11
## Weighting          : term frequency (tf)
```

```
# Create matrices for our train and test dataset
X = as.matrix(DTM)
X_test = as.matrix(DTM_test)
```

We need to account for all words in our train dataset that aren’t in test and all the words in our test dataset that aren’t in train.

```r
# Get the list of words in the training set
X_words = colnames(X)

# Get the list of words in the test set
X_test_words = colnames(X_test)

# Create 2 empty vectors to store words to add to test and words to drop from test
test_add = vector(length=0)
test_drop = vector(length=0)

# Loop through the test words and add those 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)
  }
}

# Loop through the train words and add those not in test to the vector test_add
for (word in X_words) {
  if (!word %in% X_test_words) {
    test_add <- c(test_add, word)
  }
}

# Create a matrix of 0's to insert into the test matrix
zero <- matrix(0, nrow = nrow(X), ncol=length(test_add))

# Name the columns using the words in test_add
colnames(zero) <- test_add

# Add the zero matrix to the test matrix
X2_test = cbind(X_test, zero)

# Sort the columns alphabetically so they match the X2
X2_test = X2_test[,order(colnames(X2_test))]

# Drop the words in test_drop from the test matrix
X2_test = X2_test[,!colnames(X2_test) %in% test_drop]
```

Our team chose to run PCA with regression on the dataset for prediction.

```r
library(glmnet)
```

```
## Loaded glmnet 2.0-18
```

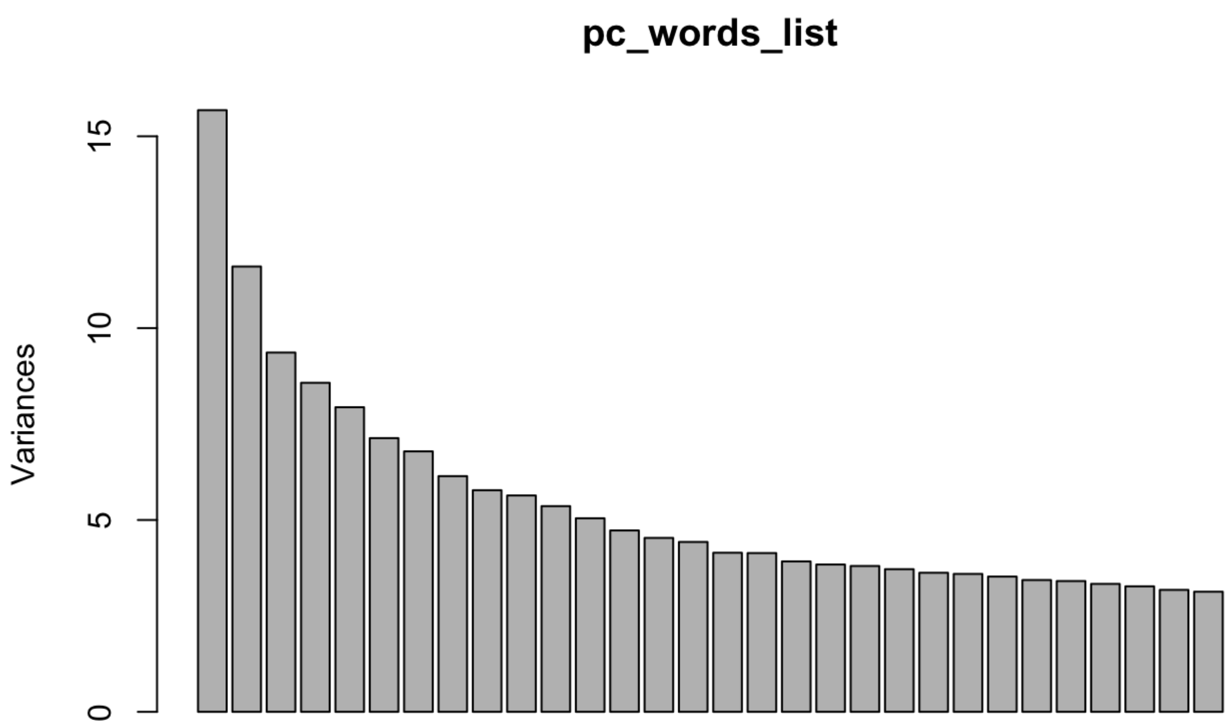
```r
library(nnet)
library(caret)
```

```
##
## Attaching package: 'caret'
```

```
## The following object is masked from 'package:mosaic':
##
##     dotPlot
```

```
## The following object is masked from 'package:purrr':
##
##     lift
```

```
A = X
b = rownames(X)

pc_words_list = prcomp(A, scale=TRUE)
screeplot(pc_words_list,npcs=30)
```



```
K = 682
V = pc_words_list$rotation[,1:K]
scores = A %*% V

#X2_test = X2_test[,1:682]
# Calculate test alphas
test_X = X2_test %*% V

# Set train x and train y
train_X = scores
train_y = rownames(scores)

# Run multinomial regression
multi = glmnet(x=train_X, y=train_y, alpha=0, family="multinomial")


# Predict
predict = predict(multi, newx=test_X, type="class", s=0)

# Check accuracy
multi_accuracy = as.integer(predict == rownames(X2_test))

# Return the total accuracy
mean(multi_accuracy)
```

```
## [1] 0.5892
```

Our accuracy achieved shown above. Our team also tried to perform randomForest (shown below) but unfortunately the model crashes our computers and takes a long time to compute so we chose not to evaluate the chunk but included code instead.

```
library(tibble)
library(dplyr)
library(randomForest)
library(caret)
library(kknn)
X_df = as.data.frame(X)
new_X = X_df %>% rownames_to_column('Author')
X2_test_df = as.data.frame(X2_test)
new_X_test = X2_test_df %>% rownames_to_column('Author')

new_X[,1] = lapply(new_X[,1],as.factor)
new_X_test[,1] = lapply(new_X_test[,1],as.factor)

new_X$missing_values <- apply(new_X, 1, function(x) any(is.na(x)))
new_X[is.na(new_X)]<-0

new_X_test$missing_values <- apply(new_X_test, 1, function(x) any(is.na(x)))
new_X_test[is.na(new_X_test)]<-0

new_X <- subset(new_X, select=-814)
new_X_test <-subset(new_X_test,select=-814)

train_mod = randomForest(Author~.,data=new_X,mtry=3)
pred_mod = predict(train_mod,new_X_test,type='class')

tbl = table(pred_mod,new_X_test$Author)
accuracy_rf= sum(diag(tbl))/sum(tbl)

cat(accuracy_rf)
```

Association Rule Mining

```
library(arules)
```

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

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

```
## The following objects are masked from 'package:mosaic':
##
##     inspect, lhs, rhs
```

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

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

```
library(arulesViz)
```

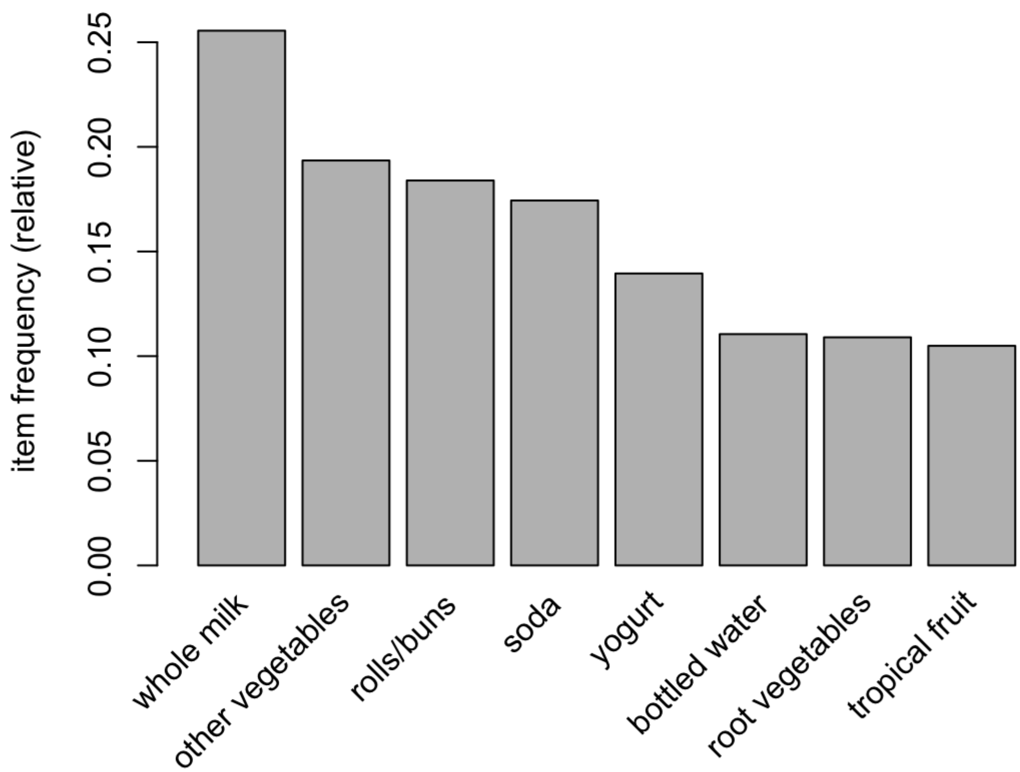
```
## Loading required package: grid
```

```
## Registered S3 method overwritten by 'seriation':
##   method         from
##   reorder.hclust gclus
```

```
library(tidyverse)

grocery_raw = read.transactions('groceries.txt',header=FALSE,format='basket',sep=',',rm.
duplicates = FALSE)
arules::inspect(grocery_raw[1:10])
```

```
##      items
## [1]  {citrus fruit,
##       margarine,
##       ready soups,
##       semi-finished bread}
## [2]  {coffee,
##       tropical fruit,
##       yogurt}
## [3]  {whole milk}
## [4]  {cream cheese,
##       meat spreads,
##       pip fruit,
##       yogurt}
## [5]  {condensed milk,
##       long life bakery product,
##       other vegetables,
##       whole milk}
## [6]  {abrasive cleaner,
##       butter,
##       rice,
##       whole milk,
##       yogurt}
## [7]  {rolls/buns}
## [8]  {bottled beer,
##       liquor (appetizer),
##       other vegetables,
##       rolls/buns,
##       UHT-milk}
## [9]  {pot plants}
## [10] {cereals,
##       whole milk}
```

```
#Visualize the frequency of items in dataset
itemFrequencyPlot(grocery_raw,support=0.1, topN=10)
```



From the relative frequency plot, whole milk is by far our most frequent active item in each transaction as it appears in over 25% of all baskets.

```
#Run Apriori Algorithm
groceryrules = apriori(grocery_raw,
    parameter=list(support=.01, confidence=.25, maxlen=5))
```

```
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##        0.25    0.1    1 none FALSE            TRUE       5    0.01      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: 98
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
## sorting and recoding items ... [88 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 3 4 done [0.00s].
## writing ... [171 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
```

```
arules::inspect(groceryrules[1:5]) #Look at top 10 rules
```

```
##     lhs              rhs                support    confidence lift
## [1] {}            => {whole milk}       0.25551601 0.2555160  1.000000
## [2] {hard cheese} => {whole milk}       0.01006609 0.4107884  1.607682
## [3] {butter milk} => {other vegetables} 0.01037112 0.3709091  1.916916
## [4] {butter milk} => {whole milk}       0.01159126 0.4145455  1.622385
## [5] {ham}         => {whole milk}       0.01148958 0.4414062  1.727509
##     count
## [1] 2513
## [2]   99
## [3]  102
## [4]  114
## [5]  113
```

```
#Support is the # of transactions that include all items in the antecedent and consequen
t parts of the rule. A percentage of the total number of transactions in the dataset.

#Confidence is the ratio of the $ of transactions that include all the items in the cons
equent and antecedent to the number of transactions that include all items in the antece
dent.

#Lift is the ratio of Confidence to Expected Confidence. Lift > 1 means the relationship
between antecendent and consequent is more significant than expected if the two sets wer
e independent. Larger the lift, more significant the association.
```

```
#Sort rules by support, find most frequent associations of items
groceryrules = sort(groceryrules, by = 'support')
inspect(groceryrules[1:5])
```

```
##     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] {yogurt}             => {whole milk}       0.05602440 0.4016035
##     lift     count
## [1] 1.000000 2513
## [2] 1.513634  736
## [3] 1.513634  736
## [4] 1.205032  557
## [5] 1.571735  551
```

From this sort, whole milk (being dominant in our item set) will appear as a consequent item in 5-8% of all baskets. As the most dominant rule states, no matter what customers buy in the grocery store, 25.6% of them will end up buying whole milk.

```
#Sort rules by confidence to find most likely to be true associations
groceryrules = sort(groceryrules, by = 'confidence')
inspect(groceryrules[1:5])
```

```
##     lhs                     rhs                   support confidence     lift count
## [1] {citrus fruit,
##      root vegetables}  => {other vegetables} 0.01037112  0.5862069 3.029608   102
## [2] {root vegetables,
##      tropical fruit}   => {other vegetables} 0.01230300  0.5845411 3.020999   121
## [3] {curd,
##      yogurt}           => {whole milk}       0.01006609  0.5823529 2.279125    99
## [4] {butter,
##      other vegetables} => {whole milk}       0.01148958  0.5736041 2.244885   113
## [5] {root vegetables,
##      tropical fruit}   => {whole milk}       0.01199797  0.5700483 2.230969   118
```

From this sort, we are almost 60% confident that whenever people buy the item combinations on the left hand side they will end up buying the respective item combinations on the right hand side.

```
#Sort rules by lift to find most significant associations of items
groceryrules = sort(groceryrules, by = 'lift')
inspect(groceryrules[1:5])
```

```
##     lhs                     rhs                    support confidence     lift count
## [1] {citrus fruit,
##      other vegetables} => {root vegetables}  0.01037112  0.3591549 3.295045   102
## [2] {other vegetables,
##      tropical fruit}   => {root vegetables}  0.01230300  0.3427762 3.144780   121
## [3] {beef}             => {root vegetables}  0.01738688  0.3313953 3.040367   171
## [4] {citrus fruit,
##      root vegetables}  => {other vegetables} 0.01037112  0.5862069 3.029608   102
## [5] {root vegetables,
##      tropical fruit}   => {other vegetables} 0.01230300  0.5845411 3.020999   121
```

From our sort, we can identify that people who buy items on the left hand side together are 3 times more likely to buy root vegetables/other vegetables versus other customers who do not buy the item sets on the left hand side.

Because whole milk is present in over 25% of our baskets, we are particularly interested in seeing what consequent items our customers will buy if they pick up whole milk at our store.

```
groceryrules_milk_left = apriori(grocery_raw,
parameter=list(support=.01, confidence=.1, maxlen=5),appearance =list(default = 'rhs',lh
s ='whole milk'))
```

```
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##         0.1    0.1    1 none FALSE            TRUE       5    0.01      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: 98
##
## set item appearances ...[1 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 done [0.00s].
## writing ... [24 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
```

```
groceryrules_milk_left = sort(groceryrules_milk_left, by ='lift')
arules::inspect(groceryrules_milk_left)
```

```
##      lhs             rhs                       support    confidence
## [1]  {whole milk} => {butter}                  0.02755465 0.1078392
## [2]  {whole milk} => {curd}                    0.02613116 0.1022682
## [3]  {whole milk} => {domestic eggs}           0.02999492 0.1173896
## [4]  {whole milk} => {whipped/sour cream}      0.03223183 0.1261441
## [5]  {whole milk} => {root vegetables}         0.04890696 0.1914047
## [6]  {whole milk} => {tropical fruit}          0.04229792 0.1655392
## [7]  {whole milk} => {yogurt}                  0.05602440 0.2192598
## [8]  {whole milk} => {pip fruit}               0.03009659 0.1177875
## [9]  {whole milk} => {other vegetables}        0.07483477 0.2928770
## [10] {whole milk} => {pastry}                  0.03324860 0.1301234
## [11] {whole milk} => {citrus fruit}            0.03050330 0.1193792
## [12] {whole milk} => {fruit/vegetable juice} 0.02663955 0.1042579
## [13] {whole milk} => {newspapers}              0.02735130 0.1070434
## [14] {whole milk} => {sausage}                 0.02989324 0.1169916
## [15] {whole milk} => {bottled water}           0.03436706 0.1345006
## [16] {whole milk} => {rolls/buns}              0.05663447 0.2216474
## [17] {}           => {yogurt}                  0.13950178 0.1395018
## [18] {}           => {rolls/buns}              0.18393493 0.1839349
## [19] {}           => {bottled water}           0.11052364 0.1105236
## [20] {}           => {tropical fruit}          0.10493137 0.1049314
## [21] {}           => {root vegetables}         0.10899847 0.1089985
## [22] {}           => {soda}                    0.17437722 0.1743772
## [23] {}           => {other vegetables}        0.19349263 0.1934926
## [24] {whole milk} => {soda}                    0.04006101 0.1567847
##      lift      count
## [1]  1.9460530  271
## [2]  1.9194805  257
## [3]  1.8502027  295
## [4]  1.7597542  317
## [5]  1.7560310  481
## [6]  1.5775950  416
## [7]  1.5717351  551
## [8]  1.5570432  296
## [9]  1.5136341  736
## [10] 1.4625865  327
## [11] 1.4423768  300
## [12] 1.4421604  262
## [13] 1.3411103  269
## [14] 1.2452520  294
## [15] 1.2169396  338
## [16] 1.2050318  557
## [17] 1.0000000 1372
## [18] 1.0000000 1809
## [19] 1.0000000 1087
## [20] 1.0000000 1032
## [21] 1.0000000 1072
## [22] 1.0000000 1715
## [23] 1.0000000 1903
## [24] 0.8991124  394
```

From rules with whole milk on LHS, we can see that customers who buy whole milk are almost 2x more likely to buy butter, curd, domestic eggs, etc. From the RHS results, we recommend the store place these items near whole milk so customers can grab them easily.

```
groceryrules_milk_right = apriori(grocery_raw,
parameter=list(support=.01, confidence=.1, maxlen=5),appearance =list(default = 'lhs',rh
s ='whole milk'))
```

```
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##         0.1    0.1    1 none FALSE            TRUE       5    0.01      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: 98
##
## set item appearances ...[1 item(s)] done [0.00s].
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
## sorting and recoding items ... [88 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 3 4 done [0.00s].
## writing ... [71 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
```

```
groceryrules_milk_right = sort(groceryrules_milk_right, by ='lift')
arules::inspect(groceryrules_milk_right)
```

```
##      lhs                            rhs              support confidence      lift count
## [1]  {curd,
##       yogurt}                    => {whole milk} 0.01006609  0.5823529 2.2791250    99
## [2]  {butter,
##       other vegetables}          => {whole milk} 0.01148958  0.5736041 2.2448850   113
## [3]  {root vegetables,
##       tropical fruit}            => {whole milk} 0.01199797  0.5700483 2.2309690   118
## [4]  {root vegetables,
##       yogurt}                    => {whole milk} 0.01453991  0.5629921 2.2033536   143
## [5]  {domestic eggs,
##       other vegetables}          => {whole milk} 0.01230300  0.5525114 2.1623358   121
## [6]  {whipped/sour cream,
##       yogurt}                    => {whole milk} 0.01087951  0.5245098 2.0527473   107
## [7]  {rolls/buns,
##       root vegetables}           => {whole milk} 0.01270971  0.5230126 2.0468876   125
## [8]  {other vegetables,
##       pip fruit}                 => {whole milk} 0.01352313  0.5175097 2.0253514   133
## [9]  {tropical fruit,
##       yogurt}                    => {whole milk} 0.01514997  0.5173611 2.0247698   149
## [10] {other vegetables,
##       yogurt}                    => {whole milk} 0.02226741  0.5128806 2.0072345   219
## [11] {other vegetables,
##       whipped/sour cream}        => {whole milk} 0.01464159  0.5070423 1.9843854   144
## [12] {fruit/vegetable juice,
##       other vegetables}          => {whole milk} 0.01047280  0.4975845 1.9473713   103
## [13] {butter}                    => {whole milk} 0.02755465  0.4972477 1.9460530   271
## [14] {curd}                      => {whole milk} 0.02613116  0.4904580 1.9194805   257
## [15] {other vegetables,
##       root vegetables}           => {whole milk} 0.02318251  0.4892704 1.9148326   228
## [16] {other vegetables,
##       tropical fruit}            => {whole milk} 0.01708185  0.4759207 1.8625865   168
## [17] {citrus fruit,
##       yogurt}                    => {whole milk} 0.01026945  0.4741784 1.8557678   101
## [18] {domestic eggs}             => {whole milk} 0.02999492  0.4727564 1.8502027   295
## [19] {other vegetables,
##       pork}                      => {whole milk} 0.01016777  0.4694836 1.8373939   100
## [20] {other vegetables,
##       pastry}                    => {whole milk} 0.01057448  0.4684685 1.8334212   104
## [21] {rolls/buns,
##       yogurt}                    => {whole milk} 0.01555669  0.4526627 1.7715630   153
## [22] {citrus fruit,
##       other vegetables}          => {whole milk} 0.01301474  0.4507042 1.7638982   128
## [23] {whipped/sour cream}        => {whole milk} 0.03223183  0.4496454 1.7597542   317
## [24] {root vegetables}           => {whole milk} 0.04890696  0.4486940 1.7560310   481
## [25] {rolls/buns,
##       tropical fruit}            => {whole milk} 0.01098119  0.4462810 1.7465872   108
## [26] {sugar}                     => {whole milk} 0.01504830  0.4444444 1.7393996   148
## [27] {hamburger meat}            => {whole milk} 0.01474326  0.4434251 1.7354101   145
## [28] {ham}                       => {whole milk} 0.01148958  0.4414062 1.7275091   113
## [29] {sliced cheese}             => {whole milk} 0.01077783  0.4398340 1.7213560   106
## [30] {bottled water,
##       other vegetables}          => {whole milk} 0.01077783  0.4344262 1.7001918   106
## [31] {other vegetables,
##       soda}                      => {whole milk} 0.01392984  0.4254658 1.6651240   137
```

```
## [32] {frozen vegetables}         => {whole milk} 0.02043721  0.4249471 1.6630940   201
## [33] {other vegetables,
##       rolls/buns}                => {whole milk} 0.01789527  0.4200477 1.6439194   176
## [34] {cream cheese}              => {whole milk} 0.01647178  0.4153846 1.6256696   162
## [35] {butter milk}               => {whole milk} 0.01159126  0.4145455 1.6223854   114
## [36] {margarine}                 => {whole milk} 0.02419929  0.4131944 1.6170980   238
## [37] {hard cheese}               => {whole milk} 0.01006609  0.4107884 1.6076815    99
## [38] {chicken}                   => {whole milk} 0.01759024  0.4099526 1.6044106   173
## [39] {white bread}               => {whole milk} 0.01708185  0.4057971 1.5881474   168
## [40] {beef}                      => {whole milk} 0.02125064  0.4050388 1.5851795   209
## [41] {tropical fruit}            => {whole milk} 0.04229792  0.4031008 1.5775950   416
## [42] {oil}                       => {whole milk} 0.01128622  0.4021739 1.5739675   111
## [43] {yogurt}                    => {whole milk} 0.05602440  0.4016035 1.5717351   551
## [44] {pip fruit}                 => {whole milk} 0.03009659  0.3978495 1.5570432   296
## [45] {onions}                    => {whole milk} 0.01209964  0.3901639 1.5269647   119
## [46] {hygiene articles}          => {whole milk} 0.01281139  0.3888889 1.5219746   126
## [47] {brown bread}               => {whole milk} 0.02521607  0.3887147 1.5212930   248
## [48] {other vegetables}          => {whole milk} 0.07483477  0.3867578 1.5136341   736
## [49] {pork}                      => {whole milk} 0.02216573  0.3844797 1.5047187   218
## [50] {soda,
##       yogurt}                    => {whole milk} 0.01047280  0.3828996 1.4985348   103
## [51] {other vegetables,
##       sausage}                   => {whole milk} 0.01016777  0.3773585 1.4768487   100
## [52] {napkins}                   => {whole milk} 0.01972547  0.3766990 1.4742678   194
## [53] {pastry}                    => {whole milk} 0.03324860  0.3737143 1.4625865   327
## [54] {dessert}                   => {whole milk} 0.01372649  0.3698630 1.4475140   135
## [55] {citrus fruit}              => {whole milk} 0.03050330  0.3685504 1.4423768   300
## [56] {fruit/vegetable juice}     => {whole milk} 0.02663955  0.3684951 1.4421604   262
## [57] {long life bakery product} => {whole milk} 0.01352313  0.3614130 1.4144438   133
## [58] {berries}                   => {whole milk} 0.01179461  0.3547401 1.3883281   116
## [59] {frankfurter}               => {whole milk} 0.02053889  0.3482759 1.3630295   202
## [60] {newspapers}                => {whole milk} 0.02735130  0.3426752 1.3411103   269
## [61] {chocolate}                 => {whole milk} 0.01667514  0.3360656 1.3152427   164
## [62] {waffles}                   => {whole milk} 0.01270971  0.3306878 1.2941961   125
## [63] {coffee}                    => {whole milk} 0.01870869  0.3222417 1.2611408   184
## [64] {sausage}                   => {whole milk} 0.02989324  0.3181818 1.2452520   294
## [65] {bottled water}             => {whole milk} 0.03436706  0.3109476 1.2169396   338
## [66] {rolls/buns}                => {whole milk} 0.05663447  0.3079049 1.2050318   557
## [67] {salty snack}               => {whole milk} 0.01118454  0.2956989 1.1572618   110
## [68] {}                          => {whole milk} 0.25551601  0.2555160 1.0000000  2513
## [69] {bottled beer}              => {whole milk} 0.02043721  0.2537879 0.9932367   201
## [70] {shopping bags}             => {whole milk} 0.02450432  0.2487100 0.9733637   241
## [71] {soda}                      => {whole milk} 0.04006101  0.2297376 0.8991124   394
```

From rules with whole milk on RHS, we can see that cuustomer who buy {curd,yogurt}, {butter, other vegetables}, or {root vegetables, yogurt/tropical fruit} are over 2.2x more likely to buy whole milk from the store. This means it will be strategic for the store to place the items on the LHS closer to whole milk as well.

```
groceryrules_other_vegetables = apriori(grocery_raw,
parameter=list(support=.01, confidence=.1, maxlen=5),appearance =list(default = 'rhs',lh
s ='other vegetables'))
```

```
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##         0.1    0.1    1 none FALSE            TRUE       5    0.01      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: 98
##
## set item appearances ...[1 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 done [0.00s].
## writing ... [26 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
```

```
inspect(groceryrules_other_vegetables)
```

```
##      lhs                     rhs                      support    confidence
## [1]  {}                   => {bottled water}         0.11052364 0.1105236
## [2]  {}                   => {tropical fruit}        0.10493137 0.1049314
## [3]  {}                   => {root vegetables}       0.10899847 0.1089985
## [4]  {}                   => {soda}                  0.17437722 0.1743772
## [5]  {}                   => {yogurt}                0.13950178 0.1395018
## [6]  {}                   => {rolls/buns}            0.18393493 0.1839349
## [7]  {}                   => {whole milk}            0.25551601 0.2555160
## [8]  {other vegetables} => {beef}                  0.01972547 0.1019443
## [9]  {other vegetables} => {pork}                  0.02165735 0.1119285
## [10] {other vegetables} => {margarine}             0.01972547 0.1019443
## [11] {other vegetables} => {butter}                0.02003050 0.1035208
## [12] {other vegetables} => {domestic eggs}         0.02226741 0.1150815
## [13] {other vegetables} => {fruit/vegetable juice} 0.02104728 0.1087756
## [14] {other vegetables} => {whipped/sour cream}    0.02887646 0.1492380
## [15] {other vegetables} => {pip fruit}             0.02613116 0.1350499
## [16] {other vegetables} => {pastry}                0.02257245 0.1166579
## [17] {other vegetables} => {citrus fruit}          0.02887646 0.1492380
## [18] {other vegetables} => {shopping bags}         0.02318251 0.1198108
## [19] {other vegetables} => {sausage}               0.02694459 0.1392538
## [20] {other vegetables} => {bottled water}         0.02480935 0.1282186
## [21] {other vegetables} => {tropical fruit}        0.03589222 0.1854966
## [22] {other vegetables} => {root vegetables}       0.04738180 0.2448765
## [23] {other vegetables} => {soda}                  0.03274021 0.1692065
## [24] {other vegetables} => {yogurt}                0.04341637 0.2243826
## [25] {other vegetables} => {rolls/buns}            0.04260295 0.2201787
## [26] {other vegetables} => {whole milk}            0.07483477 0.3867578
##      lift      count
## [1]  1.0000000 1087
## [2]  1.0000000 1032
## [3]  1.0000000 1072
## [4]  1.0000000 1715
## [5]  1.0000000 1372
## [6]  1.0000000 1809
## [7]  1.0000000 2513
## [8]  1.9430662  194
## [9]  1.9414764  213
## [10] 1.7406635  194
## [11] 1.8681223  197
## [12] 1.8138238  219
## [13] 1.5046529  207
## [14] 2.0819237  284
## [15] 1.7852365  257
## [16] 1.3112349  222
## [17] 1.8031403  284
## [18] 1.2160366  228
## [19] 1.4822091  265
## [20] 1.1601012  244
## [21] 1.7677896  353
## [22] 2.2466049  466
## [23] 0.9703476  322
## [24] 1.6084566  427
```

```
## [25] 1.1970465  419
## [26] 1.5136341  736
```

We see that people who buy other vegetables are 2.08x more likely to buy whipped/sour cream, so it makes sense to place them near each other in the grocery store.

```
groceryrules_other_vegetables_d = apriori(grocery_raw,
parameter=list(support=.01, confidence=.1, maxlen=5),appearance =list(default = 'lhs',rh
s ='other vegetables'))
```

```
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##         0.1    0.1    1 none FALSE            TRUE       5    0.01      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: 98
##
## set item appearances ...[1 item(s)] done [0.00s].
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
## sorting and recoding items ... [88 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 3 4 done [0.00s].
## writing ... [63 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
```

```
inspect(groceryrules_other_vegetables_d)
```

```
##      lhs                            rhs                    support confidence      lift
count
## [1]  {}                          => {other vegetables} 0.19349263  0.1934926 1.0000000
1903
## [2]  {butter milk}               => {other vegetables} 0.01037112  0.3709091 1.9169159
102
## [3]  {onions}                    => {other vegetables} 0.01423488  0.4590164 2.3722681
140
## [4]  {berries}                   => {other vegetables} 0.01026945  0.3088685 1.5962805
101
## [5]  {hamburger meat}            => {other vegetables} 0.01382816  0.4159021 2.1494470
136
## [6]  {salty snack}               => {other vegetables} 0.01077783  0.2849462 1.4726465
106
## [7]  {sugar}                     => {other vegetables} 0.01077783  0.3183183 1.6451186
106
## [8]  {waffles}                   => {other vegetables} 0.01006609  0.2619048 1.3535645
99
## [9]  {long life bakery product} => {other vegetables} 0.01067616  0.2853261 1.4746096
105
## [10] {dessert}                   => {other vegetables} 0.01159126  0.3123288 1.6141636
114
## [11] {cream cheese}              => {other vegetables} 0.01372649  0.3461538 1.7889769
135
## [12] {chicken}                   => {other vegetables} 0.01789527  0.4170616 2.1554393
176
## [13] {white bread}               => {other vegetables} 0.01372649  0.3260870 1.6852681
135
## [14] {chocolate}                 => {other vegetables} 0.01270971  0.2561475 1.3238103
125
## [15] {coffee}                    => {other vegetables} 0.01342145  0.2311734 1.1947400
132
## [16] {frozen vegetables}         => {other vegetables} 0.01779359  0.3699789 1.9121083
175
## [17] {beef}                      => {other vegetables} 0.01972547  0.3759690 1.9430662
194
## [18] {curd}                      => {other vegetables} 0.01718353  0.3225191 1.6668288
169
## [19] {napkins}                   => {other vegetables} 0.01443823  0.2757282 1.4250060
142
## [20] {pork}                      => {other vegetables} 0.02165735  0.3756614 1.9414764
213
## [21] {frankfurter}               => {other vegetables} 0.01647178  0.2793103 1.4435193
162
## [22] {bottled beer}              => {other vegetables} 0.01616675  0.2007576 1.0375464
159
## [23] {brown bread}               => {other vegetables} 0.01870869  0.2884013 1.4905025
184
## [24] {margarine}                 => {other vegetables} 0.01972547  0.3368056 1.7406635
194
## [25] {butter}                    => {other vegetables} 0.02003050  0.3614679 1.8681223
197
## [26] {newspapers}                => {other vegetables} 0.01931876  0.2420382 1.2508912
```

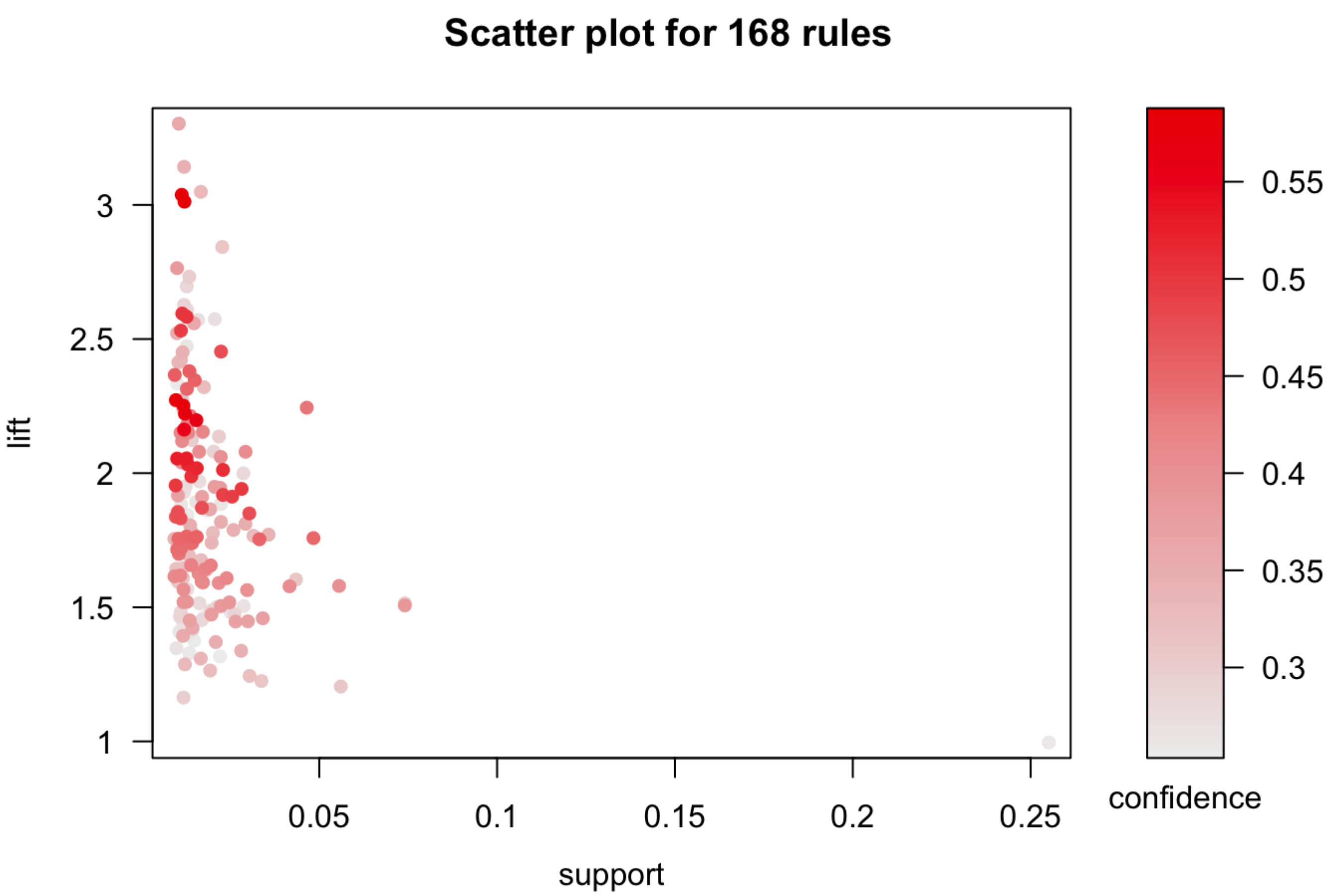
```
190
## [27] {domestic eggs}           => {other vegetables} 0.02226741  0.3509615 1.8138238
219
## [28] {fruit/vegetable juice}   => {other vegetables} 0.02104728  0.2911392 1.5046529
207
## [29] {whipped/sour cream}      => {other vegetables} 0.02887646  0.4028369 2.0819237
284
## [30] {pip fruit}               => {other vegetables} 0.02613116  0.3454301 1.7852365
257
## [31] {pastry}                  => {other vegetables} 0.02257245  0.2537143 1.3112349
222
## [32] {citrus fruit}            => {other vegetables} 0.02887646  0.3488943 1.8031403
284
## [33] {shopping bags}           => {other vegetables} 0.02318251  0.2352941 1.2160366
228
## [34] {sausage}                 => {other vegetables} 0.02694459  0.2867965 1.4822091
265
## [35] {bottled water}           => {other vegetables} 0.02480935  0.2244710 1.1601012
244
## [36] {tropical fruit}          => {other vegetables} 0.03589222  0.3420543 1.7677896
353
## [37] {root vegetables}         => {other vegetables} 0.04738180  0.4347015 2.2466049
466
## [38] {soda}                    => {other vegetables} 0.03274021  0.1877551 0.9703476
322
## [39] {yogurt}                  => {other vegetables} 0.04341637  0.3112245 1.6084566
427
## [40] {rolls/buns}              => {other vegetables} 0.04260295  0.2316197 1.1970465
419
## [41] {whole milk}              => {other vegetables} 0.07483477  0.2928770 1.5136341
736
## [42] {pork,
##       whole milk}              => {other vegetables} 0.01016777  0.4587156 2.3707136
100
## [43] {butter,
##       whole milk}              => {other vegetables} 0.01148958  0.4169742 2.1549874
113
## [44] {domestic eggs,
##       whole milk}              => {other vegetables} 0.01230300  0.4101695 2.1198197
121
## [45] {fruit/vegetable juice,
##       whole milk}              => {other vegetables} 0.01047280  0.3931298 2.0317558
103
## [46] {whipped/sour cream,
##       yogurt}                  => {other vegetables} 0.01016777  0.4901961 2.5334096
100
## [47] {whipped/sour cream,
##       whole milk}              => {other vegetables} 0.01464159  0.4542587 2.3476795
144
## [48] {pip fruit,
##       whole milk}              => {other vegetables} 0.01352313  0.4493243 2.3221780
133
## [49] {pastry,
##       whole milk}              => {other vegetables} 0.01057448  0.3180428 1.6436947
```

```
104
## [50] {citrus fruit,
##       root vegetables}          => {other vegetables} 0.01037112  0.5862069 3.0296084
102
## [51] {citrus fruit,
##       whole milk}               => {other vegetables} 0.01301474  0.4266667 2.2050797
128
## [52] {sausage,
##       whole milk}               => {other vegetables} 0.01016777  0.3401361 1.7578760
100
## [53] {bottled water,
##       whole milk}               => {other vegetables} 0.01077783  0.3136095 1.6207825
106
## [54] {root vegetables,
##       tropical fruit}           => {other vegetables} 0.01230300  0.5845411 3.0209991
121
## [55] {tropical fruit,
##       yogurt}                   => {other vegetables} 0.01230300  0.4201389 2.1713431
121
## [56] {tropical fruit,
##       whole milk}               => {other vegetables} 0.01708185  0.4038462 2.0871397
168
## [57] {root vegetables,
##       yogurt}                   => {other vegetables} 0.01291307  0.5000000 2.5840778
127
## [58] {rolls/buns,
##       root vegetables}          => {other vegetables} 0.01220132  0.5020921 2.5948898
120
## [59] {root vegetables,
##       whole milk}               => {other vegetables} 0.02318251  0.4740125 2.4497702
228
## [60] {soda,
##       whole milk}               => {other vegetables} 0.01392984  0.3477157 1.7970490
137
## [61] {rolls/buns,
##       yogurt}                   => {other vegetables} 0.01148958  0.3343195 1.7278153
113
## [62] {whole milk,
##       yogurt}                   => {other vegetables} 0.02226741  0.3974592 2.0541308
219
## [63] {rolls/buns,
##       whole milk}               => {other vegetables} 0.01789527  0.3159785 1.6330258
176
```

From the above output, we see that if one buys {citrus fruit, root vegetables}, they are 3.03x more likely to buy other vegetables.

```
#Remove redudant rules
redundantrules = is.redundant(groceryrules)
groceryrules = groceryrules[!redundantrules]


#Plot Rules
plot(groceryrules, measure = c("support", "lift"), shading = "confidence")
```

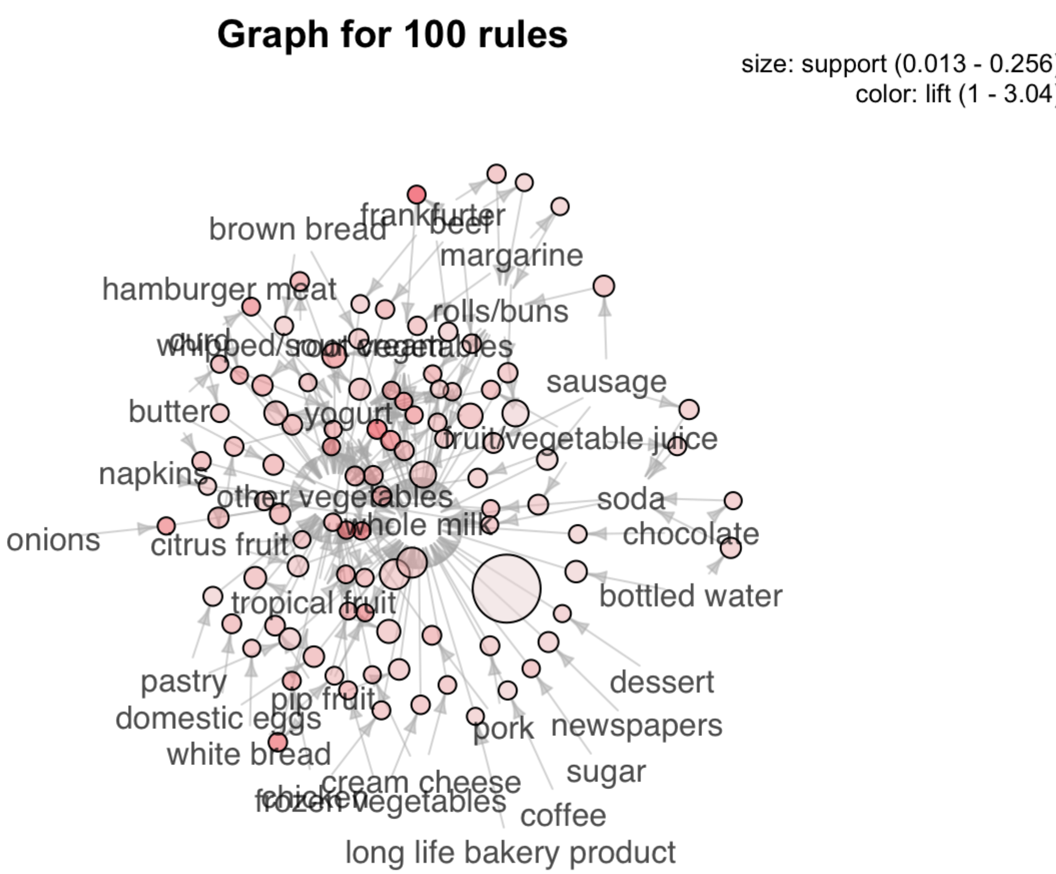
```
## To reduce overplotting, jitter is added! Use jitter = 0 to prevent jitter.
```



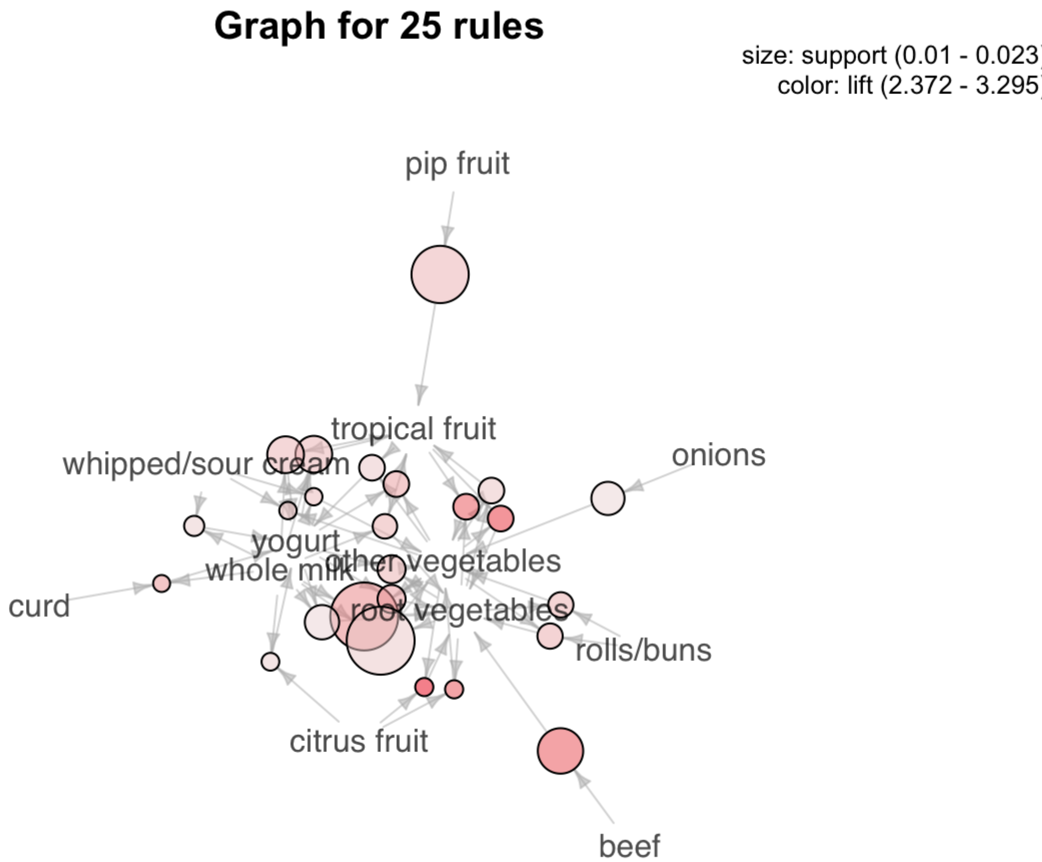

```
# graph-based visualization
sub1 = subset(groceryrules, subset=confidence > 0.01 & support > 0.005)
summary(sub1)
```

```
## set of 168 rules
##
## rule length distribution (lhs + rhs):sizes
##  1  2  3
##  1 95 72
##
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   1.000   2.000   2.000   2.423   3.000   3.000
##
## summary of quality measures:
##     support           confidence          lift            count
##  Min.   :0.01007   Min.   :0.2517   Min.   :1.000   Min.   :  99.0
##  1st Qu.:0.01174   1st Qu.:0.2969   1st Qu.:1.526   1st Qu.: 115.5
##  Median :0.01454   Median :0.3565   Median :1.787   Median : 143.0
##  Mean   :0.01972   Mean   :0.3702   Mean   :1.879   Mean   : 193.9
##  3rd Qu.:0.02135   3rd Qu.:0.4258   3rd Qu.:2.151   3rd Qu.: 210.0
##  Max.   :0.25552   Max.   :0.5862   Max.   :3.295   Max.   :2513.0
##
## mining info:
##         data ntransactions support confidence
##  grocery_raw          9835    0.01       0.25
```

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



```
?plot.rules

plot(head(sub1, 25, by='lift'), method='graph')
```



To conclude, our discovered item sets makes sense. From each type of sort we were able to gain valuable information about key items that our customers purchase consequently with other items. After general analysis, we focused on visualizing rules based on our top two most bought items (whole milk and other vegetables) in order to maximize the store's profits with items important to the customers.