BRIAN LAKEY

QUESTION 1

Loading the necessary data and libraries

```
library(mosaic)
```

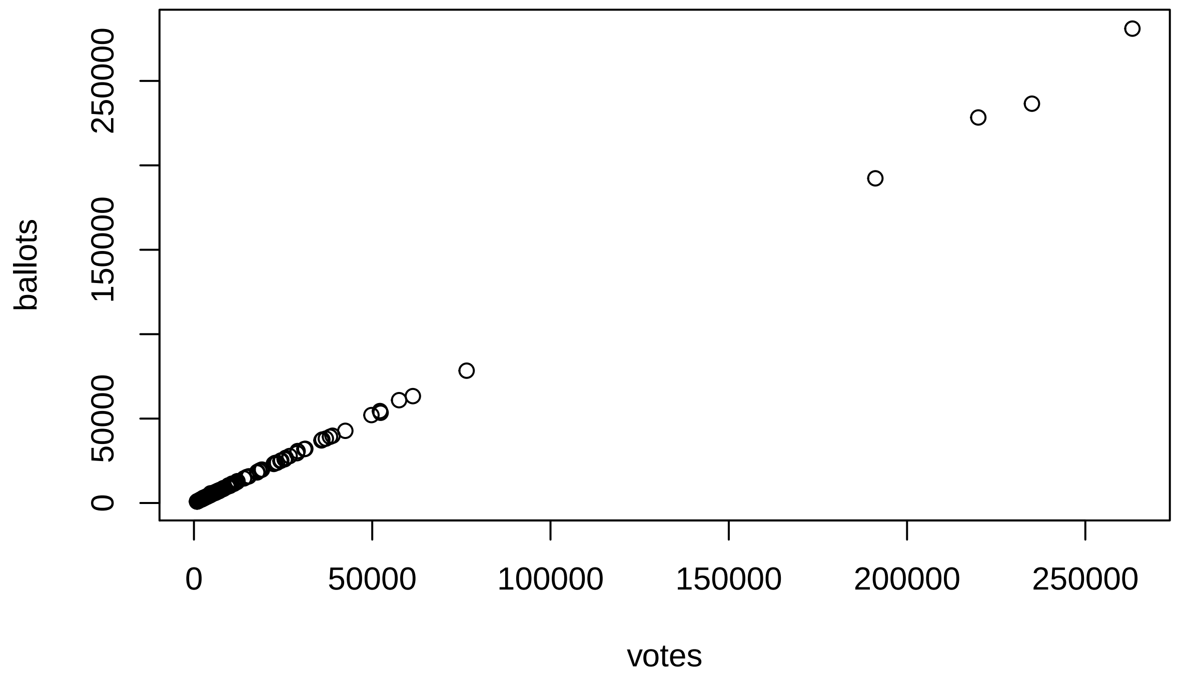
```
## Loading required package: car
## Loading required package: dplyr
##
## Attaching package: 'dplyr'
##
## The following objects are masked from 'package:stats':
##
##     filter, lag
##
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
##
## Loading required package: lattice
## Loading required package: ggplot2
## 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
```

```
voting = read.csv('C:/Users/Brian/Desktop/MS-BA/Summer/Intro to Predictive Modeling/Section 2/STA380/da
attach(voting)
head(voting)
```

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

The first step is to examine the data, and a get a feel for the spread between cast ballots and counted votes.

```
plot(ballots~votes)
```



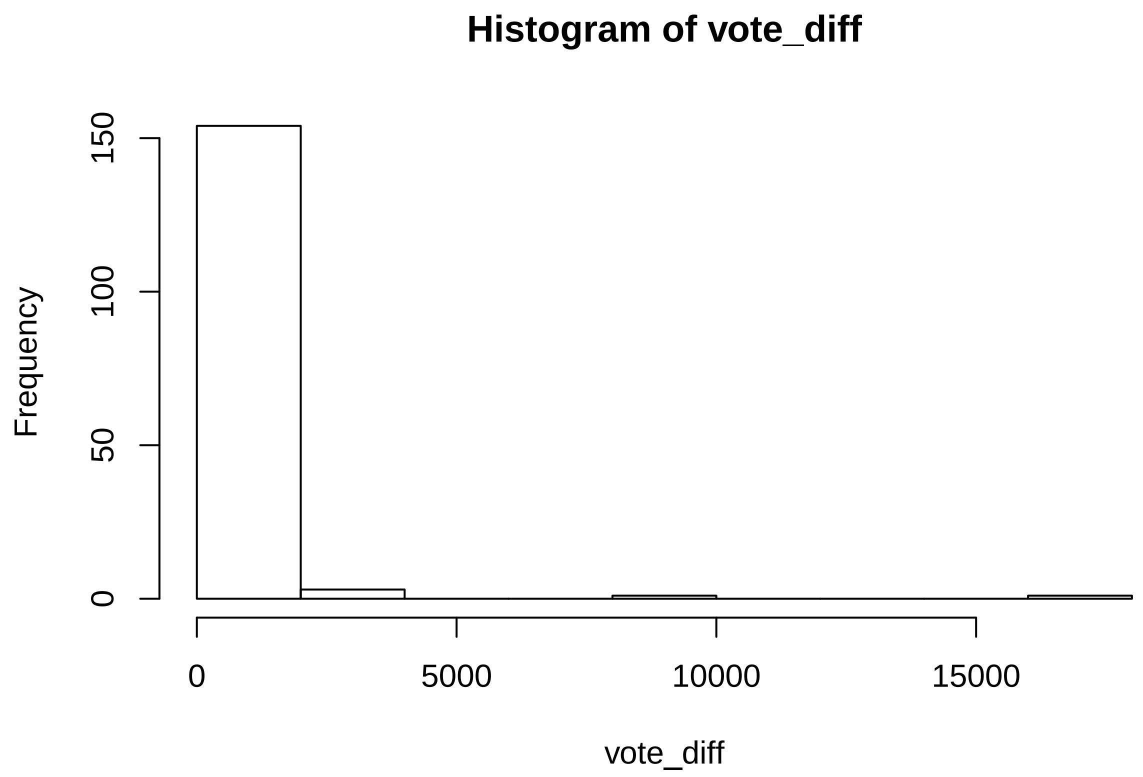
The relationship seems pretty linear, as one would expect - as cast ballots increase, so too do counted ballots.

The next step is to quantify the spread, both in absolute and relative terms (by county). Computing solely absolute deltas between cast and counted ballots would effectively be creating a population map (see https://imgs.xkcd.com/comics/heatmap.png) Appending two new variables: differential between cast and counted votes, both in absolute numbers and in proportion #to ballots cast

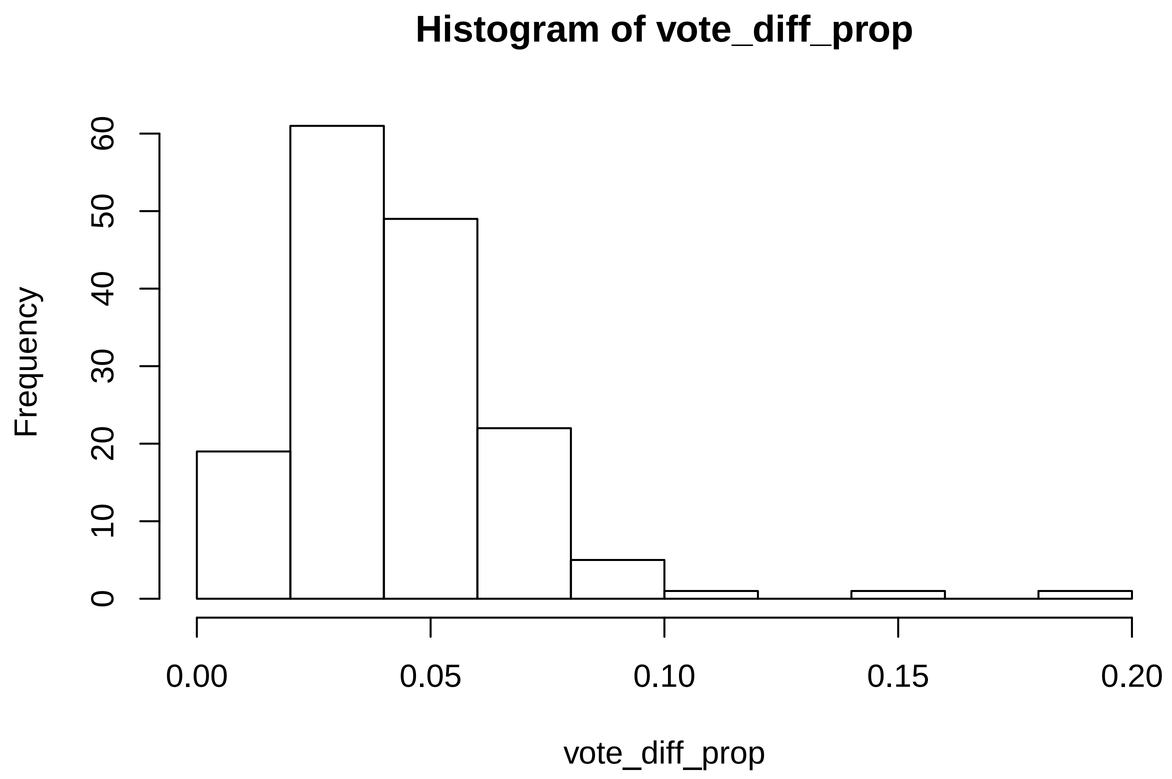
```
vote_diff=ballots-votes
vote_diff_prop=(ballots-votes)/ballots
voting=data.frame(voting, vote_diff, vote_diff_prop)
```

Some exploratory graphs of vote count differentials:

```
hist(vote_diff)
```



```r
hist(vote_diff_prop)
```



```r
range(vote_diff_prop)
```

```
## [1] 0.0000000 0.1881205
```

```r
summary(vote_diff_prop)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
## 0.00000 0.02779 0.03983 0.04379 0.05647 0.18810
```

```r
sd(vote_diff_prop)
```

```
## [1] 0.02496544
```

The majority of counties report less than 5% undercount rate, with a range up to 18% at the maximum. Standar deviation is around 2.5%

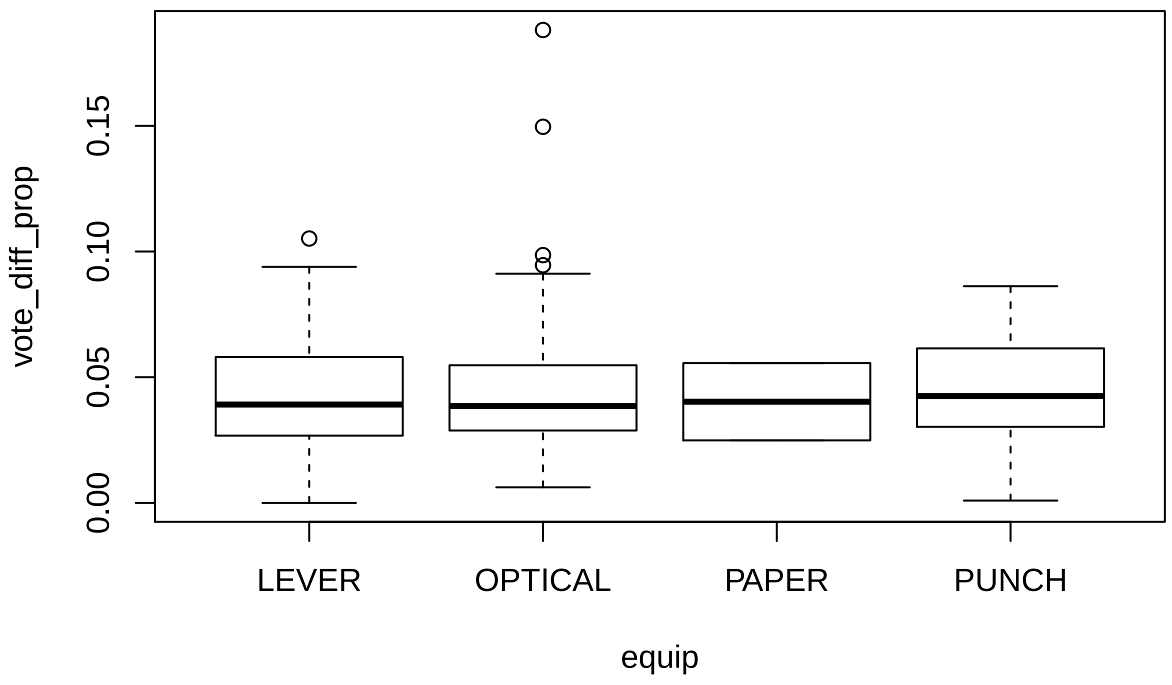
The question asks us to examine voting by machines, and if there is any evidene of systematic voting machine influence on undercount. First step is to examine the distribution of voting machines types.

```r
summary(equip) #predominantly lever and optical
```

```
##   LEVER OPTICAL   PAPER   PUNCH 
##      74      66       2      17
```

The next step is to bin undercount proportions by machine type. It is important to use the rate of undercount rather than the absolute count, as far greater ballots (and thus uncounted ballots, given the linear relationship) were cast on lever and optical machines.

```r
plot(vote_diff_prop~equip)
```



```r
equip_diff = aggregate(vote_diff_prop, list(equip), summary)
equip_diff
```

```
##   Group.1    x.Min. x.1st Qu.  x.Median    x.Mean x.3rd Qu.    x.Max.
## 1   LEVER 0.0000000 0.0267900 0.0391200 0.0418900 0.0577000 0.1052000
## 2 OPTICAL 0.0062040 0.0289200 0.0385100 0.0451800 0.0545200 0.1881000
## 3   PAPER 0.0248700 0.0325600 0.0402500 0.0402500 0.0479300 0.0556200
## 4   PUNCH 0.0009149 0.0302600 0.0424900 0.0470900 0.0614700 0.0862000
```

```r
equip_diff_anova = aov(vote_diff_prop ~ equip)
summary(equip_diff_anova)
```

```
##              Df  Sum Sq   Mean Sq F value Pr(>F)
## equip         3 0.00060 0.0002013   0.319  0.812
## Residuals   155 0.09787 0.0006314
```

From the non-significant results of the analysis of variance, it does not appear that voting equipment played a significant role in influencing vote discrepancies.

However, this isn't entirely a satisfying answer to the question of vote undercount. Let's double-check by forcing a linear regression model to equipment:

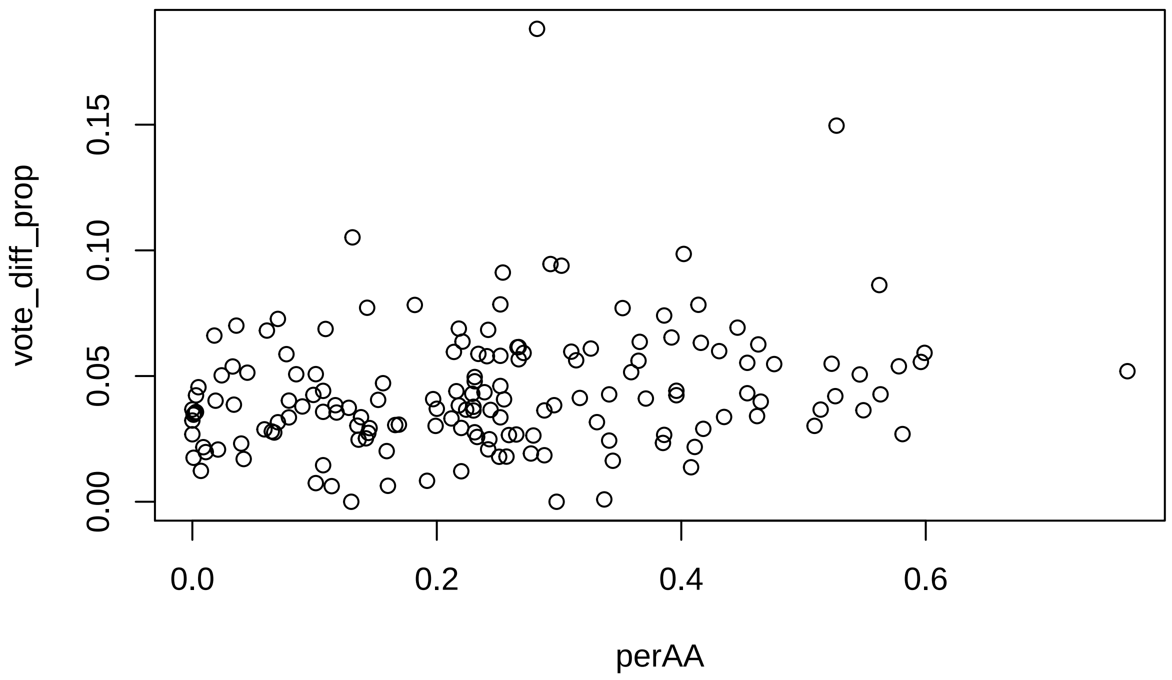
```r
county_lm=lm(vote_diff_prop~equip)
summary(county_lm)
```

```
##
## Call:
## lm(formula = vote_diff_prop ~ equip)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.046179 -0.015368 -0.004659  0.012349  0.142943
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)   0.041894   0.002921  14.342   <2e-16 ***
## equipOPTICAL  0.003284   0.004254   0.772    0.441
## equipPAPER   -0.001647   0.018007  -0.091    0.927
## equipPUNCH    0.005200   0.006758   0.769    0.443
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.02513 on 155 degrees of freedom
## Multiple R-squared:  0.006131,   Adjusted R-squared:  -0.0131
## F-statistic: 0.3187 on 3 and 155 DF,  p-value: 0.8118
```

With none of the p-values significant for any of the machine type coeffecients, it butresses our argument that voting equipment does not seemed to have played a role in vote counting

Out of curiosity, indpendent of voting machines, is there a relationship between the proportion of African American county residents and proportion of vote undercount?

```r
plot(vote_diff_prop~perAA)
```



```r
race_lm=lm(vote_diff_prop~perAA)
summary(race_lm)
```

```
##
## Call:
## lm(formula = vote_diff_prop ~ perAA)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.046183 -0.015228 -0.003821  0.012253  0.142957
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 0.035248   0.003476  10.140  < 2e-16 ***
## perAA       0.035162   0.011891   2.957  0.00359 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.02438 on 157 degrees of freedom
## Multiple R-squared:  0.05276,    Adjusted R-squared:  0.04672
## F-statistic: 8.744 on 1 and 157 DF,  p-value: 0.003586
```

While the relationship plot does not paint a strong visual picture, from the results of the linear regression, if we assume a linear relationship, proportion of African-American residents within a county does appear to have a significant relationship with vote undercount.

QUESTION 2

```r
library(mosaic)
library(fImport)
```

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

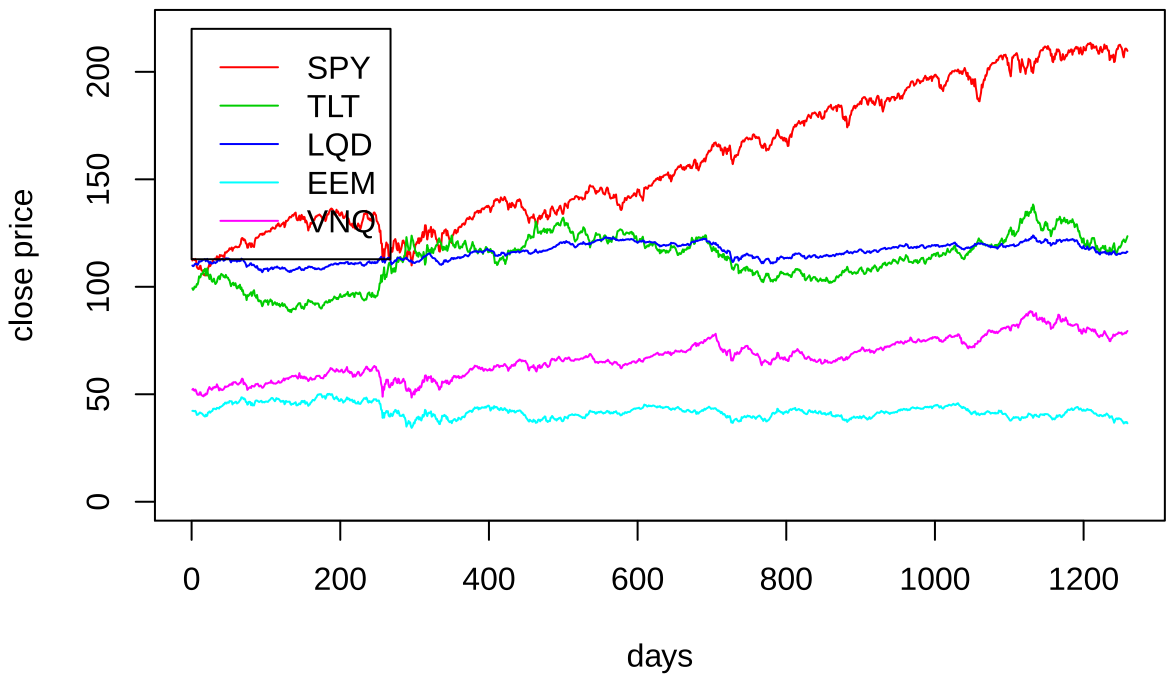
```r
library(foreach)
```

The first step is to collect the stock information over the last five years for the following-exchange traded funds: * US domestic equities (SPY: the S&P 500 stock index) * US Treasury bonds (TLT) * Investment-grade corporate bonds (LQD)
Emerging-market equities (EEM * Real estate (VNQ)

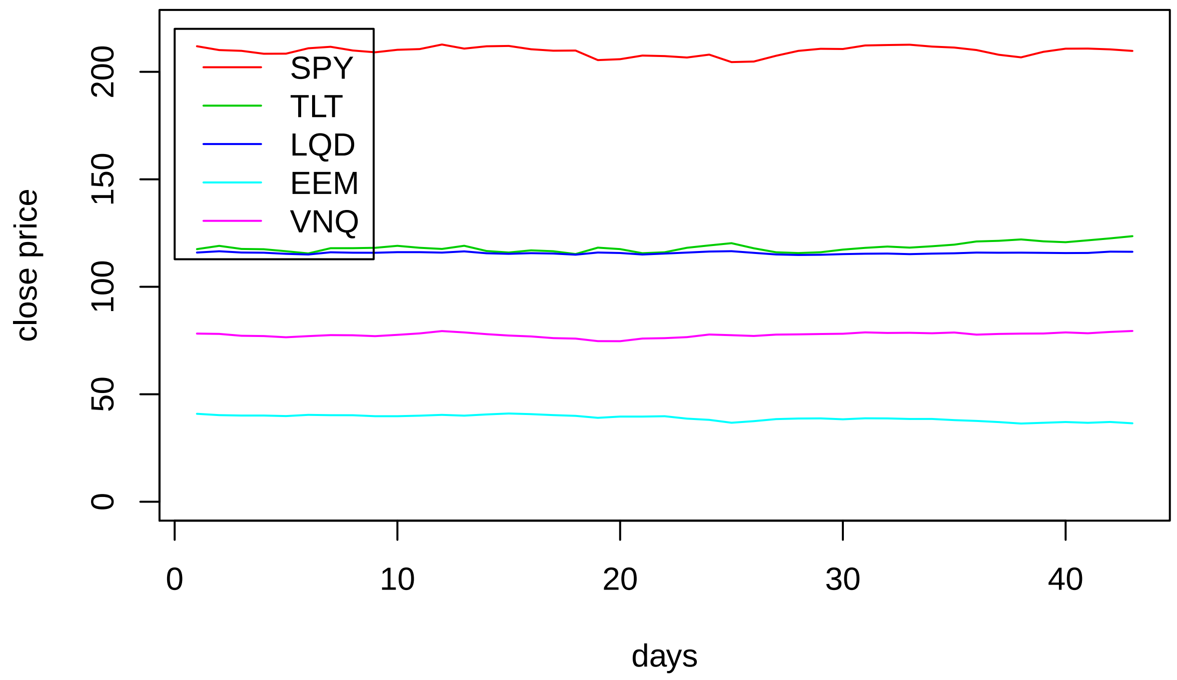
```r
candidate_stocks = c("SPY","TLT","LQD","EEM","VNQ")
candidate_prices = yahooSeries(candidate_stocks, from='2010-08-03', to='2015-08-03')
```

Any good first step in approaching data analysis is to get a feel for some of the data properties. The next few charts plot closing prices over the entire 5-year period, as well as the previous month of trading.

```r
plot(candidate_prices$SPY.Close, ylim=c(0,220), type='l', col=2, xlab='days', ylab='close price')
lines(candidate_prices$TLT.Close, type='l', col=3)
lines(candidate_prices$LQD.Close, type='l', col=4)
lines(candidate_prices$EEM.Close, type='l', col=5)
lines(candidate_prices$VNQ.Close, type='l', col=6)
legend(0,220, c("SPY","TLT","LQD","EEM","VNQ"), col = c(2,3,4,5,6), lty=1)
```



```r
prices_month=window(candidate_prices, "2015-06-03", "2015-08-03")
plot(prices_month$SPY.Close, ylim=c(0,220), type='l', col=2, xlab='days', ylab='close price')
lines(prices_month$TLT.Close, type='l', col=3)
lines(prices_month$LQD.Close, type='l', col=4)
lines(prices_month$EEM.Close, type='l', col=5)
lines(prices_month$VNQ.Close, type='l', col=6)
legend(0,220, c("SPY","TLT","LQD","EEM","VNQ"), col = c(2,3,4,5,6), lty=1)
```



As interesting as the overall performances of the ETFs, it will be more useful in protfolio management to investigate the percent returns of each ETF. To do this, we have a (very helpfully provided) helper function to compute returns from closing prices, which we will define here.

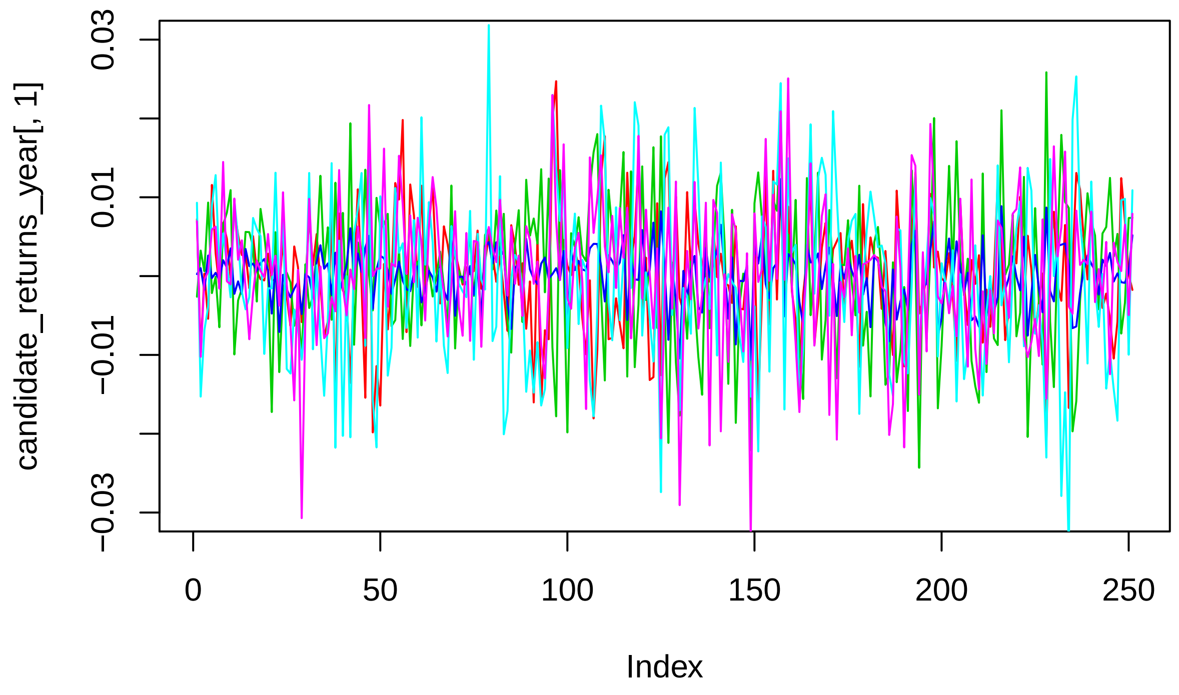
```
YahooPricesToReturns = function(series) {
  mycols = grep('Adj.Close', colnames(series))
  closingprice = series[,mycols]
  N = nrow(closingprice)
  percentreturn = as.data.frame(closingprice[2:N,]) / as.data.frame(closingprice[1:(N-1),]) - 1
  mynames = strsplit(colnames(percentreturn), '.', fixed=TRUE)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

As well as compute returns from the closing prices for each stock for the full 5-year period, the previous year, and the previous month.

```
candidate_returns = YahooPricesToReturns(candidate_prices)
candidate_returns_year = YahooPricesToReturns(candidate_prices[1007:1258,])
candidate_returns_month = YahooPricesToReturns(candidate_prices[1237:1258,])
```

Let’s look at percent returns for stocks over the last five years:

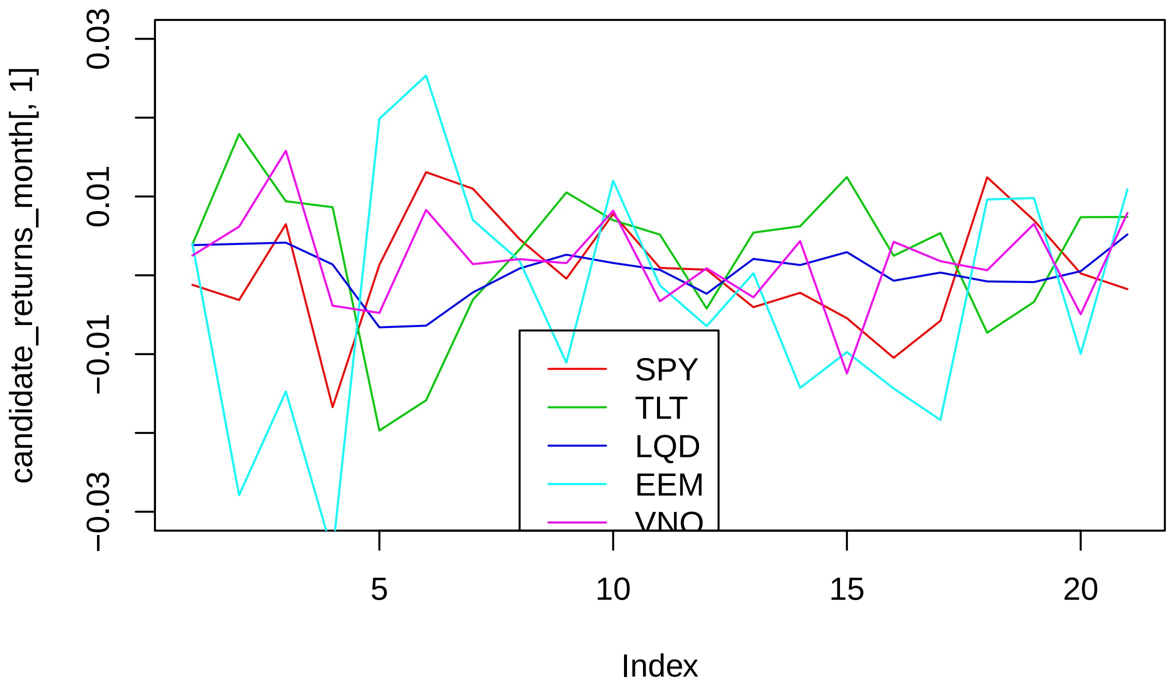
```r
plot(candidate_returns_year[,1], ylim=c(-.03,.03), type='l', col=2)
lines(candidate_returns_year[,2], type='l', col=3)
lines(candidate_returns_year[,3], type='l', col=4)
lines(candidate_returns_year[,4], type='l', col=5)
lines(candidate_returns_year[,5], type='l', col=6)
```



Looks rather volatile. Not particularly helpful.

How about for the last month:

```r
plot(candidate_returns_month[,1], ylim=c(-.03,.03), type='l', col=2) #moderately aggressive 3
lines(candidate_returns_month[,2], type='l', col=3) #more aggressive 4
lines(candidate_returns_month[,3], type='l', col=4) #conservative 1
lines(candidate_returns_month[,4], type='l', col=5) #aggressive 5
lines(candidate_returns_month[,5], type='l', col=6) #medium 2
legend(8,-0.007, c("SPY","TLT","LQD","EEM","VNQ"), col = c(2,3,4,5,6), lty=1)
```



While more informative on recent performance, we need a metric to measure risk for each of these stocks to allow us to craft a suite of portfolios matching our client's risk tolerance. Let's map each stock onto the performance of the S&P 500 (SPY), and assess the beta coeffecient as a marker.

```
lm_TLT = lm(candidate_returns_year[,2] ~ candidate_returns_year[,1])
lm_LQD = lm(candidate_returns_year[,3]~ candidate_returns_year[,1])
lm_EEM = lm(candidate_returns_year[,4] ~ candidate_returns_year[,1])
lm_VNQ = lm(candidate_returns_year[,5] ~ candidate_returns_year[,1])
summary(lm_TLT)
```

```
##
## Call:
## lm(formula = candidate_returns_year[, 2] ~ candidate_returns_year[,
##     1])
##
## Residuals:
##        Min         1Q     Median         3Q        Max
## -0.0291664 -0.0053418  0.0005345  0.0059994  0.0242008
##
## Coefficients:
##                                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)                    0.0006192  0.0005479   1.130     0.26
## candidate_returns_year[, 1] -0.4589758  0.0718563  -6.387 8.22e-10 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
```

```
## Residual standard error: 0.008665 on 249 degrees of freedom
## Multiple R-squared:  0.1408, Adjusted R-squared:  0.1373
## F-statistic:  40.8 on 1 and 249 DF,  p-value: 8.22e-10
```

```r
summary(lm_LQD)
```

```
##
## Call:
## lm(formula = candidate_returns_year[, 3] ~ candidate_returns_year[,
##     1])
##
## Residuals:
##        Min         1Q     Median         3Q        Max
## -0.0116595 -0.0018810  0.0004108  0.0020876  0.0130447
##
## Coefficients:
##                              Estimate Std. Error t value Pr(>|t|)
## (Intercept)                 7.548e-05  2.253e-04   0.335   0.7379
## candidate_returns_year[, 1] -6.793e-02  2.954e-02  -2.300   0.0223 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.003562 on 249 degrees of freedom
## Multiple R-squared:  0.02079,    Adjusted R-squared:  0.01686
## F-statistic: 5.288 on 1 and 249 DF,  p-value: 0.0223
```

```r
summary(lm_EEM)
```

```
##
## Call:
## lm(formula = candidate_returns_year[, 4] ~ candidate_returns_year[,
##     1])
##
## Residuals:
##        Min         1Q     Median         3Q        Max
## -0.0237700 -0.0056612 -0.0001886  0.0054132  0.0275485
##
## Coefficients:
##                              Estimate Std. Error t value Pr(>|t|)
## (Intercept)                -0.0009976  0.0005043  -1.978    0.049 *
## candidate_returns_year[, 1]  1.0001836  0.0661293  15.125   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.007974 on 249 degrees of freedom
## Multiple R-squared:  0.4788, Adjusted R-squared:  0.4767
## F-statistic: 228.8 on 1 and 249 DF,  p-value: < 2.2e-16
```

```r
summary(lm_VNQ)
```

```
##
## Call:
```

```
## lm(formula = candidate_returns_year[, 5] ~ candidate_returns_year[,
##     1])
##
## Residuals:
##        Min         1Q     Median         3Q        Max
## -0.0274163 -0.0040670  0.0005027  0.0048362  0.0193259
##
## Coefficients:
##                              Estimate Std. Error t value Pr(>|t|)
## (Intercept)                 0.0001231  0.0004890   0.252    0.801
## candidate_returns_year[, 1] 0.6388806  0.0641304   9.962   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.007733 on 249 degrees of freedom
## Multiple R-squared:  0.285,  Adjusted R-squared:  0.2821
## F-statistic: 99.25 on 1 and 249 DF,  p-value: < 2.2e-16
```

- SPY: baseline market performance
- TLT: -0.45
- LQD: -.068
- EEM: 1.00
- VNQ: 0.63

As we are examining the absolute value of the coefficient to determine volatility (given the p-value and standard error in the above linear regressions, it can be concluded there exists somewhat of a linear relationship), we would gauge EEM as the most volatile, and then VNQ, TLT, and LQD in descending order.

We will also look at the standard deviation of each return over the last year-long period as a separate marker of volatility.

```r
sd(candidate_returns_year[,2])
```

```
## [1] 0.009329175
```

```r
sd(candidate_returns_year[,3])
```

```
## [1] 0.003592686
```

```r
sd(candidate_returns_year[,4])
```

```
## [1] 0.01102368
```

```r
sd(candidate_returns_year[,5])
```

```
## [1] 0.009127185
```

This butresses our conclusion on ETF volatility. Standard deviations over the last year indicate rank order of risk in descending order: EEM, VNQ, TLT, and LQD. Combining these factors, it is the conclusion of the author that the stocks can be ranked in the following volatility order:

1. EEM (most volatile)
2. VNQ
3. TLT
4. LQD
5. SPY (simply tracking the market)

Any good "portfolio" combines a mix of different ETF options. For my offerings, I would like to compute three packages: one that is an even split of the five ETF options, one that represents a "safe", or conservative approach to risk management, and one that represents an "aggressive" approach. I will define the packges in the following proportions:

Even: 20% across the board Safe: 40% in LQD,40% in TLT, 10% in VNQ, and 10% in SPY Aggressive: 50% in EEM,30% in VNQ, 10% in TLT, and 10% in LQD

In order to simulate investment return in each package over a month-long period, I will use a bootstrapping method, resampling from the performance of each stock over the prior month as the sample. The initial investment is $100,000, and daily balance will be redistributed according to proportions.

```
totalwealth = 100000
even_weights = c(0.2, 0.2, 0.2, 0.2, 0.2)
safe_weights = c(0.1, 0.4, 0.4, 0, 0.1)
agg_weights = c(0, 0.1, 0.1, 0.5, 0.3)
even_holdings = even_weights * totalwealth
safe_holdings = safe_weights * totalwealth
agg_holdings = agg_weights * totalwealth
n_days = 20
even_wealthtracker = rep(0, n_days)
safe_wealthtracker = rep(0, n_days)
agg_wealthtracker = rep(0, n_days)
set.seed(27)
for(today in 1:n_days) {
  return.today = resample(candidate_returns, 1, orig.ids=FALSE)

  even_holdings = even_holdings + even_holdings*return.today
  safe_holdings = safe_holdings + safe_holdings*return.today
  agg_holdings = agg_holdings + agg_holdings*return.today

  even_totalwealth = sum(even_holdings)
  safe_totalwealth = sum(safe_holdings)
  agg_totalwealth = sum(agg_holdings)

  even_wealthtracker[today] = even_totalwealth
  safe_wealthtracker[today] = safe_totalwealth
  agg_wealthtracker[today] = agg_totalwealth

  even_holdings = even_weights * even_totalwealth
  safe_holdings = safe_weights * safe_totalwealth
  agg_holdings = agg_weights * agg_totalwealth
}
even_totalwealth
```

```
## [1] 103006.3
```

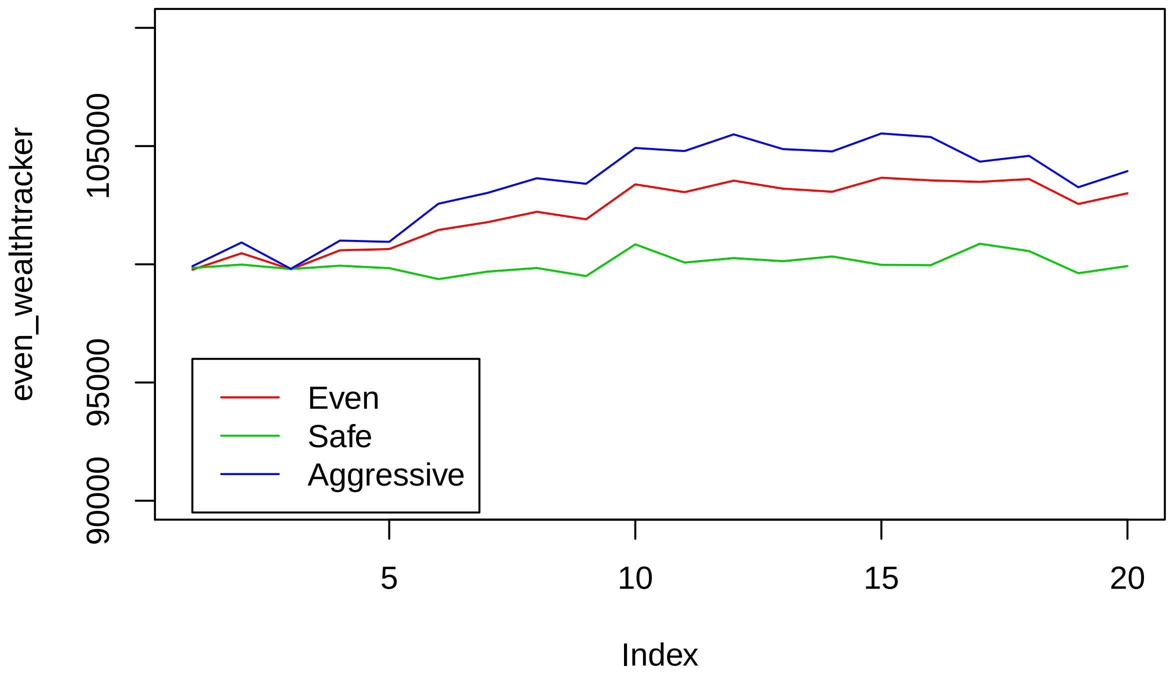
```
safe_totalwealth
```

```
## [1] 99927.49
```

```
agg_totalwealth
```

```
## [1] 103942.2
```

```
par(mfrow=c(1,1))
plot(even_wealthtracker, ylim=c(90000,110000), type='l', col=2)
lines(safe_wealthtracker, type='l', col=3)
lines(agg_wealthtracker, type='l', col=4)
legend(1,96000, c("Even","Safe","Aggressive"), col = c(2,3,4,5,6), lty=1)
```



In the above bootsampling approach, the distribution of portfolios is supported: the safe approach netted the least fluctuations in value, followed next by the even approach, with the aggressive porftolio seeing the greatest movement (in this particular case, that translated to profit.)

QUESTION 3

```r
library(mosaic)
library(ggplot2)
library(scatterplot3d)
wine = read.csv('C:/Users/Brian/Desktop/MS-BA/Summer/Intro to Predictive Modeling/Section 2/STA380/data
```

The first approach to clustering the wine types is through principal component analysis. In order to do this unsupervised technique, it is important to first removing the dependent variable- in this case, wine color.

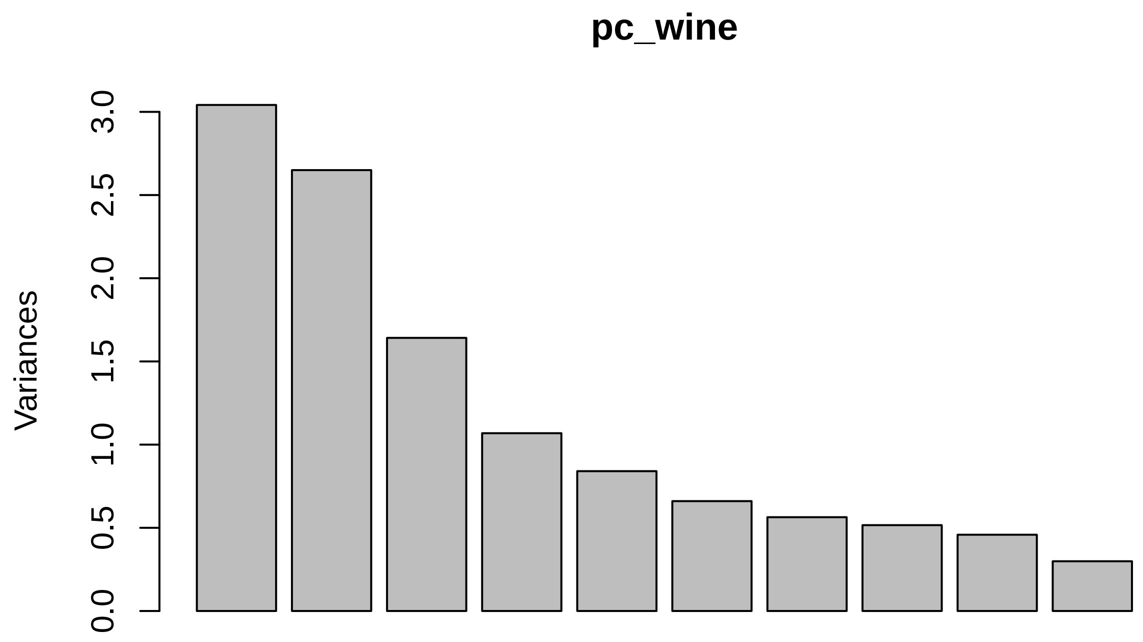
```r
wine_props = wine[,1:12]
pc_wine = prcomp(wine_props, scale.=TRUE)

pc_wine
```

```
## Standard deviations:
##  [1] 1.7440032 1.6278372 1.2812130 1.0337433 0.9167881 0.8126495 0.7508838
##  [8] 0.7183195 0.6770320 0.5468207 0.4770613 0.1810667
##
## Rotation:
##                              PC1        PC2         PC3         PC4
## fixed.acidity        -0.25692873  0.2618431 -0.46748619  0.14396377
## volatile.acidity     -0.39493118  0.1051983  0.27968932  0.08005785
## citric.acid           0.14646061  0.1440935 -0.58807557 -0.05551036
## residual.sugar        0.31890519  0.3425850  0.07550170 -0.11245623
## chlorides            -0.31344994  0.2697701 -0.04676921 -0.16529004
## free.sulfur.dioxide   0.42269137  0.1111788  0.09899801 -0.30330631
## total.sulfur.dioxide  0.47441968  0.1439475  0.10128143 -0.13223199
## density              -0.09243753  0.5549205  0.05156338 -0.15057853
## pH                   -0.20806957 -0.1529219  0.40678741 -0.47147768
## sulphates            -0.29985192  0.1196342 -0.16869128 -0.58801992
## alcohol              -0.05892408 -0.4927275 -0.21293142 -0.08003179
## quality               0.08747571 -0.2966009 -0.29583773 -0.47243936
##                               PC5         PC6         PC7          PC8
## fixed.acidity        -0.165362611  0.03003708 -0.39343530  0.001155415
## volatile.acidity     -0.147774077 -0.38266373 -0.44511080  0.310077574
## citric.acid           0.234621394  0.36224839 -0.04769762  0.444962196
## residual.sugar       -0.507921181 -0.06331719  0.09576310  0.081944829
## chlorides             0.393896604 -0.42544212  0.47329609  0.375531558
## free.sulfur.dioxide   0.248451958 -0.28318017 -0.36271398  0.120097626
## total.sulfur.dioxide  0.223966811 -0.10676882 -0.23481304  0.011279269
## density              -0.330357303  0.15455292 -0.01328572  0.042943625
## pH                    0.001457502  0.56089714 -0.07932113  0.362281828
## sulphates             0.193245549 -0.02014082 -0.17023612 -0.592220645
## alcohol              -0.116023190 -0.16947538 -0.33890566  0.226040866
## quality              -0.459129140 -0.27788835  0.27317740  0.093046206
##                              PC9        PC10         PC11          PC12
## fixed.acidity         0.42416913 -0.27243245 -0.276932032 -0.3350925313
## volatile.acidity     -0.12323319  0.49394624  0.140799120 -0.0824207438
## citric.acid          -0.24623265  0.33035538  0.229276179  0.0013466535
## residual.sugar       -0.48802361 -0.20717448  0.005144195 -0.4512148285
## chlorides            -0.04404975 -0.23887258 -0.193398397 -0.0432778637
## free.sulfur.dioxide   0.30139683 -0.30344826  0.486158400 -0.0009050256
## total.sulfur.dioxide  0.00181386  0.29478016 -0.720162498  0.0640632122
## density               0.07108094 -0.07681483 -0.003324491  0.7156665472
## pH                    0.13666158 -0.11240888 -0.139083153 -0.2067626762
## sulphates            -0.29740108  0.08546912  0.047219109 -0.0781995574
## alcohol              -0.41706026 -0.41605905 -0.191289020  0.3320120790
## quality               0.35664731  0.30782956 -0.018082206  0.0082880344
```

```r
summary(pc_wine) #first rotation accounts for ~25% of variance
```

```
## Importance of components:
##                           PC1    PC2    PC3     PC4     PC5     PC6
## Standard deviation     1.7440 1.6278 1.2812 1.03374 0.91679 0.81265
## Proportion of Variance 0.2535 0.2208 0.1368 0.08905 0.07004 0.05503
## Cumulative Proportion  0.2535 0.4743 0.6111 0.70013 0.77017 0.82520
##                            PC7    PC8    PC9    PC10    PC11    PC12
## Standard deviation     0.75088 0.7183 0.6770 0.54682 0.47706 0.18107
## Proportion of Variance 0.04699 0.0430 0.0382 0.02492 0.01897 0.00273
## Cumulative Proportion  0.87219 0.9152 0.9534 0.97830 0.99727 1.00000
```

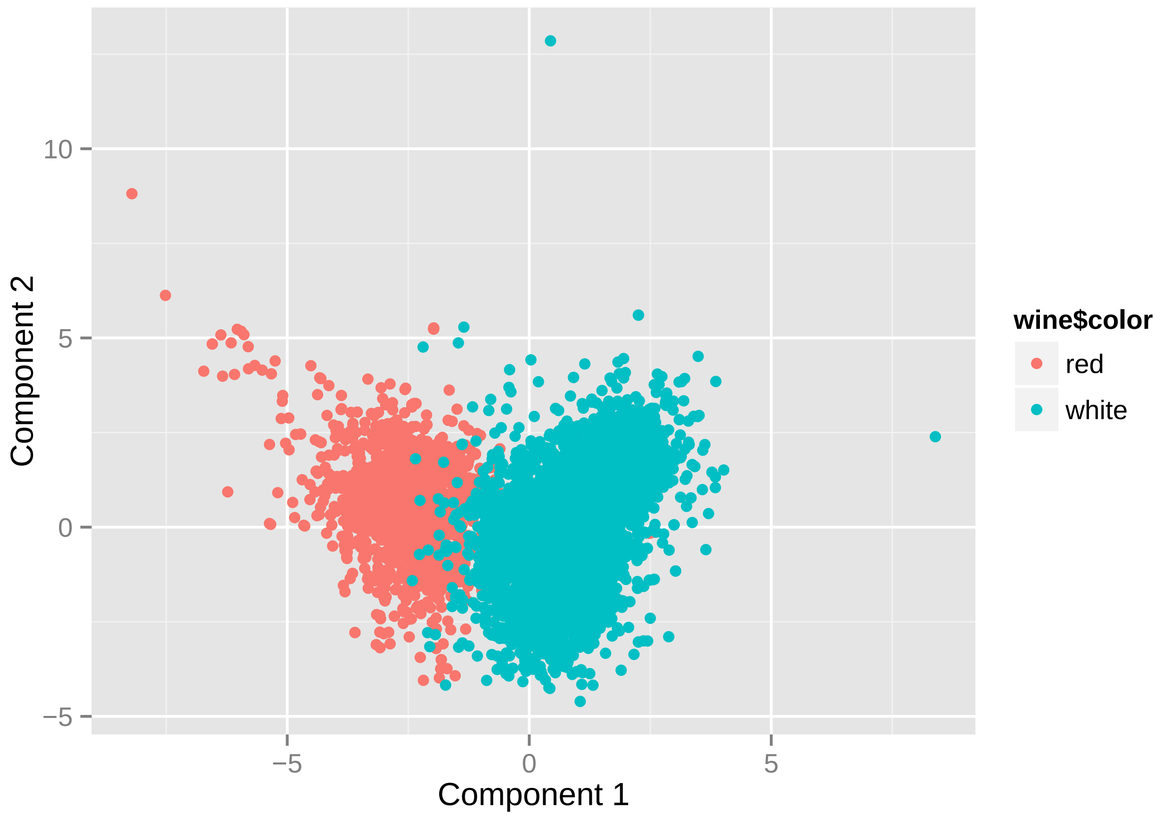
```r
plot(pc_wine)
```



You can see from the output that the first principal component, made up from a selected combination of input variables, accounts for ~25% of the variance in the data. By the time we have layered on the sixth component, we have accounted for 85.2% of the variance.

In the following chart, we see that the grouping accuracy of the first two PCA components aligns quite closely with the true color of the wine. The groups in the following chart are identified by their non-color feature, and then colored with their true wine type post hoc.

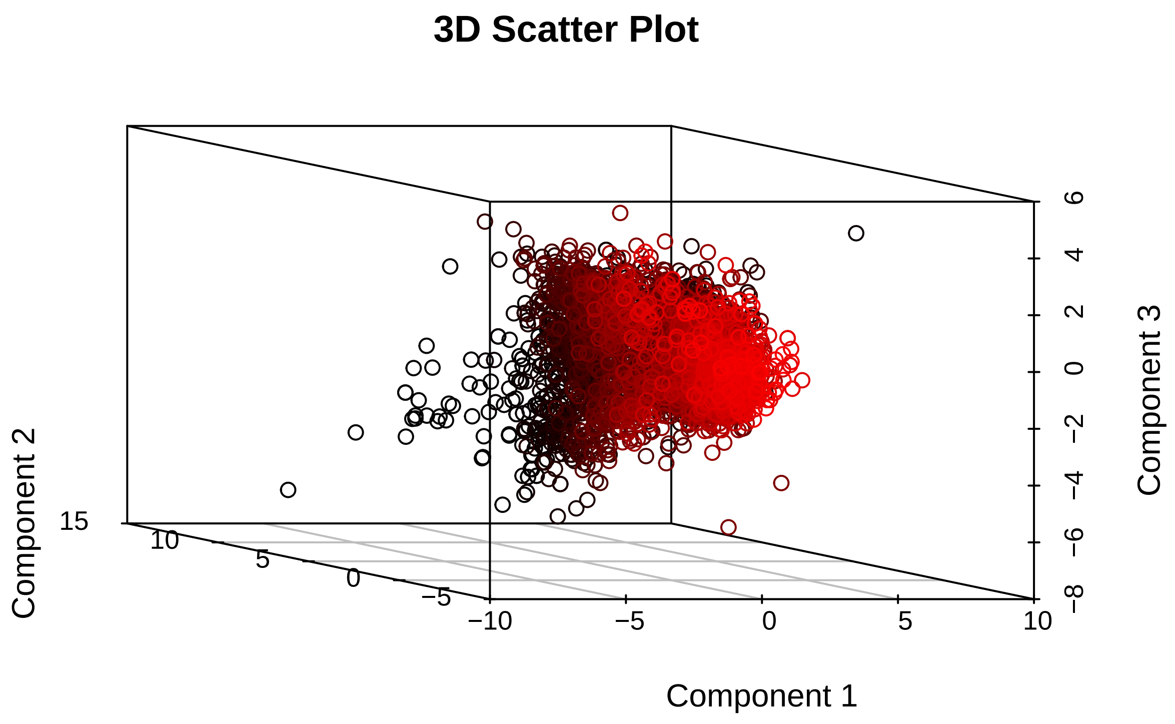
```
loadings = pc_wine$rotation
scores = pc_wine$x
qplot(scores[,1], scores[,2], color=wine$color, xlab='Component 1', ylab='Component 2')
```



Very little overlap.

Additionally, see below a three-dimensional plot of the first three principal components (although slightly harder to interpret visually):

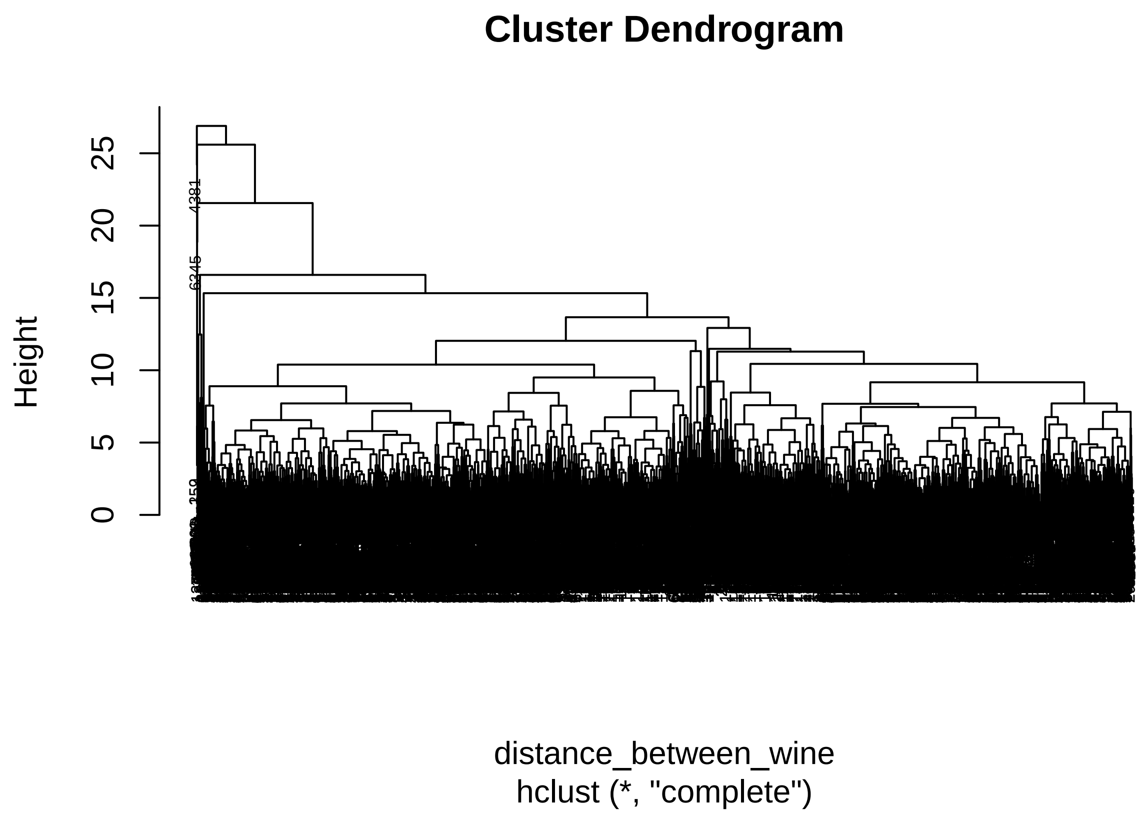
```
scatterplot3d(scores[,1:3], #making a 3d plot with the first three rotations (had to rename to white wi
              #to blue, because the points were being plotted as white - impossible to see)
              main="3D Scatter Plot",
              highlight.3d = TRUE,
              xlab = 'Component 1',
              ylab = 'Component 2',
              zlab = 'Component 3',
              angle=150)
```



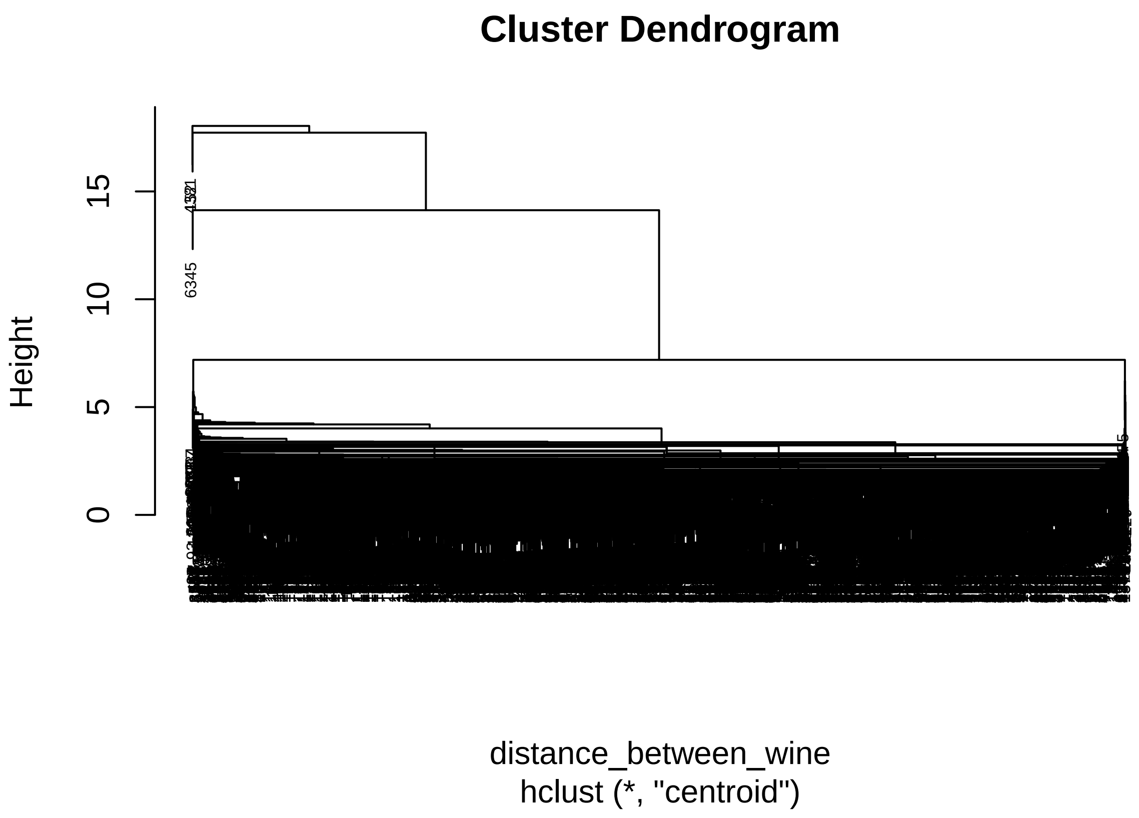
Another approache to grouping wine types is through hierarchical clusting by distance. When using any distance clustering method, in addition to removing the dependent variable, it is also important to scale your data. The scaling nullifies the magnitude or amplitude of each variable's units, so the distance between each point in each variable can be made using a consistent rubric.

The following hierarchical clusters and charts represent several different approaches to clustering. Initially, cluster trees are made using several different approaches (complete, centroid, and Ward's minumum variance method). Then these hierarchy trees are plotted.

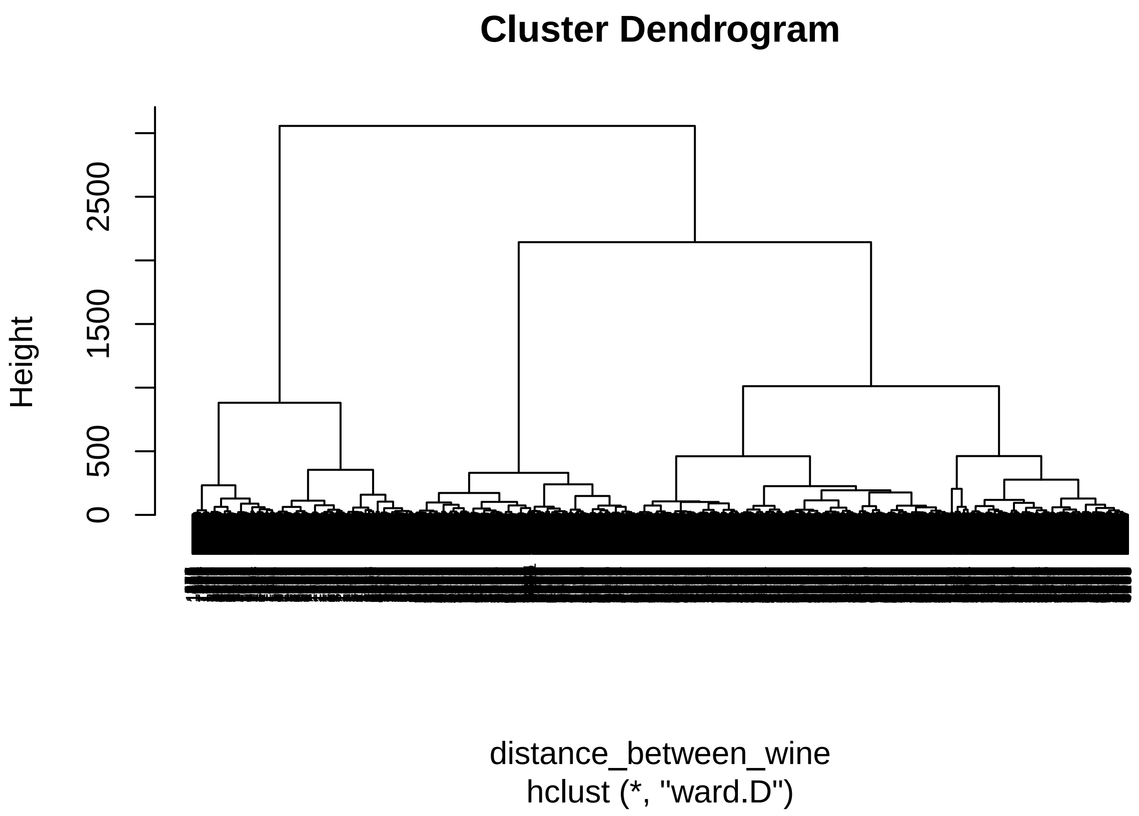
```
wine_scaled <- scale(wine_props, center=TRUE, scale=TRUE)
distance_between_wine = dist(wine_scaled)

hcomplete = hclust(distance_between_wine, method='complete')
hcentroid = hclust(distance_between_wine, method='centroid')
dward = hclust(distance_between_wine, method='ward.D')

plot(hcomplete, cex=0.5)
```



```r
plot(hcentroid, cex=0.5)
```



```
plot(dward, cex=0.5)
```

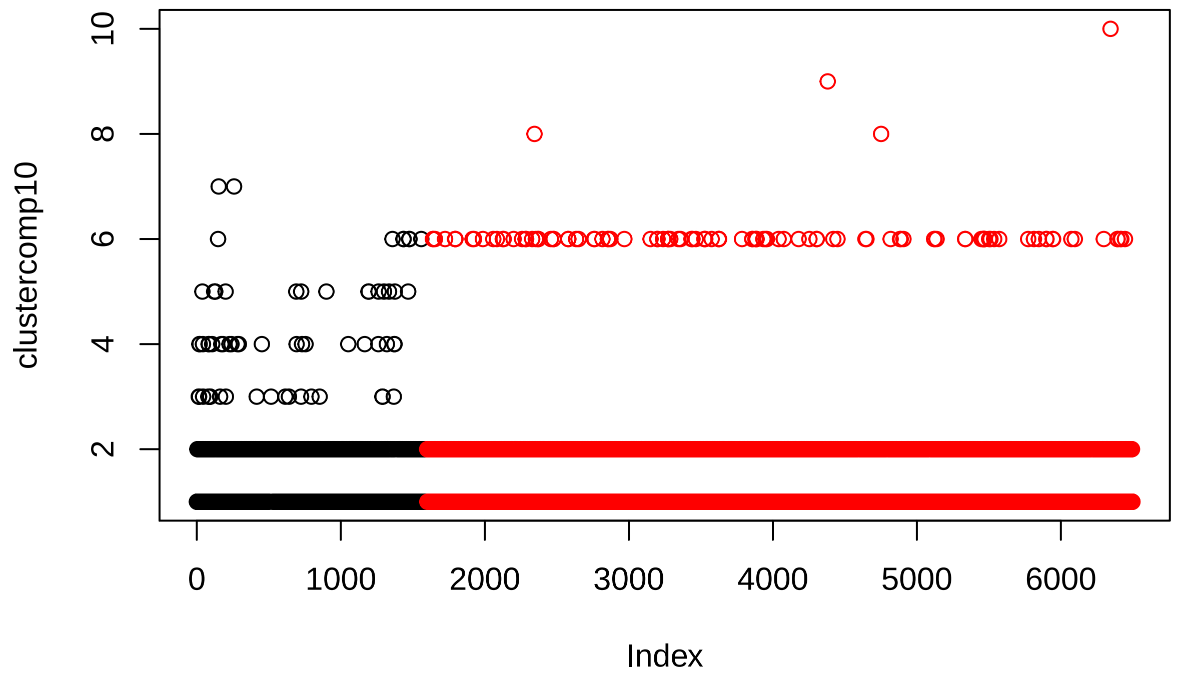


As you can see, this sort of view is not particularly useful when fully fitting all of the data points in a large data sets. Thankfully, a strength of hierarchical clustering is that a cutoff threshold can be designated, and all of the leaves can be "cut" or trimmed back. This, however, presents the challenge of choosing the right amount to cut.

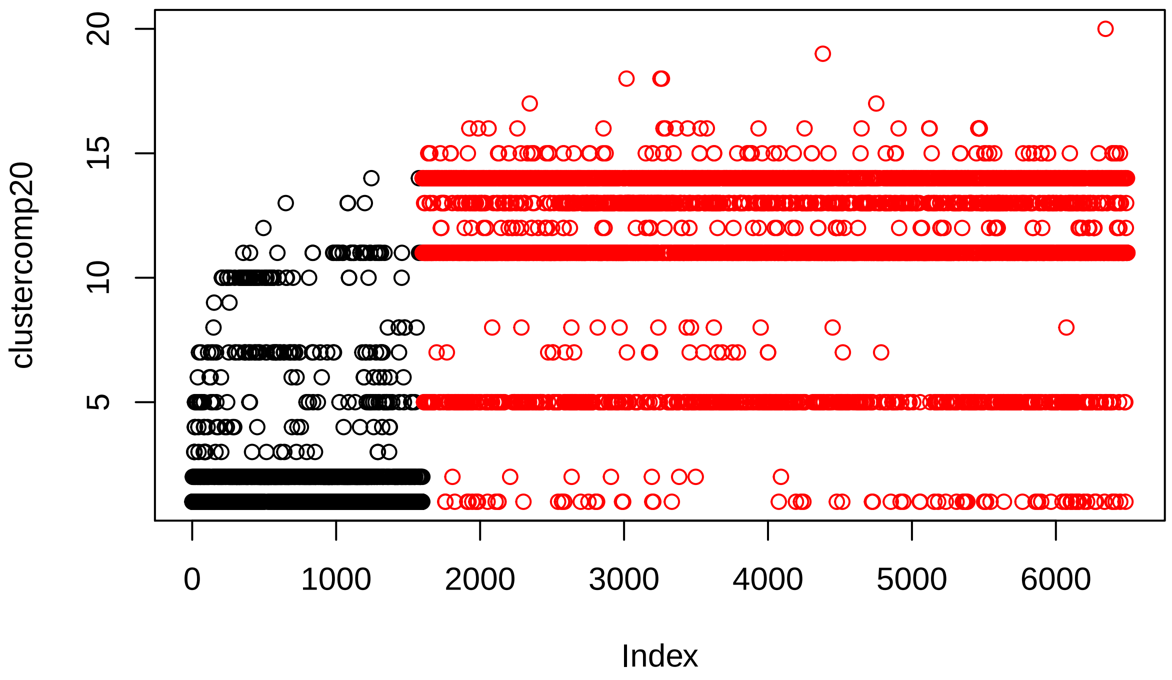
Below are several different approaches to cutting each of the different hiearchical trees into k clusters, and mapping the actual color of the wine onto the clusters.

Complete method with 10, 20, and 50 clusters:

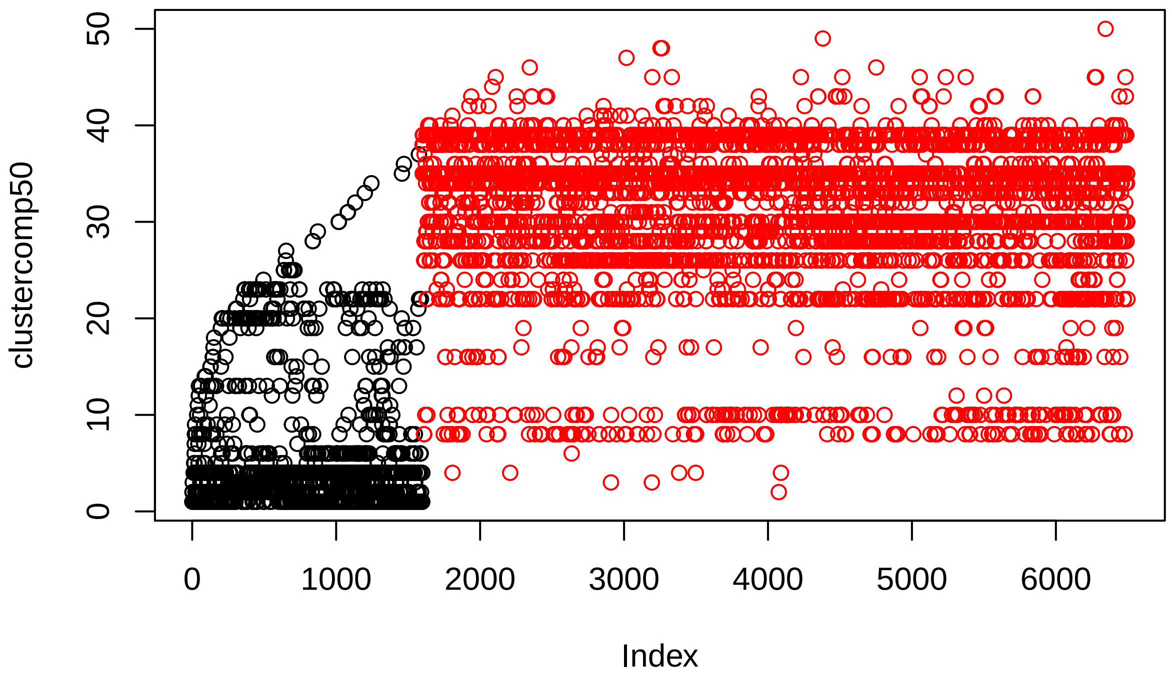
```
clustercomp10 = cutree(hcomplete, k=10)
clustercomp20 = cutree(hcomplete, k=20)
clustercomp50 = cutree(hcomplete, k=50)
plot(clustercomp10,col=(wine$color))
```



```
plot(clustercomp20,col=(wine$color))
```



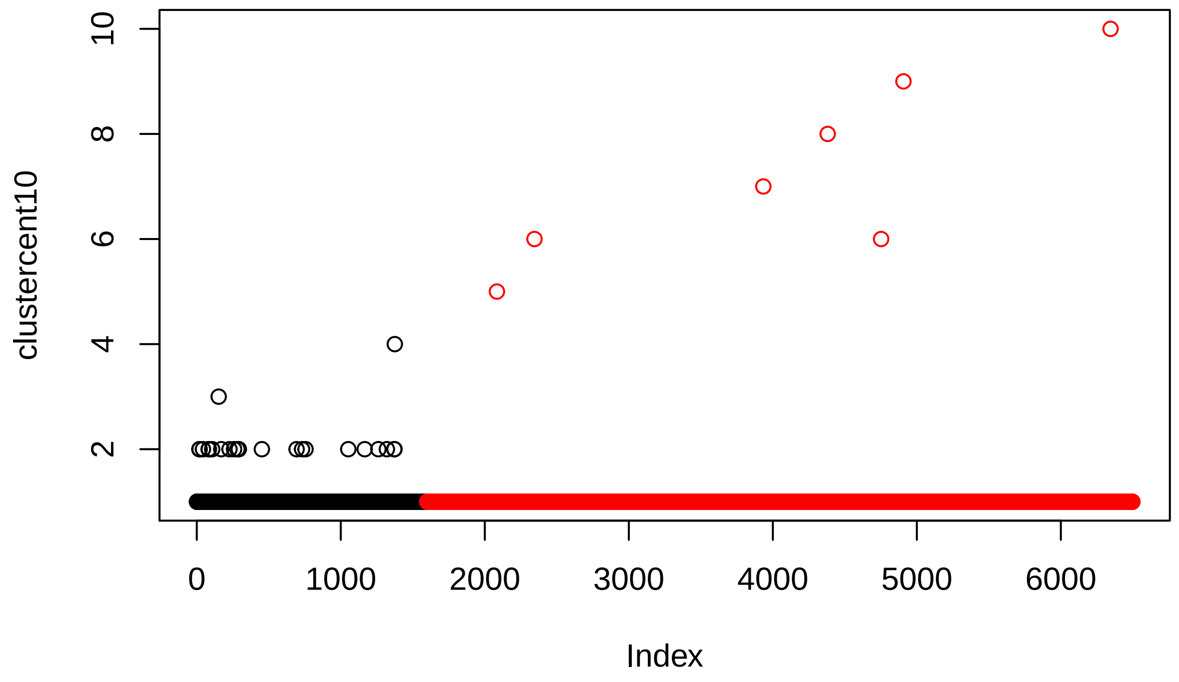
```
plot(clustercomp50,col=(wine$color))
```



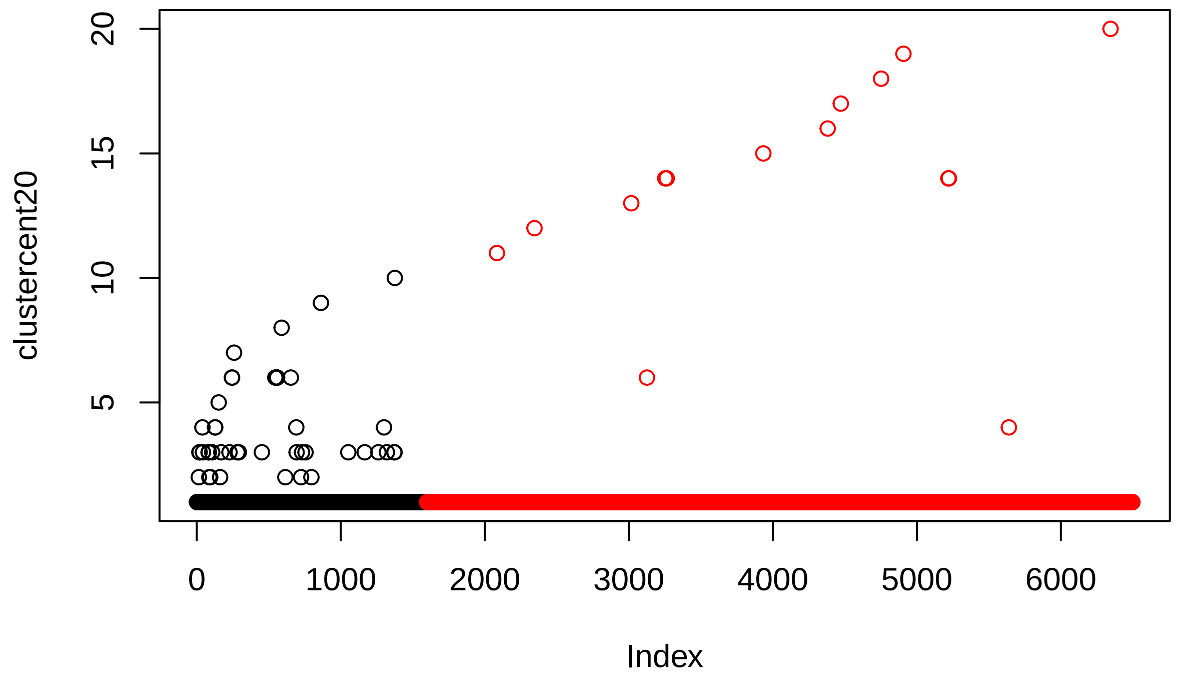
The idea here would be that every cluster (in this case horizontal line) would acurately reflect whether a given wine was red or white. That is, the entire line, or cluster, should be a single color. You can see that clusters 1 through 7 do a very good job at identifying whites, and clusters ~25 and up do a consistent job in identifying reds. However, there are some clusters that overlap, and include bother reds and whites (clusters 8, 10, and 22, for example).

Centroid method with 10, 20, and 50 clusters:

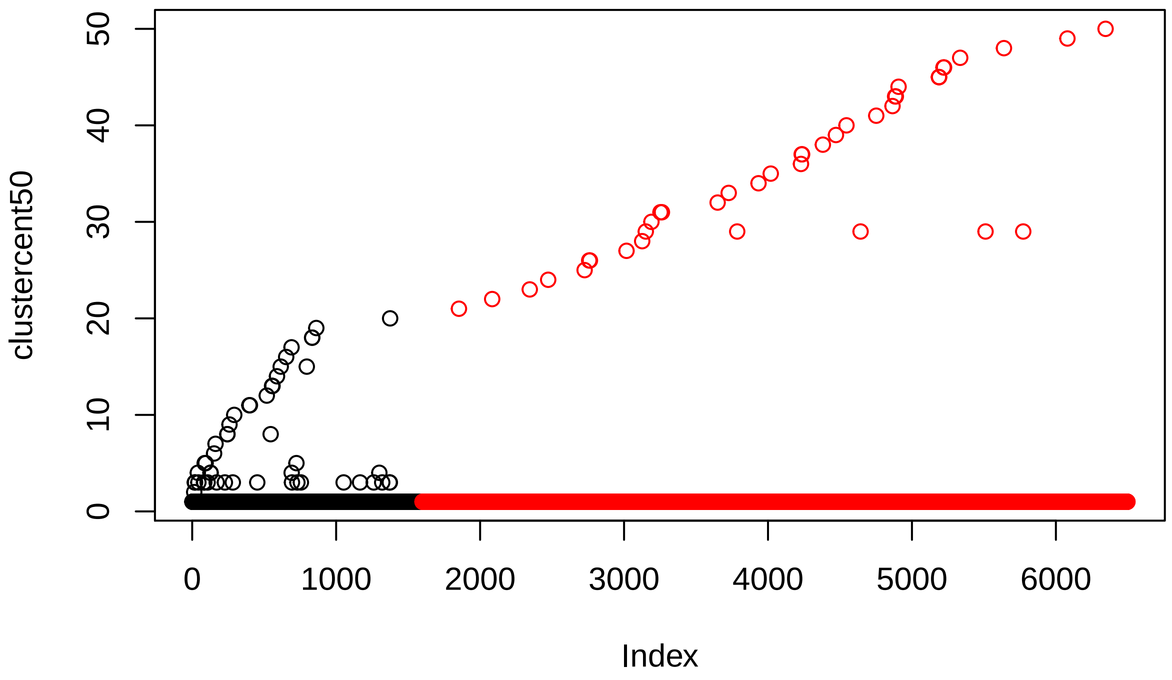
```
clustercent10 = cutree(hcentroid, k=10)
clustercent20 = cutree(hcentroid, k=20)
clustercent50 = cutree(hcentroid, k=50)
plot(clustercent10,col=(wine$color))
```



```
plot(clustercent20,col=(wine$color))
```



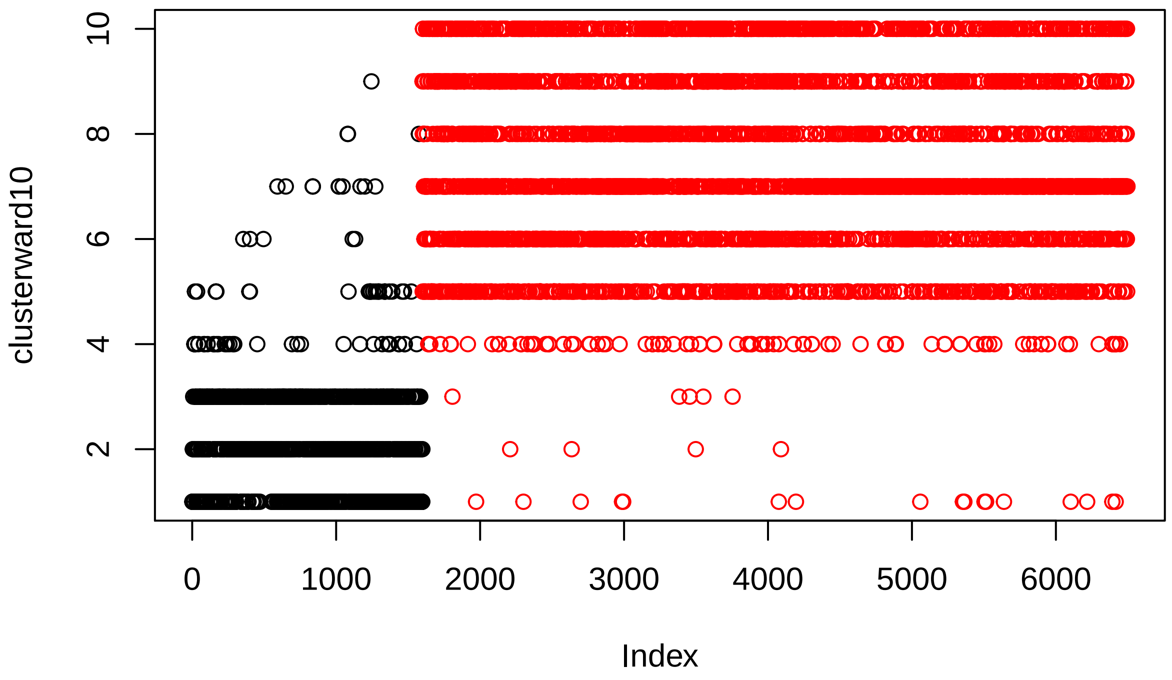
```r
plot(clustercent50,col=(wine$color))
```



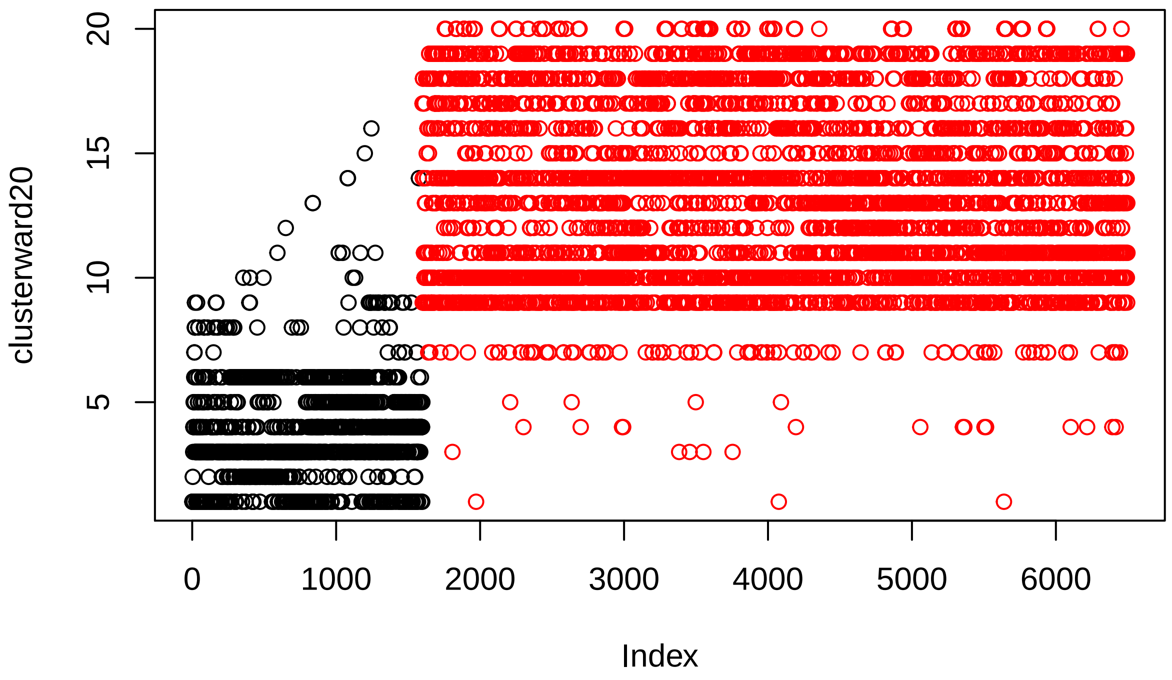
However, when using the centroid approach, clustering does a terrible job. No individual example of wine exists that defines that cluster's distance from other clusters on all dimensions.

Ward's method with 10, 20, and 50 clusters:

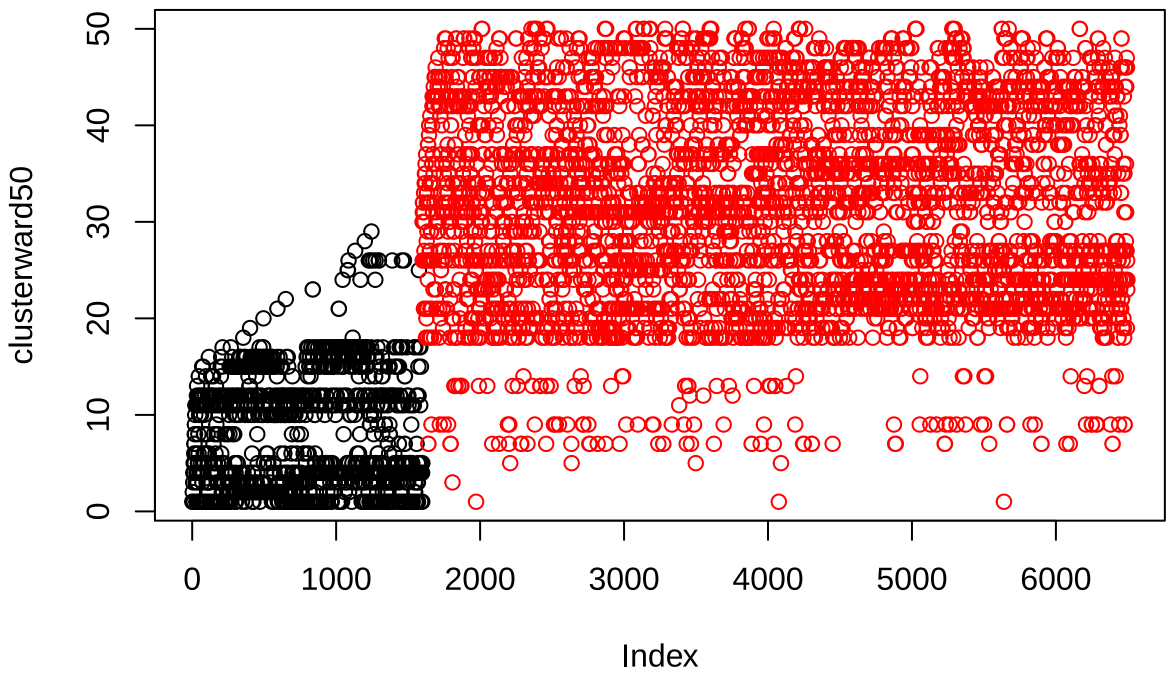
```
clusterward10 = cutree(dward, k=10)
clusterward20 = cutree(dward, k=20)
clusterward50 = cutree(dward, k=50)
plot(clusterward10,col=(wine$color))
```



```
plot(clusterward20,col=(wine$color))
```



```r
plot(clusterward50,col=(wine$color))
```



In general, the greater the number of clusters, the greater the accuracy of each cluster in predicting the wine. Ward's method tends to yield the greatest within-cluster accuracy (although I am making an admitted visual estimate) - each cluster contains a clear majority of a single color of wine. However, can it reasonably by said that there are 50 different groupings of wine in the norther Portugal region? The sommelier would likely argue this is still an underestimate.

My preference for the explanatory model here is likely the hierarchical tree method. In this case, there are clear decisions made at each branch based on features of the wine that can be explained to a lay-person, which allows for real-world buy-in from other parties. In the case of PCA, the ability to interpret a single component is limited, and any skepticism more difficult to deflect.

QUESTION 4

A good approach to giving some segmentation analysis to the NutrientH20's social media audience is to use a solid clustering method. A potential good fit for this data may be a k-means clustering analysis, where groups of social media followers are grouped based on similarity of interests. These interests are coded through a manual textual analysis of their tweets. The question is, which k-means approach will work best?

The first step in setting up k-means is to normalize the data. We will be using a "distance"-based approach to clustering, and we need to ensure that all the data points variables are on the same scale. For instance, are 15 sports mentions and 15 small business mentions equivalent? No - it is likely that small businesses are mentioned far less, so a score of 15 here could be a heavy outlier while 15 sports mentions could be typical. By normalizing the variables, we can make more ready comparisons between scales with different amplitudes.

```r
library(cluster)
library(fpc)
```

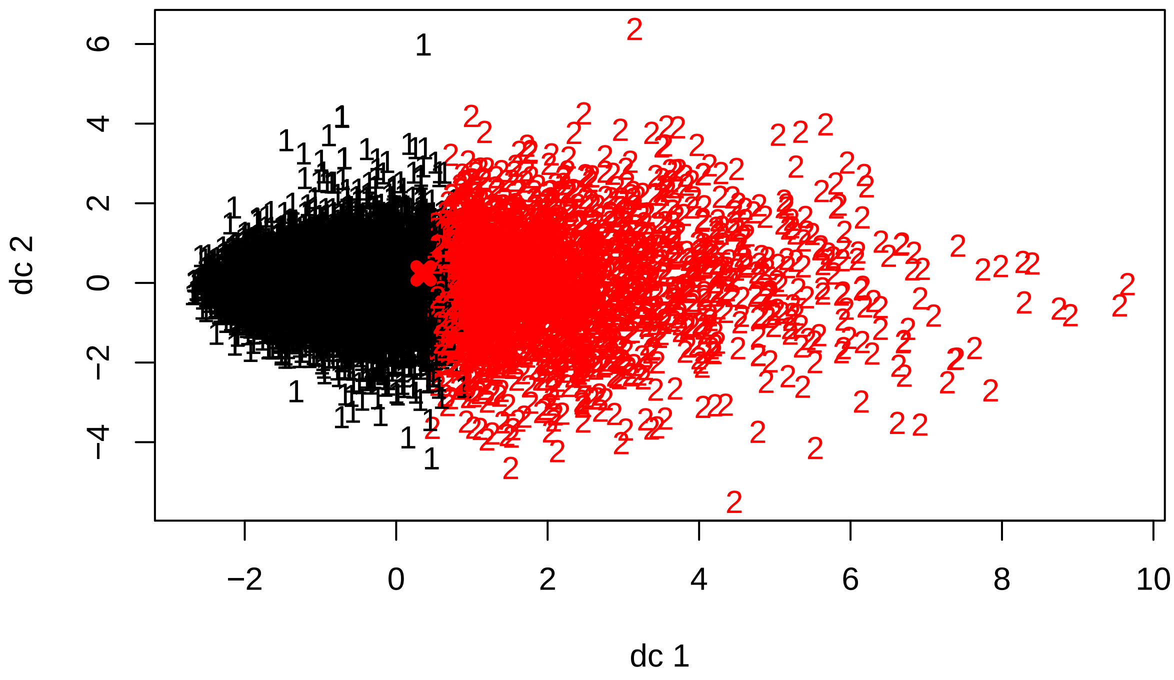
```
social = read.csv('C:/Users/Brian/Desktop/MS-BA/Summer/Intro to Predictive Modeling/Section 2/STA380/da
colnames(social)
```

```
##  [1] "X"                "chatter"          "current_events"
##  [4] "travel"           "photo_sharing"    "uncategorized"
##  [7] "tv_film"          "sports_fandom"    "politics"
## [10] "food"             "family"           "home_and_garden"
## [13] "music"            "news"             "online_gaming"
## [16] "shopping"         "health_nutrition" "college_uni"
## [19] "sports_playing"   "cooking"          "eco"
## [22] "computers"        "business"         "outdoors"
## [25] "crafts"           "automotive"       "art"
## [28] "religion"         "beauty"           "parenting"
## [31] "dating"           "school"           "personal_fitness"
## [34] "fashion"          "small_business"   "spam"
## [37] "adult"
```

```
social_strip = social[,2:length(social)]
social_scaled = scale(social_strip, center=TRUE, scale=TRUE)
```

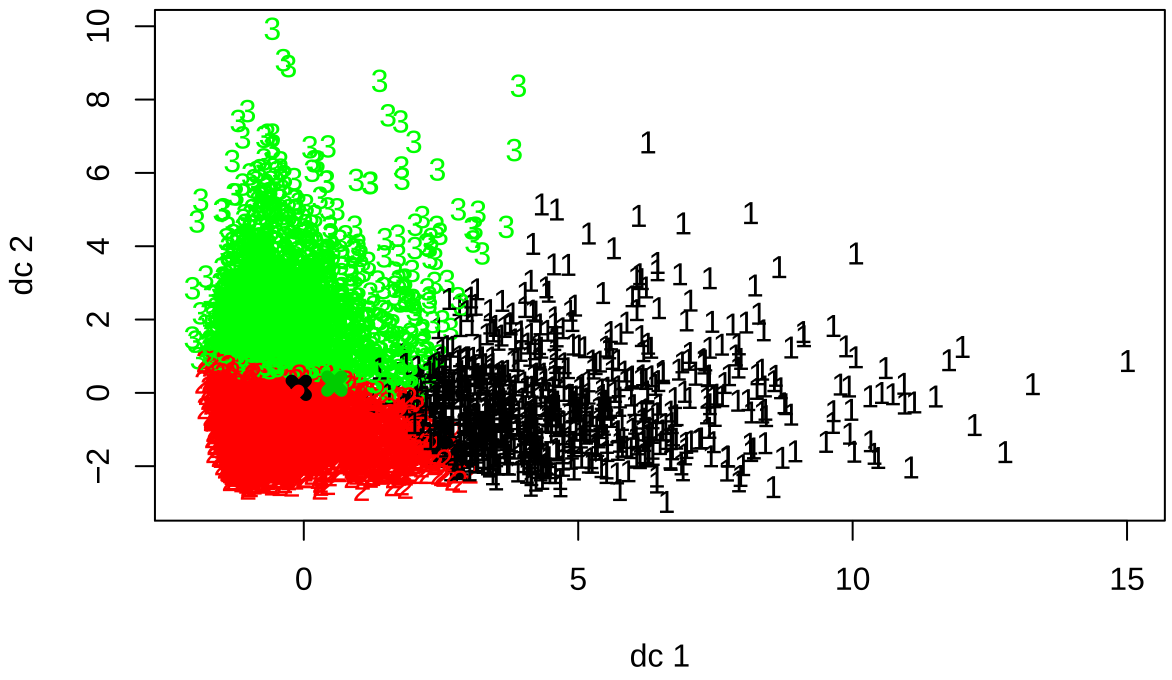
We can now run some random k-means clusters (over 50 random starts) with k groups, and see what we get. The first run will be for two segments:

```
set.seed(22)
cluster_scaled2 = kmeans(social_scaled, centers=2, nstart=50)
plotcluster(social_scaled, cluster_scaled2$cluster)
points(cluster_scaled2$centers, col = 1:2, pch = 4, lwd=6)
```



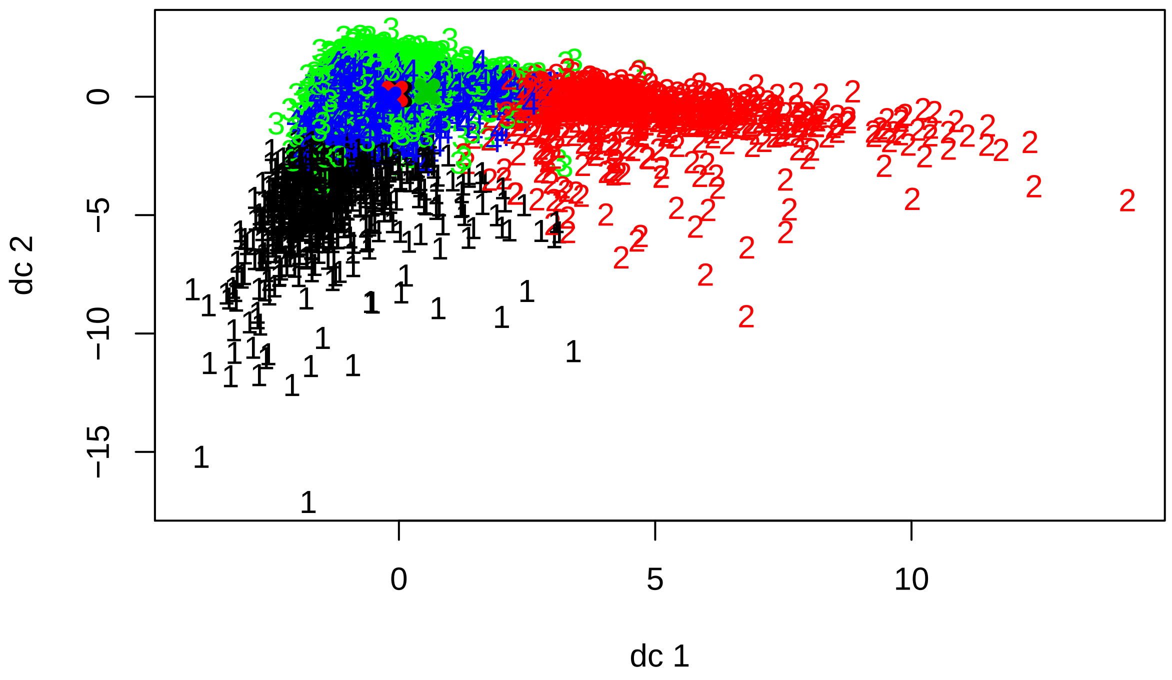
While the clusters seem to be grouped disparately (With the "black" segments to the left, and the "red" segment to the right), by overlaying the centers, you can see that the centers of these two groups appear to be rather close in distance. Although it is admittedly hard to see the black center.

```r
set.seed(23)
cluster_scaled3 = kmeans(social_scaled, centers=3, nstart=50)
plotcluster(social_scaled, cluster_scaled3$cluster)
points(cluster_scaled3$centers, col = 1:3, pch = 4, lwd=6)
```



Plotting three clusters seems to make a clear distiction between the groups. The centers are slightly less tightly grouped.

```r
set.seed(24)
cluster_scaled4 = kmeans(social_scaled, centers=4, nstart=50)
plotcluster(social_scaled, cluster_scaled4$cluster)
points(cluster_scaled4$centers, col = 1:4, pch = 4, lwd=6)
```



However, once you increase the clusters to 4, we start to see some overlap between groups (when plotting in these dimensions).

While all of these charts paint a picture of potential different customer segments, by themselves they do little to tell us about the composition of each segments. Additionally, with a large data set, we're rolling the dice a bit by relying on 50 random start points for our clusters. The next two objectives should be to fit a clustering algorithm with intelligent k-center selection, and then do define the interests of each of the k-means segments.

One strong approach is to use k-means++ to define our cluster centers. This approach selects new cluster centers "repulsively", by looking for candidate centers in proportion to their distance fromm centers already selected.

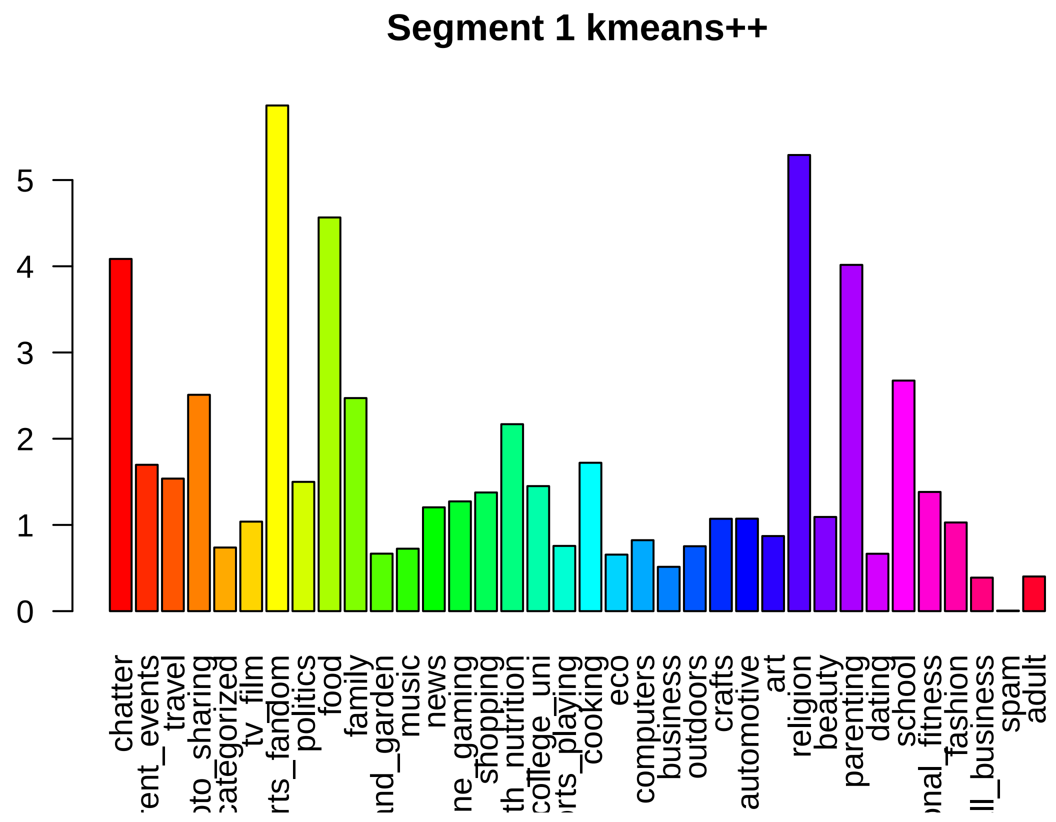
The following r script identifies three potential customer segments. Three was selected as the number of clusters for a few reasons, chiefly that a) a marketing team's preference would likely to be to have a smaller number of potential segments, rather than overfitting follower interests to an unweildy number of segments, b) from our earlier exploratory k-means clustering, 3 clusters seemed to maximize between-center distancce while minimizing segment overlap.

```r
library(flexclust)
```

```
## Loading required package: grid
## Loading required package: modeltools
## Loading required package: stats4
```

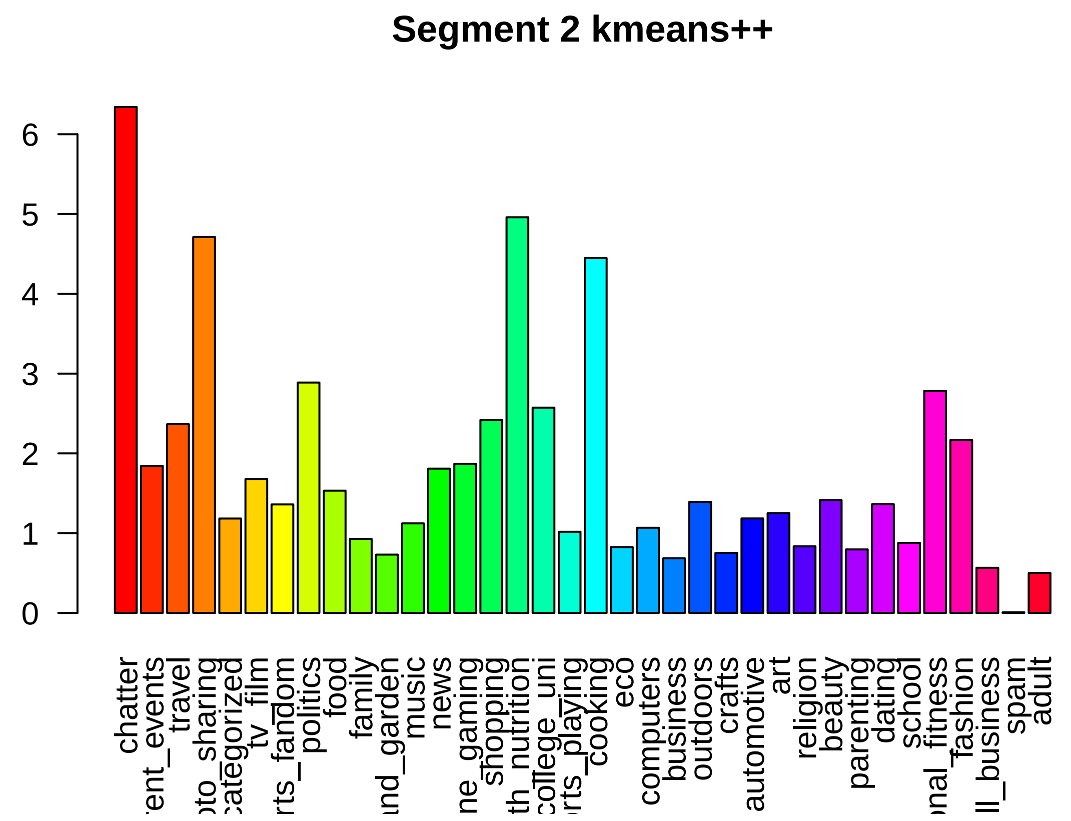
```r
set.seed(5)
social_clust = kcca(social_scaled, k=3, family=kccaFamily("kmeans"), control=list(initcent="kmeanspp"))
mu = attr(social_scaled,"scaled:center")
```

```r
sigma = attr(social_scaled,"scaled:scale")

socialclust_1seg = parameters(social_clust)[1,]*sigma + mu
socialclust_2seg = parameters(social_clust)[2,]*sigma + mu
socialclust_3seg = parameters(social_clust)[3,]*sigma + mu

barplot(socialclust_1seg, main='Segment 1 kmeans++',col=rainbow(36),las=2)
```



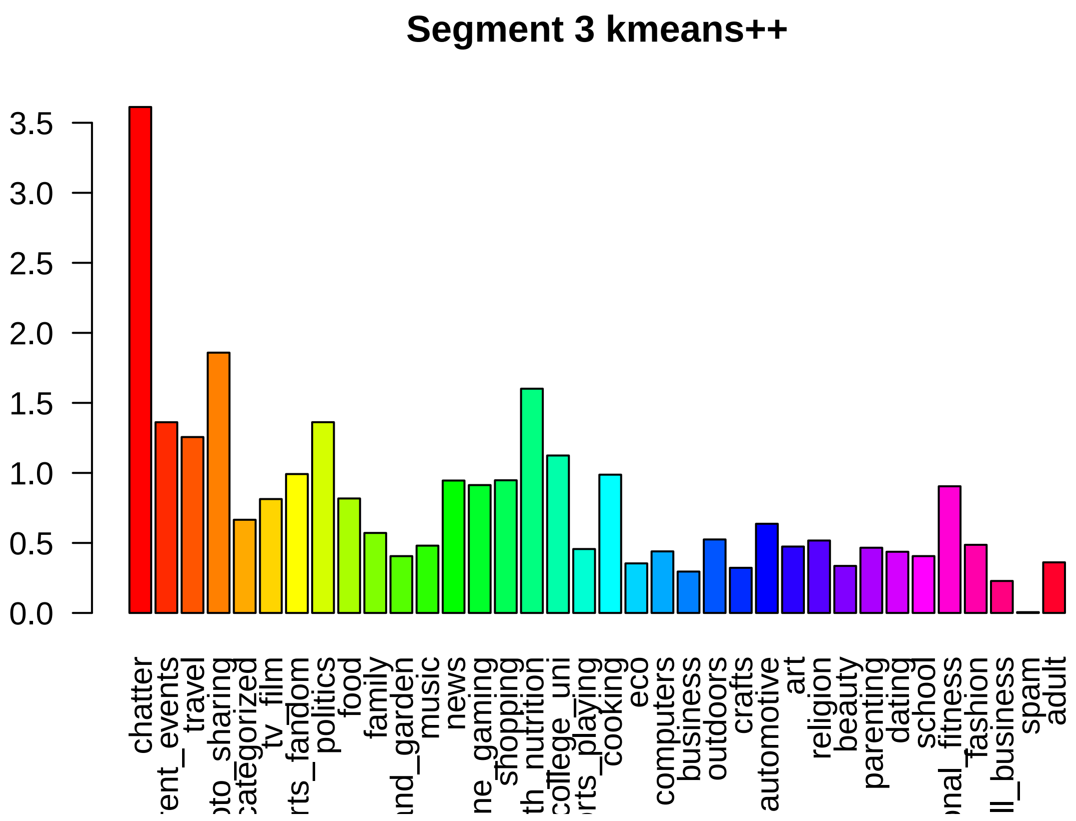
The above chart plots the mean "mentions" in customer segment 1 for each of the captured dimensions. Through this visual plot, we can identify a few main trends in defining this population. Segment One tends to have a high focus on family, religion, and parenting - this segment likely puts importance on traditional values. Additionally, Segment One expresses an interest in sports and food. Put together, I'd likely characterize this segment as a "typical American family", although that is quite a loaded term. With a focus on family, food, and sports, they are rather a traditional segment.

```r
barplot(socialclust_2seg, main='Segment 2 kmeans++',col=rainbow(36),las=2)
```



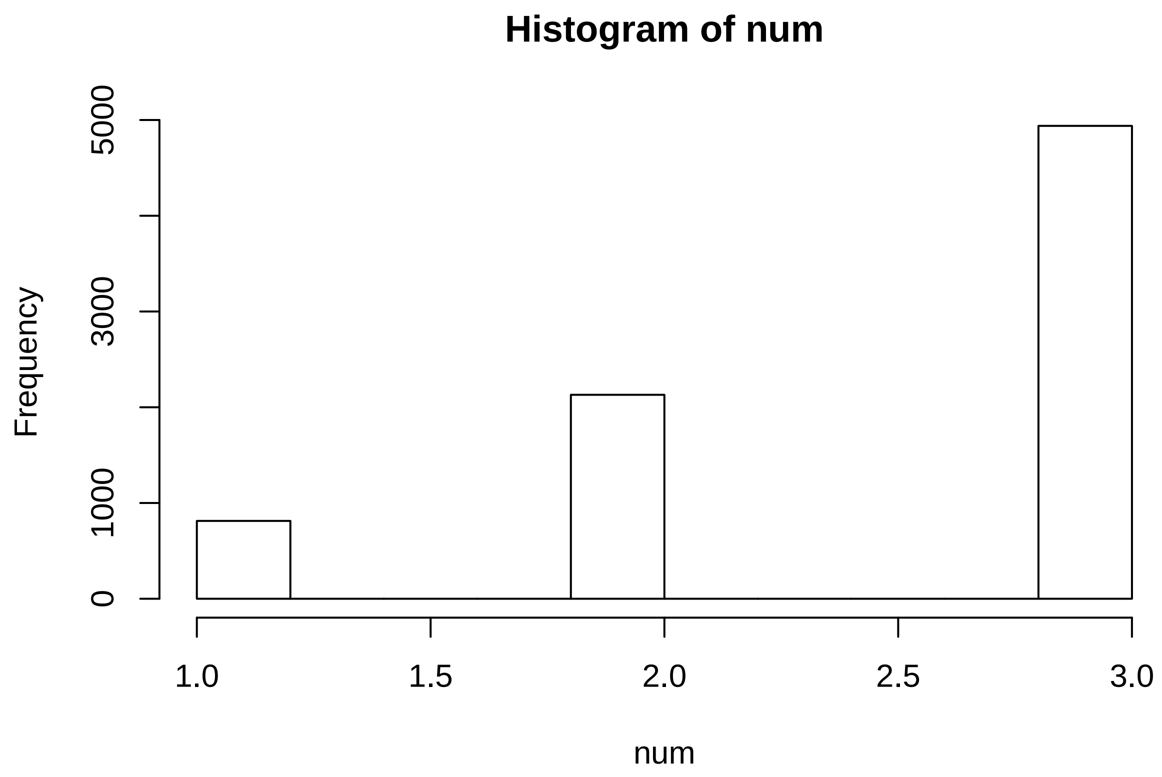
Segment Two represents a slightly different demographic. This group prioritizes (relatively) health and nutrition, cooking, personal fitness, and fashion. Additionally, Segment Two gets high marks on photo sharing and chatter (although this metric is not inherently paricularly meaningful in its definition and some extent high for every group). This segment might be characterize by a health-conscious, socially-active young person who likes to share information they like. This would probably be the primary target for a marketing effort, as this segment is likely to spread a message they identify with.

```r
barplot(socialclust_3seg, main='Segment 3 kmeans++',col=rainbow(36),las=2)
```



Segment Three does not appear to have any significant defining factors, but rather is defined by not being included in segments One and Two. From a marketing perspective, this analysis does not give any real insight into targeting this group.

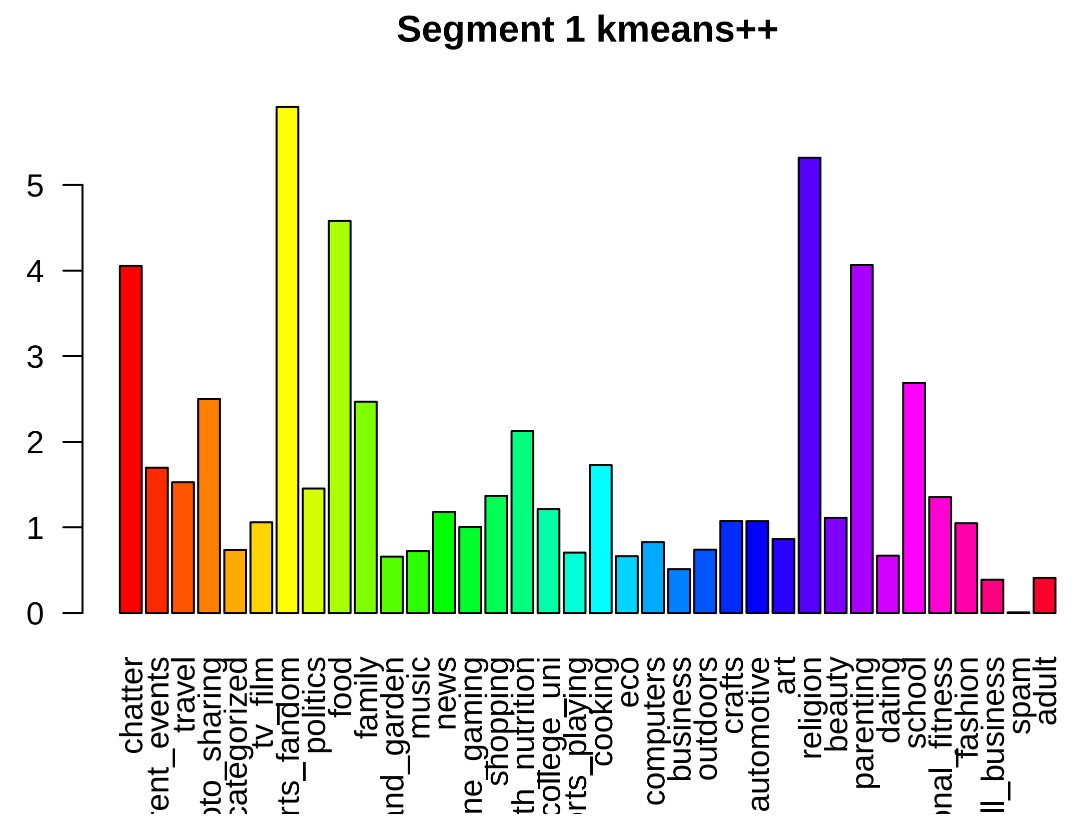
```r
num = clusters(social_clust)
hist(num)
```



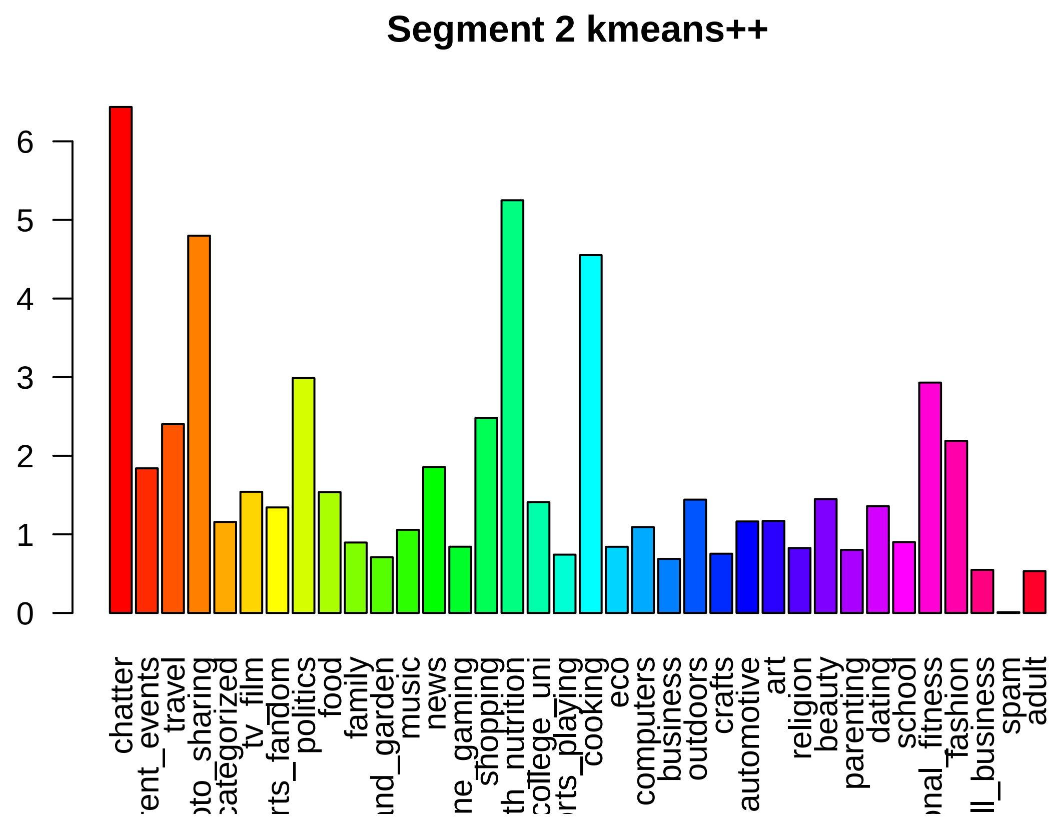
Unfortunately, from the above histogram, it appears that the majority of folllowers are bucketed into segment three. However, this does not address the issues of network influence, likelihood to share, etc. that may be important in social media.

Lastly, it would be interesting to map the data onto four clusters, and see what happens. Really, there are a functionally infinite number of clusters that could be assigned, but let's cap the exploration at four.

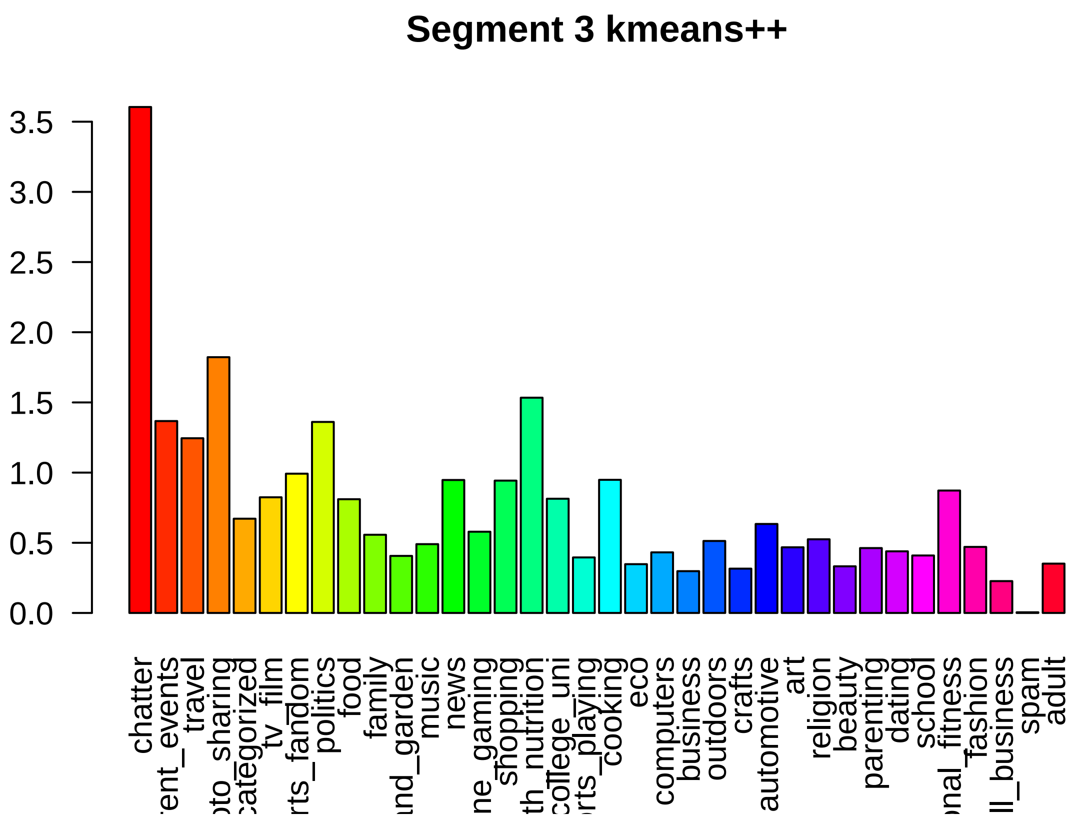
```
set.seed(5)
social_clust4 = kcca(social_scaled, k=4, family=kccaFamily("kmeans"), control=list(initcent="kmeanspp")
socialclust4_1seg = parameters(social_clust4)[1,]*sigma + mu
socialclust4_2seg = parameters(social_clust4)[2,]*sigma + mu
socialclust4_3seg = parameters(social_clust4)[3,]*sigma + mu
socialclust4_4seg = parameters(social_clust4)[4,]*sigma + mu
barplot(socialclust4_1seg, main='Segment 1 kmeans++',col=rainbow(36),las=2)
```



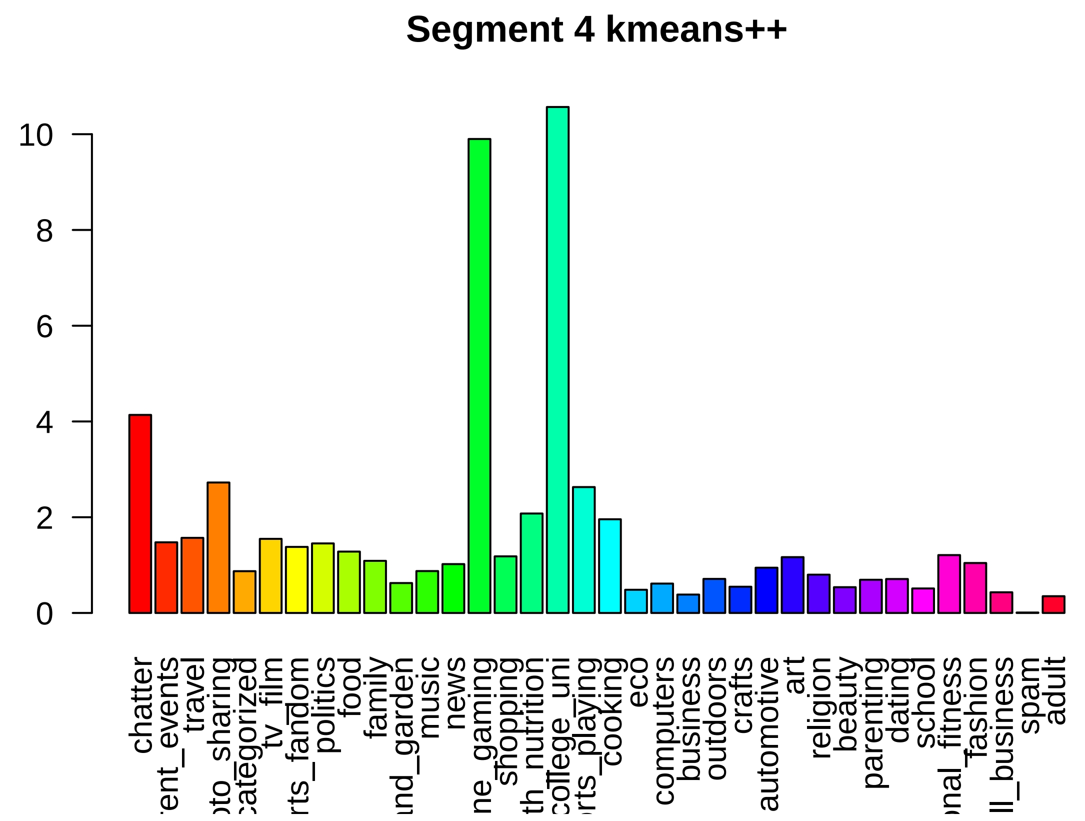
```r
barplot(socialclust4_2seg, main='Segment 2 kmeans++',col=rainbow(36),las=2)
```



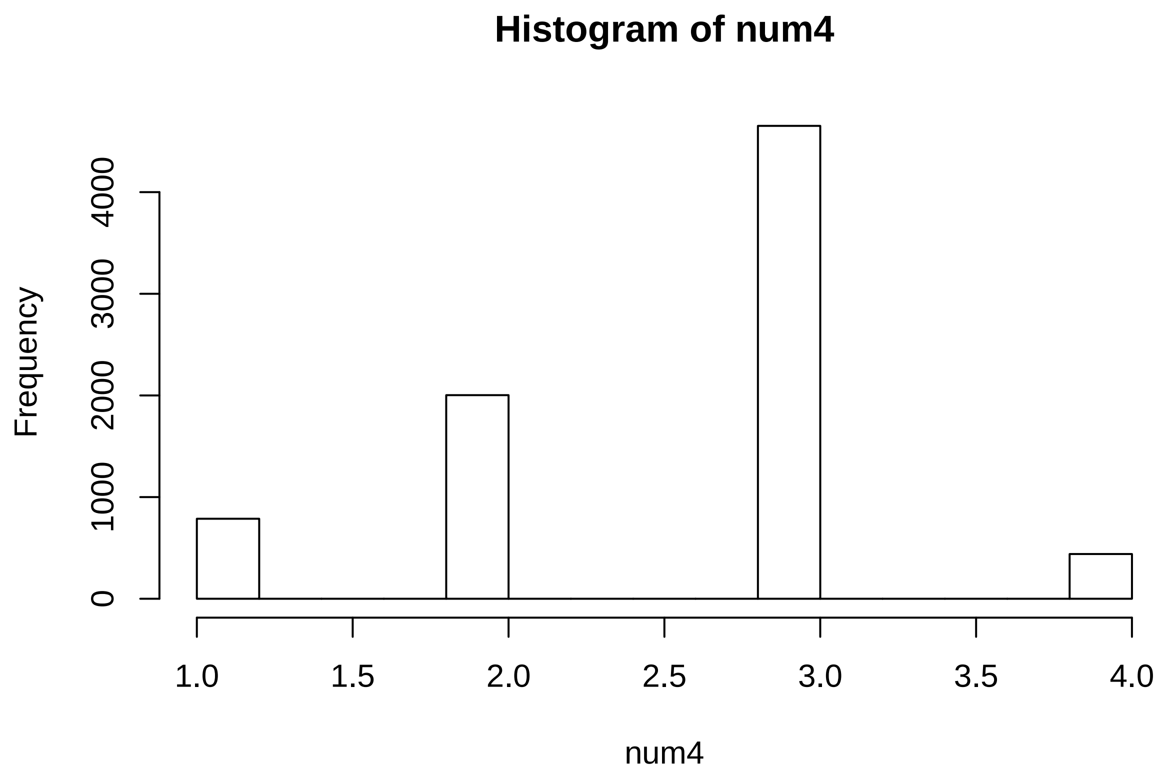
```r
barplot(socialclust4_3seg, main='Segment 3 kmeans++',col=rainbow(36),las=2)
```



```
barplot(socialclust4_4seg, main='Segment 4 kmeans++',col=rainbow(36),las=2)
```



```
num4 = clusters(social_clust4)
hist(num4)
```



In adding a fourth cluster, we have not really changed the profile of the previous three clusters. Cluster 1 maintains its user profile, as does cluster 2, and cluster 3 remains a bit of a catch-all. However, cluster 4 has now identified a small proportion of followers who have a very focused interest in online gaming and colleges/universities. While the minority group, their clear definition as online college gamers potentially makes accessing this group very straightforward.