Assignment1

Qijing Zhang (Vicky)

August 6, 2015

Question 1: Georgia voting

```
library(doBy)
```

```
## Loading required package: survival
```

```
library(ggplot2)
georgia <-
  read.csv(
  "/Users/vickyzhang/Documents/MSBA/predictive2/STA380/data/georgia2000.csv", row.names =
  1
  )
names(georgia)
```

```
## [1] "ballots" "votes"   "equip"   "poor"    "urban"   "atlanta" "perAA"
## [8] "gore"    "bush"
```

```
head(georgia)
```

```
##           ballots votes   equip poor urban atlanta perAA gore bush
## APPLING      6617  6099   LEVER    1     0       0 0.182 2093 3940
## ATKINSON     2149  2071   LEVER    1     0       0 0.230  821 1228
## BACON        3347  2995   LEVER    1     0       0 0.131  956 2010
## BAKER        1607  1519 OPTICAL    1     0       0 0.476  893  615
## BALDWIN     12785 12126   LEVER    0     0       0 0.359 5893 6041
## BANKS        4773  4533   LEVER    0     0       0 0.024 1220 3202
```

```
georgia['undercount'] = georgia['ballots'] - georgia['votes']
# look at how equipment affects undercount, without pivot
# syntax: y~x, x is the col you want to group by, y is the value you want to be grouped!
equip_undercount = summaryBy(undercount~equip, data=georgia, FUN = function(x) c(m = mean(x)) )
equip_undercount
```

```
##     equip undercount.m
## 1   LEVER     229.9459
## 2 OPTICAL     592.2727
## 3   PAPER      56.5000
## 4   PUNCH    2262.4706
```

```
# Punchcards obviously lead to the higest average undercount. Paper leads to the lowest.

######## investigate whether this has an effect on poor and minority communities
# use contingency table
t1 = xtabs(undercount~poor+equip, data = georgia)
t1
```

```
##     equip
## poor LEVER OPTICAL PAPER PUNCH
##    0  6816   31633     0 37033
##    1 10200    7457   113  1429
```

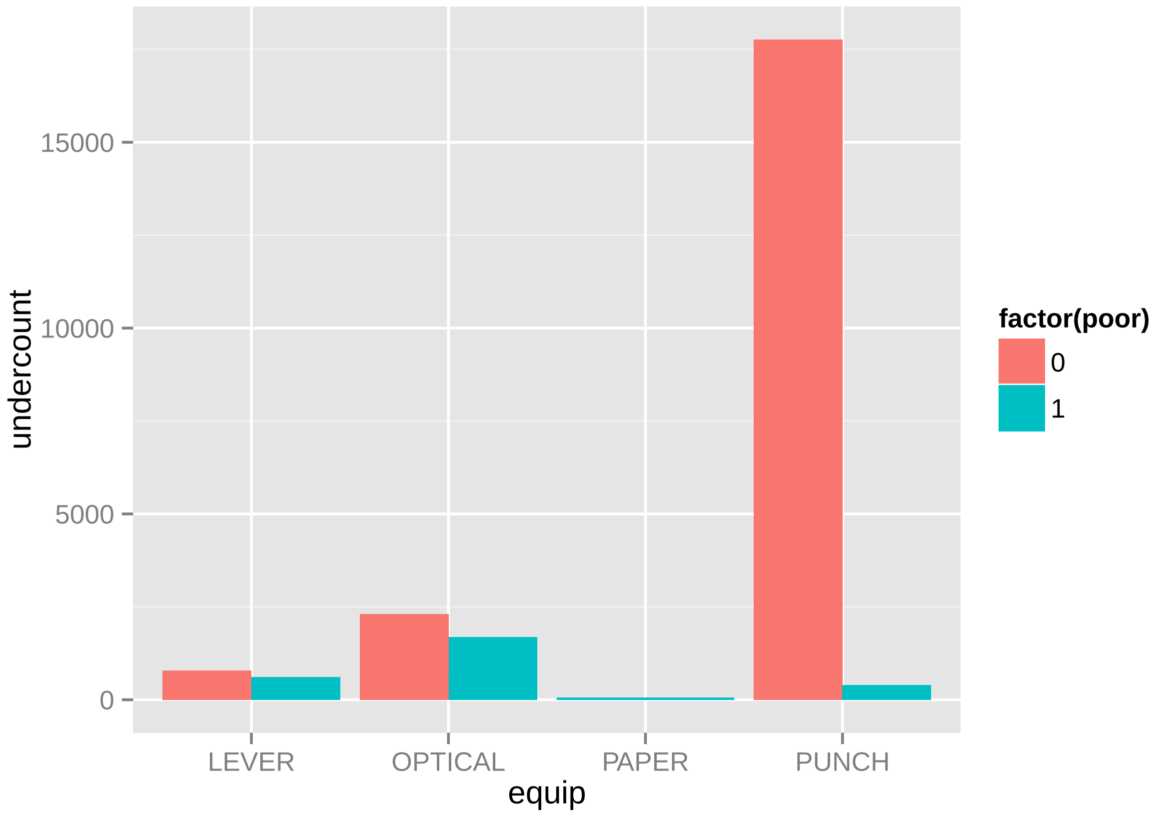
```r
# get average of each category, using summaryBy()
georgia_poor = georgia[georgia['poor'] == 0,]
xtab_poor = summaryBy(undercount~equip, data=georgia_poor, FUN = function(x) c(m = mean(x)) )
xtab_poor
```

```
##     equip undercount.m
## 1   LEVER     235.0345
## 2 OPTICAL     659.0208
## 3   PUNCH    3703.3000
```

```r
georgia_nonpoor = georgia[georgia['poor'] == 1,]
xtab_nonpoor =
  summaryBy(undercount~equip, data=georgia_nonpoor, FUN = function(x) c(m = mean(x)) )
xtab_nonpoor
```

```
##     equip undercount.m
## 1   LEVER     226.6667
## 2 OPTICAL     414.2778
## 3   PAPER      56.5000
## 4   PUNCH     204.1429
```

```r
# the effect of low income on undercount, by equipment
# poor is seen as a continuous variable, so you have to factor() it!!
ggplot(georgia,
       aes(x = equip, y = undercount, fill = factor(poor))) + geom_bar(stat =
       "identity", position = position_dodge())
```



###

Conclusion If a poor community uses punch cards, it would have a very high undercount, and the votes among those people are not adequately represented. If such a community wants to have its votes adequately represented, use paper (not available in those communities yet though), lever, or optical.

```
######## effect of minority community on undercount, linear regression
mean(georgia$perAA)
```

```
## [1] 0.2429811
```

```
# code perAA as 1 if perAA is higher than mean, 0 otherwise
georgia[georgia['perAA'] > 0.24,'perAA'] = 1
georgia[georgia['perAA'] <= 0.24,'perAA'] = 0

# contingency tables, giving the sum(undercount) for each perAA * equip
# combination. Tried to do average(undercount) but didn't succeed. will do
# average in summaryBy() function below.
t2 = xtabs(undercount~perAA+equip,data = georgia)
t2
```

```
##      equip
## perAA LEVER OPTICAL PAPER PUNCH
##     0  8589   23059     0  5620
##     1  8427   16031   113 32842
```

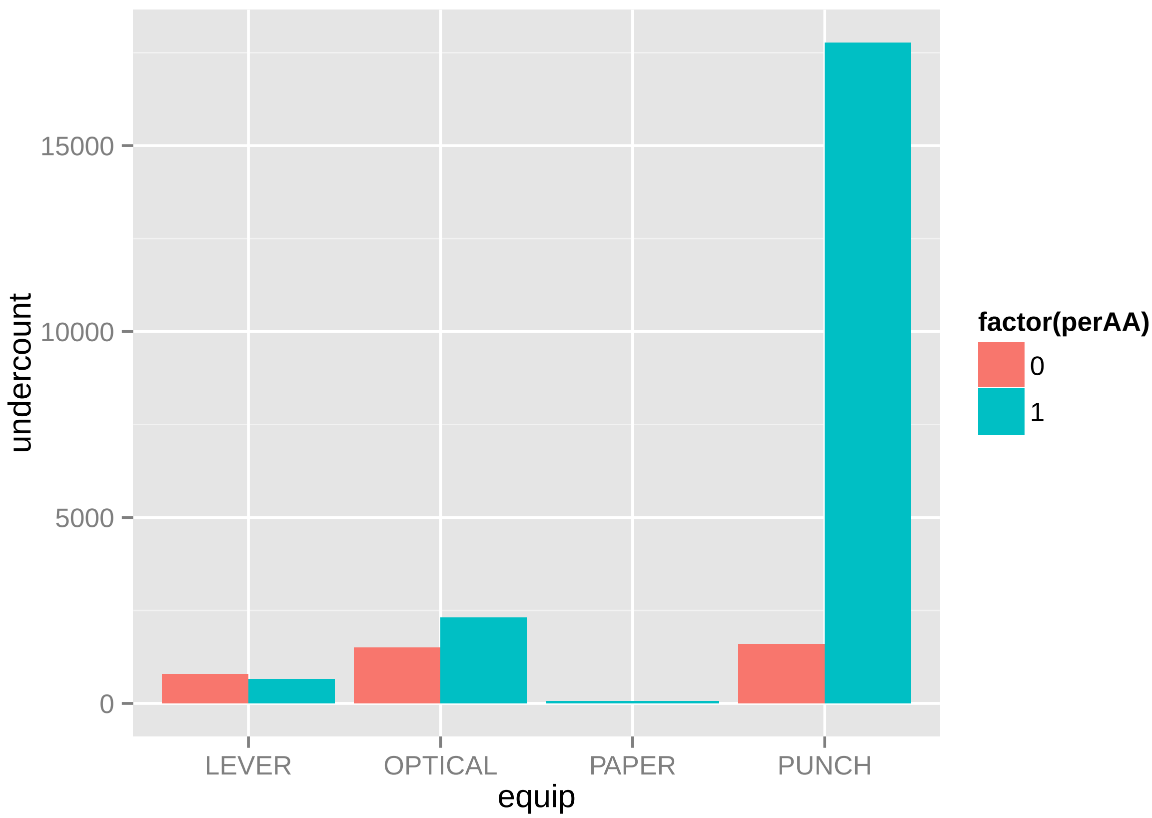
```
# look at the average undercount of each category
georgia_AA = georgia[georgia['perAA'] == 1,]
xtab_AA = summaryBy(undercount~equip, data=georgia_AA, FUN = function(x) c(m = mean(x)) )
xtab_AA
```

```
##     equip undercount.m
## 1   LEVER     210.6750
## 2 OPTICAL     667.9583
## 3   PAPER      56.5000
## 4   PUNCH    2985.6364
```

```
georgia_nonAA = georgia[georgia['perAA'] == 0,]
xtab_nonAA = summaryBy(undercount~equip, data=georgia_nonAA, FUN = function(x) c(m = mean(x)) )
xtab_nonAA
```

```
##     equip undercount.m
## 1   LEVER     252.6176
## 2 OPTICAL     549.0238
## 3   PUNCH     936.6667
```

```
ggplot(georgia, aes(x = equip, y = undercount, fill = factor(perAA))) + geom_bar(stat =
      "identity", position = position_dodge())
```



Conclusion:

If a community has a higher African-American percentage than average, it would see a much higher undercount if it uses punch than optical or lever. The choice of equipment does have different effects on undercount in minority communities.

Quesiton 2: Bootstrapping

```r
library(mosaic)
```

```
## Loading required package: car
## Loading required package: dplyr
##
## Attaching package: 'dplyr'
##
## The following objects are masked from 'package:stats':
##
##     filter, lag
##
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
##
## Loading required package: lattice
## Loading required package: mosaicData
##
## Attaching package: 'mosaic'
##
## The following objects are masked from 'package:dplyr':
##
##     count, do, tally
##
## The following object is masked from 'package:car':
##
##     logit
##
## The following objects are masked from 'package:stats':
##
##     binom.test, cor, cov, D, 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
library(fImport)
```

```
## Loading required package: timeDate
## Loading required package: timeSeries
```

```r
library(foreach)
library(pracma)
```

```
##
## Attaching package: 'pracma'
##
## The following objects are masked from 'package:mosaic':
```

```
##
##     cross, deg2rad, dot, logit, pdist, rad2deg, rand
##
## The following object is masked from 'package:car':
##
##     logit
```

```r
mystocks = c("SPY", 'TLT', 'LQD', 'EEM', 'VNQ')
myprices = yahooSeries(mystocks, from='2011-01-01', to='2015-07-30')

YahooPricesToReturns = function(series) {
  mycols = grep('Adj.Close', colnames(series))
  closingprice = series[,mycols]
  N = nrow(closingprice)
  percentreturn = as.data.frame(closingprice[2:N,]) / as.data.frame(closingprice[1:(N-1),]) - 1
  mynames = strsplit(colnames(percentreturn), '.', fixed=TRUE)
  mynames = lapply(mynames, function(x) return(paste0(x[1], ".PctReturn")))
  colnames(percentreturn) = mynames
  as.matrix(na.omit(percentreturn))
}

######## return of individual assets (both arithmetic and compound)
# Compute the returns from the closing prices
myreturns = YahooPricesToReturns(myprices)

# arithmetic average daily return of each stock, but this is not an accurate
# measure of overall return
average_return = apply(myreturns, MARGIN = 2, FUN = mean)
average_return
```

```
## SPY.PctReturn TLT.PctReturn LQD.PctReturn EEM.PctReturn VNQ.PctReturn
##  5.643882e-04  3.935468e-04  2.066796e-04 -6.085922e-05  4.989246e-04
```

```r
# Accurate measure - compute compound average daily return of stock.
# First, add 1 to all values in df, which is as easy as (myreturns + 1)!!!
myreturn1 = myreturns + 1
head(myreturn1)
```

```
##            SPY.PctReturn TLT.PctReturn LQD.PctReturn EEM.PctReturn
## 2011-01-04     0.9994490     1.0011775     1.0012861     1.0045738
## 2011-01-05     1.0051976     0.9779727     0.9926605     0.9975166
## 2011-01-06     0.9980414     1.0043735     1.0014788     0.9894190
## 2011-01-07     0.9980375     1.0053342     1.0050757     0.9907738
## 2011-01-10     0.9987416     1.0054142     1.0017445     0.9896296
## 2011-01-11     1.0035438     0.9943996     0.9995418     1.0106929
##            VNQ.PctReturn
## 2011-01-04     0.9808510
## 2011-01-05     1.0036153
## 2011-01-06     0.9906340
## 2011-01-07     1.0003636
## 2011-01-10     0.9998182
## 2011-01-11     0.9978186
```

```r
# take nth root of the product of all returns, where n = nrow(myreturn1), chain
# the results into a list
nth = nrow(myreturn1)
compound_returns = foreach(i=1:ncol(myreturn1), .combine = 'c') %do% {
  compound_return = nthroot(prod(myreturn1[,i]), nth)
}
compound_returns = compound_returns - 1
compound_returns
```

```
## [1]  0.0005199355  0.0003475315  0.0002005995 -0.0001570169  0.0004332609
```

```r
######## risk of individual assets
# get a list of standard deviations of each stock, representing risk
stdevs = foreach(i=1:ncol(myreturns), .combine='c') %do% {
    sd(myreturns[,i])
}
stdevs
```

```
## [1] 0.009420369 0.009596946 0.003486944 0.013854692 0.011457092
```

```r
# Observation: The standard deviation of emerging markets is the largest, which
# fits expectation.

###### compute Sharpe ratio of individual assets to give a risk/return metric
# get risk-free rate from TLT
risk_free_return = compound_returns[2]
sharpe = (compound_returns - risk_free_return) / stdevs
sharpe
```

```
## [1]  0.018301194  0.000000000 -0.042137757 -0.036417152  0.007482654
```

Conclusion

As expected, the Sharpe ratio of TLT is 0. Both investment-grade corporate bonds and emerging markets have a Sharpe ratio below 0 with the largest absolute values, which reflects their high risk.

```r
########### choosing portfolio
##### the even split
totalwealth = 10000
weights = c(0.2, 0.2, 0.2, 0.2, 0.2)

# expected daily return for the even-split portfolio.
###### Note: I used formulas mentioned in class to calculate mean and variance
# here, but there's a smarter way to do it. See 'smart way' below.
expected_return_evensplit = sum(weights * compound_returns)
expected_return_evensplit
```

```
## [1] 0.0002688621
```

```r
# expected stdev
# get covariance
cova = cov(myreturns)
cova[1,1]^2
```

```
## [1] 7.875381e-09
```

```r
var_terms = 0
for (i in 1:ncol(cova)) {
  var_terms = var_terms + cova[i, i] * weights[i] ^ 2
}
var_terms
```

```
## [1] 2.064884e-05
```

```r
# calculate sum of covariance terms
cova_terms = 0
for (i in 2:nrow(cova)) {
  for(j in 2:ncol(cova)) {
    cova_terms = cova_terms + 2*weights[i]*weights[j]*cova[i, j]
    #print(paste('i=',i,'j=',j))
  }
}
cova_terms
```

```
## [1] 4.187201e-05
```

```r
variance_evensplit = var_terms + cova_terms
std_evensplit = sqrt(variance_evensplit)
Sharpe_evensplit = (expected_return_evensplit - risk_free_return) / std_evensplit
Sharpe_evensplit # negative!
```

```
## [1] -0.009949319
```

```r
# ####### smart way - calculate daily return first!!!!
# calculate the returns of daily portfolio return first
evensplit = weights * (myreturns + 1) # apply weights first
# and then sum across each row to get portfolio return
returns_evensplit = apply(evensplit,MARGIN = 1, FUN = sum)
head(returns_evensplit)
```

```
## 2011-01-04 2011-01-05 2011-01-06 2011-01-07 2011-01-10 2011-01-11
##  0.9974675  0.9953925  0.9967893  0.9999170  0.9990696  1.0011993
```

```r
returns_evensplit = returns_evensplit - 1
head(returns_evensplit)
```

```
##    2011-01-04    2011-01-05    2011-01-06    2011-01-07    2011-01-10
## -2.532504e-03 -4.607453e-03 -3.210664e-03 -8.303279e-05 -9.303865e-04
##    2011-01-11
##  1.199342e-03
```

```r
# get compound return
nth = length(returns_evensplit)
compound_return = nthroot(prod(returns_evensplit+1), nth)
compound_return = compound_return - 1
compound_return
```

```
## [1] 0.0003027417
```

```r
# get stdev
sd(returns_evensplit)
```

```
## [1] 0.00596693
```

```r
###### to get a safer portfolio, definitely involve market and risk-free assets,
# and then choose real-estate because it's less volatile than
# investment-grade bond and emerging market
myreturns_safe = myreturns + 1
returns_safe = 1/3 * myreturns[,1] + 1/3 * myreturns[,2] + 1/3 *myreturns[,5]
nth = length(returns_safe)
expected_return_safe = nthroot(prod(returns_safe+1), nth)
expected_return_safe = expected_return_safe - 1
expected_return_safe # -0.999532
```

```
## [1] 0.0004680473
```

```r
sd(returns_safe) # slightly smaller than sd(returns_evensplit), 0.005930875 < 0.00596693
```

```
## [1] 0.005930875
```

```r
###### to get a more aggressive portfolio, definitely involve investment-grade
# bond and emerging market
myreturns_safe = myreturns + 1
returns_aggre = 1/2 * myreturns[,3] + 1/2 *myreturns[,4]
nth = length(returns_aggre)
expected_return_aggre = nthroot(prod(returns_aggre+1), nth)
expected_return_aggre = expected_return_aggre - 1
expected_return_aggre # -0.9999526
```

```
## [1] 4.743117e-05
```

```r
sd(returns_aggre) # slightly bigger than sd(returns_evensplit), 0.007137946 > 0.00596693
```

```
## [1] 0.007137946
```

```r
######### simulation
# create a df to put the results in
results = data.frame(matrix(ncol = 3, nrow = 2), row.names = c('totalwealth', 'VaR'))
colnames(results) = c('evensplit', 'safe', 'aggressive')
```

```
# Now simulate many different possible trading years for even-split portfolio
set.seed(1)
n_days = 20
sim1 = foreach(i=1:500, .combine='rbind') %do% {
    totalwealth = 10000
    weights = c(0.2, 0.2, 0.2, 0.2, 0.2)
    holdings = weights * totalwealth
    wealthtracker = rep(0, n_days) # Set up a placeholder to track total wealth
    for(today in 1:n_days) {
        return.today = resample(myreturns, 1, orig.ids=FALSE)
        holdings = holdings + holdings*return.today # holdings is a series
        totalwealth = sum(holdings) # so need to sum it to get total wealth
        wealthtracker[today] = totalwealth
        holdings = weights * totalwealth # rebalanced at 0 transaction cost
    }
    wealthtracker
}
# final wealth
total_evensplit = totalwealth
total_evensplit
```

```
## [1] 10050.32
```

```
results['totalwealth', 'evensplit'] = total_evensplit
# VaR
VaR_evensplit = quantile(sim1[,n_days], 0.05) - 10000
VaR_evensplit
```

```
##        5%
## -346.6541
```

```
results['VaR', 'evensplit'] = VaR_evensplit


##### bootstrapping for safe porfolio
returns_safe = myreturns[,c(1, 2, 5)]

# seed is only effective for one operation, have to reset after each time it's used
set.seed(1)
n_days = 20
sim2 = foreach(i=1:500, .combine='rbind') %do% {
    totalwealth = 10000
    weights = c(1/3, 1/3, 1/3)
    holdings = weights * totalwealth
    wealthtracker_safe = rep(0, n_days) # Set up a placeholder to track total wealth
    for(today in 1:n_days) {
        return.today = resample(returns_safe, 1, orig.ids=FALSE)
        holdings = holdings + holdings*return.today # holdings is a series
        totalwealth = sum(holdings) # so need to sum it to get total wealth
        wealthtracker_safe[today] = totalwealth
        holdings = weights * totalwealth # rebalanced at 0 transaction cost
    }
```

```r
    wealthtracker_safe
}
# final wealth
total_safe = totalwealth
total_safe
```

```
## [1] 10122.09
```

```r
results['totalwealth', 'safe'] = total_safe
# VaR
VaR_safe = quantile(sim2[,n_days], 0.05) - 10000
VaR_safe
```

```
##        5%
## -326.1435
```

```r
results['VaR', 'safe'] = VaR_safe

###### Bootstrapping for aggressive portfolio
returns_aggre = myreturns[,c(3, 4)]

set.seed(1)
n_days = 20
sim3 = foreach(i=1:500, .combine='rbind') %do% {
    totalwealth = 10000
    weights = c(0.5, 0.5)
    holdings = weights * totalwealth
    wealthtracker_aggre = rep(0, n_days) # Set up a placeholder to track total wealth
    for(today in 1:n_days) {
        return.today = resample(returns_aggre, 1, orig.ids=FALSE)
        holdings = holdings + holdings*return.today # holdings is a series
        totalwealth = sum(holdings) # so need to sum it to get total wealth
        wealthtracker_aggre[today] = totalwealth
        holdings = weights * totalwealth # rebalanced at 0 transaction cost
    }
    wealthtracker_aggre
}
# final wealth
total_aggre = totalwealth
total_aggre
```

```
## [1] 9941.889
```

```r
results['totalwealth', 'aggressive'] = total_aggre

# VaR
VaR_aggre = quantile(sim3[,n_days], 0.05) - 10000
VaR_aggre
```

```
##        5%
## -450.8268
```

```r
results['VaR', 'aggressive'] = VaR_aggre

results
```

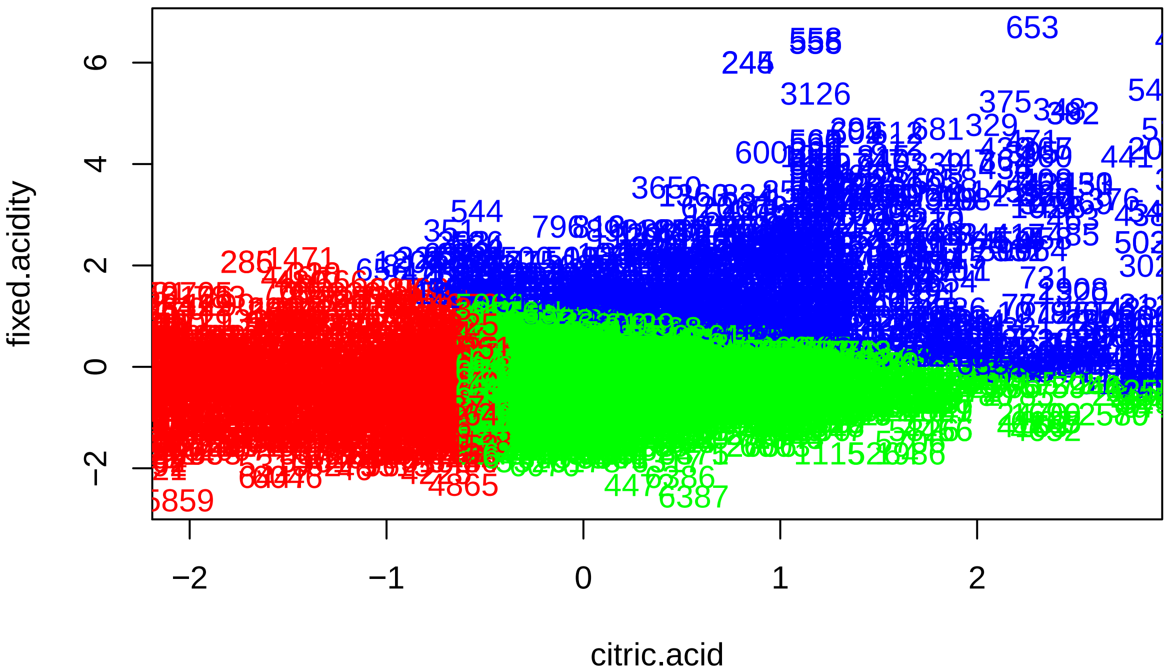
```
##               evensplit       safe aggressive
## totalwealth 10050.3176 10122.0916  9941.8890
## VaR           -346.6541  -326.1435  -450.8268
```

Conclusion:

We can see clearly that the aggressive portfolio yields the lowest total wealth yet the largest VaR absolute value, which means it gives the lowest return and highest risk (given the seed we have). Safe portfolio actually performs the best, with highest total wealth and lowest VaR absolute value. Evensplit is somewhere between aggressive and safe portfolio. So I would recommend my client to choose 'safe' portfolio.

Question 3: Wine

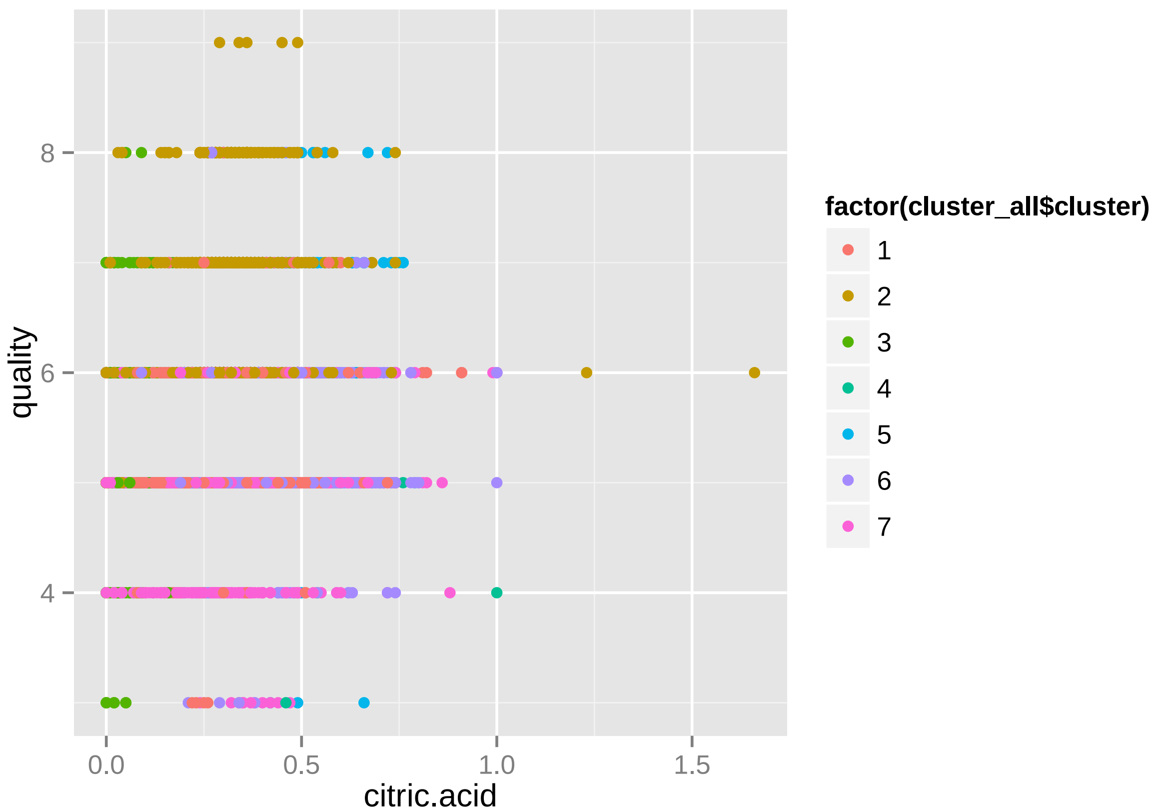
```r
wine <-
  read.csv(
  "/Users/vickyzhang/Documents/MSBA/predictive2/STA380/data/wine.csv", header =
  TRUE, stringsAsFactors = FALSE
  ) # use the last flag to force color to be character
# code color as number
wine[wine$color == 'red','color'] = 1.0
wine[wine$color == 'white','color'] = 0.0
wine$color = as.numeric(wine$color)
wine$quality = as.numeric(wine$quality)
# str(wine)
# is.numeric(wine)


wine_scaled <- scale(wine, center=TRUE, scale=TRUE)
# just cluster 2 features
set.seed(1)
cluster_wine <- kmeans(wine_scaled[,c("fixed.acidity","citric.acid")], centers=3)

plot(wine_scaled[,"citric.acid"], wine_scaled[,"fixed.acidity"], xlim=c(-2,2.75),
    type="n", xlab="citric.acid", ylab="fixed.acidity")
text(wine_scaled[,"citric.acid"], wine_scaled[,"fixed.acidity"], labels=rownames(wine),
    col=rainbow(3)[cluster_wine$cluster])
```




```r
# cluster all features
set.seed(1)
cluster_all <- kmeans(wine_scaled, centers=7, nstart = 50)
cluster_all$centers
```

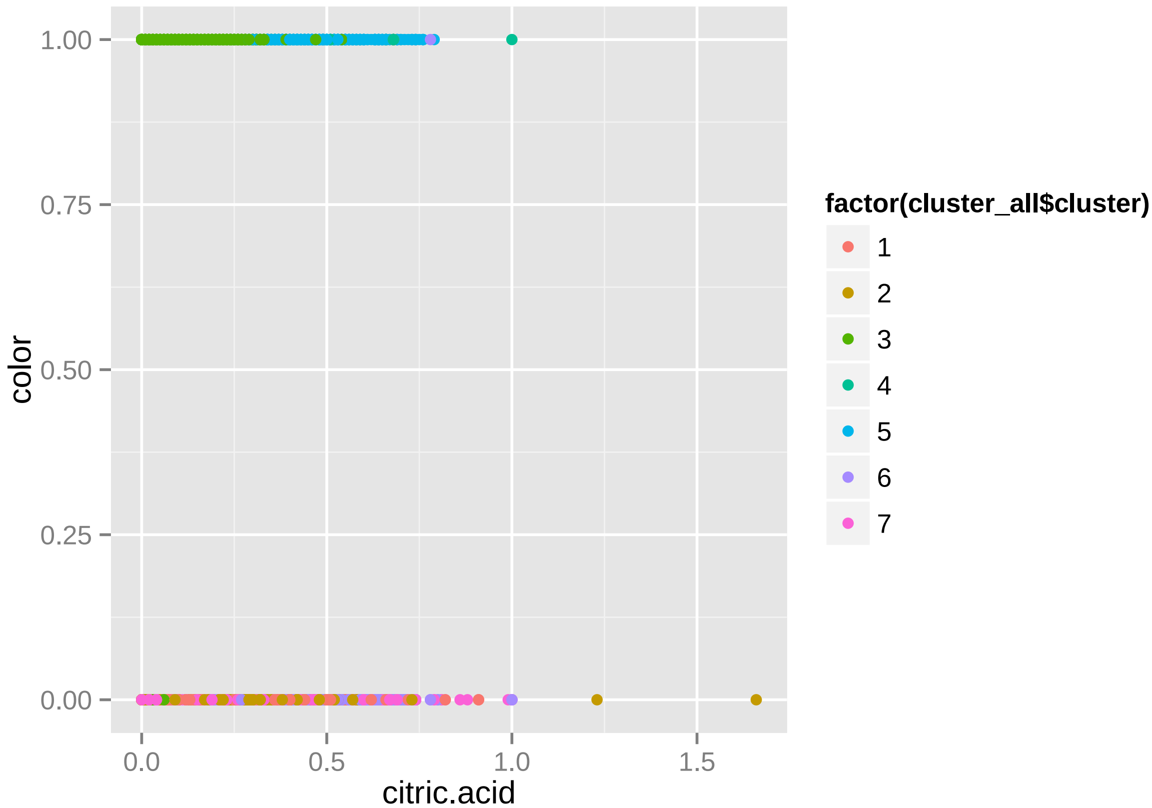
```
##   fixed.acidity volatile.acidity citric.acid residual.sugar  chlorides
## 1   -0.45627926       -0.4623410 -0.03019831      0.1352314 -0.1126683
## 2   -0.48466400       -0.3855687  0.03409887     -0.4167497 -0.5710563
## 3    0.12836500        1.6614953 -1.20767796     -0.6346649  0.7177026
## 4    0.76468203        1.1166055  1.22513121     -0.4911024  9.1331215
## 5    2.03482452        0.3765072  0.96538062     -0.5783566  0.8308879
## 6   -0.08094310       -0.3446103  0.38748280      1.7358889 -0.1764518
## 7   -0.03145708       -0.3248022  0.08416269     -0.4063439 -0.2399427
##   free.sulfur.dioxide total.sulfur.dioxide     density          pH
## 1          0.88017515           0.92372664  0.01803799  0.28733493
## 2          0.02141051          -0.03976414 -1.20174021  0.02544669
## 3         -0.78386706          -1.16392412  0.53784664  0.97110277
## 4         -0.68126169          -0.67309494  0.78680342 -0.95468324
## 5         -0.91418956          -1.35017406  0.93711797  0.01986169
## 6          0.81372446           0.92960630  1.10975841 -0.62069190
## 7         -0.51308358          -0.04356175 -0.46179954 -0.50971145
##    sulphates    alcohol    quality      color
## 1 -0.1949340 -0.4345690 -0.1634969 -0.5671736
## 2 -0.2687299  1.1937129  0.9429578 -0.5498851
## 3  0.4798927 -0.2228147 -0.4722043  1.7283600
## 4  3.7032480 -0.9125990 -0.7844721  1.3631588
## 5  1.3148583  0.1949863  0.1832541  1.7423429
## 6 -0.2184903 -0.9253102 -0.1761398 -0.5692294
## 7 -0.4291427 -0.1490935 -0.5900553 -0.5653087
```

```r
# the following plot shows some relation between cluster number and quality, but
# not very distinct.
qplot(citric.acid, quality, data=wine, color=factor(cluster_all$cluster))
```

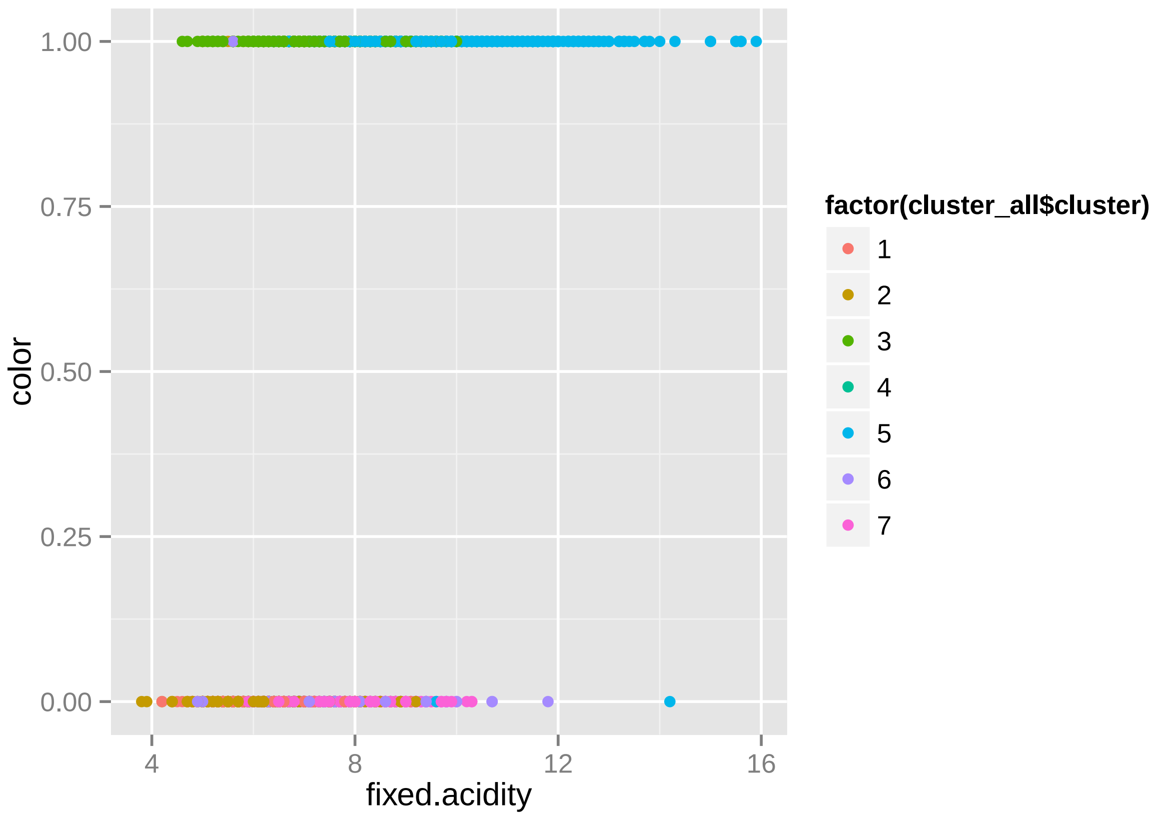


In the following 2 plots, cluster 1, 2, 3 => red wine; cluster 4, 5, 6, 7 => white wine. The clusters are generally effective in recognizing the color of wine. There are a few wrong predictions but not many.

```r
qplot(citric.acid, color, data=wine, color=factor(cluster_all$cluster))
```



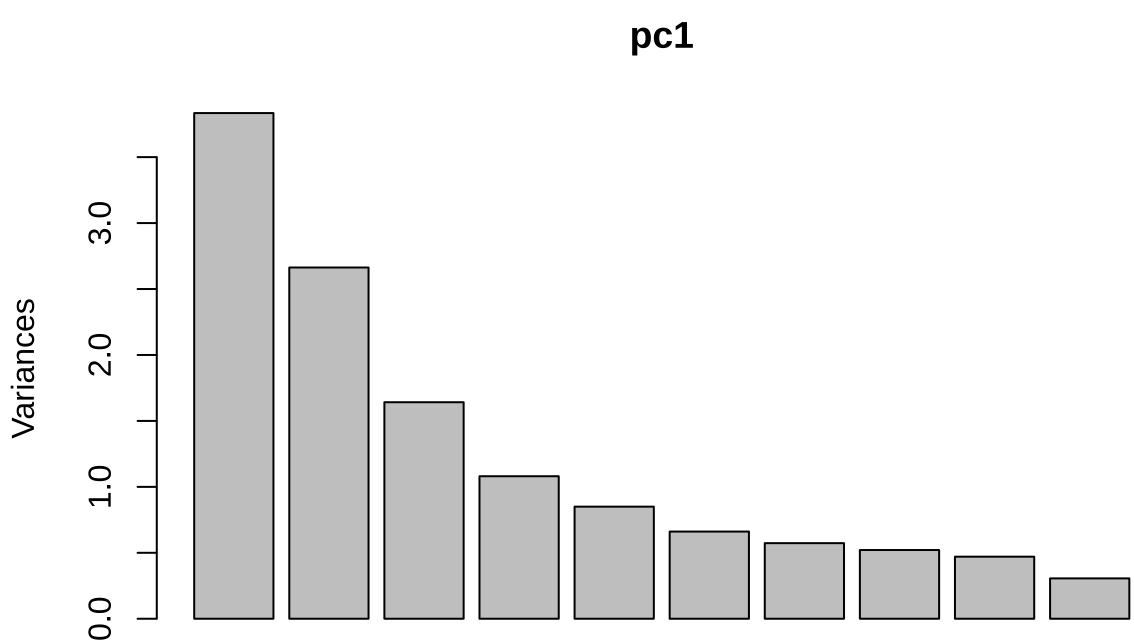

```
qplot(fixed.acidity, color, data=wine, color=factor(cluster_all$cluster))
```



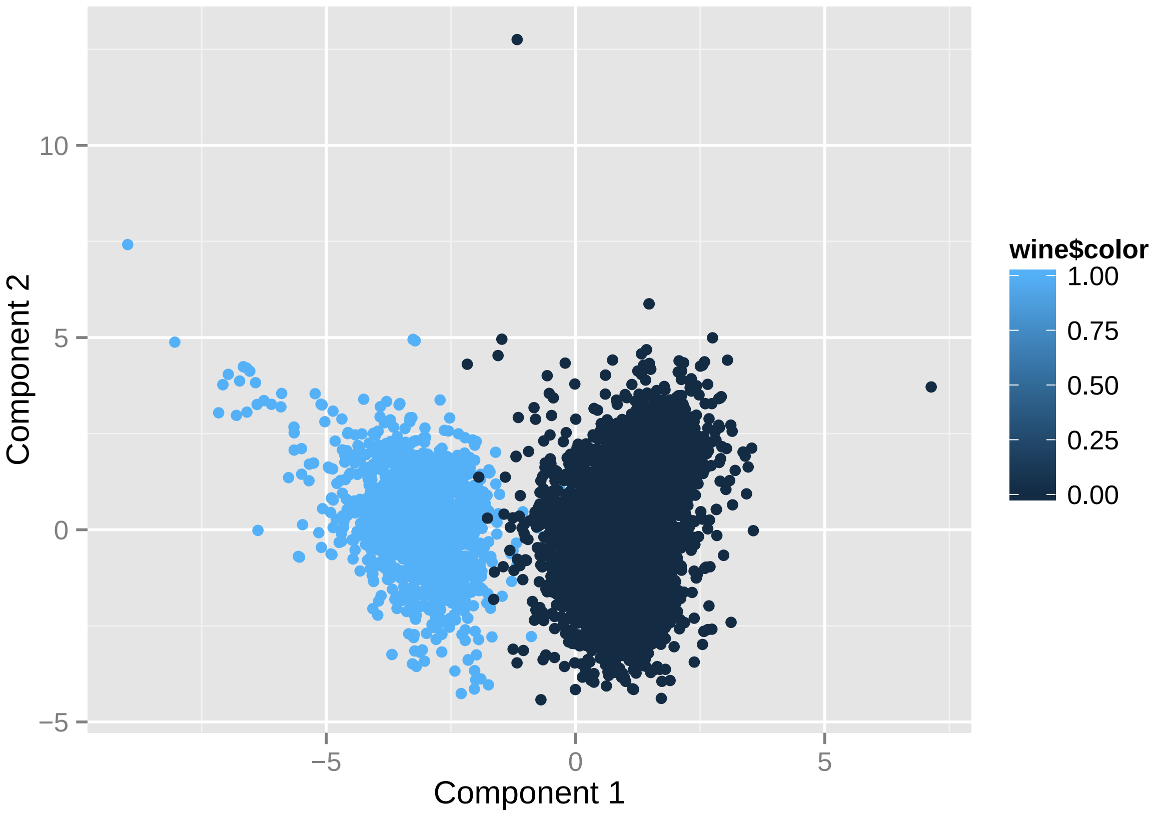
```r
######### PCA
pc1 = prcomp(wine, scale.=TRUE)
#pc1
summary(pc1)
```

```
## Importance of components:
##                           PC1    PC2    PC3     PC4     PC5     PC6
## Standard deviation     1.9581 1.6319 1.2812 1.03947 0.92182 0.81294
## Proportion of Variance 0.2949 0.2048 0.1263 0.08312 0.06537 0.05084
## Cumulative Proportion  0.2949 0.4998 0.6260 0.70916 0.77452 0.82536
##                            PC7     PC8     PC9    PC10    PC11    PC12
## Standard deviation     0.75694 0.72179 0.68590 0.55304 0.50673 0.34535
## Proportion of Variance 0.04407 0.04008 0.03619 0.02353 0.01975 0.00917
## Cumulative Proportion  0.86944 0.90951 0.94570 0.96923 0.98898 0.99815
##                           PC13
## Standard deviation     0.15495
## Proportion of Variance 0.00185
## Cumulative Proportion  1.00000
```

```r
plot(pc1)
```

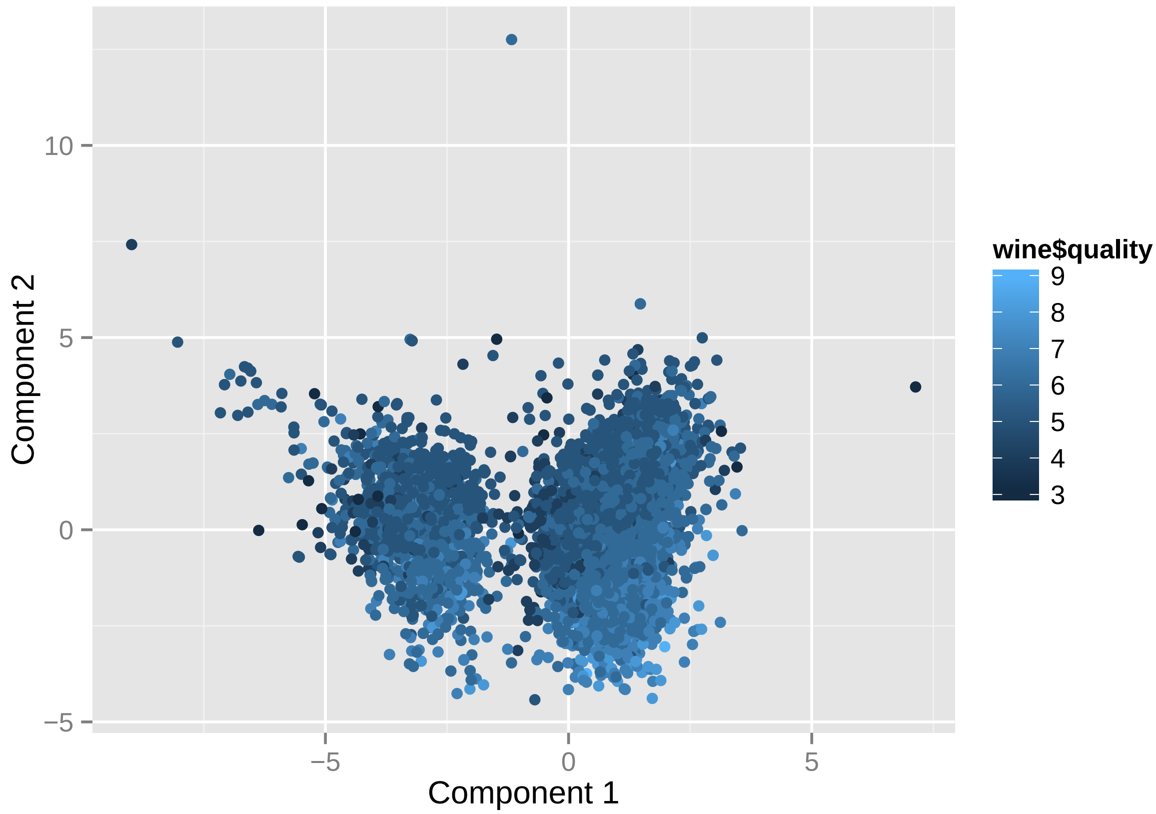



```r
# not very informative plot
#biplot(pc1)

# more informative plot
loadings = pc1$rotation
scores = pc1$x
qplot(scores[,1], scores[,2], color=wine$color, xlab='Component 1', ylab='Component 2')
```



We can clearly see the diff between red and white wine, but there are some overlap area in between the two clusters. For those points, PCA does not do a very good job telling which color it is. In contrast, recall that there is almost no vague area in clustering. In other words, for each data point, as long as you know which cluster it is in, you can get an almost accurate prediction on its color, using clustering. So I would say clustering does a better job than PCA here.

```
qplot(scores[,1], scores[,2], color=wine$quality, xlab='Component 1', ylab='Component 2')
```



As for wine quality, the split between good wine and bad wine isn't very clear under PCA. As we can see from the graph, the dots of different colors are all mixed together. Similar situation in clustering.

Question 4: Twitter marketing segment analysis

```
twitter <-
  read.csv(
  "/Users/vickyzhang/Documents/MSBA/predictive2/STA380/data/social_marketing.csv", header =
  TRUE
  ) # use the last flag to force color to be character
```

Get a brief summary of all features. Most features have a median of 0 and mean of less than 1, which means users don't post more than one tweet on any topic, on average. However the mean and median of chatter are both higher than normal, so either the annotators didn't do a good job or people just like to talk about random things that are hard to categorize.

```
summary(twitter)
```

```
##           X            chatter       current_events      travel
##  123pxkyqj:   1   Min.   : 0.000   Min.   :0.000   Min.   : 0.000
##  12grikctu:   1   1st Qu.: 2.000   1st Qu.:1.000   1st Qu.: 0.000
##  12klxic7j:   1   Median : 3.000   Median :1.000   Median : 1.000
##  12t4msroj:   1   Mean   : 4.399   Mean   :1.526   Mean   : 1.585
##  12yam5913:   1   3rd Qu.: 6.000   3rd Qu.:2.000   3rd Qu.: 2.000
##  132y8f6aj:   1   Max.   :26.000   Max.   :8.000   Max.   :26.000
```

```
## (Other)  :7876
## photo_sharing   uncategorized     tv_film      sports_fandom
## Min.   : 0.000  Min.   :0.000   Min.   : 0.00   Min.   : 0.000
## 1st Qu.: 1.000  1st Qu.:0.000   1st Qu.: 0.00   1st Qu.: 0.000
## Median : 2.000  Median :1.000   Median : 1.00   Median : 1.000
## Mean   : 2.697  Mean   :0.813   Mean   : 1.07   Mean   : 1.594
## 3rd Qu.: 4.000  3rd Qu.:1.000   3rd Qu.: 1.00   3rd Qu.: 2.000
## Max.   :21.000  Max.   :9.000   Max.   :17.00   Max.   :20.000
##
##    politics          food            family        home_and_garden
## Min.   : 0.000   Min.   : 0.000   Min.   : 0.0000   Min.   :0.0000
## 1st Qu.: 0.000   1st Qu.: 0.000   1st Qu.: 0.0000   1st Qu.:0.0000
## Median : 1.000   Median : 1.000   Median : 1.0000   Median :0.0000
## Mean   : 1.789   Mean   : 1.397   Mean   : 0.8639   Mean   :0.5207
## 3rd Qu.: 2.000   3rd Qu.: 2.000   3rd Qu.: 1.0000   3rd Qu.:1.0000
## Max.   :37.000   Max.   :16.000   Max.   :10.0000   Max.   :5.0000
##
##      music              news         online_gaming       shopping
## Min.   : 0.0000   Min.   : 0.000   Min.   : 0.000   Min.   : 0.000
## 1st Qu.: 0.0000   1st Qu.: 0.000   1st Qu.: 0.000   1st Qu.: 0.000
## Median : 0.0000   Median : 0.000   Median : 0.000   Median : 1.000
## Mean   : 0.6793   Mean   : 1.206   Mean   : 1.209   Mean   : 1.389
## 3rd Qu.: 1.0000   3rd Qu.: 1.000   3rd Qu.: 1.000   3rd Qu.: 2.000
## Max.   :13.0000   Max.   :20.000   Max.   :27.000   Max.   :12.000
##
## health_nutrition  college_uni     sports_playing     cooking
## Min.   : 0.000   Min.   : 0.000   Min.   :0.0000   Min.   : 0.000
## 1st Qu.: 0.000   1st Qu.: 0.000   1st Qu.:0.0000   1st Qu.: 0.000
## Median : 1.000   Median : 1.000   Median :0.0000   Median : 1.000
## Mean   : 2.567   Mean   : 1.549   Mean   :0.6392   Mean   : 1.998
## 3rd Qu.: 3.000   3rd Qu.: 2.000   3rd Qu.:1.0000   3rd Qu.: 2.000
## Max.   :41.000   Max.   :30.000   Max.   :8.0000   Max.   :33.000
##
##       eco           computers          business         outdoors
## Min.   :0.0000   Min.   : 0.0000   Min.   :0.0000   Min.   : 0.0000
## 1st Qu.:0.0000   1st Qu.: 0.0000   1st Qu.:0.0000   1st Qu.: 0.0000
## Median :0.0000   Median : 0.0000   Median :0.0000   Median : 0.0000
## Mean   :0.5123   Mean   : 0.6491   Mean   :0.4232   Mean   : 0.7827
## 3rd Qu.:1.0000   3rd Qu.: 1.0000   3rd Qu.:1.0000   3rd Qu.: 1.0000
## Max.   :6.0000   Max.   :16.0000   Max.   :6.0000   Max.   :12.0000
##
##     crafts          automotive           art             religion
## Min.   :0.0000   Min.   : 0.0000   Min.   : 0.0000   Min.   : 0.000
## 1st Qu.:0.0000   1st Qu.: 0.0000   1st Qu.: 0.0000   1st Qu.: 0.000
## Median :0.0000   Median : 0.0000   Median : 0.0000   Median : 0.000
## Mean   :0.5159   Mean   : 0.8299   Mean   : 0.7248   Mean   : 1.095
## 3rd Qu.:1.0000   3rd Qu.: 1.0000   3rd Qu.: 1.0000   3rd Qu.: 1.000
## Max.   :7.0000   Max.   :13.0000   Max.   :18.0000   Max.   :20.000
##
##     beauty           parenting           dating             school
## Min.   : 0.0000   Min.   : 0.0000   Min.   : 0.0000   Min.   : 0.0000
## 1st Qu.: 0.0000   1st Qu.: 0.0000   1st Qu.: 0.0000   1st Qu.: 0.0000
## Median : 0.0000   Median : 0.0000   Median : 0.0000   Median : 0.0000
## Mean   : 0.7052   Mean   : 0.9213   Mean   : 0.7109   Mean   : 0.7677
```

```
##  3rd Qu.: 1.0000   3rd Qu.: 1.0000   3rd Qu.: 1.0000   3rd Qu.: 1.0000  
##  Max.   :14.0000   Max.   :14.0000   Max.   :24.0000   Max.   :11.0000  
##                                                                         
##  personal_fitness    fashion        small_business        spam        
##  Min.   : 0.000   Min.   : 0.0000   Min.   :0.0000   Min.   :0.00000  
##  1st Qu.: 0.000   1st Qu.: 0.0000   1st Qu.:0.0000   1st Qu.:0.00000  
##  Median : 0.000   Median : 0.0000   Median :0.0000   Median :0.00000  
##  Mean   : 1.462   Mean   : 0.9966   Mean   :0.3363   Mean   :0.00647  
##  3rd Qu.: 2.000   3rd Qu.: 1.0000   3rd Qu.:1.0000   3rd Qu.:0.00000  
##  Max.   :19.000   Max.   :18.0000   Max.   :6.0000   Max.   :2.00000  
##                                                                       
##      adult        
##  Min.   : 0.0000  
##  1st Qu.: 0.0000  
##  Median : 0.0000  
##  Mean   : 0.4033  
##  3rd Qu.: 0.0000  
##  Max.   :26.0000  
## 
```

Get rid of rows with NA, get rid of random userID. There's no problem with just using row index as user ID. Also, no need to do scaling here because all data are on the same scale.

```r
twitter = twitter[complete.cases(twitter),][,-1]
# cluster on all features
cluster_all <- kmeans(twitter, centers=7, nstart = 50)
cluster_all$centers
```

```
##    chatter current_events   travel photo_sharing uncategorized  tv_film
## 1 4.036442       1.539121 1.324759      2.429796     0.9603430 1.039657
## 2 2.932176       1.358112 1.121812      1.512480     0.7175800 1.041508
## 3 3.528830       1.640857 1.210873      2.092257     0.7199341 1.024712
## 4 4.002028       1.718053 1.456389      6.008114     1.2089249 1.010142
## 5 4.062837       1.682226 6.497307      2.258528     0.7307002 1.168761
## 6 9.999173       1.853598 1.196030      5.682382     0.9023987 1.062862
## 7 4.105793       1.428212 1.521411      2.654912     0.8463476 1.438287
##   sports_fandom   politics      food    family home_and_garden     music
## 1     1.2722401  1.3301179 2.1521972 0.8049303       0.6141479 0.7491961
## 2     0.9755833  0.9956593 0.7932718 0.5586001       0.4302767 0.5458492
## 3     6.3064250  0.9736409 4.7413509 2.4629325       0.6507414 0.7051071
## 4     1.3002028  1.3711968 1.1156187 0.9290061       0.6044625 1.2048682
## 5     1.9802513 10.0879713 1.5691203 0.9192101       0.5691203 0.6193896
## 6     1.3490488  1.4855252 1.0264682 0.9040529       0.5889165 0.8006617
## 7     1.4559194  1.2720403 1.3602015 1.1108312       0.5617128 0.7758186
##        news online_gaming  shopping health_nutrition college_uni
## 1 1.2411576     0.9474812 1.3483387       12.5251876   1.0375134
## 2 0.8060228     0.5515464 0.7232773        0.9568638   0.8979924
## 3 1.0494234     0.8270181 1.1499176        1.5881384   1.0939044
## 4 1.0831643     1.0466531 1.7322515        2.0567951   1.4543611
## 5 5.1633752     0.8563734 1.1992819        1.4380610   1.3554758
## 6 0.8147229     0.7758478 3.5723739        1.3258892   1.2266336
## 7 0.8589421    10.5239295 1.2292191        1.6120907  10.8715365
##   sports_playing    cooking       eco computers   business   outdoors
```

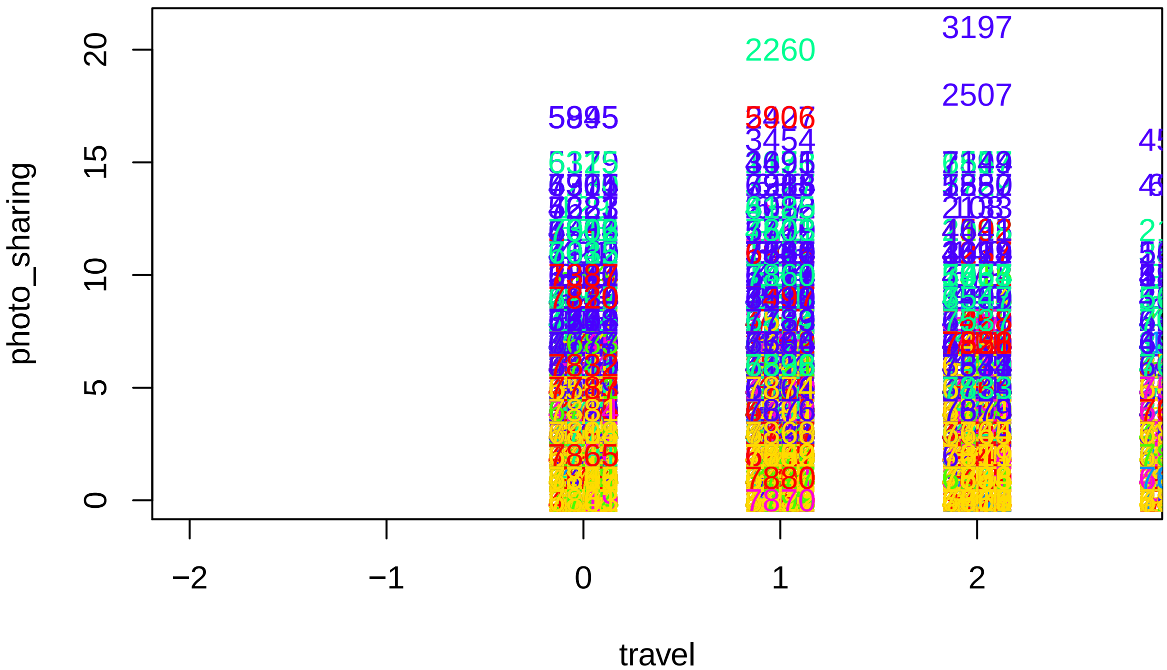
```
## 1      0.6366559  3.4072883 0.8574491 0.5605573 0.4501608 2.4973205
## 2      0.4153554  0.8298969 0.3415627 0.3594683 0.3024959 0.4500814
## 3      0.7084020  1.2767710 0.6095552 0.7215815 0.4645799 0.6754530
## 4      0.8519270 12.1663286 0.5253550 0.7464503 0.5578093 0.8235294
## 5      0.6355476  1.2531418 0.5870736 2.7055655 0.6678636 0.8797127
## 6      0.5922250  1.1695616 0.6923077 0.6054591 0.5988420 0.5161290
## 7      2.5012594  1.5793451 0.4685139 0.5617128 0.3727960 0.6297229
##      crafts automotive       art  religion    beauty parenting    dating
## 1 0.6002144  0.6784566 0.8210075 0.8703108 0.5219721 0.8360129 1.0032154
## 2 0.3372219  0.5721649 0.6418882 0.5596853 0.3827998 0.4612046 0.4571351
## 3 1.0444811  1.0197694 0.8023064 5.4958814 1.0362438 4.1565074 0.6079077
## 4 0.5841785  0.8498986 0.9046653 1.0141988 3.8762677 0.9006085 0.6004057
## 5 0.6211849  2.0089767 0.6499102 1.2208259 0.4991023 1.0341113 1.0789946
## 6 0.6261373  1.0488007 0.6641853 0.7245658 0.5301902 0.7427626 1.1803143
## 7 0.5994962  0.9420655 1.2166247 0.9244332 0.5062972 0.8589421 0.7279597
##      school personal_fitness   fashion small_business        spam
## 1 0.6302251        6.2411576 0.8210075      0.2647374 0.006430868
## 2 0.4354314        0.6294086 0.5092241      0.2637005 0.007596310
## 3 2.7018122        1.0593081 0.9242175      0.3937397 0.006589786
## 4 0.9716024        1.3103448 5.7464503      0.4563895 0.004056795
## 5 0.7504488        0.9389587 0.7001795      0.4991023 0.007181329
## 6 0.8916460        0.9627792 0.8701406      0.4317618 0.003308519
## 7 0.6120907        1.0201511 0.9471033      0.4231738 0.007556675
##       adult
## 1 0.3247588
## 2 0.4495388
## 3 0.4299835
## 4 0.4442191
## 5 0.2351885
## 6 0.3606286
## 7 0.4332494
```

Conclusion

Going through each cluster, we can delineate a 'portrait' for each group:
1. group 1 - online-gaming(10) college(10) student. got 10+ on both online_gaming and college_uni. 2+ on sports_playing too, which fits into the picture. Inference: likely male, aged 18-22. could market these products to them: World of WarCraft expansion pack, ergonomic keyboard / mouse / computer chair, microwavable dinner.
2. group 2 - traveler(6) passionate about politics(10), news(5) and automotive(2). Inference: likely male. Potential buyer of : Online news/critics subscription, traveler magazine, Lonely Planet, Travel channel, suitcases, cars
3. group 3 - uninterested in everything. Potential buyer of : basic living necessities (since we can't figure out what else they need)
4. group 4 - photo-sharing(5) shopper(3). Inference: likely to be women. Potential buyer of : fashion, and possibly everything else
5. group 5 - photo-sharing(2) sports-loving(6) foodie(4), care about family(2), religion(5), parenting(4), school(2). inference: married and have kids. more likely to be female. Potential customer of : sports tickets, Yelp, religious material, parenting websites
6. group 6 - photo-sharing(6), health-nutrition(2), cooking(12), beauty(3), fashion(5). likely to be a women or even a wife. Potential customer of : Martha Stewart, Food Network, Instagram
7. group 7 - photo-sharing(2), food(2), health-nutrition(12), cooking(3), outdoors(2), personal_fitness(6). seems to be some very health-aware people. Potential buyer of : whey protein, pre-workout, fitness coaching,

nutrition planning

```r
plot(twitter[,"travel"], twitter[,"photo_sharing"], xlim=c(-2,2.75),
    type="n", xlab="travel", ylab="photo_sharing")
text(twitter[,"travel"], twitter[,"photo_sharing"], labels=rownames(twitter),
    col=rainbow(7)[cluster_all$cluster])
```




```r
# doesn't show much information, all colors seem to be mixed together

# try PCA
pc1 = prcomp(twitter, scale.=TRUE)
#pc1
```

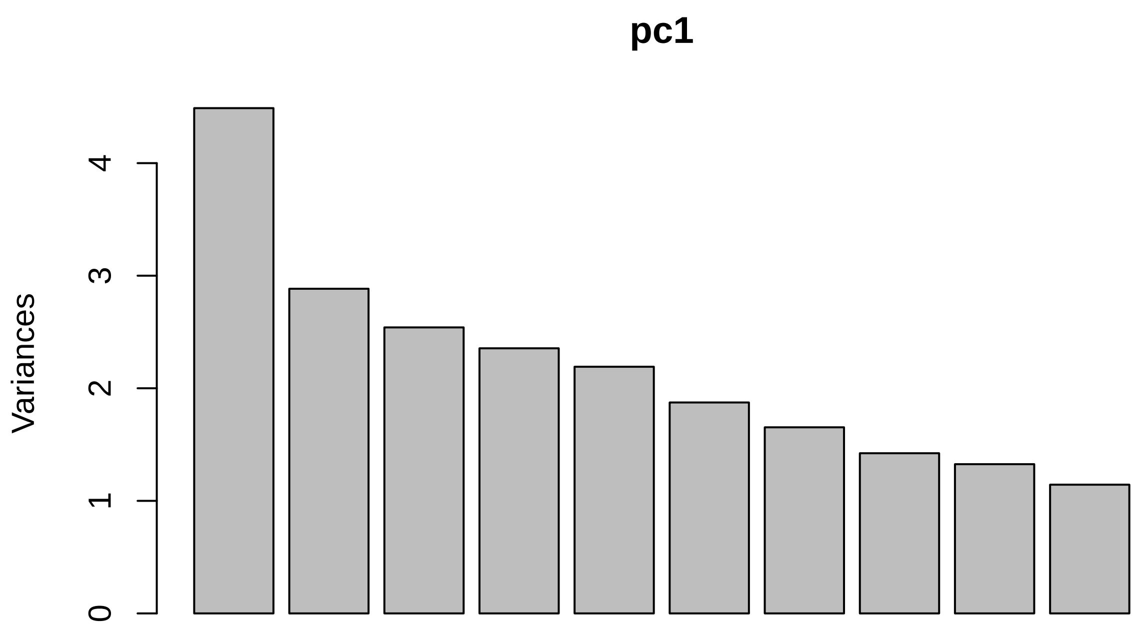
Not showing pc1 results here because it’s very long and I found it not as informative as clustering here. For clustering, interpretability is quite good, you can immediately start talking about each cluster as a group of users; but as for PCA, the principal components don’t have direct meanings.

```r
summary(pc1)
```

```
## Importance of components:
##                           PC1     PC2     PC3     PC4     PC5     PC6
## Standard deviation     2.1186 1.69824 1.59388 1.53457 1.48027 1.36885
## Proportion of Variance 0.1247 0.08011 0.07057 0.06541 0.06087 0.05205
## Cumulative Proportion  0.1247 0.20479 0.27536 0.34077 0.40164 0.45369
##                            PC7     PC8     PC9    PC10    PC11    PC12
## Standard deviation     1.28577 1.19277 1.15127 1.06930 1.00566 0.96785
## Proportion of Variance 0.04592 0.03952 0.03682 0.03176 0.02809 0.02602
## Cumulative Proportion  0.49961 0.53913 0.57595 0.60771 0.63580 0.66182
##                           PC13    PC14    PC15    PC16   PC17    PC18
## Standard deviation     0.96131 0.94405 0.93297 0.91698 0.9020 0.85869
## Proportion of Variance 0.02567 0.02476 0.02418 0.02336 0.0226 0.02048
```

```
## Cumulative Proportion  0.68749 0.71225 0.73643 0.75979 0.7824 0.80287
##                           PC19    PC20    PC21    PC22    PC23    PC24
## Standard deviation     0.83466 0.80544 0.75311 0.69632 0.68558 0.65317
## Proportion of Variance 0.01935 0.01802 0.01575 0.01347 0.01306 0.01185
## Cumulative Proportion  0.82222 0.84024 0.85599 0.86946 0.88252 0.89437
##                           PC25    PC26    PC27    PC28    PC29    PC30
## Standard deviation     0.64881 0.63756 0.63626 0.61513 0.60167 0.59424
## Proportion of Variance 0.01169 0.01129 0.01125 0.01051 0.01006 0.00981
## Cumulative Proportion  0.90606 0.91735 0.92860 0.93911 0.94917 0.95898
##                           PC31   PC32    PC33    PC34    PC35    PC36
## Standard deviation     0.58683 0.5498 0.48442 0.47576 0.43757 0.42165
## Proportion of Variance 0.00957 0.0084 0.00652 0.00629 0.00532 0.00494
## Cumulative Proportion  0.96854 0.9769 0.98346 0.98974 0.99506 1.00000
```

```r
plot(pc1)
```




```r
# not very informative plot
#biplot(pc1)

# more informative plot
loadings = pc1$rotation
scores = pc1$x
# There are 36 principal components and each of them don't contribute much to
# cumulative proportion
```