Predictive Modeling Part 2 HW

Martina Galvan, Zeba Pathan, Deepti Rao, Tairan Deng

8/18/2020

Visual story telling part 1: green buildings

Overall speaking we think developers argument is too simplified on the cost-benefit model. Therere other factors like, renovation rate, leasing rate, building class that also affect cost of operation of the building, we did the following calculation and plots analysis.

Renovation Rate Comparision

By adding green features to the building, there's a one time cost, but there could be future increased maintenance cost, indicated by times of renovation in the long run.

We process data by divide total amout of renovation by total building ages of green/nongreen buildings. Then what we got is the renovation rate expection through building life. Accourding to our calculation, the chance per year that green building need renovation is 0.89%, that of non-green building got change of 0.80%. Therefore green buildings are 11.8% more likely to need renovation work every year.

Since renovation cost is hard to predict, were not sure if higher rent income can offset the further upcoming renovation expense. But in terms of maintenance, this is a conculsion against developers model.

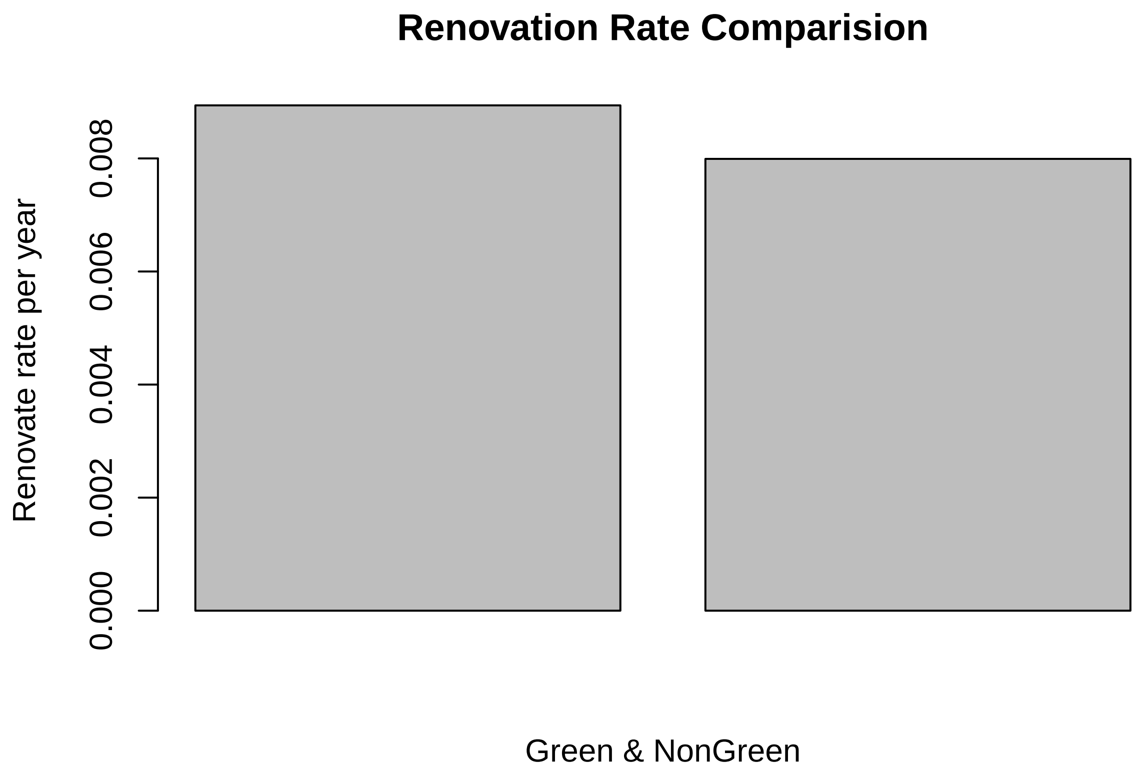
```
library(plyr)
green<-read.csv('C:/Users/DENG/Desktop/UTA Doc/Summer/predictive modeling/PART2 EXERCISE/STA380/data/gr

#compare renovation rate of Green/Nongreen
reno_bygreen<-aggregate(x = green$renovated,by = list(green$green_rating),FUN = sum)
age_bygreen<-aggregate(x = green$age,by = list(green$green_rating),FUN = sum)
print(reno_bygreen)
```

```
##   Group.1    x
## 1       0 2850
## 2       1  146
```

```
print(age_bygreen)
```

```
##   Group.1      x
## 1       0 356610
## 2       1  16334
```

```r
g_RenoRate<-reno_bygreen[2,2]/age_bygreen[2,2]
notg_RenoRate<-reno_bygreen[1,2]/age_bygreen[1,2]
extra_rate<-(g_RenoRate-notg_RenoRate)/notg_RenoRate
#combine renovation rates and plot
Fix_rate<-c(g_RenoRate,notg_RenoRate)
barplot(Fix_rate,main="Renovation Rate Comparision",
        xlab="Green & NonGreen", ylab="Renovate rate per year")
```

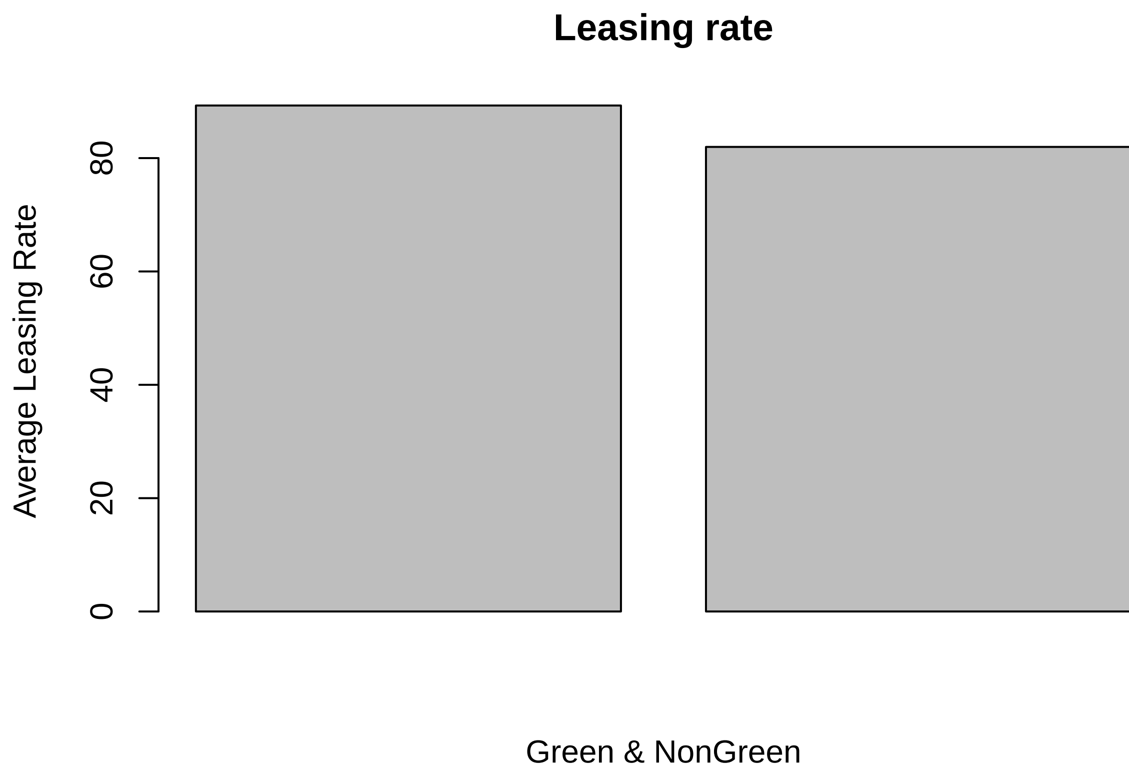


Leasing Rate Comparision Another factor affecting realized income of the building is leasing rate. We first suspect that green buildings may have lower leasing rate because of their higher rents, thus offsets the gain from extra rent rate. We did the average leasing rate comparision between green and non green buildings and found some interest results.

The average leasing rate for non-green building is 81.97206, the average leasing rate of green building is 89.28190, which is over 10% higher than non-green building. This result indicates that green buildin owners were able to rent out higher capacity of their building at a higher rental expense(per square feet). This observation clearly supported developer's argument on investing in green buildings.

```
library(plyr)
Leasing_rate_df<-aggregate(green[, 6], list(green$green_rating), mean)
Leasing_rate<-c(Leasing_rate_df[2,2],Leasing_rate_df[1,2])
Leasing_rate
```

```
## [1] 89.28190 81.97206
```



Plot result is shown below.

Building class Comparision

In this part were analyzing relationship between green rating and classes of buildings. We are comparing scale of class A buildings and class B buildings amoung green/non green buildings. Since building quality is also a factor of evaluation the market trend. As we can see, 79.7% of green buildings are Class A, which means average quality of green buildings are rather good, compared to 36% of non-green buildings, need less to say, non-green buildings have a much higher percentage of class B buildings. Its hard to draw a conclusion from this observation, since green building is a rather new concept, older building tends to have more worn-out conditions. Also we do not know if building green rating is a factor that determines the buildings class. If thats the case, we can not assume that building class rating necessarily represent luxuriness of the building.

```
class_average<-aggregate(green[, 10:11], list(green$green_rating), mean)
class_average
```

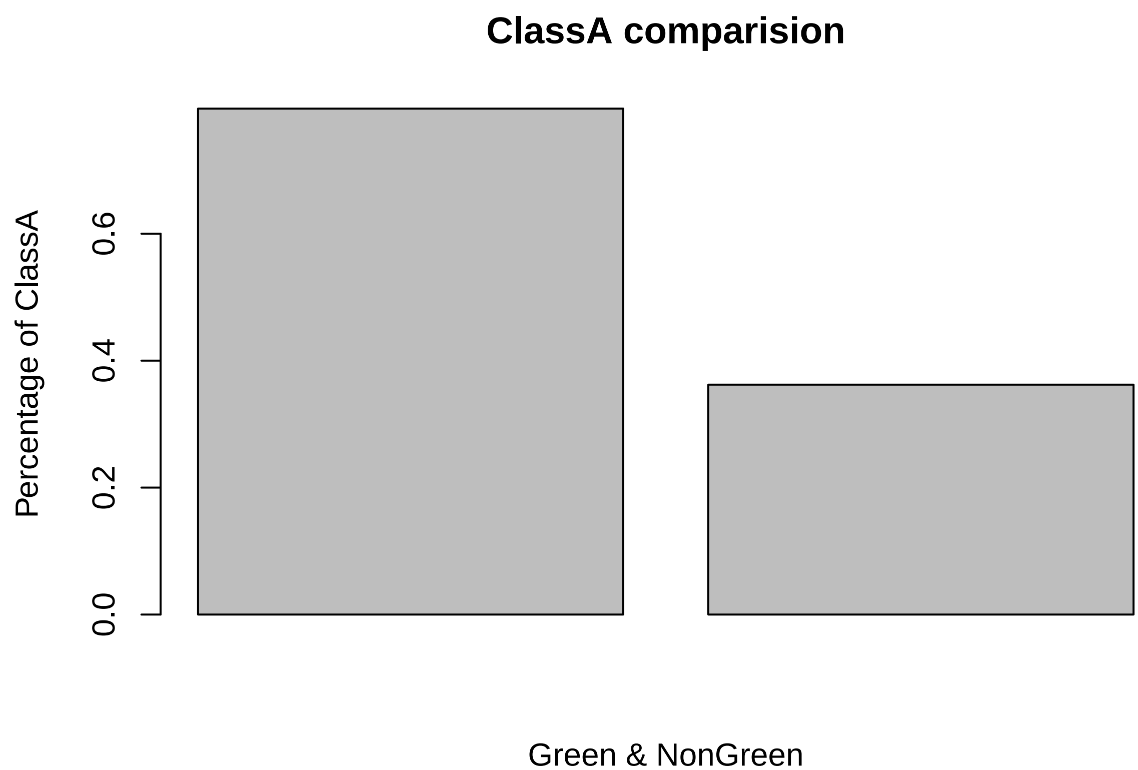
```
##   Group.1   class_a   class_b
## 1       0 0.3621862 0.4848107
## 2       1 0.7970803 0.1927007
```

```
class_average_rateA<-c(class_average[2,2],class_average[1,2])
class_average_rateA
```

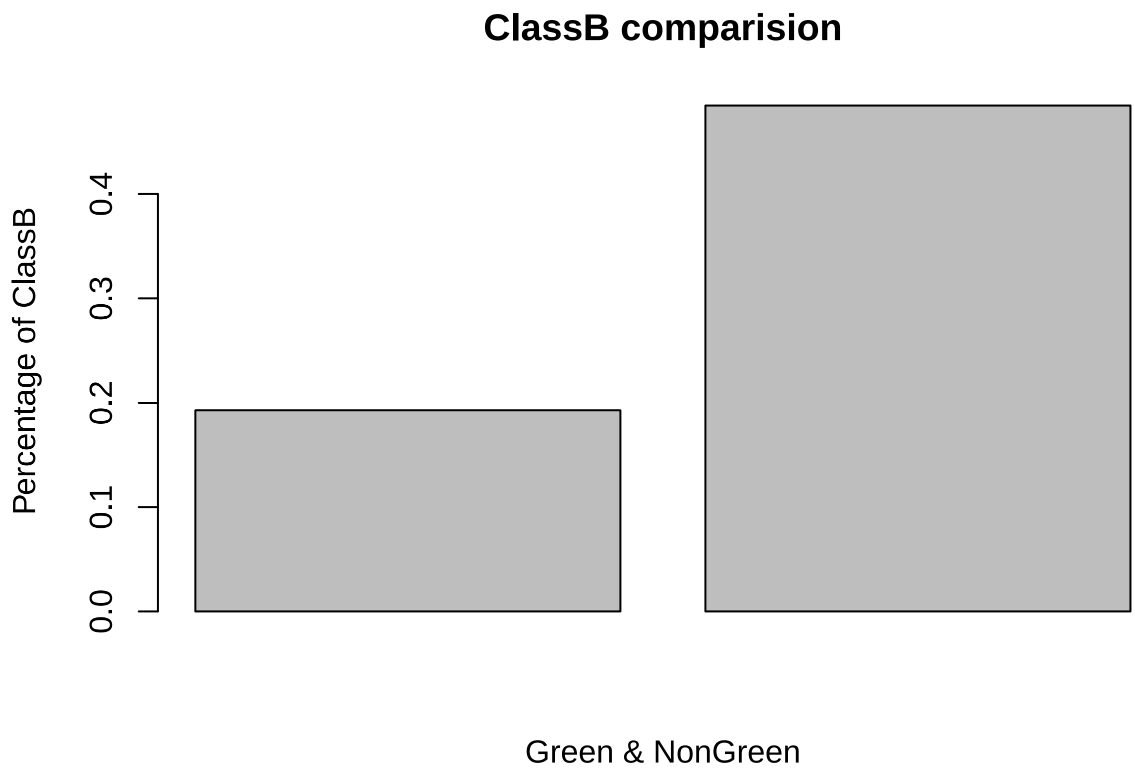
```
## [1] 0.7970803 0.3621862
```

```
class_average_rateB<-c(class_average[2,3],class_average[1,3])
class_average_rateB
```

```
## [1] 0.1927007 0.4848107
```



Please see following plots



Visual story telling part 2: flights at ABIA

This dataset includes a huge variety of data and we have plenty of room to play with it. We decided to focus on delay time and weather as a delay reason. After plotting, we're also doing a regression on arrival delay and departure delay time.

Plot delay time

In this question were `trying to visualization of frequency of flight cancellation due to weather. Based on weather data when Austin have extreme weather in 2008,` were trying to find connection in between.

From the plot we can see delays are more severe from mid March to May and in December in 2008. Refer to extreme weather record. Date Loss in Million 03-31-2008 120
04-10-2008 25
05-14-2008 50
08-18-2008 25
12-12-2008 85

Reference:SIGNIFICANT WEATHER, 2000S https://texasalmanac.com/topics/environment/significant-weather-2000s

By comparing the timeline of extreme weather dataframe and our plot results, we have following conculsion. From the plot result of arrival and depature delay, we can easily tell that delay time is highly associated with extreme weather in Austin area in 2008, including storm, snow storm and hail.

```
library(ggplot2)
library(dplyr)
```

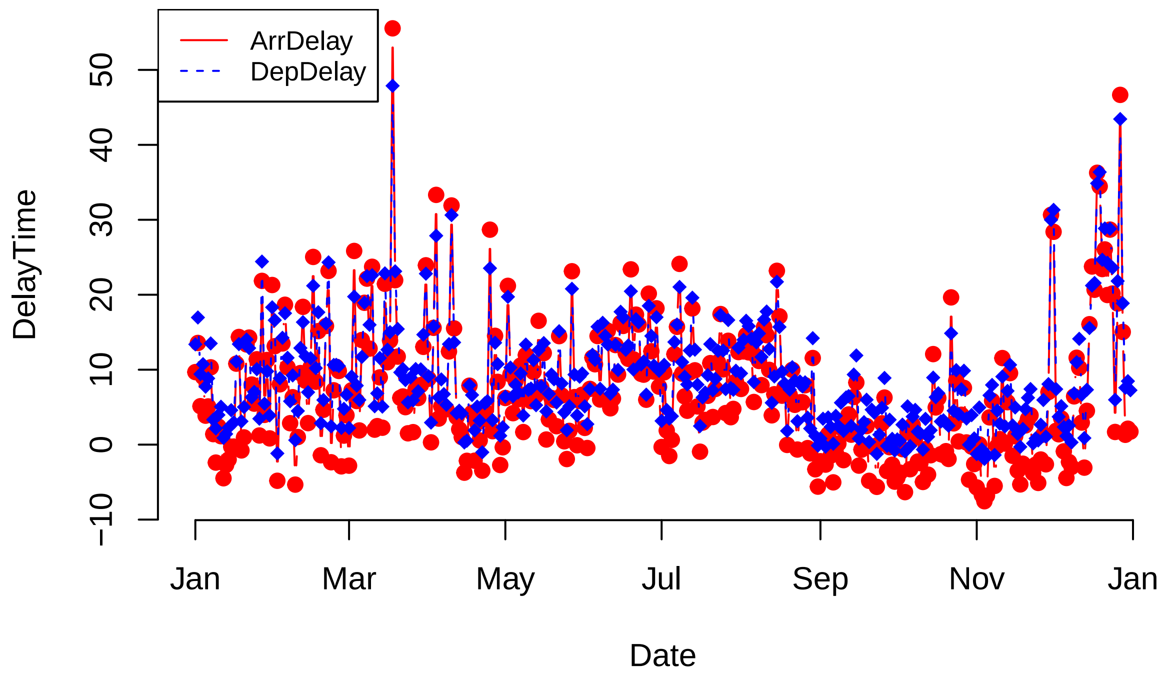
```
##
## Attaching package: 'dplyr'

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

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

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

```
library(data.table)
```

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

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

```
ABIA<-read.csv('C:/Users/DENG/Desktop/UTA Doc/Summer/predictive modeling/PART2 EXERCISE/STA380/data/ABIA
ABIA$date<-as.Date(with(ABIA,paste(Year,Month,DayofMonth,sep='-')),"%Y-%m-%d")
delay<-aggregate(ABIA[, 15:16], list(ABIA$date), mean,na.rm=T)

diaster<-data.frame('Date'=c('03-31-2008','04-10-2008','05-14-2008','08-18-2008','12-12-2008'),'Loss'=c
diaster
```

```
##         Date Loss
## 1 03-31-2008  120
## 2 04-10-2008   25
## 3 05-14-2008   50
## 4 08-18-2008   25
## 5 12-12-2008   85

## Warning in xy.coords(x, y): NAs introduced by coercion
```



Delay time on regression model

After we found the correlation between weather and delay time, we want to examine if weather factor effect arrival time and departure time in similar way. Since arrival time and delay time are under same weather circumstances, over this topic we only build regression model between them.

```
Arr_Delay<-lm(delay$DepDelay~delay$ArrDelay,data=delay)
summary(Arr_Delay)
```

```
##
## Call:
## lm(formula = delay$DepDelay ~ delay$ArrDelay, data = delay)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -4.7239 -0.8489  0.0429  0.8227  4.8762
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)    3.445434   0.093931   36.68   <2e-16 ***
## delay$ArrDelay 0.813765   0.008401   96.86   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.423 on 364 degrees of freedom
```

```
## Multiple R-squared:  0.9627, Adjusted R-squared:  0.9626
## F-statistic:  9383 on 1 and 364 DF,  p-value: < 2.2e-16
```

From the above regression results we can see that arrival delay and departure delay times are highly associated, which make sense. If a flight arrive late at Austin airport, it's likely for that flight to also take off late. The coefficient of arrival time over departure time shows rather low Std. error, which means all flights are affected by late arrival in a very similar way. Additionally, the intercept of this model is very low, only 3.44 min compared to common delay time over hours. This intercept is not significant compared to the total y value, departure delay time. This means that departure delay time is mainly effected by arrival delay time. The coefficient of 0.81 indicates that for every 10 more min of arrival delay time, the departure delay time only increase by 8 min, indicating that Austin airport actually operates more efficient on improving delays under extreme weather.

Market_Segmentation

```r
library(ggplot2)
library(LICORS)  # for kmeans++
library(foreach)
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

## 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.
##
## Have you tried the ggformula package for your plots?
```

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

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

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

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

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

```
## 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
```

```r
social<-read.csv("C:/Users/DENG/Desktop/UTA Doc/Summer/predictive modeling/PART2 EXERCISE/STA380/data/s
Y = social[,(2:37)]
X = scale(Y, center=TRUE, scale=TRUE)
```

In order to begin the clustering analysis, we first needed to read in the data and scale it as seen above. Scaling the data in a case like this is important and is standard practice because so that numbers can be compared to means with standard deviations.

Printing the head after scaling gets us decimal point numbers because they are in terms of standard deviations. Scaling the data is done by subtracting the mean and dividing by the standard deviation.

Here, we extract the centers and scales from the rescaled data (which are named attributes)

The seed was set so that when another data scientist opens this file, they are able to reproduce the same clusters as were used in this analysis.

Subsequently, the kmeans cluster was run. We ran a total of 6 clusters and set n as 25, so that 25 samples of clusters were created and the clusters with the lowest within cluster sum of squares were kept.

```
##       chatter current_events       travel photo_sharing uncategorized
## 1 -0.01978665    -0.06523228 -0.213064232   -0.14668700   -0.09020329
## 2 -0.01322585     0.02584364 -0.149652050   -0.01500432    0.16376774
## 3 -0.04392445     0.11791953 -0.104926567   -0.02887017   -0.07188474
## 4  0.01997425    -0.01441394  0.001179381    0.04762999    0.11220114
## 5  0.04239873     0.11103642  1.762348461   -0.05700940   -0.03989065
## 6  0.16937879     0.19897957 -0.039478942    1.25267323    0.51629414
##        tv_film sports_fandom   politics        food      family home_and_garden
## 1 -0.044099066    -0.2881426 -0.2565449 -0.35390342 -0.25620888      -0.1098637
## 2 -0.051866184    -0.1994093 -0.1763095  0.41219220 -0.07943388       0.1559208
## 3 -0.008218818     1.9956542 -0.2036725  1.78927038  1.44172188       0.1677619
## 4  0.442380560    -0.1157139 -0.1647376 -0.07504745  0.18657154       0.1280783
## 5  0.077844260     0.1946632  2.3660540  0.02470268  0.04381456       0.1231953
## 6  0.009090473    -0.1942673 -0.1141903 -0.17775062  0.04790152       0.1611161
##         music        news online_gaming     shopping health_nutrition
## 1 -0.11832619 -0.24413216   -0.23149592 -0.060696813      -0.32804349
## 2  0.05802189 -0.04750369   -0.13504771  0.037219717       2.10096843
## 3  0.05427946 -0.07964823   -0.07525406  0.045501429      -0.15986076
## 4  0.33128838 -0.19217111    3.08639816 -0.016286239      -0.18239834
## 5 -0.03738467  1.95767757   -0.14154027 -0.005308791      -0.20638699
## 6  0.56509367 -0.06963991   -0.05307164  0.380921908      -0.06377776
##    college_uni sports_playing    cooking         eco   computers    business
## 1 -0.224110093    -0.22566789 -0.3311775 -0.15952897 -0.23314415 -0.12540115
## 2 -0.207910291    -0.03139765  0.3737284  0.52691392 -0.07346128  0.06780737
## 3 -0.122811520     0.10111053 -0.1177924  0.19540434  0.06853509  0.11561728
## 4  3.079589514     1.99975960 -0.1542229 -0.03226477 -0.05589906  0.03025585
## 5 -0.079836272    -0.01040025 -0.2153661  0.10592762  1.54693130  0.35651739
## 6 -0.003849398     0.18236003  2.5696137  0.08586275  0.07153792  0.28699368
##      outdoors      crafts  automotive          art    religion     beauty
## 1 -0.31504294 -0.18704023 -0.18219110 -0.062587846 -0.29777139 -0.2647399
## 2  1.62031045  0.09006259 -0.12083117  0.009414380 -0.17436406 -0.2108243
## 3 -0.08066041  0.70076128  0.16504066  0.084783773  2.17949821  0.2944705
## 4 -0.10093331  0.10468027  0.04616838  0.308653617 -0.12952947 -0.1982355
## 5  0.11058686  0.15290925  1.11088881 -0.003891175 -0.03374476 -0.1743681
## 6  0.03420672  0.15120803  0.05439785  0.136795837 -0.11806235  2.3894052
##     parenting      dating      school personal_fitness      fashion
## 1 -0.30516086 -0.09401577 -0.24447607      -0.33344148 -0.263936411
## 2 -0.10620247  0.18305088 -0.14434820       2.07047764 -0.107018846
## 3  2.06902438  0.03877666  1.63137287      -0.11432402  0.006917579
## 4 -0.14801846  0.01652805 -0.20324072      -0.18926634 -0.054189180
## 5  0.01707471  0.20088949 -0.03525502      -0.19210751 -0.179364771
## 6 -0.06535754  0.15733694  0.19696343      -0.04579539  2.498279621
##   small_business         spam        adult
## 1    -0.09609806  0.004313442  0.007570295
## 2    -0.06915453  0.003346418  0.007099232
## 3     0.10384435 -0.014411952 -0.003722006
## 4     0.22238753  0.031711284  0.032585000
## 5     0.23867309 -0.007267965 -0.092230656
## 6     0.27603909 -0.035852804  0.018725972
```

Above, we see what is in cluster 1. Each number is the coordinate for each feature for that centroid (cluster) on a z score scale.

```
##          chatter   current_events           travel    photo_sharing
```

```
##      4.328927076     1.443489755     1.098039216     2.296100463
##    uncategorized         tv_film   sports_fandom        politics
##      0.728574576     0.997135933     0.971359330     1.011015642
##             food          family home_and_garden           music
##      0.769112139     0.573694646     0.439744437     0.557391496
##             news   online_gaming        shopping health_nutrition
##      0.692663582     0.586693104     1.279576999     1.092311082
##      college_uni  sports_playing         cooking             eco
##      0.900198282     0.419035030     0.862304472     0.389513109
##        computers        business        outdoors          crafts
##      0.374091210     0.336417713     0.401630315     0.363075567
##       automotive             art        religion          beauty
##      0.580964970     0.622824411     0.525225821     0.353602115
##        parenting          dating          school personal_fitness
##      0.458911655     0.543291474     0.477197621     0.660057281
##          fashion  small_business            spam           adult
##      0.513989866     0.276933245     0.006829698     0.417052214

##          chatter  current_events          travel   photo_sharing
##      4.352080990     1.559055118     1.242969629     2.655793026
##    uncategorized         tv_film   sports_fandom        politics
##      0.966254218     0.984251969     1.163104612     1.254218223
##             food          family home_and_garden           music
##      2.129358830     0.773903262     0.635545557     0.739032621
##             news   online_gaming        shopping health_nutrition
##      1.105736783     0.845894263     1.456692913    12.013498313
##      college_uni  sports_playing         cooking             eco
##      0.947131609     0.608548931     3.280089989     0.917885264
##        computers        business        outdoors          crafts
##      0.562429696     0.470191226     2.742407199     0.589426322
##       automotive             art        religion          beauty
##      0.664791901     0.740157480     0.761529809     0.425196850
##        parenting          dating          school personal_fitness
##      0.760404949     1.037120360     0.596175478     6.442069741
##          fashion  small_business            spam           adult
##      0.800899888     0.293588301     0.006749156     0.416197975

##          chatter  current_events          travel   photo_sharing
##      4.243741765     1.675889328     1.345191041     2.617918314
##    uncategorized         tv_film   sports_fandom        politics
##      0.745718050     1.056653491     5.906455863     1.171277997
##             food          family home_and_garden           music
##      4.574440053     2.496706192     0.644268775     0.735177866
##             news   online_gaming        shopping health_nutrition
##      1.038208169     1.006587615     1.471673254     1.848484848
##      college_uni  sports_playing         cooking             eco
##      1.193675889     0.737812912     1.594202899     0.662714097
##        computers        business        outdoors          crafts
##      0.729907773     0.503293808     0.685111989     1.088274045
##       automotive             art        religion          beauty
##      1.055335968     0.862977602     5.268774704     1.096179183
##        parenting          dating          school personal_fitness
##      4.056653491     0.779973650     2.706192358     1.187088274
##          fashion  small_business            spam           adult
```

```
##      1.009222661      0.400527009      0.005270092      0.396574440

##          chatter   current_events           travel    photo_sharing
##      4.469248292      1.507972665      1.587699317      2.826879271
##    uncategorized          tv_film    sports_fandom         politics
##      0.917995444      1.804100228      1.343963554      1.289293850
##             food           family  home_and_garden            music
##      1.264236902      1.075170843      0.615034169      1.020501139
##             news    online_gaming         shopping health_nutrition
##      0.801822323      9.503416856      1.359908884      1.747152620
##      college_uni   sports_playing          cooking              eco
##     10.471526196      2.589977221      1.469248292      0.487471526
##        computers         business         outdoors           crafts
##      0.583143508      0.444191344      0.660592255      0.601366743
##       automotive              art         religion           beauty
##      0.892938497      1.227790433      0.847380410      0.441913440
##        parenting           dating           school personal_fitness
##      0.697038724      0.740318907      0.526195900      1.006833713
##          fashion   small_business             spam            adult
##      0.897494305      0.473804100      0.009111617      0.462414579

##          chatter   current_events           travel    photo_sharing
##      4.548387097      1.667155425      5.612903226      2.541055718
##    uncategorized          tv_film    sports_fandom         politics
##      0.775659824      1.199413490      2.014662757      8.960410557
##             food           family  home_and_garden            music
##      1.441348974      0.913489736      0.611436950      0.640762463
##             news    online_gaming         shopping health_nutrition
##      5.318181818      0.828445748      1.379765396      1.639296188
##      college_uni   sports_playing          cooking              eco
##      1.318181818      0.629032258      1.259530792      0.593841642
##        computers         business         outdoors           crafts
##      2.473607038      0.670087977      0.916422287      0.640762463
##       automotive              art         religion           beauty
##      2.347507331      0.718475073      1.030791789      0.473607038
##        parenting           dating           school personal_fitness
##      0.947214076      1.068914956      0.725806452      1.000000000
##          fashion   small_business             spam            adult
##      0.668621701      0.483870968      0.005865103      0.236070381

##          chatter   current_events           travel    photo_sharing
##      4.996515679      1.778745645      1.494773519      6.118466899
##    uncategorized          tv_film    sports_fandom         politics
##      1.296167247      1.085365854      1.174216028      1.442508711
##             food           family  home_and_garden            music
##      1.081881533      0.918118467      0.639372822      1.261324042
##             news    online_gaming         shopping health_nutrition
##      1.059233449      1.066202091      2.078397213      2.280487805
##      college_uni   sports_playing          cooking              eco
##      1.538327526      0.817073171     10.811846690      0.578397213
##        computers         business         outdoors           crafts
##      0.733449477      0.621951220      0.824041812      0.639372822
##       automotive              art         religion           beauty
```

```
##      0.904181185     0.947735192     0.869337979     3.878048780
##        parenting          dating          school personal_fitness
##      0.822299652     0.991289199     1.001742160     1.351916376
##          fashion  small_business            spam           adult
##      5.564459930     0.506968641     0.003484321     0.437282230
```

by multiplying by the standard deviation and adding the mean, we see something that is more useful and can now compare the numbers for each category between each cluster, and try to identify what goes in these clusters.

Multiplying by sigma and adding mu is a way of unscaling the data and recentering it. The result we get from running this line is the unscaled version. this is much easier to interpret it so we don't have to interpret it based on z scores or standard deviations.

We defined clusters as being groups of correlated interests based on what these users tweeted about. The clusters are as follows:

Cluster 1 had higher numbers for personal fitness and health nutrition. Our group concluded that these might be people who are really into working out and eating healthy. Cluster 2 had high numbers for online gaming, sports playing, and college and university. We concluded that these are likely college aged gamers. Cluster 3 had high numbers for politics, computers, news. Politics was the highest here by far so we concluded that this cluster is likely made up of politically engaged people. Cluster 4 had high numbers for food, religion, parenting, school, sports, fandom, and crafts. This is a widely distributed amount of interests, but we concluded that since parenting had a particularly high number, the cluster could be made up of parents. Most of the topics are things parents might talk about (school, and sports). Cluster 5 had high numbers for fashion, cooking, photo sharing, beauty. Our group thought this could be fashion bloggers or lifestyle influencers. Cluster 6 was made up of all negative numbers except for spam and adult. As a result, we concluded that these are spam twitter accounts with explicit adult content.

Above is another look at which categories are in which clusters.

Here, we apply the k means++.

```
##          chatter  current_events          travel   photo_sharing
##      4.328927076     1.443489755     1.098039216     2.296100463
##    uncategorized         tv_film   sports_fandom        politics
##      0.728574576     0.997135933     0.971359330     1.011015642
##             food          family home_and_garden           music
##      0.769112139     0.573694646     0.439744437     0.557391496
##             news   online_gaming        shopping health_nutrition
##      0.692663582     0.586693104     1.279576999     1.092311082
##      college_uni   sports_playing         cooking             eco
##      0.900198282     0.419035030     0.862304472     0.389513109
##        computers        business        outdoors          crafts
##      0.374091210     0.336417713     0.401630315     0.363075567
##       automotive             art        religion          beauty
##      0.580964970     0.622824411     0.525225821     0.353602115
##        parenting          dating          school personal_fitness
##      0.458911655     0.543291474     0.477197621     0.660057281
##          fashion  small_business            spam           adult
##      0.513989866     0.276933245     0.006829698     0.417052214
```

```
##          chatter  current_events          travel   photo_sharing
##      4.243741765     1.675889328     1.345191041     2.617918314
##    uncategorized         tv_film   sports_fandom        politics
##      0.745718050     1.056653491     5.906455863     1.171277997
```

```
##            food          family  home_and_garden           music
##     4.574440053     2.496706192     0.644268775     0.735177866
##            news   online_gaming        shopping health_nutrition
##     1.038208169     1.006587615     1.471673254     1.848484848
##     college_uni  sports_playing         cooking             eco
##     1.193675889     0.737812912     1.594202899     0.662714097
##       computers        business        outdoors          crafts
##     0.729907773     0.503293808     0.685111989     1.088274045
##      automotive             art        religion          beauty
##     1.055335968     0.862977602     5.268774704     1.096179183
##       parenting          dating          school personal_fitness
##     4.056653491     0.779973650     2.706192358     1.187088274
##         fashion  small_business            spam           adult
##     1.009222661     0.400527009     0.005270092     0.396574440
```

```
##         chatter  current_events          travel   photo_sharing
##     4.469248292     1.507972665     1.587699317     2.826879271
##   uncategorized         tv_film   sports_fandom        politics
##     0.917995444     1.804100228     1.343963554     1.289293850
##            food          family  home_and_garden           music
##     1.264236902     1.075170843     0.615034169     1.020501139
##            news   online_gaming        shopping health_nutrition
##     0.801822323     9.503416856     1.359908884     1.747152620
##     college_uni  sports_playing         cooking             eco
##    10.471526196     2.589977221     1.469248292     0.487471526
##       computers        business        outdoors          crafts
##     0.583143508     0.444191344     0.660592255     0.601366743
##      automotive             art        religion          beauty
##     0.892938497     1.227790433     0.847380410     0.441913440
##       parenting          dating          school personal_fitness
##     0.697038724     0.740318907     0.526195900     1.006833713
##         fashion  small_business            spam           adult
##     0.897494305     0.473804100     0.009111617     0.462414579
```

Here, we initialize the k means and see the values per cluster per topic.

```
## [1] 89277.35 27194.42 29625.16 16467.02 29195.01 22721.60
```

```
## [1] 89277.35 29625.16 27194.42 16467.02 29195.01 22721.60
```

These are vectors of the within cluster sum of square for each cluster. This is done in preparation for the next step, which is finding the sum. The values from clust1 (the clustering done without k means) are 89277.35, 27194.42, 29625.16, 16467.02, 29195.01, and 22721.60. The values from clust2 (the clustering done with k means++) are 89277.35, 29625.16, 27194.42, 16467.02, 29195.01, and 22721.60.

```
## [1] 214480.6
```

```
## [1] 214480.6
```

This gives us the within sample sum of squares. The result is 214480.6.

```
## [1] 214480.6
```

```
## [1] 214480.6
```

As an alternative, we used this method, which also gives us total sum of squares.Again, the value is 214480.6.

```
## [1] 69235.45
```

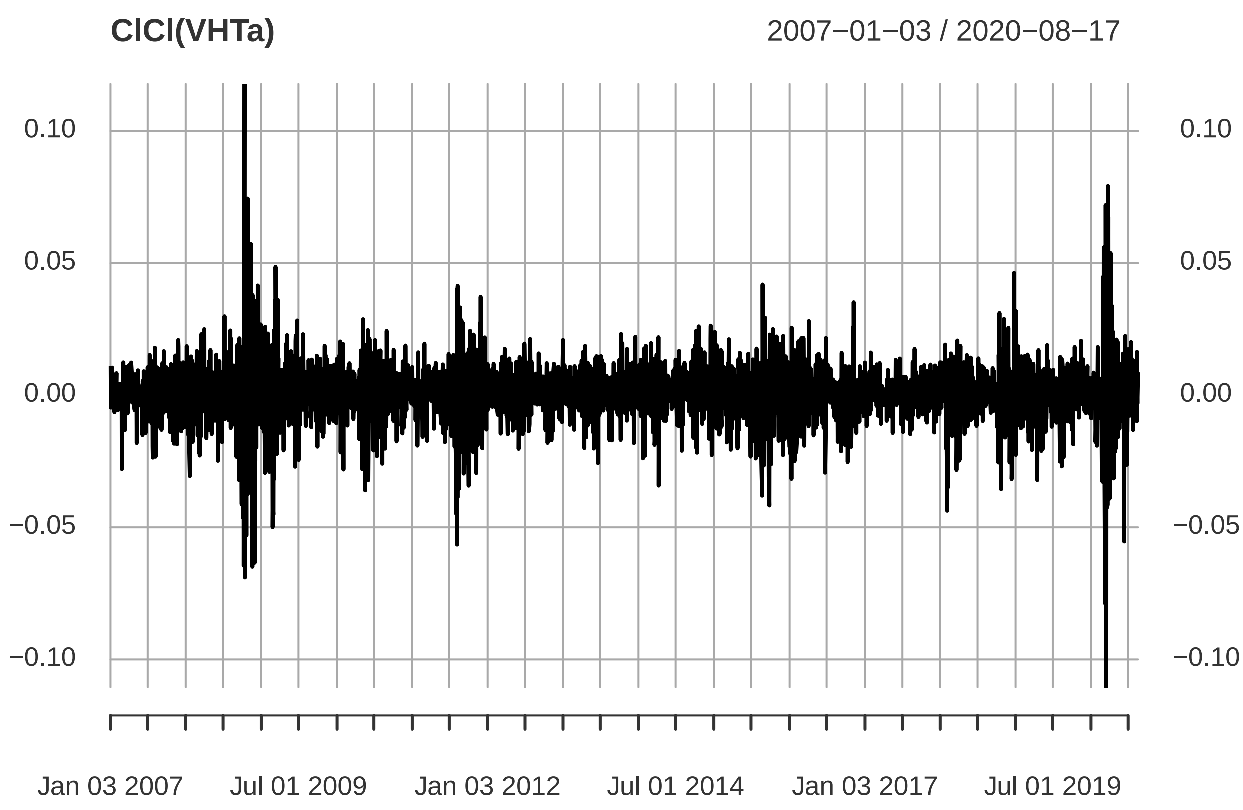
```
## [1] 69235.45
```

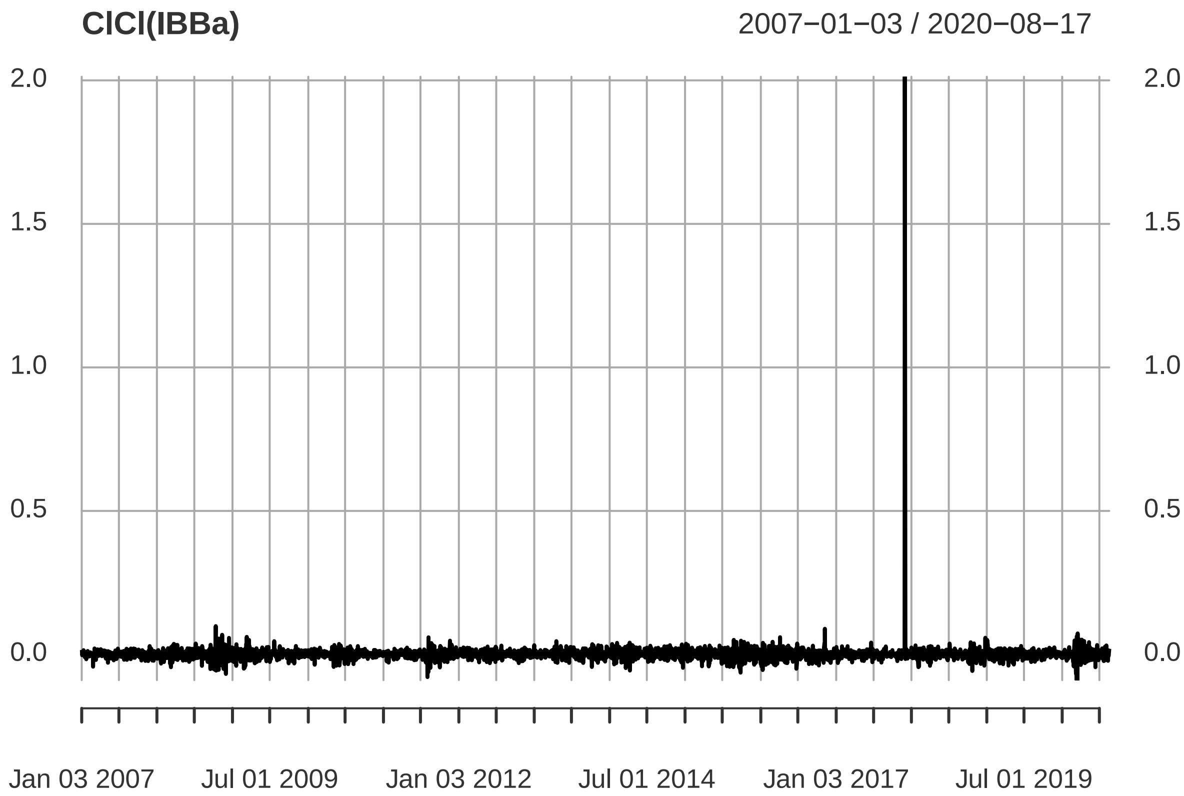
This gives us the between sample sum of squares, which we expect to be high. The value is 69235.45.

Portfolio Modeling Report

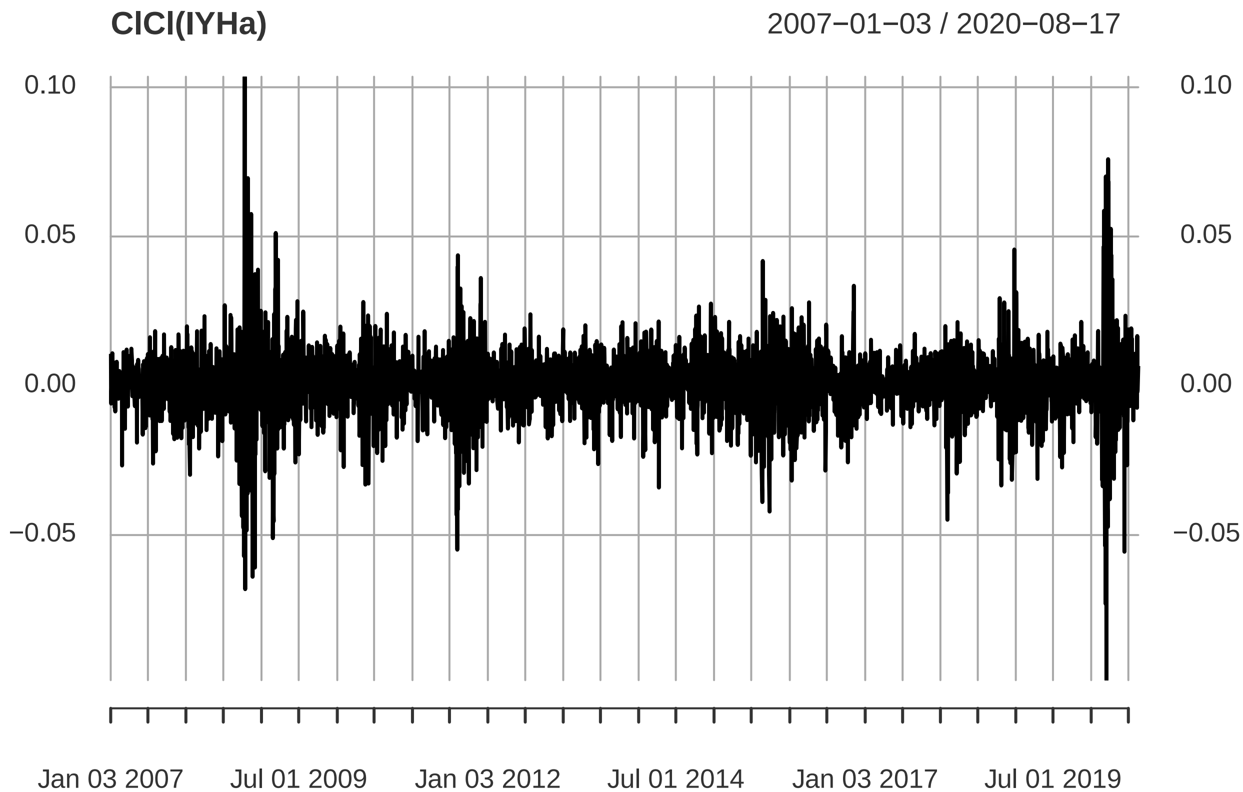
Our group chose the biotech industry and imported three stocks.

Stocks are sometimes split up from EFTs in order to pay dividends, which affects the price of the stock, so we had to adjust for this.



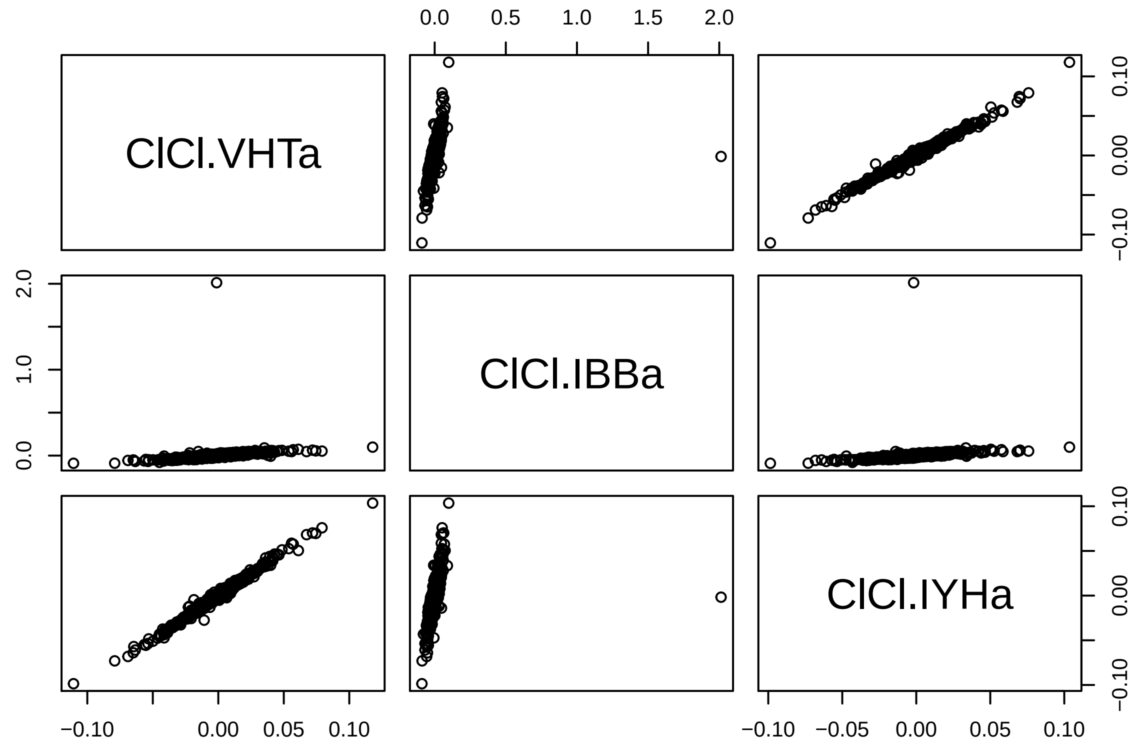


ClCl(IBBa)
2007−01−03 / 2020−08−17
2.0
1.5
1.0
0.5
0.0
2.0
1.5
1.0
0.5
0.0
Jan 03 2007
Jul 01 2009
Jan 03 2012
Jul 01 2014
Jan 03 2017
Jul 01 2019

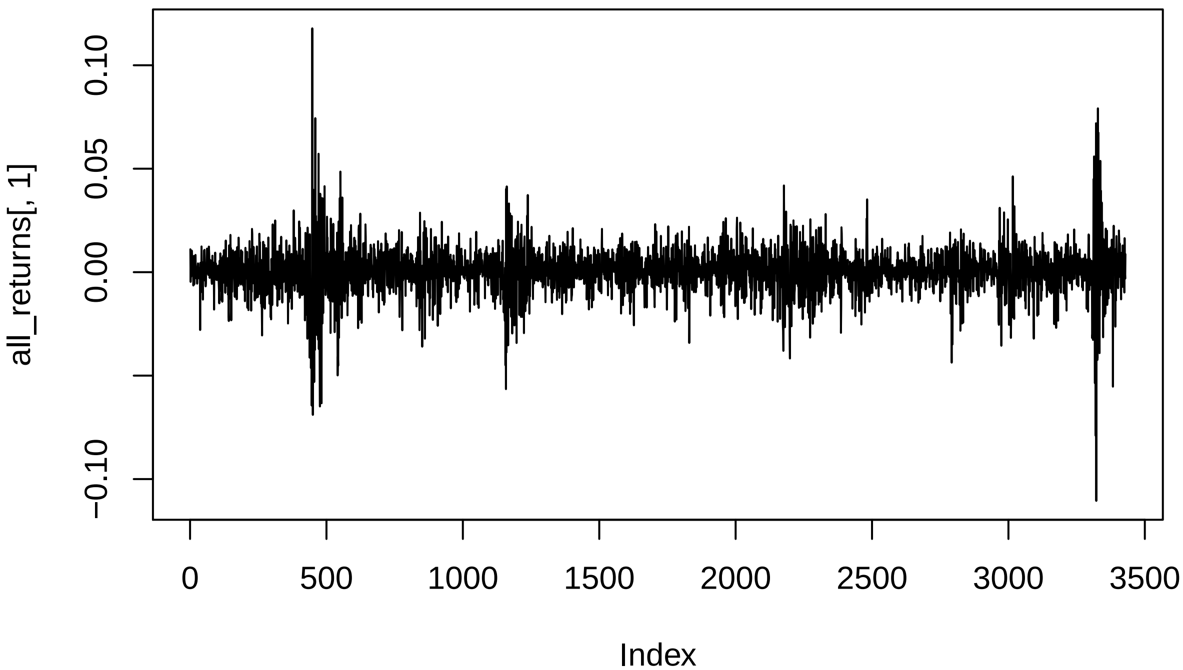


The fluctuation in each stock over time is visible in the plots.

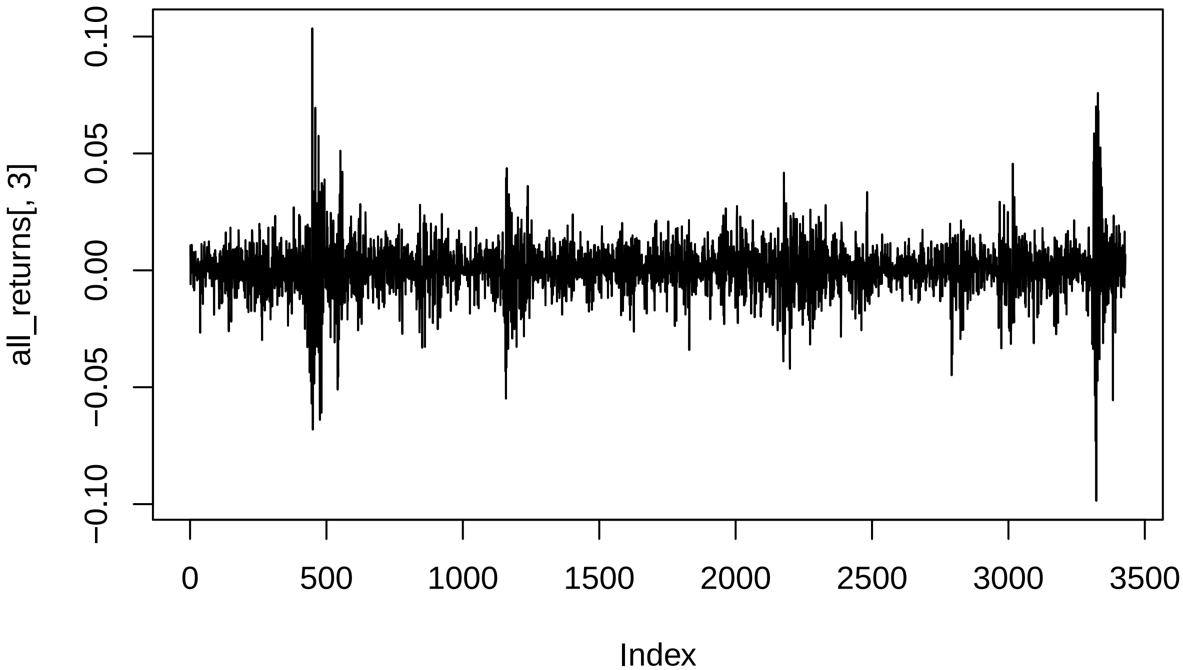
The seed was set in order to create reproducible results. THen the changes were bound into a single matrix.



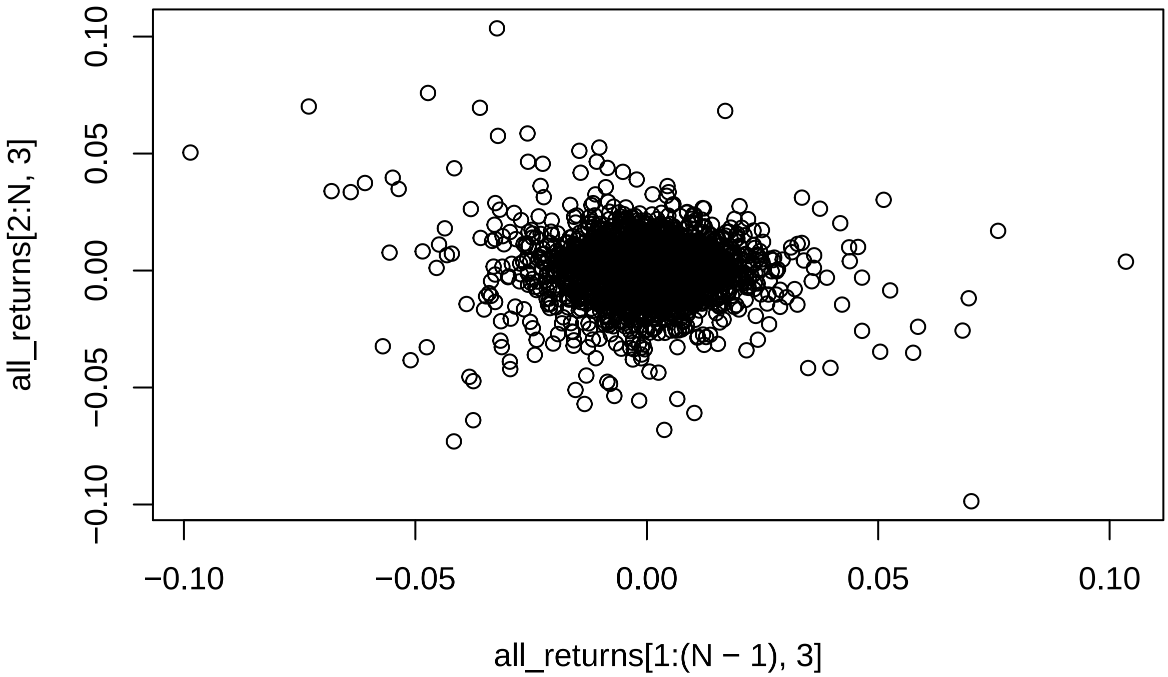
ClCl.VHTa
ClCl.IBBa
ClCl.IYHa
0.0
0.5
1.0
1.5
2.0
−0.10
0.00
0.05
0.10
−0.05



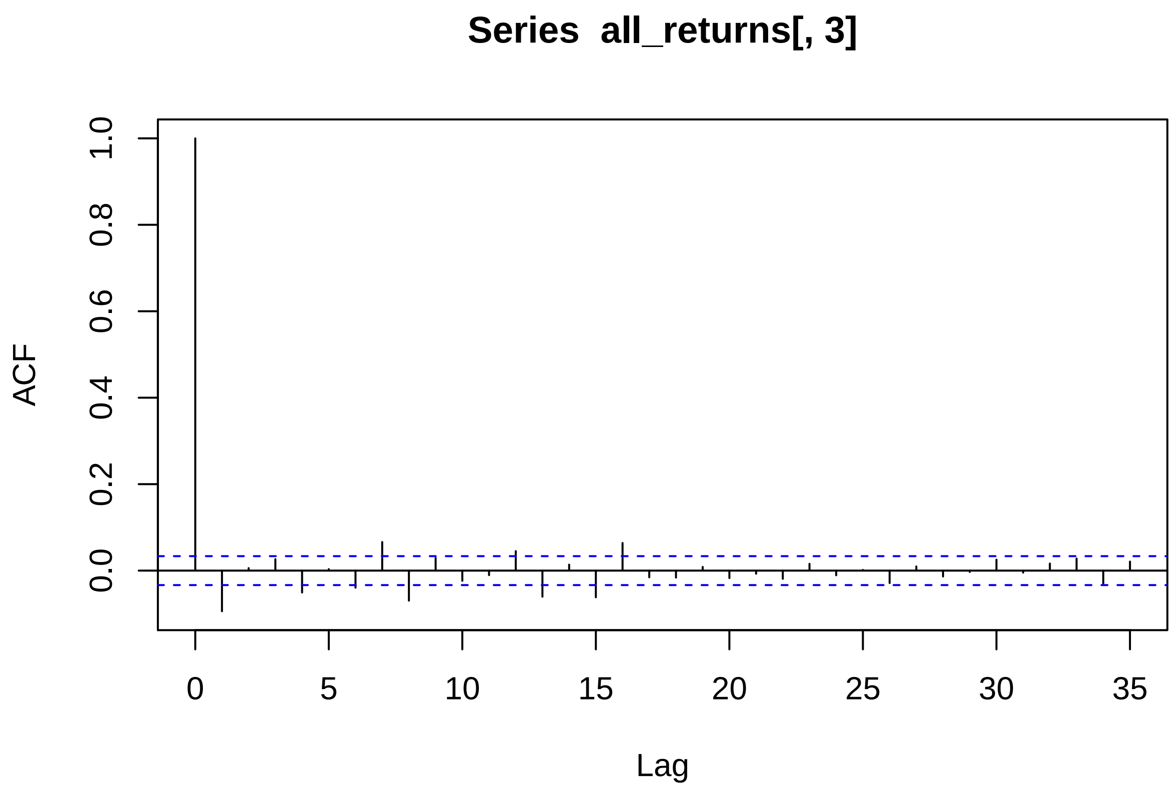
We can see that there is a strong correlation. These returns can be viewed as draws from the joint distribution



Above, we have plotted the market returns over time.



In this plot, we wanted to see whether the returns from one day are correlated with the returns from the subsequent day. This does not appear to be true as there is no visible correlation.



Our autocorrelation plot did not return anything.

```
##               ClCl.VHTa   ClCl.IBBa    ClCl.IYHa
## 2011-01-14 0.0008687924 0.001457219 0.0008996252
```

Above, we sampled a random return from the empirical joint distribution. This is a simulation of a random day and the returns from that day.

Above, we set up each portfolio. The total wealth was set at $100000, and the weights of each stock varied per portfolio. Portfolio 1 had weights of 0.3,0.2, and 0.5. Portfolio 2 had weights of 0.5, 0.4, and 0.1. Potfolio 3 had weights of 0.1, 0.6, and 0.3. These weights correspond to VHT, IBB, and IYH respectively.

```
##            ClCl.VHTa ClCl.IBBa ClCl.IYHa
## 2011-01-14  30026.06  20029.14  50044.98
```

```
## [1] 100100.2
```

```
##            ClCl.VHTa ClCl.IBBa ClCl.IYHa
## 2011-01-14  50043.44  40058.29     10009
```

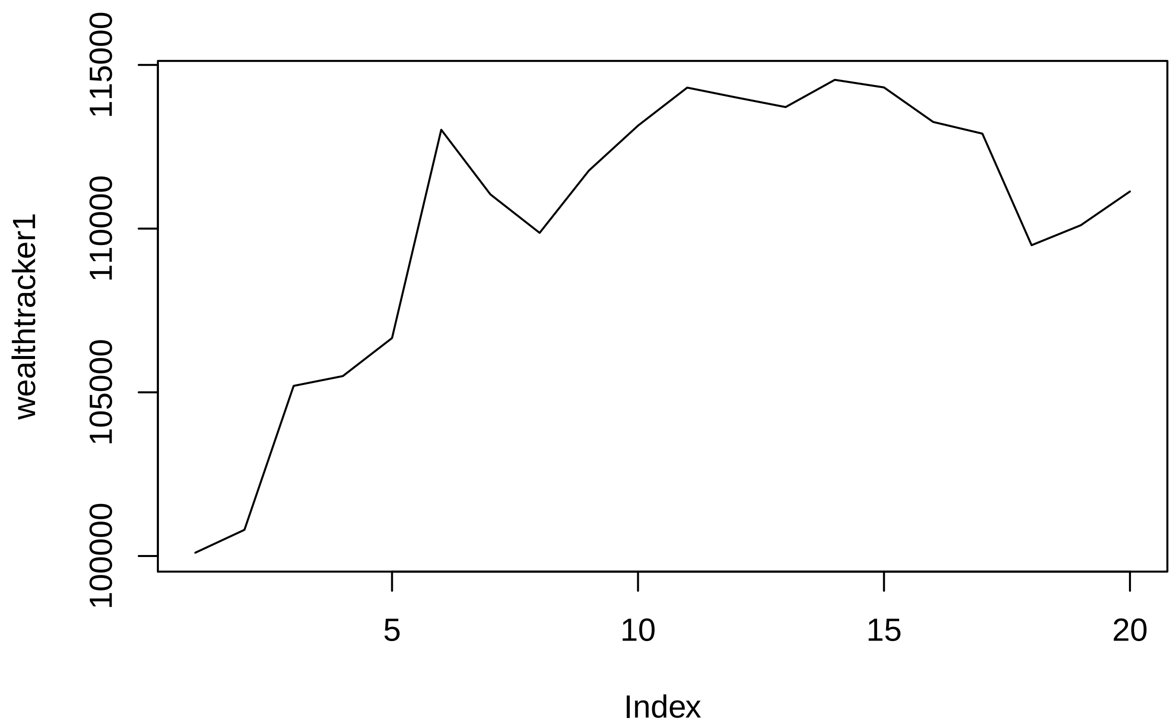
```
## [1] 100110.7
```

```
##            ClCl.VHTa ClCl.IBBa ClCl.IYHa
## 2011-01-14  10008.69  60087.43  30026.99
```

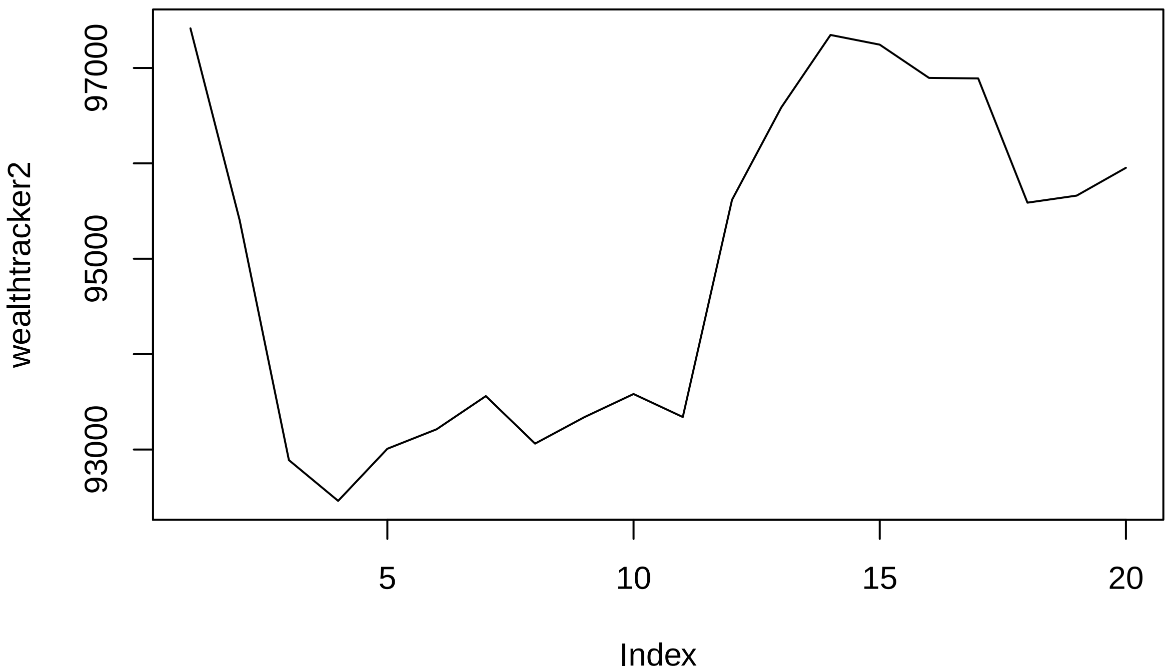
```
## [1] 100123.1
```

The total wealth was then calculated per portfolio, taking into account the weights of the ETFs and the value of each respective ETF. The total wealth are 100100.2, 100110.7, and 100123.1 for portfolios 1, 2 and 3 respectively.

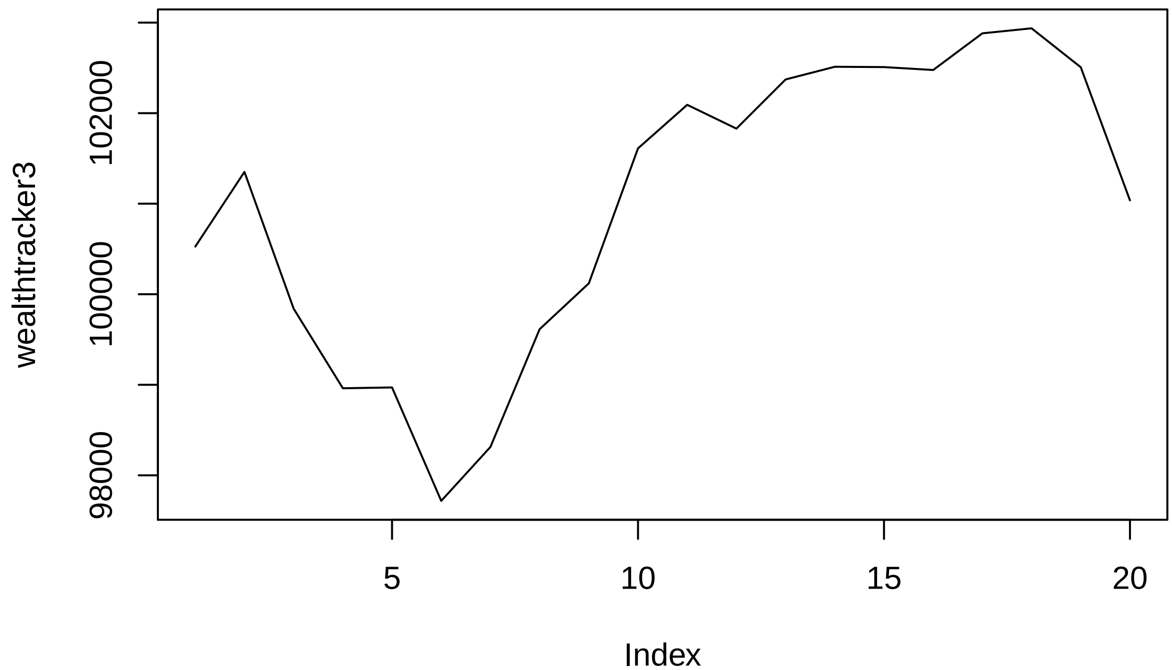
```
## [1] 111134.8
```




```
## [1] 95953.42
```

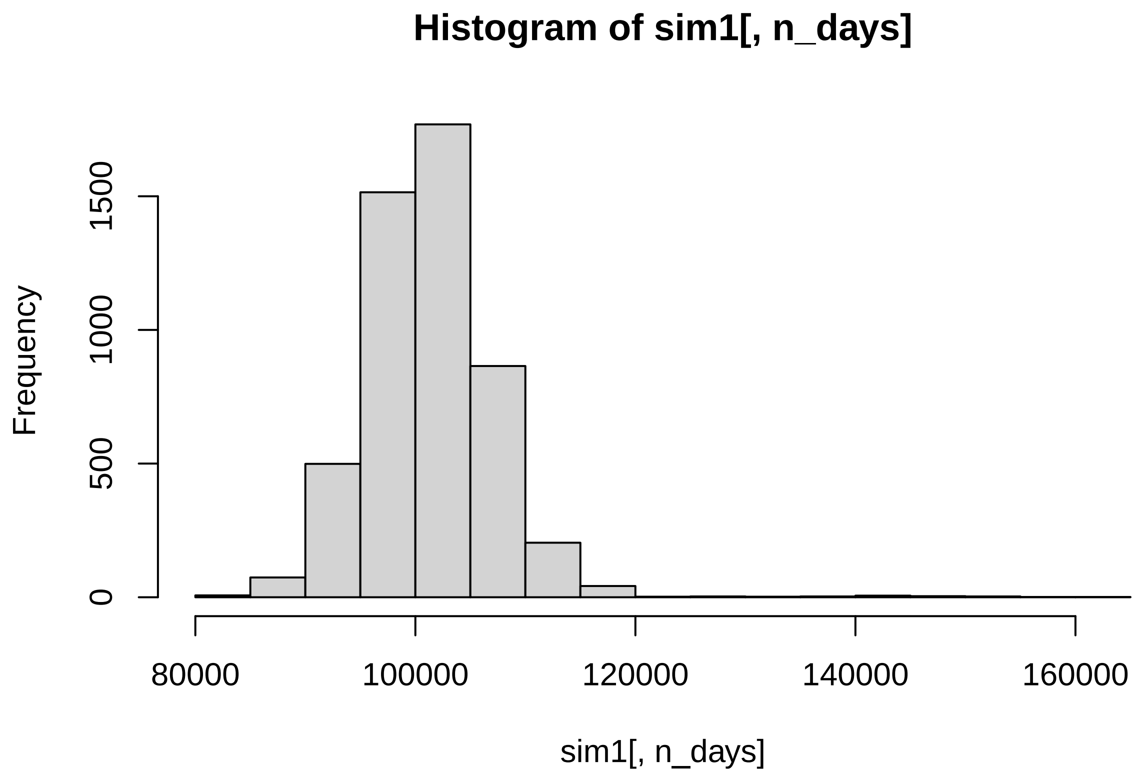



```
## [1] 101036.9
```

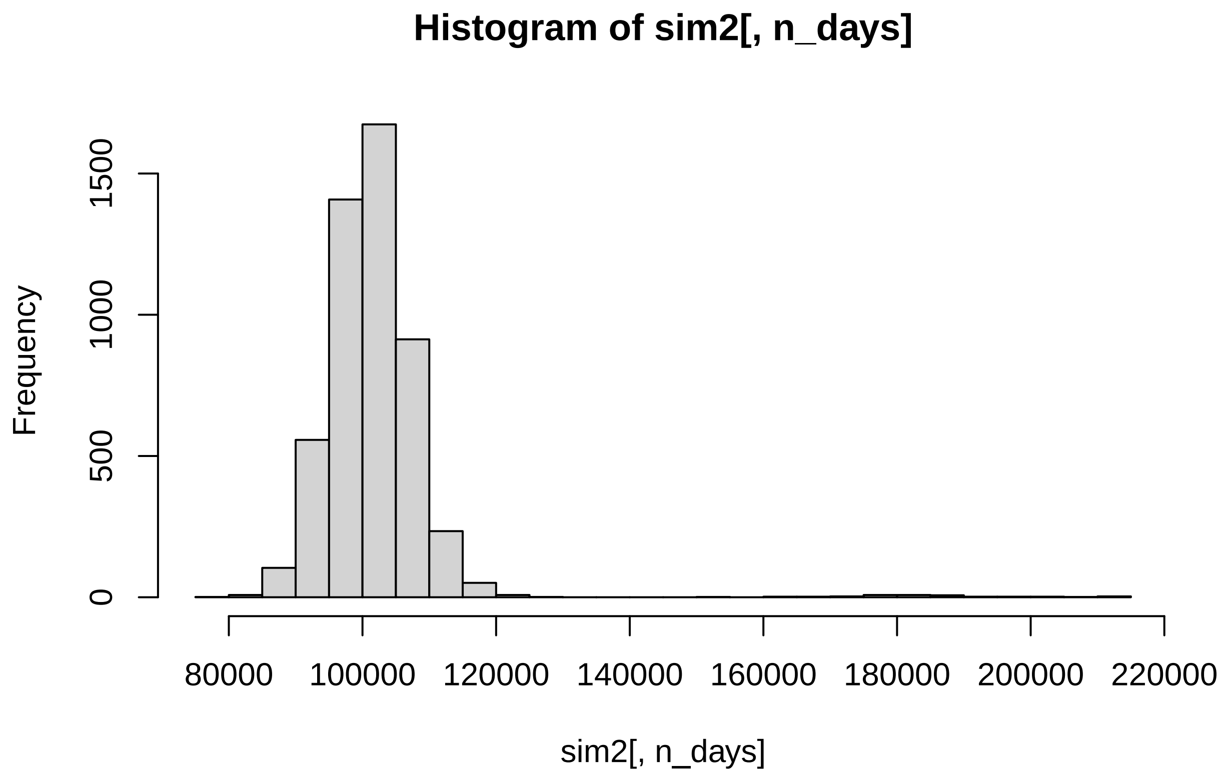


For each portfolio, we looped over two trading weeks and ran the code 5 times to account for variability in performance trajectory. each 'wealthtracker' tracks the total amount per day. Then the holdings are added with the returns from each day and the sum for each portfolio is the total wealth.

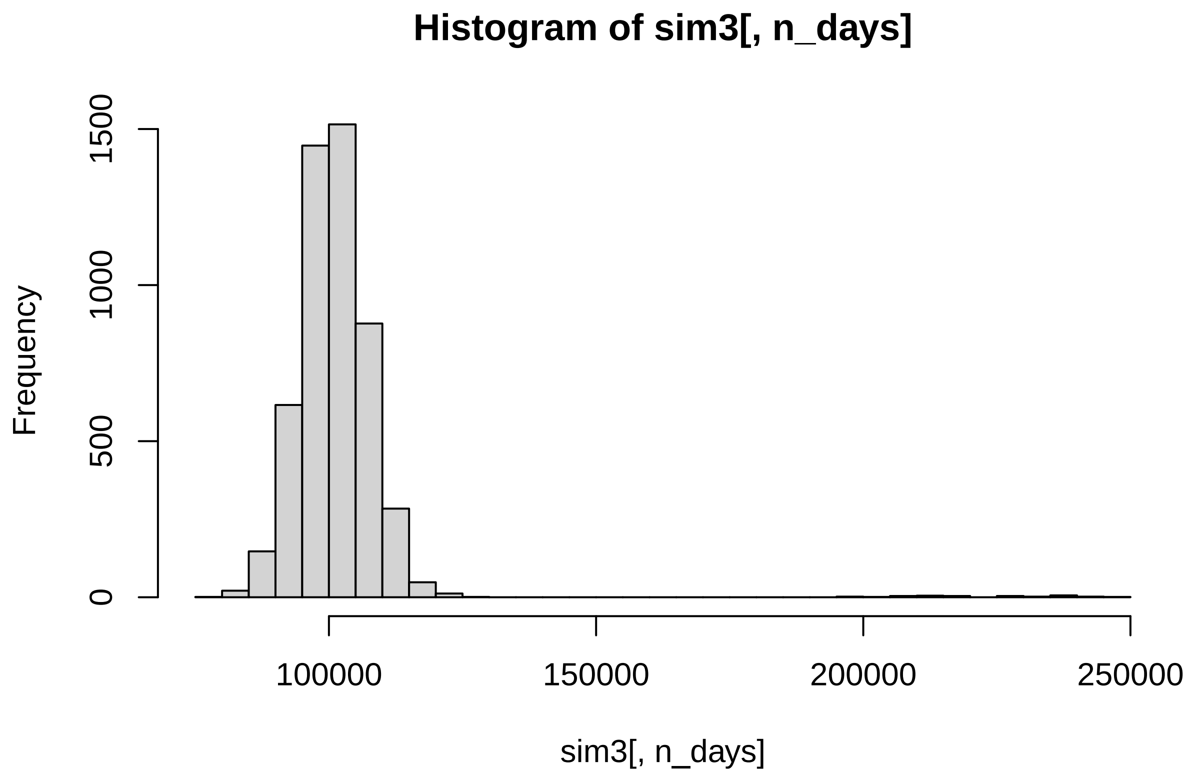
```
##                [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
## result.1 100100.19 100799.44 105196.20 105496.04 106656.96 113019.62 111044.44
## result.2  97694.21  95554.64  93525.00  93118.06  93564.32  93619.22  94052.33
## result.3 100484.15 100998.16  99836.90  98920.25  98996.73  97797.87  98691.59
## result.4 101176.66 101883.15 103127.57 104029.23 102925.60 103722.10 103055.35
## result.5 100137.27  99966.27  98819.22  99253.24  99108.84  98515.69  98990.56
## result.6 100843.57 100900.29 100750.66  98108.76  98382.38  98357.42  98156.94
##                [,8]      [,9]     [,10]     [,11]     [,12]     [,13]     [,14]
## result.1 109870.34 111770.07 113145.32 114306.75 114005.07 113712.59 114544.44
## result.2  93446.07  93706.71  94040.48  93993.61  96004.70  96831.97  97497.79
## result.3  99377.08  99962.18 101268.04 102284.17 101997.06 102481.38 102488.71
## result.4 104092.15 104683.73 105474.72 105851.49 108947.92 105534.45 105512.28
## result.5  98450.98  98360.96 100178.89  99866.16  99930.83  99082.61  99879.62
## result.6  98236.54  98575.47 102875.15 101890.14 102646.86 106180.07 105805.53
##               [,15]     [,16]     [,17]     [,18]     [,19]     [,20]
## result.1 114312.77 113257.66 112900.73 109493.23 110103.44 111134.76
## result.2  97355.97  97213.07  97090.30  96070.67  96110.17  96405.49
## result.3 102087.05 102367.67 102543.20 102790.80 102636.87 101571.52
## result.4 105682.83 107334.67 106994.78 106899.18 107400.19 106036.12
## result.5  99165.51  99455.76  99697.96  98400.65  97152.46  97385.97
## result.6 104832.26 105238.80 103774.38 102642.78 103140.63 101621.87
```




```
##                [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
## result.1  98792.31  98492.70  97199.70  96784.42  96447.11  96667.04  97021.11
## result.2 100124.36  96124.52  96523.59  97494.42  97622.73  97634.59  99113.50
## result.3 101150.25  98279.62  99302.57  99115.41  97579.51  96795.28  97112.15
## result.4 101200.37 101804.26 101667.37 101463.65 101308.91 102243.85 101163.13
## result.5 100229.97 100226.81 100994.71  97214.97  97019.84  95705.88  95826.71
## result.6 101216.53 100195.75  99344.85  98468.81  98332.50  98026.71  98830.78
##                [,8]      [,9]     [,10]     [,11]     [,12]     [,13]     [,14]
## result.1  98417.50  97777.22  97603.36  99442.82  98258.01  98611.52  97511.57
## result.2 100480.34  98892.35  98807.22  98018.14  96986.61  96969.55  97041.72
## result.3  98836.27  99246.58  99931.88 100302.98 100062.35 100964.80 102290.34
## result.4 103220.06 102699.03 102122.37 101471.84  96372.20  95700.50  96390.67
## result.5  96261.47  96851.39  99116.25  98703.11  98987.79  99379.08  98270.98
## result.6  99144.25  98267.11 102116.13 101325.40 103051.51 100806.04 100529.01
##               [,15]     [,16]     [,17]    [,18]     [,19]     [,20]
## result.1  97378.02  97648.54  98708.68 99105.11 100022.97  99853.43
## result.2  97546.22  95667.66  95415.66 95716.24  98376.67 100638.08
## result.3 100744.94 100039.21 101180.68 99249.01  99608.51  98117.98
## result.4  95001.39  94784.27  95432.80 93928.37  94410.43  94858.23
## result.5  97191.35  97815.34  98248.45 98542.46  99833.79 100059.99
## result.6 100085.00 100025.34  99749.97 98965.17  97342.10  96371.81
```



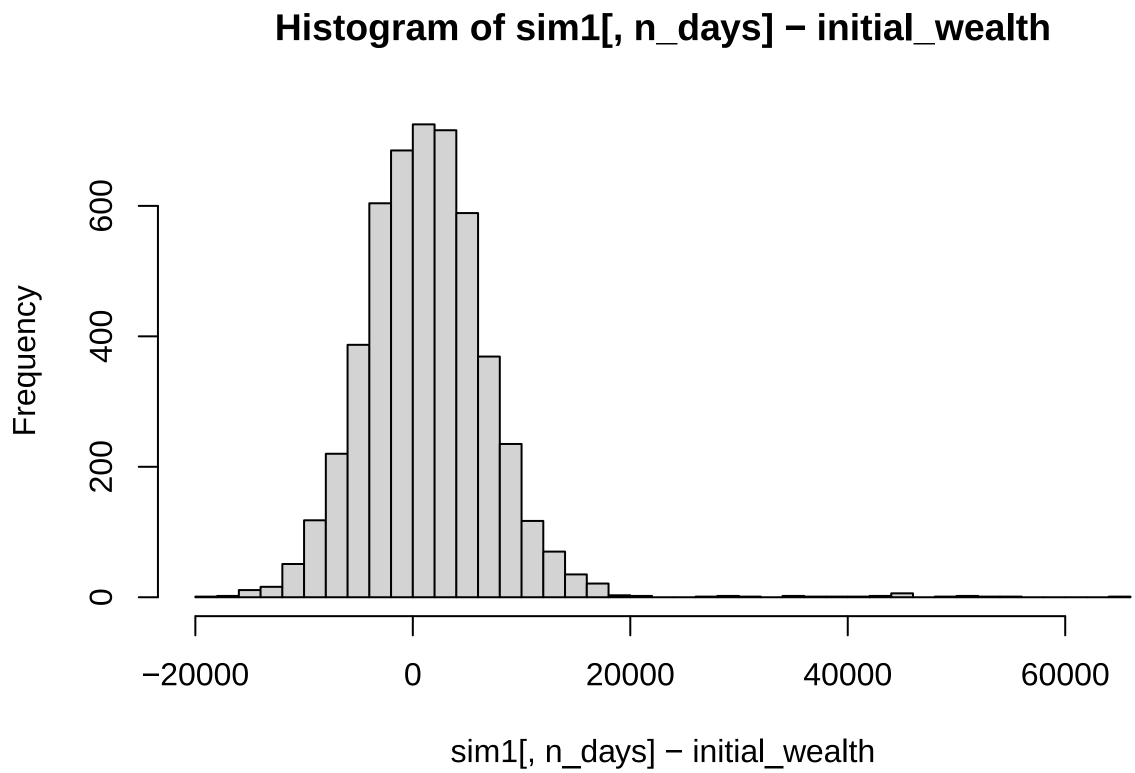

```
##                [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
## result.1  99690.49  99518.32  99149.47 100209.96  99627.44 100244.02  99877.02
## result.2  97755.96  96604.20  97991.76  98463.86  97517.45  98488.63  97628.34
## result.3 101319.73 101473.12 102323.30 103414.67 104047.90 103022.01 108738.82
## result.4  99401.57  98206.62  98555.09  99730.28 100120.50 102385.73 101007.87
## result.5  99947.67 101514.67 100267.94  99059.26 101042.75 102014.87 102629.21
## result.6 101909.86 105712.36 106582.98 106691.53 107468.32 107346.14 106266.44
##                [,8]      [,9]     [,10]     [,11]     [,12]     [,13]     [,14]
## result.1  98971.16 100884.07  99526.69  99746.18 103461.67 105268.07 104815.99
## result.2  96351.83  96648.11  96413.20  94814.67  94653.32  95077.94  95121.05
## result.3 109426.02 111423.28 113051.37 113102.65 111180.52 110625.71 110474.16
## result.4 101588.01 104978.96 105346.99 105950.85 106421.72 104944.09 102041.81
## result.5 102825.98 103673.62 105409.81 104109.81 104901.77 104488.58 103962.20
## result.6 104787.95 103664.12 105117.86 106875.61 105831.77 105328.17 106234.38
##               [,15]     [,16]     [,17]     [,18]     [,19]     [,20]
## result.1 104811.45 105297.76 106853.93 106781.90 105407.10 106654.32
## result.2  94993.45  92548.49  92546.35  93449.10  94693.56  97037.10
## result.3 110221.69 109439.20 110190.45 110763.21 112297.52 109932.73
## result.4 101817.93 101324.69 101452.76 100757.95 100023.99  98933.81
## result.5 102441.09 101209.00 100733.10  98855.46  99329.39  99046.86
## result.6 106444.48 106769.15 105935.67 104776.48 105932.95 107571.92
```



For each portfolio, we are simulating the trajectory of wealth over 20 days. This is done 5000 times for each portfolio to account for any variability in simulations. The 'wealthtrackers' tracks the wealth of each portfolio over this 20 business day period. In the histograms, each row is a simulated trajectory and each column is wealth data.

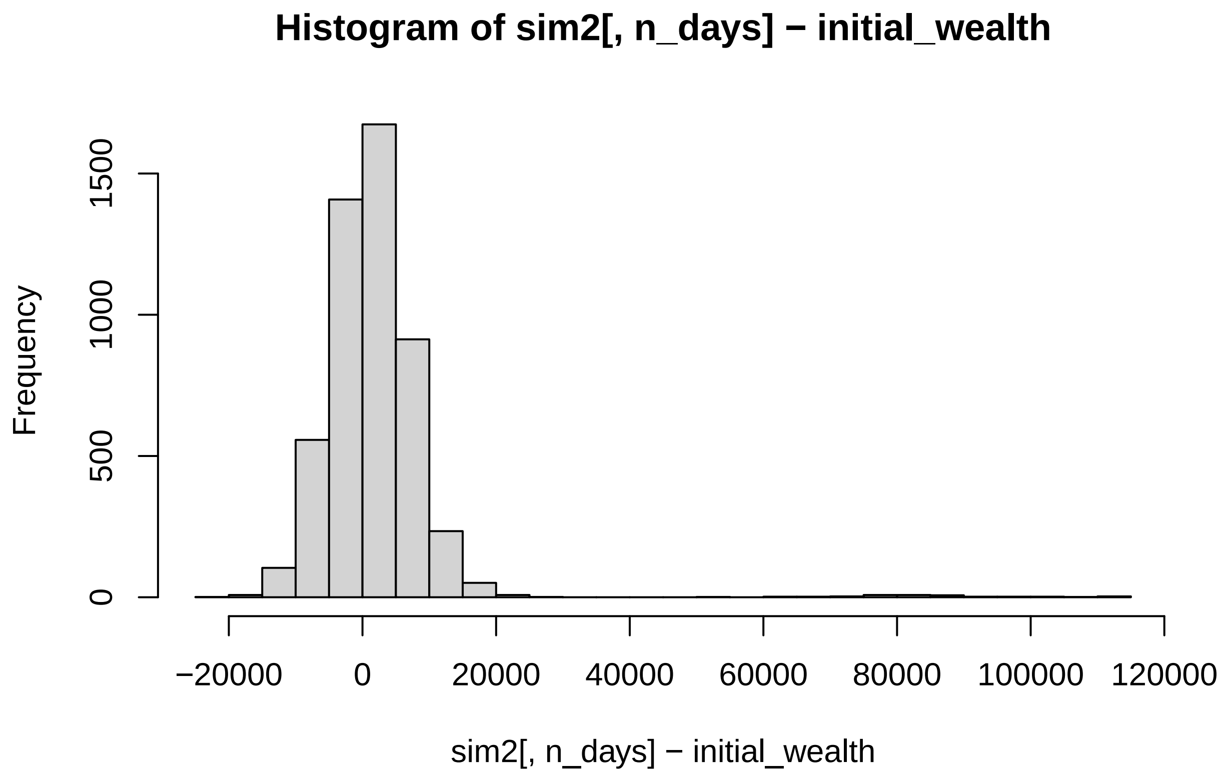
```
## [1] 101327.7
```

```
## [1] 1327.703
```



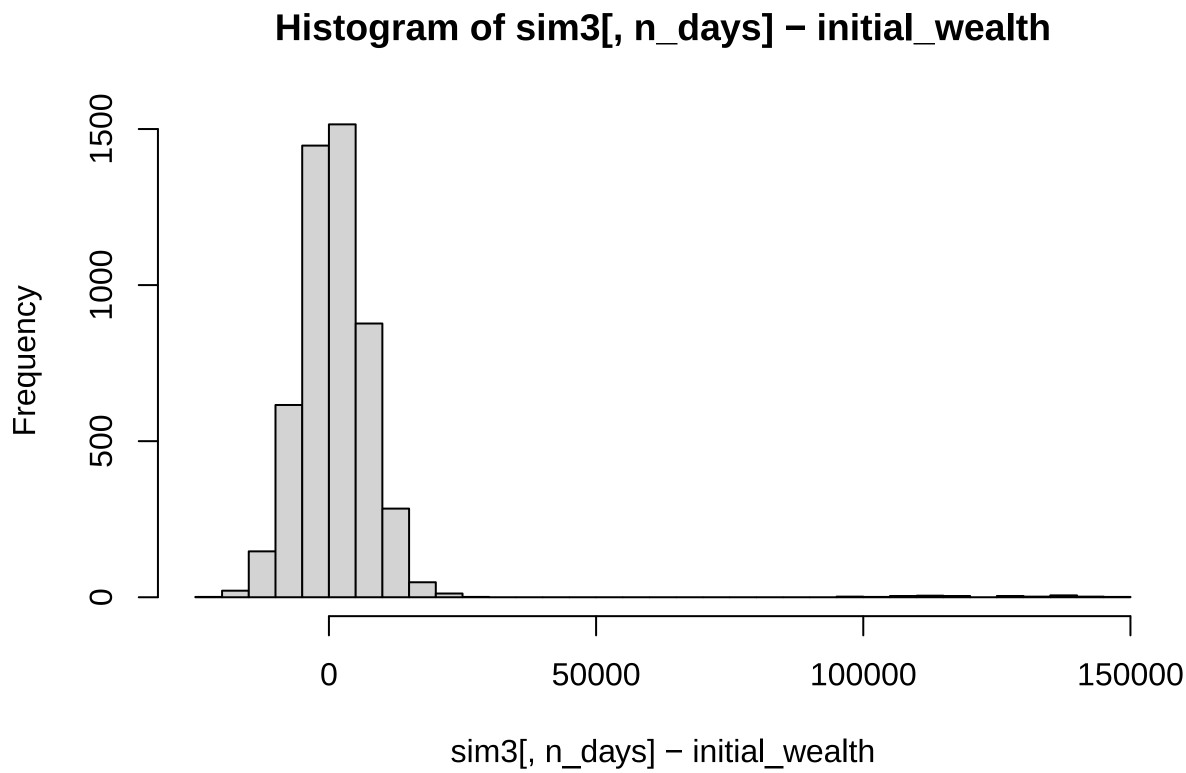

```
## [1] 101842.9
```

```
## [1] 1842.927
```




```
## [1] 101671.6
```

```
## [1] 1671.594
```



Here, the results of the simulations per portfolio are averaged in order to accounts for simulation variability. Then the initial wealth is subtracted in order to calculate the profit or loss of the portfolio. These results are plotted in histograms.

```
##        5%
## -7307.644
```

```
##        5%
## -7982.749
```

```
##        5%
## -8945.975
```

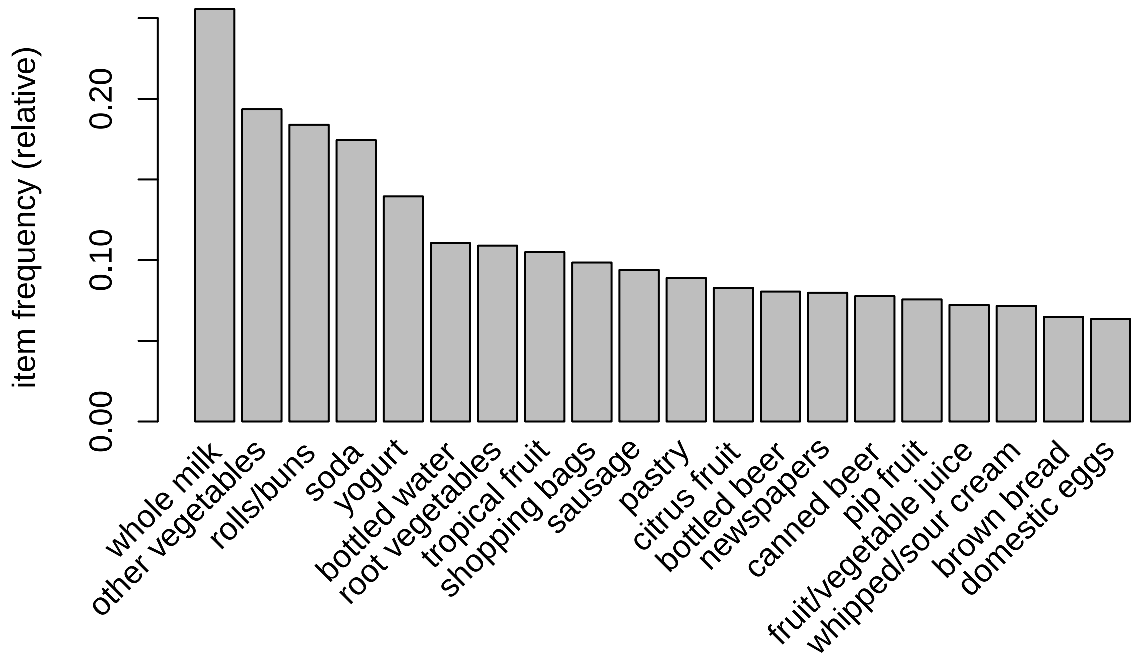
The value at risk is the amount that a portfolio might lost at a given percentage in a normal day, given market conditions. Here, the 5% value at risk is calculted for each portfolio. The 4-week (20 trading day) value at risk of this first portfolio is -\7300.149. The 4-week (20 trading day) value at risk of this second portfolio is -\8001.347. The 4-week (20 trading day) value at risk of this third portfolio is -\8858.699. In this simulation, the first portfolio returns the lowest value at risk, but the third portfolio may return the most.

Association Rule Mining

```
## 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
```

From our summary, we can see a few things about our dataset. We see that we have 9835 rows (baskets) and 169 columns. Our most frequent items are whole milk, with 2513 transactions, other vegetables, with 1903 transactions, and rolls/buns, with 1809 transactions. Our density is 0.02609146, which is the percentage of non-empty cells. The density shows that 43,364 items were purchased throughout all of the transactions in the data. Another thing we can observe is that the large majority of transactions only had one item. The transaction with the highest number of items had 32 items, and there was only one such transaction. It's interesting that more people opted to buy fewer amounts of items at a time. We also see that 50 percent of the transactions had between 2 and 6 items in them.



Our frequency plot reflects the same things we see from the summary.

We then cast 'groceries' as a special arules "transactions" class. We did this so we can run the apriori algorithm, which is an algorithm for finding rules over a support threshold.

```
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##        0.25    0.1    1 none FALSE            TRUE       5   0.007      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: 68
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.01s].
## sorting and recoding items ... [104 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 3 4 done [0.00s].
## writing ... [363 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
```

```
##     lhs        rhs                support     confidence coverage   lift
```

```
## [1] {herbs} => {root vegetables}  0.007015760 0.43125    0.01626843 3.956477
## [2] {herbs} => {other vegetables} 0.007727504 0.47500    0.01626843 2.454874
## [3] {herbs} => {whole milk}       0.007727504 0.47500    0.01626843 1.858983
##     count
## [1] 69
## [2] 76
## [3] 76
```

Above we show the first 3 out of 9835 transactions as a sample of our data.

```
##     lhs        rhs               support    confidence coverage   lift
## [1] {herbs} => {root vegetables} 0.00701576 0.43125    0.01626843 3.956477
##     count
## [1] 69
```

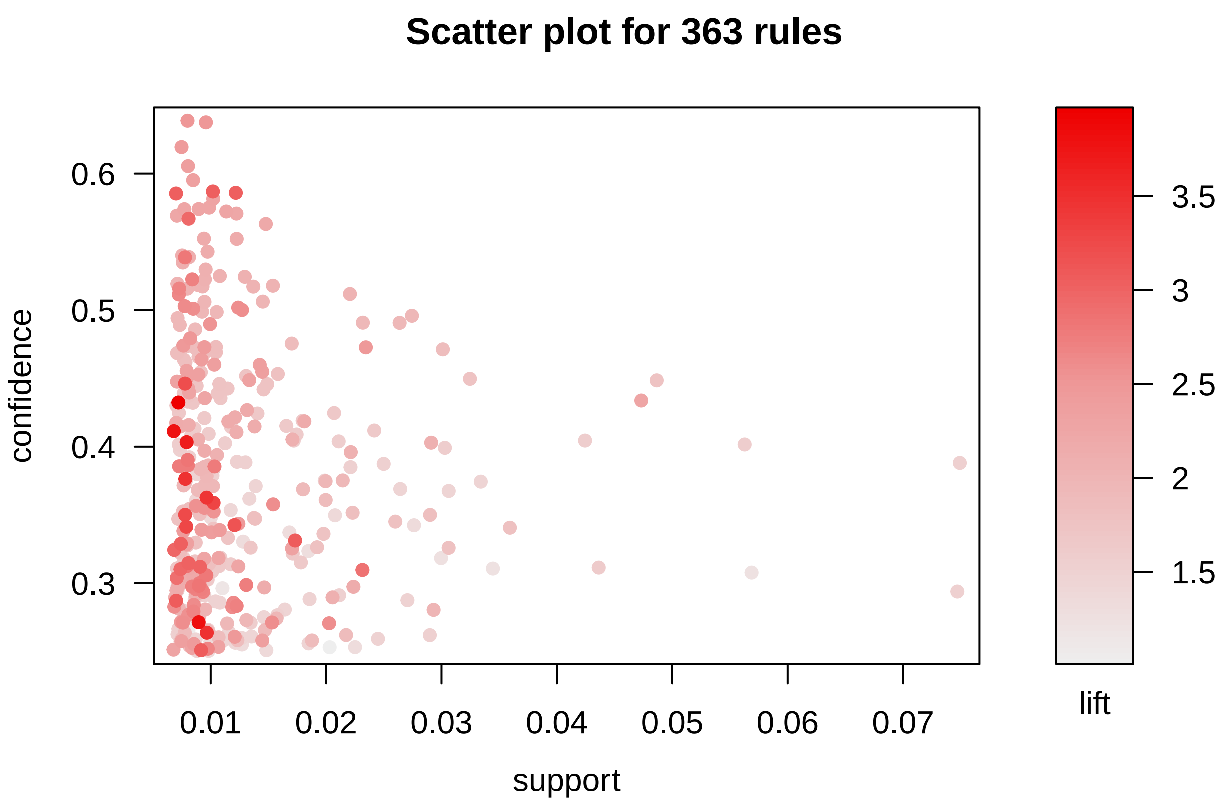
```
##      lhs                          rhs                    support confidence   coverage     lift count
## [1]  {baking powder}           => {whole milk}       0.009252669  0.5229885 0.01769192 2.046793    91
## [2]  {frozen vegetables,
##       other vegetables}        => {whole milk}       0.009659380  0.5428571 0.01779359 2.124552    95
## [3]  {curd,
##       yogurt}                  => {whole milk}       0.010066090  0.5823529 0.01728521 2.279125    99
## [4]  {curd,
##       other vegetables}        => {whole milk}       0.009862735  0.5739645 0.01718353 2.246296    97
## [5]  {pork,
##       root vegetables}         => {other vegetables} 0.007015760  0.5149254 0.01362481 2.661214    69
## [6]  {margarine,
##       rolls/buns}              => {whole milk}       0.007930859  0.5379310 0.01474326 2.105273    78
## [7]  {butter,
##       root vegetables}         => {whole milk}       0.008235892  0.6377953 0.01291307 2.496107    81
## [8]  {butter,
##       yogurt}                  => {whole milk}       0.009354347  0.6388889 0.01464159 2.500387    92
## [9]  {butter,
##       other vegetables}        => {whole milk}       0.011489578  0.5736041 0.02003050 2.244885   113
## [10] {domestic eggs,
##       root vegetables}         => {other vegetables} 0.007320793  0.5106383 0.01433655 2.639058    72
## [11] {domestic eggs,
##       root vegetables}         => {whole milk}       0.008540925  0.5957447 0.01433655 2.331536    84
## [12] {domestic eggs,
##       yogurt}                  => {whole milk}       0.007727504  0.5390071 0.01433655 2.109485    76
## [13] {domestic eggs,
##       other vegetables}        => {whole milk}       0.012302999  0.5525114 0.02226741 2.162336   121
## [14] {fruit/vegetable juice,
##       yogurt}                  => {whole milk}       0.009456024  0.5054348 0.01870869 1.978094    93
## [15] {tropical fruit,
##       whipped/sour cream}      => {other vegetables} 0.007829181  0.5661765 0.01382816 2.926088    77
## [16] {tropical fruit,
##       whipped/sour cream}      => {whole milk}       0.007930859  0.5735294 0.01382816 2.244593    78
## [17] {root vegetables,
##       whipped/sour cream}      => {whole milk}       0.009456024  0.5535714 0.01708185 2.166484    93
## [18] {whipped/sour cream,
##       yogurt}                  => {whole milk}       0.010879512  0.5245098 0.02074225 2.052747   107
## [19] {rolls/buns,
##       whipped/sour cream}      => {whole milk}       0.007829181  0.5347222 0.01464159 2.092715    77
```

```
## [20] {other vegetables,
##       whipped/sour cream}    => {whole milk}       0.014641586  0.5070423 0.02887646 1.984385   144
## [21] {pip fruit,
##       root vegetables}       => {other vegetables} 0.008134215  0.5228758 0.01555669 2.702304    80
## [22] {pip fruit,
##       root vegetables}       => {whole milk}       0.008947636  0.5751634 0.01555669 2.250988    88
## [23] {pip fruit,
##       yogurt}                => {whole milk}       0.009557702  0.5310734 0.01799695 2.078435    94
## [24] {other vegetables,
##       pip fruit}             => {whole milk}       0.013523132  0.5175097 0.02613116 2.025351   133
## [25] {pastry,
##       yogurt}                => {whole milk}       0.009150991  0.5172414 0.01769192 2.024301    90
## [26] {citrus fruit,
##       root vegetables}       => {other vegetables} 0.010371124  0.5862069 0.01769192 3.029608   102
## [27] {citrus fruit,
##       root vegetables}       => {whole milk}       0.009150991  0.5172414 0.01769192 2.024301    90
## [28] {sausage,
##       tropical fruit}        => {whole milk}       0.007219115  0.5182482 0.01392984 2.028241    71
## [29] {root vegetables,
##       sausage}               => {whole milk}       0.007727504  0.5170068 0.01494662 2.023383    76
## [30] {root vegetables,
##       tropical fruit}        => {other vegetables} 0.012302999  0.5845411 0.02104728 3.020999   121
## [31] {root vegetables,
##       tropical fruit}        => {whole milk}       0.011997966  0.5700483 0.02104728 2.230969   118
## [32] {tropical fruit,
##       yogurt}                => {whole milk}       0.015149975  0.5173611 0.02928317 2.024770   149
## [33] {root vegetables,
##       yogurt}                => {whole milk}       0.014539908  0.5629921 0.02582613 2.203354   143
## [34] {rolls/buns,
##       root vegetables}       => {other vegetables} 0.012201322  0.5020921 0.02430097 2.594890   120
## [35] {rolls/buns,
##       root vegetables}       => {whole milk}       0.012709710  0.5230126 0.02430097 2.046888   125
## [36] {other vegetables,
##       yogurt}                => {whole milk}       0.022267412  0.5128806 0.04341637 2.007235   219
## [37] {other vegetables,
##       root vegetables,
##       tropical fruit}        => {whole milk}       0.007015760  0.5702479 0.01230300 2.231750    69
## [38] {root vegetables,
##       tropical fruit,
##       whole milk}            => {other vegetables} 0.007015760  0.5847458 0.01199797 3.022057    69
## [39] {other vegetables,
##       tropical fruit,
##       yogurt}                => {whole milk}       0.007625826  0.6198347 0.01230300 2.425816    75
## [40] {tropical fruit,
##       whole milk,
##       yogurt}                => {other vegetables} 0.007625826  0.5033557 0.01514997 2.601421    75
## [41] {other vegetables,
##       root vegetables,
##       yogurt}                => {whole milk}       0.007829181  0.6062992 0.01291307 2.372842    77
## [42] {root vegetables,
##       whole milk,
##       yogurt}                => {other vegetables} 0.007829181  0.5384615 0.01453991 2.782853    77

##     lhs                    rhs                support confidence   coverage     lift count
```

```
## [1] {butter,
##      root vegetables}  => {whole milk} 0.008235892  0.6377953 0.01291307 2.496107    81
## [2] {butter,
##      yogurt}           => {whole milk} 0.009354347  0.6388889 0.01464159 2.500387    92
## [3] {other vegetables,
##      tropical fruit,
##      yogurt}           => {whole milk} 0.007625826  0.6198347 0.01230300 2.425816    75
## [4] {other vegetables,
##      root vegetables,
##      yogurt}           => {whole milk} 0.007829181  0.6062992 0.01291307 2.372842    77
```

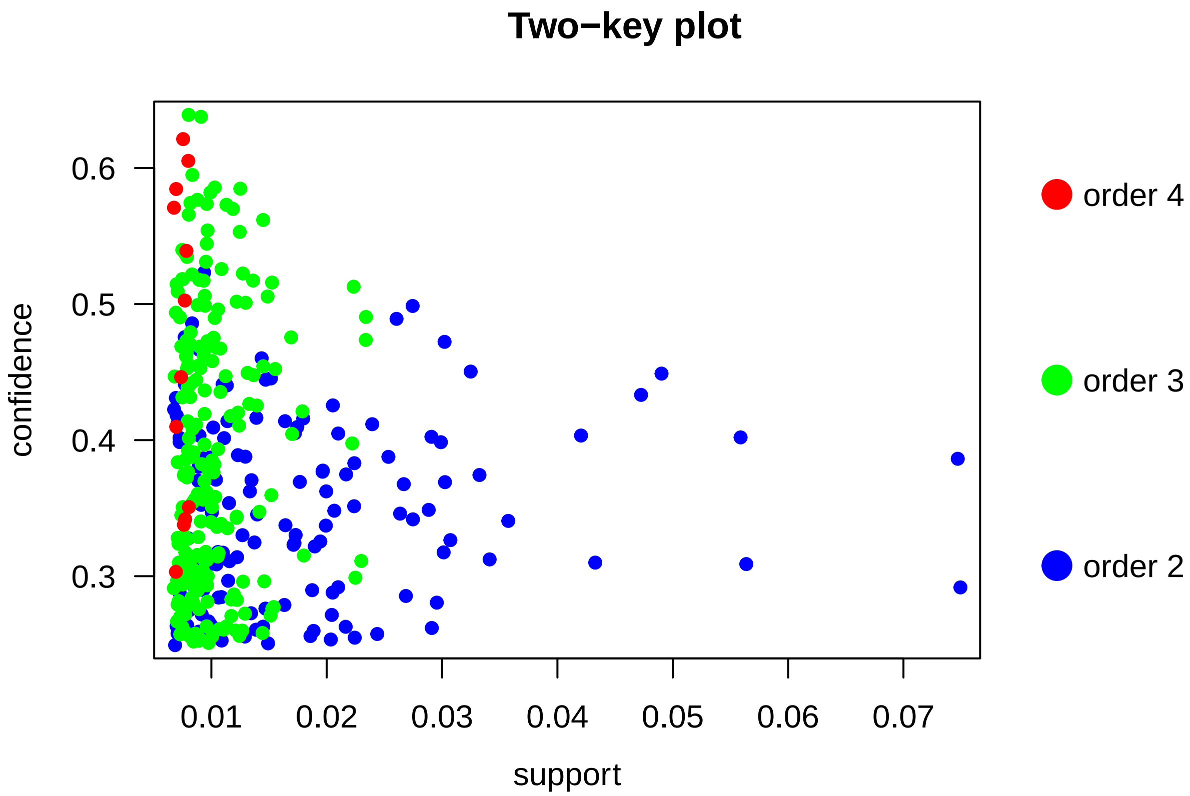
We then inspected subsets with varying levels of lift and confidence. We see that when we set it to a lift higher than 3, herbs and root vegetables have the highest lift at 3.95, meaning that if someone buys herbs, they are 3.95 times more likely to buy root vegetables. With a confidence above 0.5, butter and yogurt have the highest confidence at 0.64, which means that if someone buys butter, they are 64% likely to also buy yogurt. When we set the lift to above 2 and the confidence to above 0.6, we see that butter and yogurt have the highest lift and confidence, with a lift of 2.50 and a confidence of 0.64.

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



When we plot our ‘grocRules,’ we see that low support, as well as confidence below 0.5, tend to have the highest lift because the points are a deeper red. We might be seeing this trend because the higher the lift is, the more sure we are that those items being bought together is not a coincidence, and so this would apply to a smaller amount of transactions.

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



With all points, we see that as the confidence gets higher, the number of points decreases. We again see that order 2, or the blue plot points, have the highest support aka it applies to the largest amount of cases.

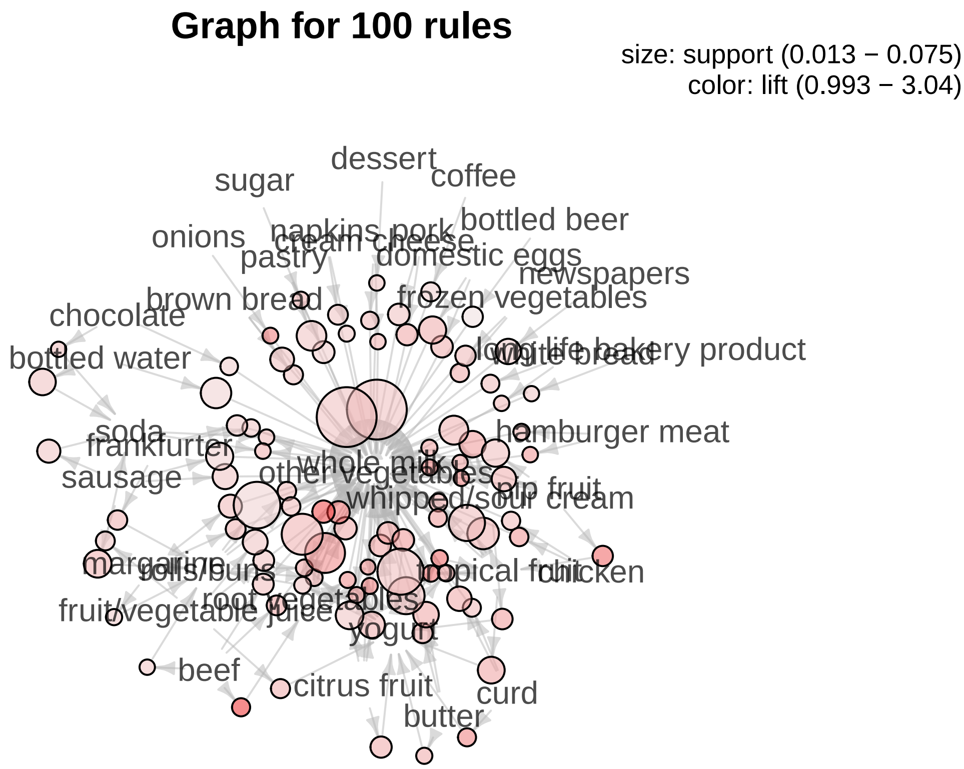
```
##     lhs                   rhs                  support    confidence coverage
## [1] {yogurt}           => {whole milk}       0.05602440 0.4016035  0.1395018
## [2] {rolls/buns}       => {whole milk}       0.05663447 0.3079049  0.1839349
## [3] {other vegetables} => {whole milk}       0.07483477 0.3867578  0.1934926
## [4] {whole milk}       => {other vegetables} 0.07483477 0.2928770  0.2555160
##     lift     count
## [1] 1.571735 551
## [2] 1.205032 557
## [3] 1.513634 736
## [4] 1.513634 736
```

```
##     lhs                   rhs              support confidence   coverage     lift count
## [1] {butter,
##      root vegetables}  => {whole milk} 0.008235892  0.6377953 0.01291307 2.496107    81
## [2] {butter,
##      yogurt}           => {whole milk} 0.009354347  0.6388889 0.01464159 2.500387    92
## [3] {other vegetables,
##      tropical fruit,
##      yogurt}           => {whole milk} 0.007625826  0.6198347 0.01230300 2.425816    75
## [4] {other vegetables,
##      root vegetables,
##      yogurt}           => {whole milk} 0.007829181  0.6062992 0.01291307 2.372842    77
```

We are interested in looking at the transactions of the blue dots with high support, so we set the support to be above 0.05. All four of the resulting transactions have whole milk in them. This is also the case when we set the confidence to be above 0.6. These findings are consistent with whole milk being the most frequently bought item in our data set.

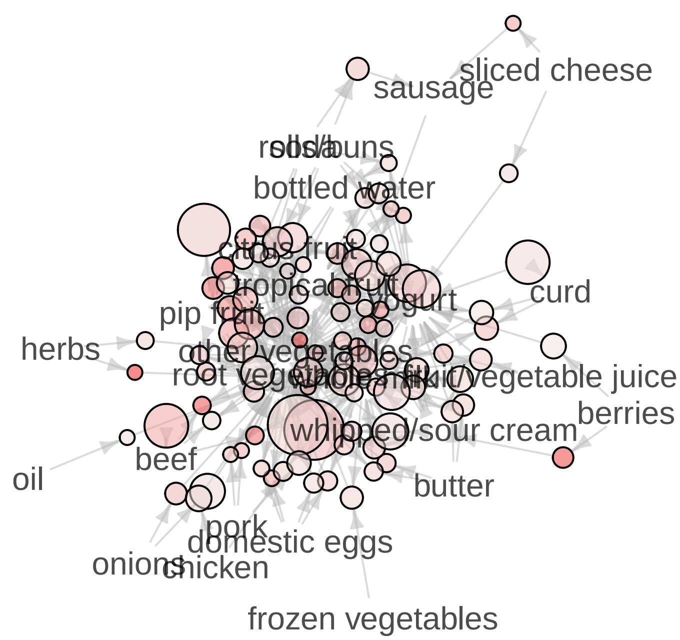
```
## set of 363 rules
##
## rule length distribution (lhs + rhs):sizes
##   2   3   4
## 137 214  12
##
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   2.000   2.000   3.000   2.656   3.000   4.000
##
## summary of quality measures:
##     support           confidence        coverage            lift
##  Min.   :0.007016   Min.   :0.2500   Min.   :0.01200   Min.   :0.9932
##  1st Qu.:0.008134   1st Qu.:0.2962   1st Qu.:0.02166   1st Qu.:1.6060
##  Median :0.009659   Median :0.3551   Median :0.02888   Median :1.9086
##  Mean   :0.012945   Mean   :0.3743   Mean   :0.03675   Mean   :2.0072
##  3rd Qu.:0.013777   3rd Qu.:0.4420   3rd Qu.:0.04230   3rd Qu.:2.3289
##  Max.   :0.074835   Max.   :0.6389   Max.   :0.25552   Max.   :3.9565
##      count
##  Min.   : 69.0
##  1st Qu.: 80.0
##  Median : 95.0
##  Mean   :127.3
##  3rd Qu.:135.5
##  Max.   :736.0
##
## mining info:
##       data ntransactions support confidence
##  groceries          9835   0.007       0.25
```

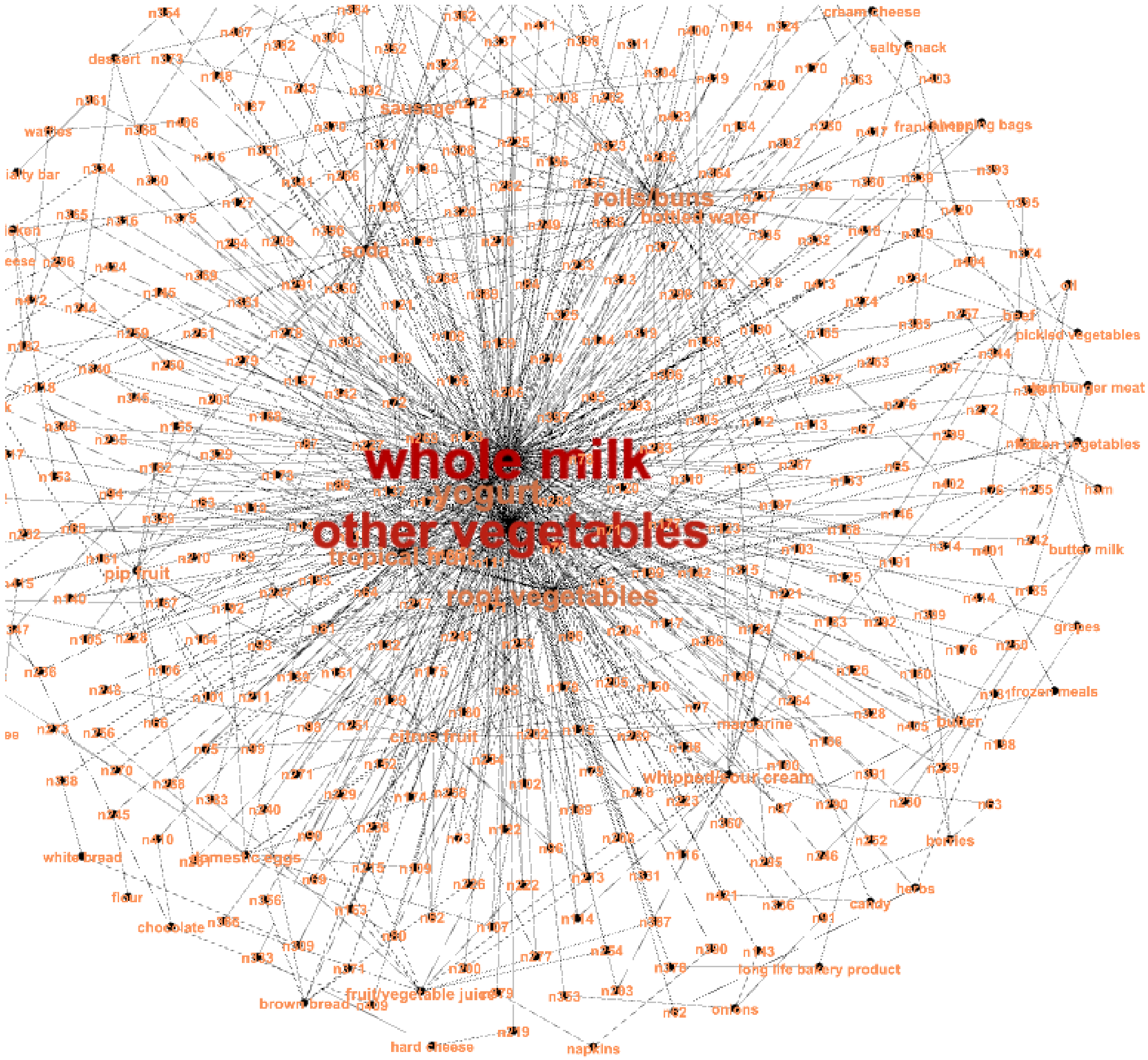
```
## Warning: plot: Too many rules supplied. Only plotting the best 100 rules using
## 'support' (change control parameter max if needed)
```


Graph for 100 rules
size: support (0.013 − 0.075)
color: lift (0.993 − 3.04)
dessert
sugar
coffee
bottled beer
onions
napkins
pork
cream cheese
pastry
domestic eggs
newspapers
brown bread
frozen vegetables
chocolate
long life bakery product
white bread
bottled water
hamburger meat
soda
frankfurter
whole milk
other vegetables
sausage
pip fruit
whipped/sour cream
margarine
rolls/buns
tropical fruit
chicken
root vegetables
fruit/vegetable juice
yogurt
beef
citrus fruit
curd
butter

Graph for 100 rules

size: support (0.007 − 0.023)
color: lift (2.279 − 3.956)





Our Gephy graph reinforces what we observed earlier. We ranked it by degree, so we see that whole milk and other vegetables especially have the largest amount of connections.

Author Attribution

We read in the c50train data as our training set and used the readerPlain function to translate all of the articles into English. Then we cleaned the file names to remove the file path and only include the author concatenated with the text name. We then placed all the documents in a vector and created a text mining corpus, which contained 2,500 documents.

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

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

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

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

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

## <<SimpleCorpus>>
## Metadata:  corpus specific: 1, document level (indexed): 0
## Content:  documents: 2500
```

For pre-processing and tokenization, we made everything lowercase and removed numbers, punctuation, and white space. This will help ensure the model focuses on true word values. After this, we still have 2,500 documents. We also removed the basic English stopwords to exclude filler words such as 'because, 'and', etc. This doesn't change the number of documents.

```
## <<DocumentTermMatrix (documents: 2500, terms: 32573)>>
## Non-/sparse entries: 545361/80887139
## Sparsity           : 99%
## Maximal term length: 55
## Weighting          : term frequency (tf)
```

We created a doc-term-matrix from our document corpus, which shows that our 2,500 documents have 32,570 terms.

```
## <<DocumentTermMatrix (documents: 2500, terms: 3396)>>
## Non-/sparse entries: 430471/8059529
## Sparsity           : 95%
## Maximal term length: 55
## Weighting          : term frequency (tf)
```

We removed the terms that were not present in over 99% of documents to remove insignificant words that rarely occur. This decreases our number of terms down to 3,393 terms from 32,570 terms.

We constructed TF IDF weights on the original training set to remove words with zero TF IDF weight since they have zero importance. This decreased the terms to 3,377. Then we fed the cleaned training set into the principal components matrix and found our PC summaries. It looks like about 844 summaries provide us around 80% of the variation in the 3,377 features.

For the test set, we read in the c50test data and performed the same data cleaning process to make the two data sets comparable.

Unfortunately, we were unable to set a column of authors as the response variable for the classification models. We tried to use a random forest model and were planning on doing KNN as well. For the random forest, we decided to use the data up to the 844th summary since it provided 80% of the variation in features.

Random Forest Classification train_pcs = data.frame(pca_c50train$x[,1:844]) authors_train = as.factor(train_authors) #make authors categorical set.seed(1) rf.docs = randomForest(authors_train~., data = train_pcs, mtry=6, importance = TRUE)