PStat 131 Final Project 2016 Election

Stevyn Fessler, Chris Bell, Xi Sun

3/23/2018

```r
library(tidyverse)
library(plyr)
library(ggplot2)
library(cluster)
library(NbClust)
library(tree)
library(randomForest)
library(maptree)
library(ROCR)
library(class)
library(rpart)
```

Question 1

There are many reasons as to why predicting the outcome of an election is so hard. One of the main reasons that it is so difficult is that the president is chosen on a national level, but states vote seperately. Another reason is that because no one knows the result untill after it is an un observed variable. The data the polls brings in deals with how people think they'll vote but not how they actually vote. One other reason is that there are many different polls at state and national levels with varying amounts of credibility, yet they all have to be considered. These are all some of the reasons that predicting the election is so difficult.

Question 2

Nate Silver did a couple of things that were unique in his approach include a method in which he would calculate the probability that Obama would win if the election was called on that specific day. Another thing that he did that was unique was when he saw a nationwide shift, he would take into account the levels of support that a candidate has in certain regions before applying the shift to those regions.

Question 3

There were many explanations as to what could have gone wrong in 2016. A few of the explanations given in the article were that while in is normal for polls to have error, in 2016 the error was all in the same direction. Another reason is that the polls may have underestimated a certain demographic, particularly whites without college degrees. This combined with the fact that Trump outperformed almost all of his swing state predictions. One of the issues that relates to why election predicting is to hard is that Democrats had a lower than expected turnout. This relates to how polls gather info on how they think they will vote but not how they will or if they will vote. Some of the things that could be done to make future predictions better could include creating a heavier focus on swing states and making sure to use time scaling to see how the predictions change as time moves on and compare national shifts with state shifts in support.

Data Preparing

```
election.raw = read.csv("election.csv") %>% as.tbl
census_meta = read.csv("metadata.csv", sep = ";") %>% as.tbl
census = read.csv("census.csv") %>% as.tbl
census$CensusTract = as.factor(census$CensusTract)
```

Question 4

```
election_federal <- subset(election.raw, election.raw$fips=="US")

election_state <- c()
for (i in 1:nrow(election.raw)){
  for (j in 1:length(state.abb)){
    if (election.raw$fips[i]==state.abb[j]){
      election_state <- rbind(election_state, election.raw[i,])
    }
  }
}

abc <- rbind(election_federal, election_state)
election <- election.raw[!(election.raw$fips %in% abc$fips),]

head(election_federal)
```

```
## # A tibble: 6 x 5
##   county   fips       candidate  state    votes
##   <fctr> <fctr>          <fctr> <fctr>    <int>
## 1   <NA>     US    Donald Trump     US 62984825
## 2   <NA>     US Hillary Clinton     US 65853516
## 3   <NA>     US    Gary Johnson     US  4489221
## 4   <NA>     US       Jill Stein     US  1429596
## 5   <NA>     US    Evan McMullin     US   510002
## 6   <NA>     US  Darrell Castle     US   186545
```

```
head(election_state)
```

```
## # A tibble: 6 x 5
##   county   fips       candidate  state   votes
##   <fctr> <fctr>          <fctr> <fctr>   <int>
## 1   <NA>     CA Hillary Clinton     CA 8753788
## 2   <NA>     CA    Donald Trump     CA 4483810
## 3   <NA>     CA    Gary Johnson     CA  478500
## 4   <NA>     CA       Jill Stein     CA  278657
## 5   <NA>     CA  Gloria La Riva     CA   66101
## 6   <NA>     FL    Donald Trump     FL 4617886
```

```
head(election)
```

```
## # A tibble: 6 x 5
##               county   fips       candidate  state   votes
##               <fctr> <fctr>          <fctr> <fctr>   <int>
## 1 Los Angeles County   6037 Hillary Clinton     CA 2464364
## 2 Los Angeles County   6037    Donald Trump     CA  769743
```

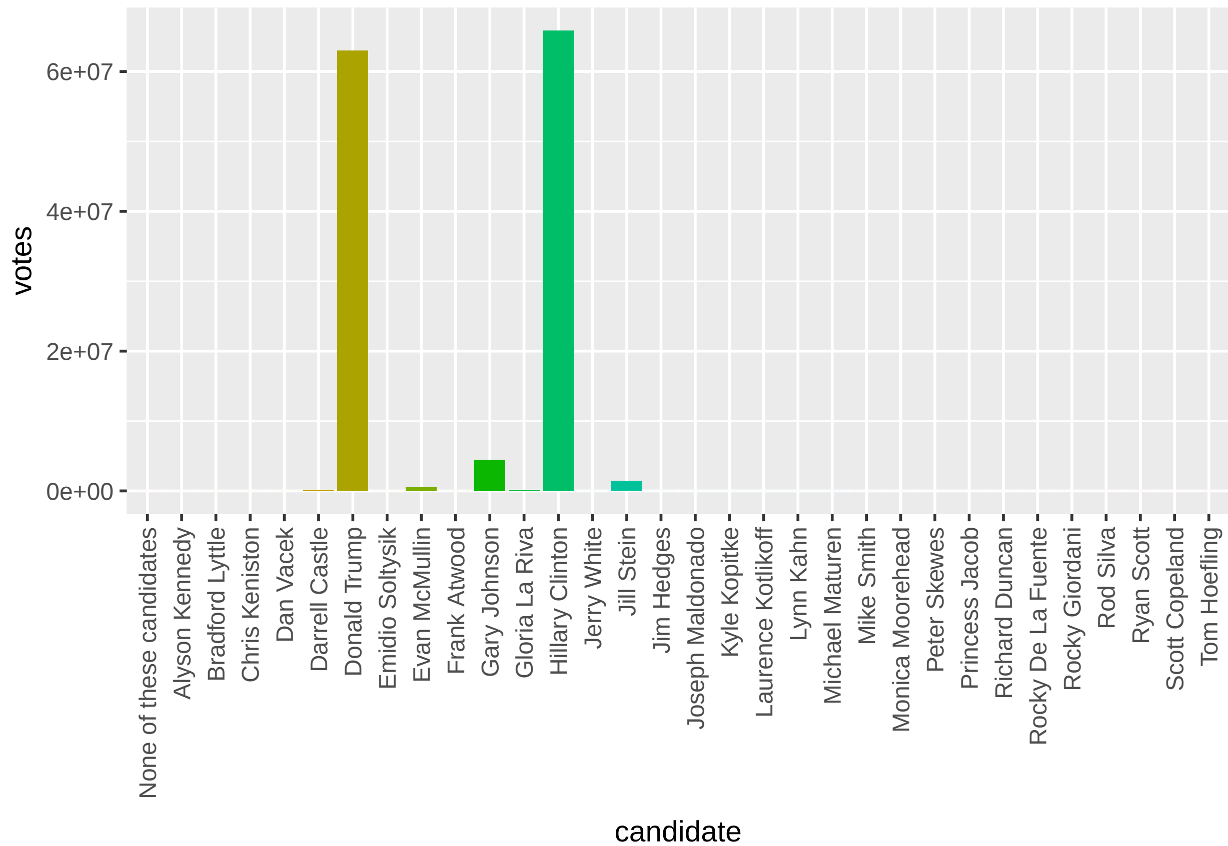
```
## 3 Los Angeles County   6037    Gary Johnson     CA   88968
## 4 Los Angeles County   6037      Jill Stein     CA   76465
## 5 Los Angeles County   6037  Gloria La Riva     CA   21993
## 6        Cook County  17031 Hillary Clinton     IL 1611946
```

Question 5

```r
length(election_federal$candidate)
```

```
## [1] 32
```

```r
ggplot(election_federal, aes(x=candidate, y=votes, fill=candidate)) +
  geom_bar(stat="identity") +
  guides(fill=FALSE) +
  theme(axis.text.x=element_text(angle=90, hjust=1, vjust=0.5))
```



The Graph below shows the names for 31 candidates in the 2016 election. However, it also shows that only 5 of those candidates had enough votes to appear on the graph and of those five almost all of the votes went to Donald Trump or Hillary Clinton.

Question 6

```r
pct.county <- ddply(election,
                    ~fips,
                    summarise,
```

```
                    candidate=candidate,
                    pct=votes/sum(votes),
                    state =state,
                    votes=votes,
                    county = county)

county_winner <- pct.county %>%
  group_by(fips) %>%
  top_n(n=1, wt=pct)

head(county_winner)
```

```
## # A tibble: 6 x 6
## # Groups:   fips [6]
##     fips       candidate       pct  state  votes            county
##   <fctr>          <fctr>     <dbl> <fctr>  <int>            <fctr>
## 1  10001    Donald Trump 0.4981282     DE  36991       Kent County
## 2  10003 Hillary Clinton 0.6230005     DE 162919 New Castle County
## 3  10005    Donald Trump 0.5916578     DE  62611     Sussex County
## 4   1001    Donald Trump 0.7339553     AL  18172    Autauga County
## 5   1003    Donald Trump 0.7732042     AL  72883    Baldwin County
## 6   1005    Donald Trump 0.5226140     AL   5454    Barbour County
```

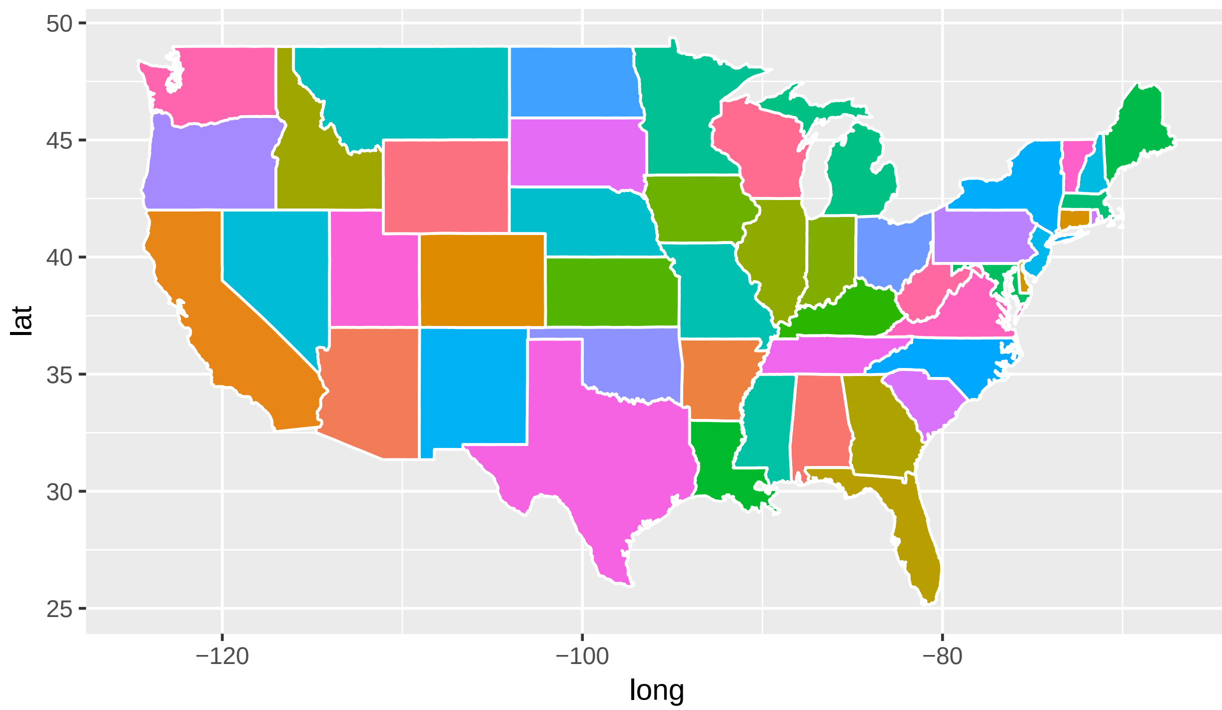
```
pct.state <- ddply(election_state,
                   ~fips,
                   summarise,
                   candidate=candidate,
                   pct=votes/sum(votes))

state_winner <- pct.state %>%
  group_by(fips) %>%
  top_n(n=1, wt=pct)

head(state_winner)
```

```
## # A tibble: 6 x 3
## # Groups:   fips [6]
##     fips       candidate       pct
##   <fctr>          <fctr>     <dbl>
## 1     AK    Donald Trump 0.5280650
## 2     AL    Donald Trump 0.6272447
## 3     AR    Donald Trump 0.6057410
## 4     AZ    Donald Trump 0.4903224
## 5     CA Hillary Clinton 0.6225644
## 6     CO Hillary Clinton 0.4815698
```

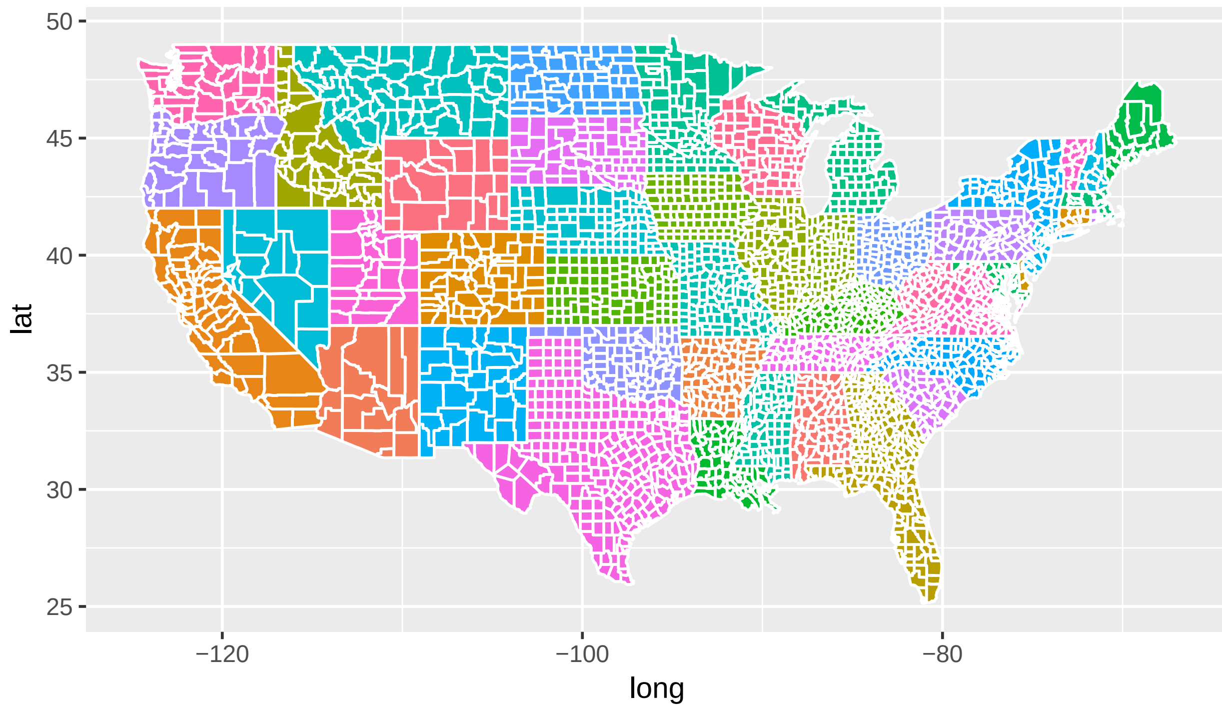
Question 7

```
states <- map_data("state")

ggplot(data = states) +
  geom_polygon(aes(x = long, y = lat, fill = region, group = group), color = "white") +
  coord_fixed(1.3) +
```

```
  guides(fill=FALSE)
```

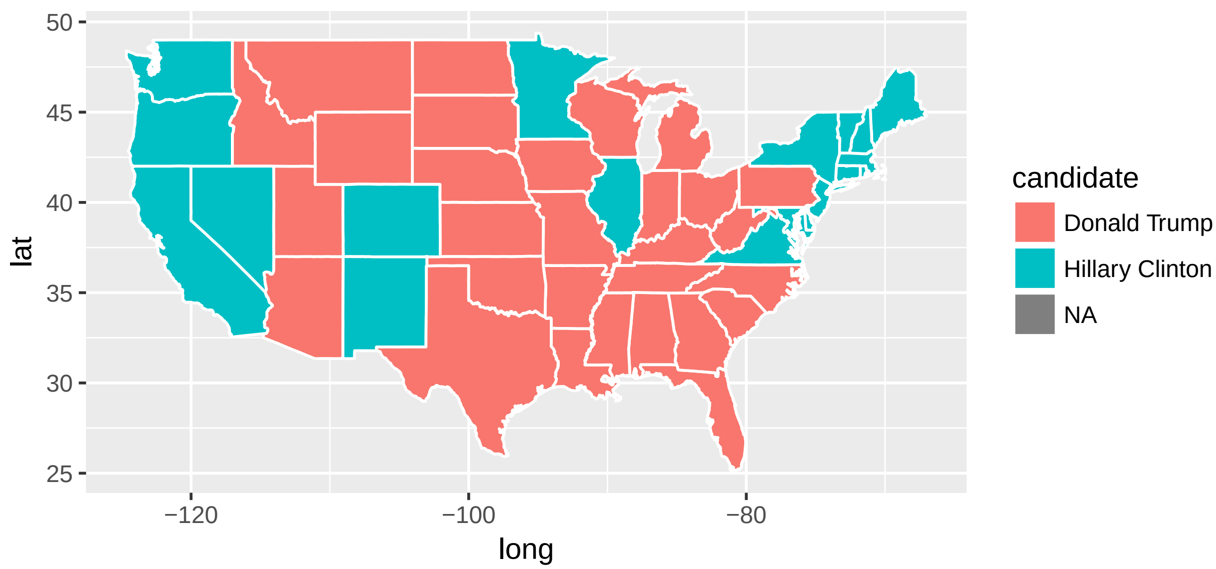



```
conties <- map_data("county")

ggplot(data=conties) +
  geom_polygon(aes(x=long, y=lat, fill=region, group=group), color="white") +
  coord_fixed(1.3) +
  guides(fill=FALSE)
```



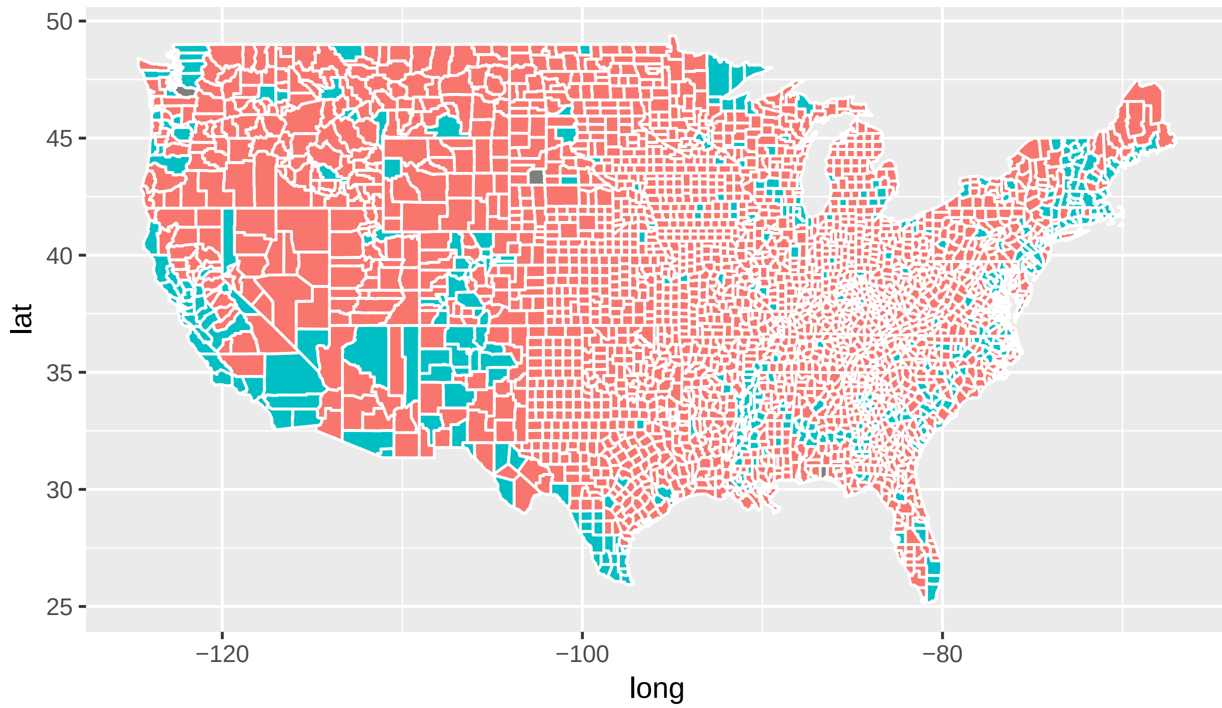
Question 8

```
fips <- state.abb[match(states$region, tolower(state.name))]

states.fips <- data.frame(states, fips)

states2 <- left_join(states.fips, state_winner, by="fips")

ggplot(data = states) +
  geom_polygon(data=states2,
               aes(x=long, y=lat, fill=candidate, group=group),
               color = "white") +
  coord_fixed(1.3)
```



Question 9

```
county <- conties

county.split <- maps::county.fips %>%
  separate(polyname, c("region", "subregion"), ",")

county.split <- cbind(as.character(county.split[,1]), county.split[,2:3])

colnames(county.split)[1] <- "fips"

county2 <- left_join(county, county.split, by=c("region", "subregion"))
county3 <- left_join(county2, county_winner, by="fips")

ggplot(data = county3) +
  geom_polygon(data=county3, aes(x=long, y=lat, fill=candidate, group=group), color = "white") +
  coord_fixed(1.3) +
  guides(fill=FALSE)
```



We will use the cleaned census data for visualization on question 10, therefore question 10 will appear after question 11.

Question 11 Part 1

```
census.del <- filter(census, complete.cases(census))

Men <- (census.del$Men/census.del$TotalPop)*100
Citizen <- (census.del$Citizen/census.del$TotalPop)*100
Employed <- (census.del$Employed/census.del$TotalPop)*100

census.del[, "Men"] <- Men
census.del[, "Citizen"] <- Citizen
census.del[, "Employed"] <- Employed

Minority <- census.del$Hispanic +
            census.del$Black +
            census.del$Native +
            census.del$Asian +
            census.del$Pacific
census.del <- data.frame(census.del, Minority)

census.del <- census.del[, -which(names(census.del) %in%
                                    c("Walk", "PublicWork", "Construction"))]
```

```r
head(census.del)
```

```
##   CensusTract   State  County TotalPop      Men Women Hispanic White Black
## 1  1001020100 Alabama Autauga     1948 48.25462  1008      0.9  87.4   7.7
## 2  1001020200 Alabama Autauga     2156 49.11874  1097      0.8  40.4  53.3
## 3  1001020300 Alabama Autauga     2968 45.95687  1604      0.0  74.5  18.6
## 4  1001020400 Alabama Autauga     4423 49.10694  2251     10.5  82.8   3.7
## 5  1001020500 Alabama Autauga    10763 45.73074  5841      0.7  68.5  24.8
## 6  1001020600 Alabama Autauga     3851 46.40353  2064     13.1  72.9  11.9
##   Native Asian Pacific  Citizen Income IncomeErr IncomePerCap
## 1    0.3   0.6     0.0 77.15606  61838     11900        25713
## 2    0.0   2.3     0.0 77.08720  32303     13538        18021
## 3    0.5   1.4     0.3 78.67251  44922      5629        20689
## 4    1.6   0.0     0.0 74.74565  54329      7003        24125
## 5    0.0   3.8     0.0 71.22549  51965      6935        27526
## 6    0.0   0.0     0.0 68.60556  63092      9585        30480
##   IncomePerCapErr Poverty ChildPoverty Professional Service Office
## 1            4548     8.1          8.4         34.7    17.0   21.3
## 2            2474    25.5         40.3         22.3    24.7   21.5
## 3            2817    12.7         19.7         31.4    24.9   22.1
## 4            2870     2.1          1.6         27.0    20.8   27.0
## 5            2813    11.4         17.5         49.6    14.2   18.2
## 6            7550    14.4         21.9         24.2    17.5   35.4
##   Production Drive Carpool Transit OtherTransp WorkAtHome MeanCommute
## 1       15.2  90.2     4.8       0         2.3        2.1        25.0
## 2       22.0  86.3    13.1       0         0.7        0.0        23.4
## 3       12.4  94.8     2.8       0         0.0        2.5        19.6
## 4       16.4  86.6     9.1       0         2.6        1.6        25.3
## 5       15.8  88.0    10.5       0         0.6        0.9        24.8
## 6       14.9  82.7     6.9       0         6.0        4.5        19.8
##   Employed PrivateWork SelfEmployed FamilyWork Unemployment Minority
## 1 48.40862        77.1          4.6          0          5.4      9.5
## 2 34.92579        77.0          6.1          0         13.3     56.4
## 3 46.26011        64.1         12.3          0          6.2     20.8
## 4 40.28940        75.7          3.1          0         10.8     15.8
## 5 46.79922        67.1          5.3          0          4.2     29.3
## 6 40.50896        79.4          5.8          0         10.9     25.0
```

Question 11 Part 2

```r
census.del2 <- census.del %>%
  group_by(State, County) %>%
  add_tally(sum(TotalPop))

names(census.del2)[names(census.del2)=="n"] <- "CountyTotal"

Weight <- census.del2$TotalPop/census.del2$CountyTotal

census.subct <- data.frame(census.del2, Weight)
```

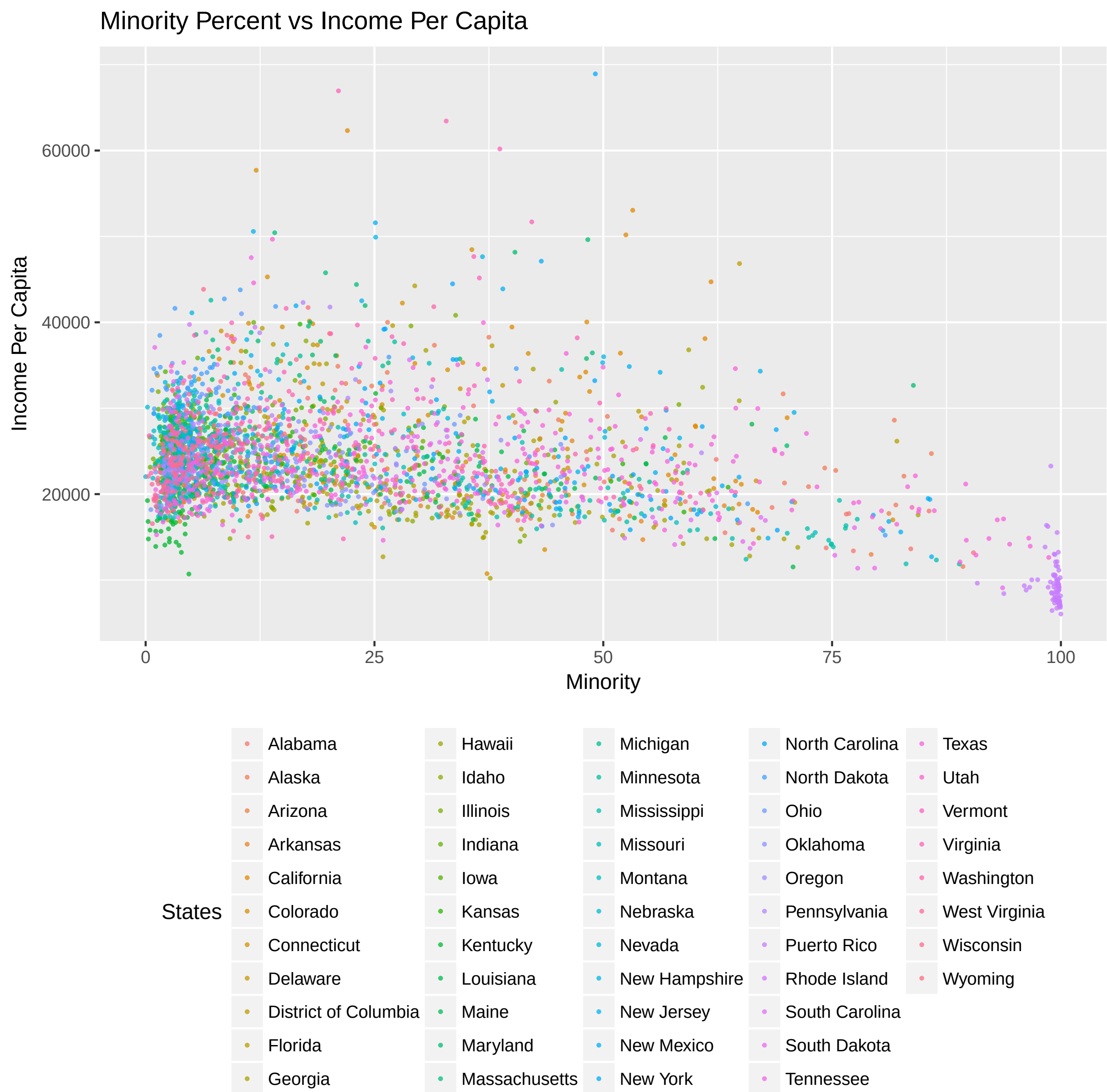
Question 11 Part 3 & 4

```
census.ct <- census.subct %>%
  group_by(State, County) %>%
  summarise_at(vars(TotalPop:Weight), mean)
head(census.ct)
```

```
## # A tibble: 6 x 36
## # Groups:   State [1]
##     State  County TotalPop      Men    Women Hispanic    White     Black
##    <fctr>  <fctr>    <dbl>    <dbl>    <dbl>    <dbl>    <dbl>     <dbl>
## 1 Alabama Autauga 4601.750 48.36759 2373.000 2.850000 73.15000 20.825000
## 2 Alabama Baldwin 6294.226 48.82690 3219.581 4.064516 83.45484  9.625806
## 3 Alabama Barbour 2992.444 52.22358 1381.667 4.800000 46.61111 46.488889
## 4 Alabama    Bibb 5651.000 53.24573 2632.750 2.475000 77.50000 17.725000
## 5 Alabama  Blount 6412.222 49.53062 3244.222 8.888889 87.80000  1.355556
## 6 Alabama Bullock 3559.333 51.78404 1672.667 2.766667 22.00000 72.466667
## # ... with 28 more variables: Native <dbl>, Asian <dbl>, Pacific <dbl>,
## #   Citizen <dbl>, Income <dbl>, IncomeErr <dbl>, IncomePerCap <dbl>,
## #   IncomePerCapErr <dbl>, Poverty <dbl>, ChildPoverty <dbl>,
## #   Professional <dbl>, Service <dbl>, Office <dbl>, Production <dbl>,
## #   Drive <dbl>, Carpool <dbl>, Transit <dbl>, OtherTransp <dbl>,
## #   WorkAtHome <dbl>, MeanCommute <dbl>, Employed <dbl>,
## #   PrivateWork <dbl>, SelfEmployed <dbl>, FamilyWork <dbl>,
## #   Unemployment <dbl>, Minority <dbl>, CountyTotal <dbl>, Weight <dbl>
```

Question 10

```
ggplot() + geom_point(data = census.ct, aes(Minority, IncomePerCap,color = census.ct$State), size = .5,
```



The Scatterplot above shows the Income per Capita compared to the percentage of the population that are minorities. Each point represents a unique county within either a state, DC, or Puerto Rico. We can see that there is some slight correlation between income and percentage of minorities. The higher the percentage of minorities the less income per capita in a county.

Question 12

```
census.subct2 <- census.subct[, 4:ncol(census.subct)]
pc.census.subct <- prcomp(census.subct2)
pc.subct <- pc.census.subct$x
subct.pc <- data.frame(pc.subct)
head(subct.pc)
```

```
##         PC1        PC2         PC3        PC4        PC5        PC6
## 1 -1041580   4871.206  3758.05874 -3466.8919 -2906.0191 -1174.5827
## 2 -1041625 -24805.729 -1998.73551 -7734.2190 -1527.2696   957.3530
## 3 -1041609 -13257.720  2080.65654  1126.5607 -1622.6782  -232.5840
## 4 -1041594  -3181.689  3111.09445   940.5835  -188.9382   -36.0470
## 5 -1041595  -3814.383  -244.61498  1964.2855  7370.8815   512.2022
## 6 -1041577   7901.386   -71.85555  -290.0689  -776.5284 -3393.2172
##          PC7       PC8        PC9      PC10       PC11       PC12
## 1  42.652629 -45.71414  -2.706350  7.273573 -10.700236   1.090680
## 2   1.070415  22.66795 -31.879474 22.845329   3.493737   2.624855
## 3  93.022695 -26.46305 -10.223634 13.016356  -7.732274  -9.247962
## 4   8.686037 -30.66701   4.255256  6.301381 -16.710345   7.095454
## 5 302.961977 -14.07746 -19.345572  6.434501  -4.971781 -12.574450
## 6 115.689192 -23.32654   2.959755  6.098167   3.263702   8.913942
##         PC13       PC14        PC15      PC16       PC17       PC18
## 1  0.4164734 -3.0482889  -0.1441516 -1.756269  -4.636772 -1.9629464
## 2 -1.8681576  5.7749187  -1.2536464 -2.577693  -1.734599  1.7615628
## 3  4.1242753  4.3483050  -2.7263015  5.415055 -12.180391 -5.1936642
## 4  6.2128913  8.9694762   1.8201036 -1.305101   2.699007 -4.1159591
## 5  2.1727623  0.7537019 -11.4399572 -7.741069  -7.391125  2.2823116
## 6  1.8355572  3.1849597   3.0792899  3.025831   3.283700  0.1581135
##          PC19       PC20        PC21       PC22        PC23       PC24
## 1  3.87709301  2.4218958  0.03079033  1.0601141  0.06426869 -1.1212229
## 2  3.36291103 -1.3155632 -2.32066332 -2.4246190  1.87874047 -1.5626241
## 3  2.57264634  5.0372218 -5.40127878 -0.1350388 -1.41132137  0.2037725
## 4  0.08042313  2.2837875  0.57637364 -2.1532622  3.26812968  2.1837067
## 5  3.99756155 -0.2326278 -2.49299335 -1.6770415  3.80502301 -2.4951940
## 6 -8.79351038  7.4114772 -3.32864202  0.9402121 -0.82388741  2.5657114
##        PC25       PC26       PC27       PC28        PC29       PC30
## 1 -1.823366 -1.4741706 -0.1555711 -0.5348014 -1.28682674  0.1866775
## 2  1.536534 -0.9194317  0.8995463  0.8715323  0.04927785 -0.4488934
## 3  2.667171 -0.8490568  1.5054519  0.0339869 -1.87956938 -0.9490494
## 4 -3.765097 -0.7575186  1.3968121 -0.1519115  0.94387612  0.4477034
## 5 -1.191036  0.2201392  1.3055341  3.4574611  2.60857296 -0.2131685
## 6 -3.115814 -2.5947006  1.9492875 -0.7007477 -0.09863811  1.9597324
##          PC31      PC32        PC33          PC34
## 1 -0.29065743 0.1580542 -0.02151739  1.824720e-12
## 2 -0.32303102 0.1543218 -0.02770026  3.366201e-12
## 3 -0.29565888 0.2484390 -0.05287911  4.747033e-12
## 4 -0.02084545 0.1602690  0.02638878 -8.466301e-13
## 5  0.03668122 0.1446216  0.08854063 -4.456397e-12
## 6 -0.24337700 0.1885632  0.04439608 -2.799975e-12
```

```
summary(pc.census.subct)
```

```
## Importance of components:
##                              PC1       PC2       PC3       PC4  PC5  PC6
## Standard deviation     1.927e+06 3.154e+04 7.708e+03 4.488e+03 2235 1527
## Proportion of Variance 9.997e-01 2.700e-04 2.000e-05 1.000e-05    0    0
## Cumulative Proportion  9.997e-01 1.000e+00 1.000e+00 1.000e+00    1    1
##                          PC7   PC8   PC9  PC10  PC11  PC12  PC13  PC14
## Standard deviation     169.3 40.05 22.85 16.39 14.68 10.65 9.544 8.156
## Proportion of Variance   0.0  0.00  0.00  0.00  0.00  0.00 0.000 0.000
## Cumulative Proportion    1.0  1.00  1.00  1.00  1.00  1.00 1.000 1.000
##                         PC15  PC16  PC17  PC18  PC19 PC20  PC21 PC22  PC23
```

```
## Standard deviation     7.221 6.887 6.383 6.053 5.625 5.09 4.765 4.23 3.861
## Proportion of Variance 0.000 0.000 0.000 0.000 0.000 0.00 0.000 0.00 0.000
## Cumulative Proportion  1.000 1.000 1.000 1.000 1.000 1.00 1.000 1.00 1.000
##                         PC24  PC25  PC26  PC27  PC28  PC29  PC30  PC31
## Standard deviation     3.499 2.878 2.432 2.123 1.876 1.842 1.533 0.995
## Proportion of Variance 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
## Cumulative Proportion  1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000
##                          PC32   PC33      PC34
## Standard deviation     0.4477 0.0826 1.913e-10
## Proportion of Variance 0.0000 0.0000 0.000e+00
## Cumulative Proportion  1.0000 1.0000 1.000e+00
```

```r
census.ct2 <- census.ct[, 3:ncol(census.ct)]
pc.census.ct <- prcomp(census.ct2)
pc.ct <- pc.census.ct$x
ct.pc <- data.frame(pc.ct)
head(ct.pc)
```

```
##         PC1       PC2         PC3       PC4        PC5       PC6       PC7
## 1  43802.85 -3768.187   956.17081  834.6755   439.5024  393.4899 -72.02159
## 2 -96086.83 -1949.164 -1407.48258 1962.8070  2691.3329 -376.1157 -20.78861
## 3  72330.85 15110.277   415.48296  251.6863 -1061.2769  108.7560  89.97443
## 4  76558.18  7130.621  2440.91707  366.3961  1442.2920  873.8584 158.22626
## 5  41387.49  2338.540  3488.76356 2355.4039  1717.2446 -976.0526 -19.31670
## 6  88561.91 13381.289    85.34153 2902.7391  -667.8090  189.0576  84.96449
##         PC8        PC9          PC10      PC11       PC12       PC13
## 1  11.06519  12.879906  -2.382204497 -3.017225  3.7790706  6.0811096
## 2  19.38906  -2.642667   2.668046988 -2.489090  0.6712149  5.0424223
## 3 -27.61447  42.512467 -10.923271230  6.456375 -0.3036778 -0.4747089
## 4  25.63285  13.997325  -5.201966723 -2.938133 -3.2878290 -1.1929414
## 5  34.00864 -11.866322   0.009722318  2.440199 -1.4830869 -1.6024197
## 6 -46.63262  66.509284 -11.371209541 -5.578639 -3.7088213 -9.2074075
##        PC14       PC15       PC16       PC17       PC18       PC19
## 1  1.943107 -0.4015538 -0.6038176 -1.1364789  0.7887315  0.5340313
## 2  1.376539  2.2835389 -0.6342368  1.3840665  1.5542532 -2.3711436
## 3 -1.956219 -4.3949189  1.9926756 -0.9144222 -1.6299579 -4.3098216
## 4  6.635150 -5.2397779  1.9151465 -2.0950941 -0.5792294  2.3356183
## 5  5.144780 -4.0789245  4.0650116  2.1729609  0.2900468 -0.2518585
## 6  6.353620 -2.2289188  4.7625717 -1.3721596  2.2848686 -5.3523539
##         PC20        PC21       PC22       PC23        PC24      PC25
## 1 -1.8691793 -1.19418415 -0.3490629  0.5739089 -0.68668924 1.1258399
## 2  0.5804598 -0.64851699  0.1312563  0.6994323  0.19123323 0.3838623
## 3 -1.7283747  0.09778255 -0.6657844 -0.5217046 -2.94098089 1.6349367
## 4  1.4364446  1.57220213  1.0475415  1.3356205 -2.52093514 0.2697769
## 5 -0.2102285  0.10223120  1.5710080  2.6038253  0.01425523 0.9747514
## 6  1.0349583  3.90691257 -1.0631081 -0.8713284 -1.75527086 1.5832893
##          PC26       PC27        PC28        PC29        PC30        PC31
## 1  0.07373209 -0.7068069 -0.07896022 -0.03368132  0.07168155  0.12586516
## 2 -0.14366015  0.2908329  0.30601277 -0.66634961  0.23300221 -0.21740424
## 3 -0.13617586 -0.1407877 -0.23188336 -0.54716108  0.12116239  0.03435296
## 4  1.83083127  1.6516710  0.01554409  0.55166212 -0.22141516 -0.66782586
## 5 -0.54174595  0.2454813  0.04857120 -0.66130196  0.69859451 -0.12882519
## 6 -0.53114429  1.8474901 -0.11320995  0.12284482 -0.14257318  0.16610582
##           PC32       PC33         PC34
## 1  0.037438506  0.04004560  1.416653e-12
```

```
## 2 -0.008856174  0.01129161 -1.672304e-12
## 3 -0.021916611  0.14735162  6.391634e-12
## 4  0.127951003  0.03343784 -1.145152e-11
## 5 -0.080443829 -0.02651660  2.796080e-13
## 6  0.024677783 -0.05266250 -5.861206e-13
```

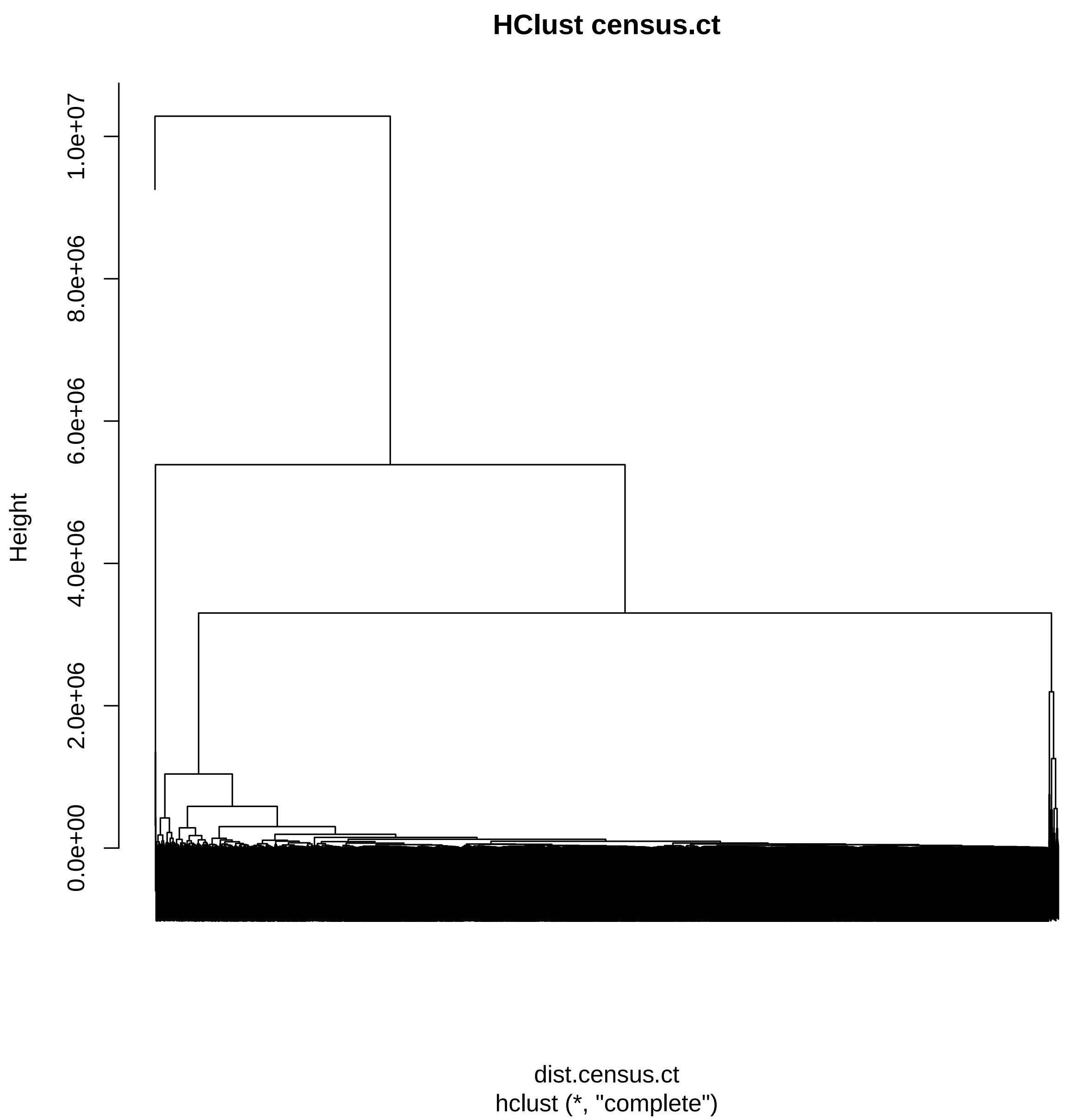
```r
summary(pc.census.ct)
```

```
## Importance of components:
##                              PC1       PC2       PC3       PC4       PC5
## Standard deviation     3.179e+05 1.407e+04 2.678e+03 2.022e+03 1.204e+03
## Proportion of Variance 9.979e-01 1.960e-03 7.000e-05 4.000e-05 1.000e-05
## Cumulative Proportion  9.979e-01 9.999e-01 9.999e-01 1.000e+00 1.000e+00
##                          PC6   PC7   PC8   PC9  PC10  PC11  PC12  PC13
## Standard deviation     662.9 83.44 30.05 16.94 10.53 7.442 6.879 5.974
## Proportion of Variance   0.0  0.00  0.00  0.00  0.00 0.000 0.000 0.000
## Cumulative Proportion    1.0  1.00  1.00  1.00  1.00 1.000 1.000 1.000
##                         PC14  PC15  PC16  PC17  PC18  PC19 PC20  PC21
## Standard deviation     5.152 4.575 3.774 3.335 3.238 2.967  2.6 2.553
## Proportion of Variance 0.000 0.000 0.000 0.000 0.000 0.000  0.0 0.000
## Cumulative Proportion  1.000 1.000 1.000 1.000 1.000 1.000  1.0 1.000
##                         PC22  PC23  PC24  PC25  PC26  PC27  PC28   PC29
## Standard deviation     2.244 2.212 1.867 1.845 1.627 1.444 1.049 0.8689
## Proportion of Variance 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.0000
## Cumulative Proportion  1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.0000
##                          PC30   PC31   PC32   PC33      PC34
## Standard deviation     0.7898 0.3905 0.3302 0.1663 3.136e-11
## Proportion of Variance 0.0000 0.0000 0.0000 0.0000 0.000e+00
## Cumulative Proportion  1.0000 1.0000 1.0000 1.0000 1.000e+00
```

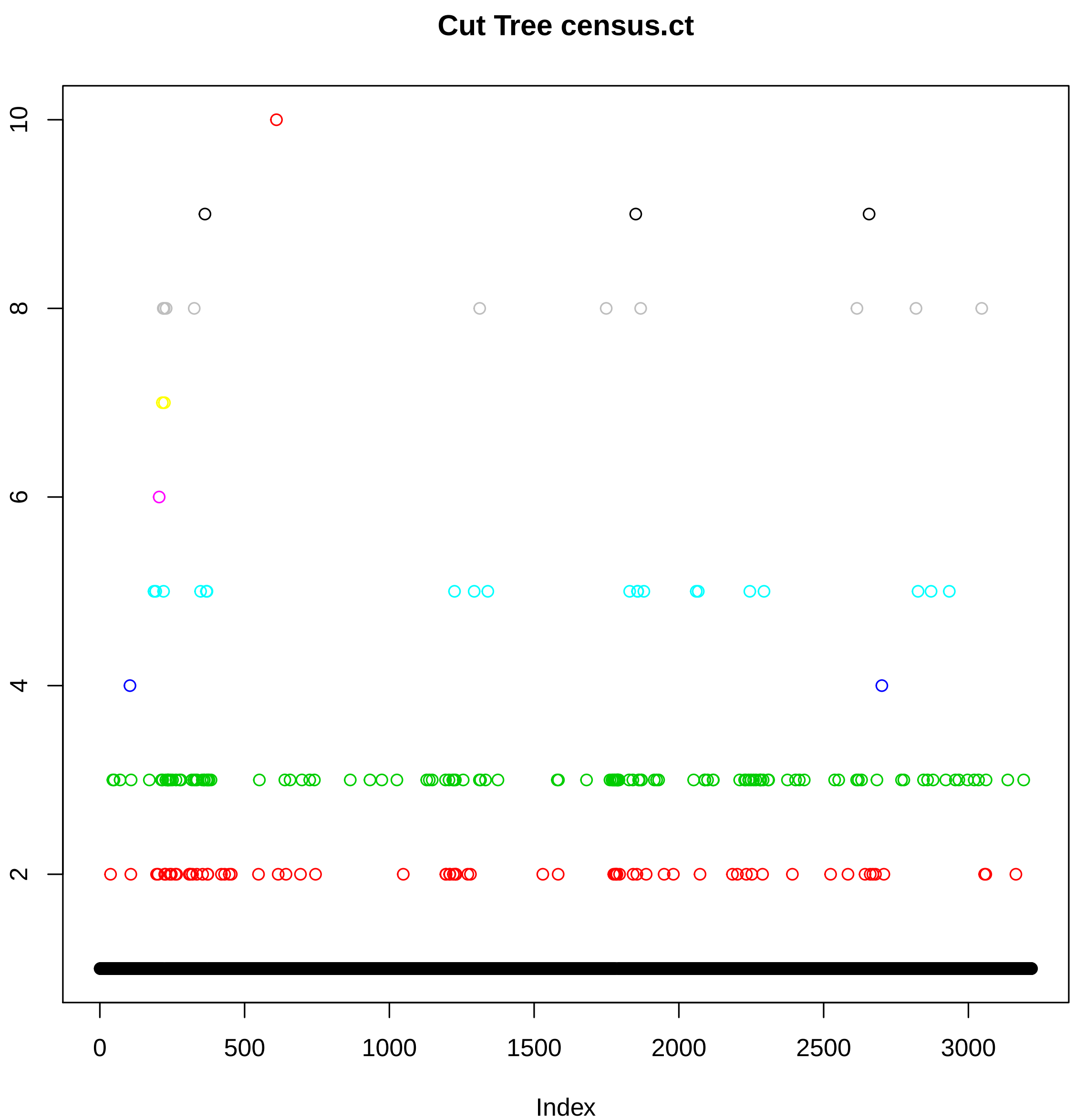
The most Prominent Loadings are PC1 and PC2.

Question 13

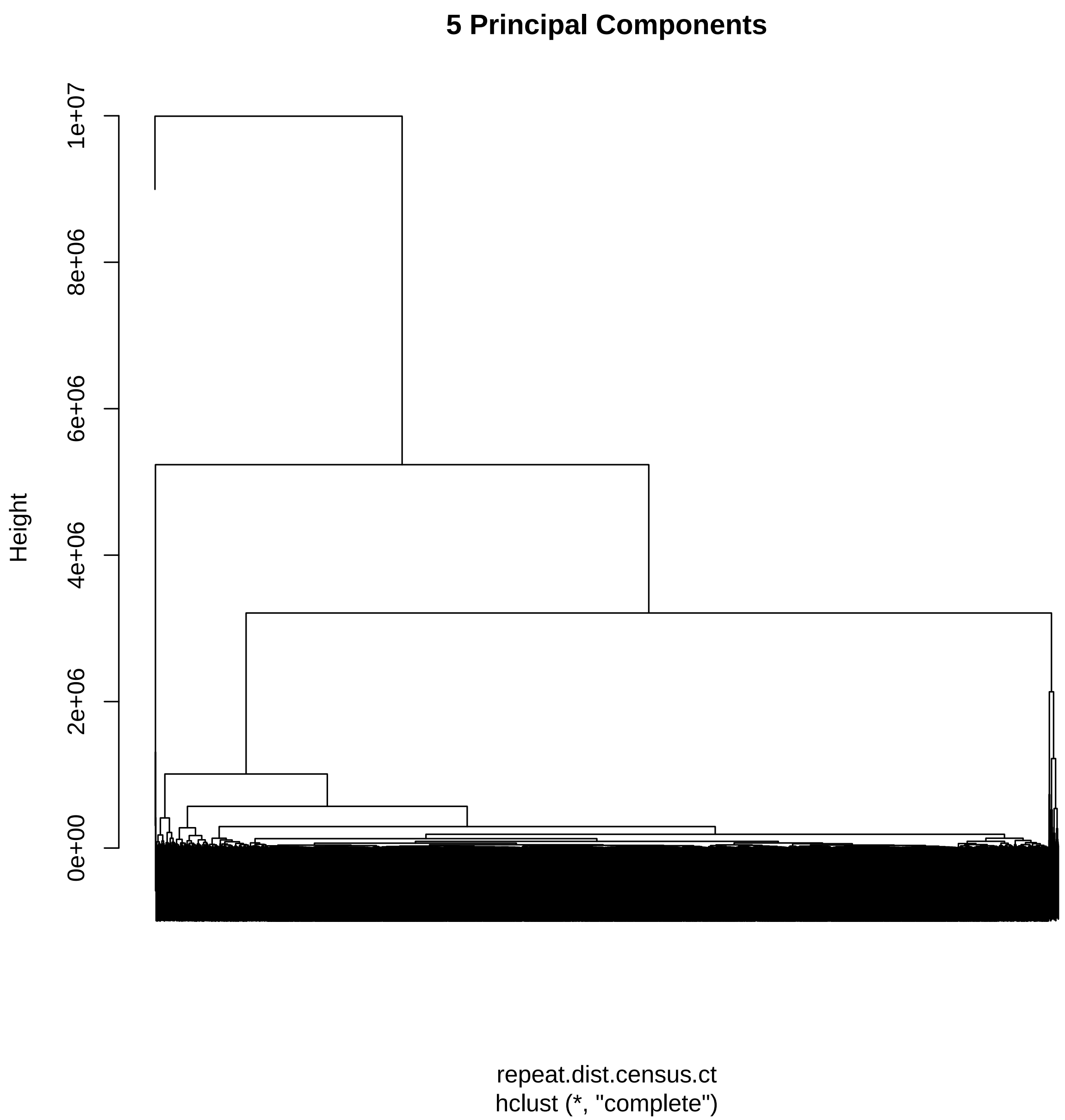
```r
dist.census.ct <- dist(census.ct, method="euclidean")
hc.census.ct <- hclust(dist.census.ct, method="complete")
plot(hc.census.ct, main="HClust census.ct", labels = FALSE)
```



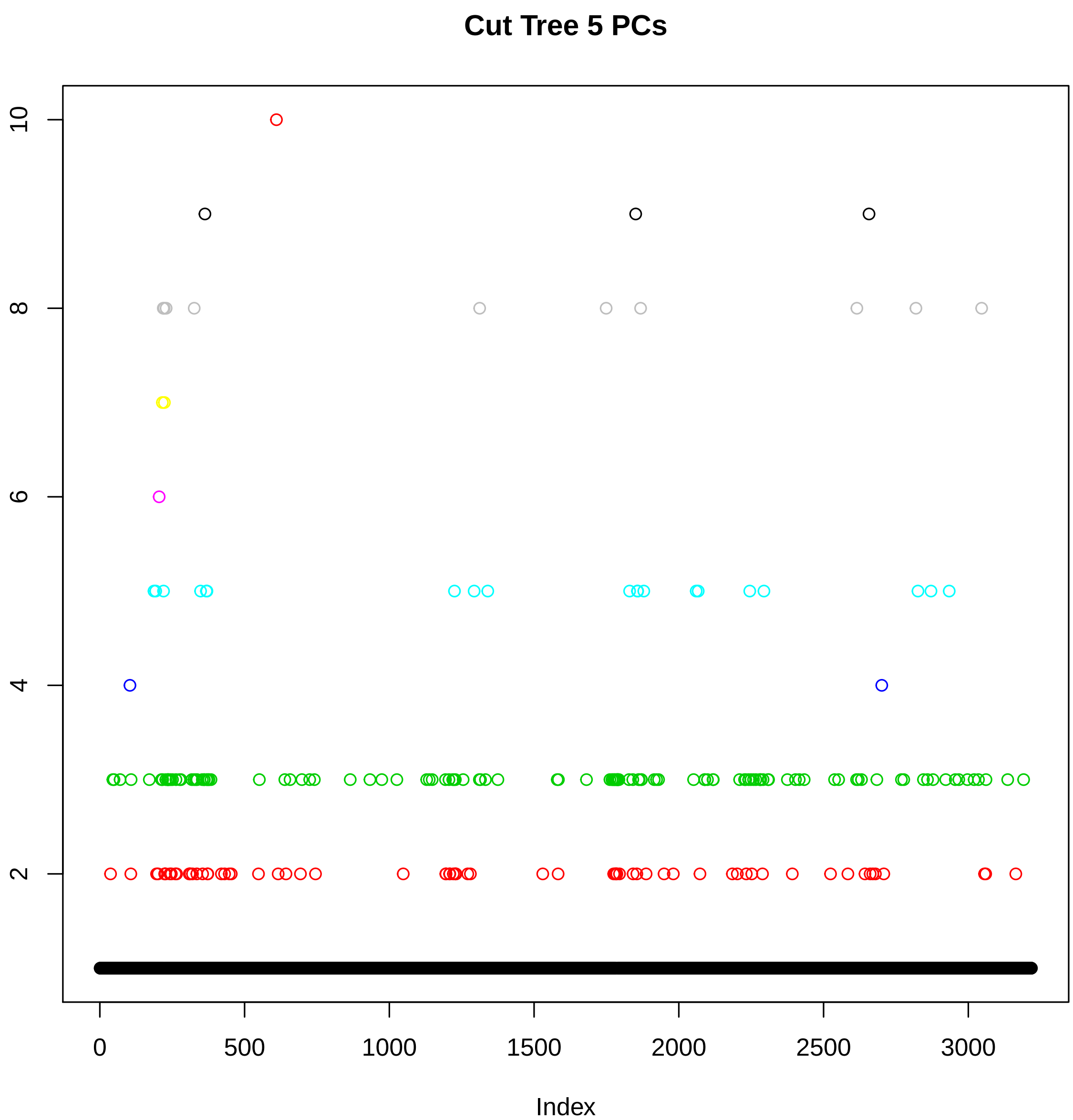

```r
hc.census.cut <- cutree(hc.census.ct, 10)
plot(hc.census.cut, col=hc.census.cut, ylab = "", main = "Cut Tree census.ct")
```




```
repeat.census.ct <- ct.pc[,1:5]
repeat.dist.census.ct <- dist(repeat.census.ct, method="euclidean")
repeat.hc.census.ct <- hclust(repeat.dist.census.ct, method="complete")
plot(repeat.hc.census.ct, main="5 Principal Components", labels = FALSE)
```




```r
repeat.hc.census.cut <- cutree(repeat.hc.census.ct, 10)
plot(repeat.hc.census.cut, col=repeat.hc.census.cut, ylab = "", main="Cut Tree 5 PCs")
```



San Mateo clusters

```
census.ct[which(census.ct$County == "San Mateo") ,]
```

```
## # A tibble: 1 x 36
## # Groups:   State [1]
##         State    County TotalPop      Men   Women Hispanic    White
##        <fctr>    <fctr>    <dbl>    <dbl>   <dbl>    <dbl>    <dbl>
## 1 California San Mateo 4813.348 49.19307 2445.29 23.43871 43.61097
## # ... with 29 more variables: Black <dbl>, Native <dbl>, Asian <dbl>,
## #   Pacific <dbl>, Citizen <dbl>, Income <dbl>, IncomeErr <dbl>,
## #   IncomePerCap <dbl>, IncomePerCapErr <dbl>, Poverty <dbl>,
```

```
## #   ChildPoverty <dbl>, Professional <dbl>, Service <dbl>, Office <dbl>,
## #   Production <dbl>, Drive <dbl>, Carpool <dbl>, Transit <dbl>,
## #   OtherTransp <dbl>, WorkAtHome <dbl>, MeanCommute <dbl>,
## #   Employed <dbl>, PrivateWork <dbl>, SelfEmployed <dbl>,
## #   FamilyWork <dbl>, Unemployment <dbl>, Minority <dbl>,
## #   CountyTotal <dbl>, Weight <dbl>
```

```
repeat.hc.census.cut[which(census.ct$County == "San Mateo")]
```

```
## [1] 2
```

```
#census.ct[which(hc.census.cut==2),]
#census.ct[which(repeat.hc.census.cut==2),]
```

According to the result, we can see that in both census.ct and repeated with first 5 PCs in ct.pc, San Mateo county is in the same cluster. In this case, we can conclude that whether or not we are using the original data set or the data set with principle components, San Mateo county has similar features as other counties in the same clusters in both cases.

By observing the result of two lines of code with "#"(since each contains 69 rows of dataframe, we will not print that out), we can assume that the similar features are total populations and employment information. Therefore, the first 5 PCs may have already contained those features for clustering which will result in the same clustering in both cases.

Classification

```
tmpwinner <- county_winner %>% ungroup %>%
  mutate(state = state.name[match(state, state.abb)]) %>%
  mutate_at(vars(state, county), tolower) %>%
  mutate(county = gsub(" county| columbia| city| parish", "", county))

census.ct$State <- tolower(census.ct$State)

tmpcensus <- census.ct %>%
  mutate_at(vars(1, 2), tolower)

election.cl <- tmpwinner %>%
  left_join(tmpcensus, by = c("state"="State", "county"="County")) %>%
  na.omit

attr(election.cl, "location") <- election.cl %>%
  select(c(county, fips, state, votes, pct))

election.cl <- election.cl %>%
  select(-c(county, fips, state, votes, pct))

set.seed(10)
n = nrow(election.cl)
in.trn <- sample.int(n, 0.8*n)
trn.cl <- election.cl[in.trn,]
tst.cl <- election.cl[-in.trn,]

set.seed(20)
nfold <- 10
```

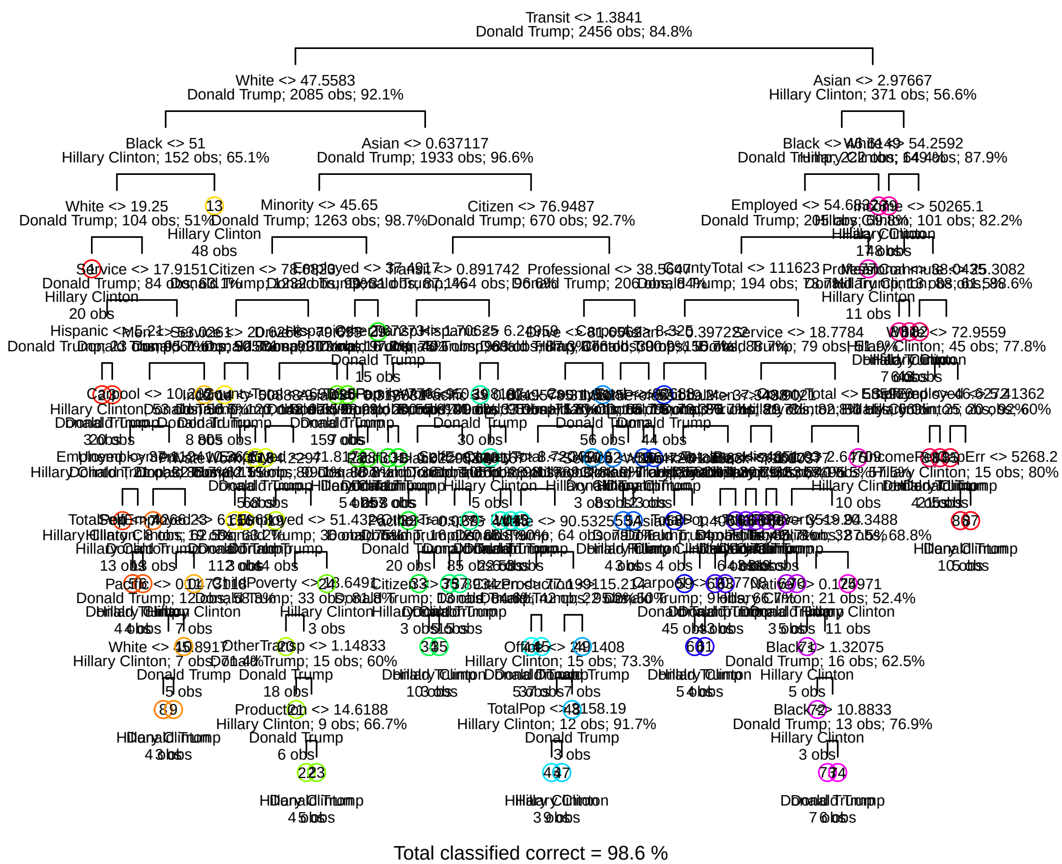
```r
folds <- sample(cut(1:nrow(trn.cl), breaks=nfold, labels=FALSE))
calc_error_rate <- function(predicted.value, true.value){
  return(mean(true.value!=predicted.value))
}
records <- matrix(NA, nrow=3, ncol=2)
colnames(records) <- c("train.error","test.error")
rownames(records) <- c("tree","knn","lda")
```

Question another 13

```r
trntree <- tree(candidate~., data = trn.cl)
trncontrol <- tree.control(nobs=2456, minsize = 7, mindev = 1e-6)
trntree <- tree(candidate~., data = trn.cl, control = trncontrol)
summary(trntree)
```

```
##
## Classification tree:
## tree(formula = candidate ~ ., data = trn.cl, control = trncontrol)
## Variables actually used in tree construction:
##  [1] "Transit"         "White"           "Black"
##  [4] "Service"         "Hispanic"        "Men"
##  [7] "Carpool"         "Employed"        "TotalPop"
## [10] "Unemployment"    "SelfEmployed"    "Pacific"
## [13] "Asian"           "Minority"        "Citizen"
## [16] "Income"          "PrivateWork"     "Drive"
## [19] "CountyTotal"     "ChildPoverty"    "OtherTransp"
## [22] "Production"      "FamilyWork"      "Professional"
## [25] "Office"          "WorkAtHome"      "Poverty"
## [28] "Native"          "MeanCommute"     "IncomePerCapErr"
## Number of terminal nodes:  87
## Residual mean deviance:  0.05358 = 126.9 / 2369
## Misclassification error rate: 0.01425 = 35 / 2456
```

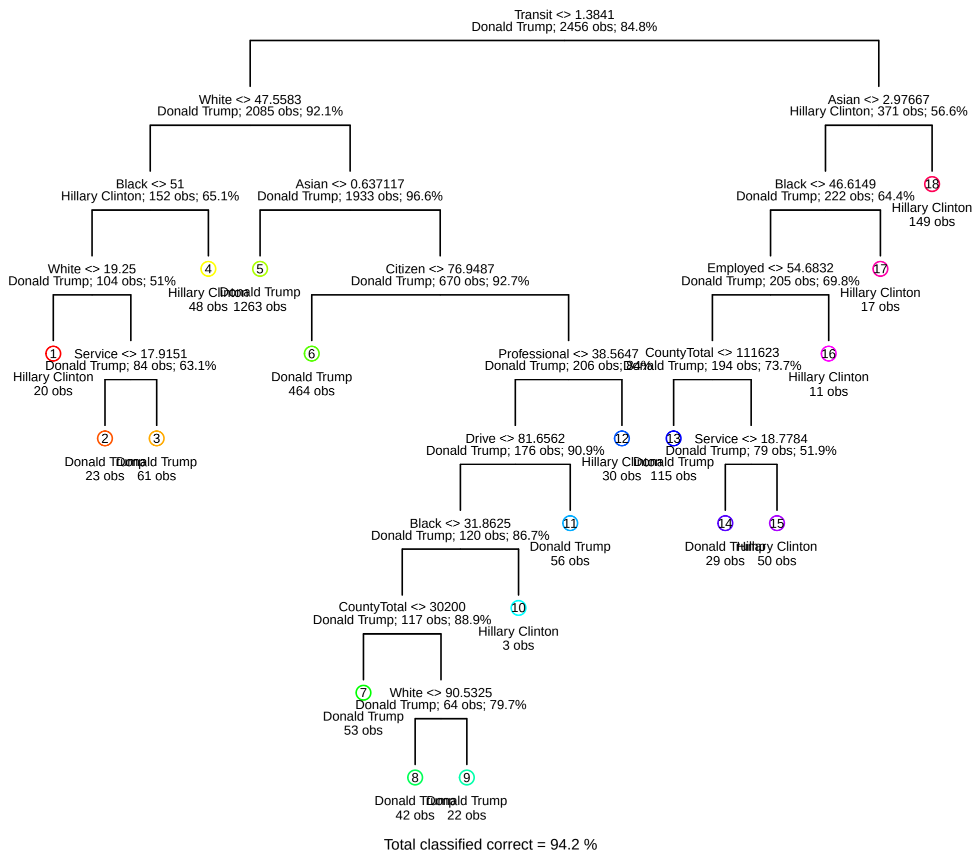
```r
draw.tree(trntree, nodeinfo=TRUE, cex=.6)
```



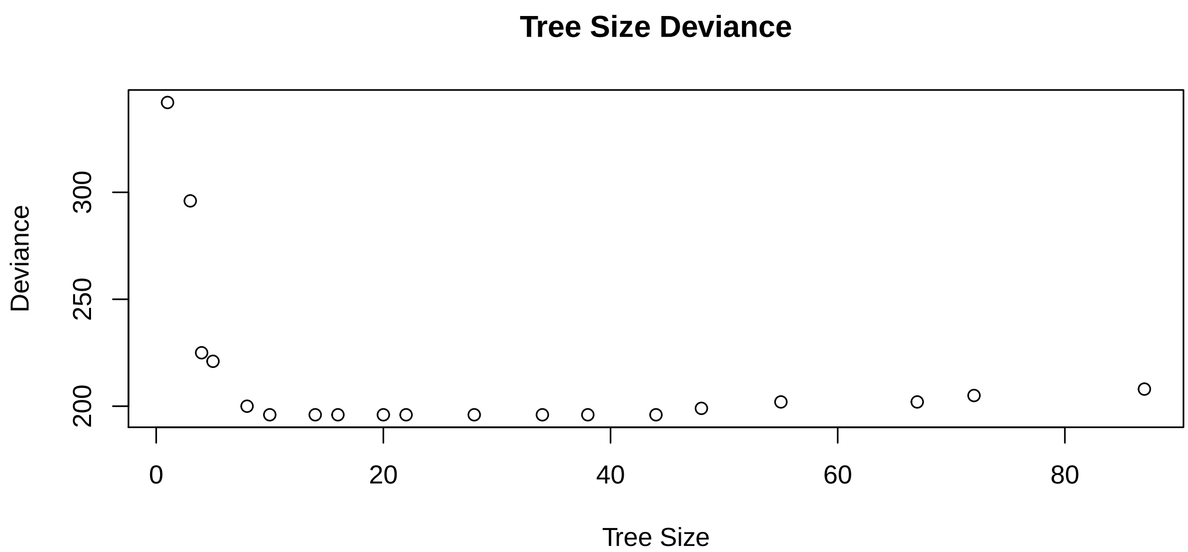

```r
prunedtree <- prune.tree(trntree,best=15)
summary(prunedtree)
```

```
##
## Classification tree:
## snip.tree(tree = trntree, nodes = c(34L, 98L, 738L, 99L, 47L,
## 739L, 10L, 48L, 22L, 35L, 7L))
## Variables actually used in tree construction:
##  [1] "Transit"      "White"        "Black"        "Service"
##  [5] "Asian"        "Citizen"      "Professional" "Drive"
##  [9] "CountyTotal"  "Employed"
## Number of terminal nodes:  18
## Residual mean deviance:  0.317 = 772.9 / 2438
## Misclassification error rate: 0.05782 = 142 / 2456
```

```r
draw.tree(prunedtree, cex=0.5, nodeinfo=TRUE)
```




```r
good.test <- tst.cl$candidate
cvtree <- cv.tree(trntree,rand=folds,prune.misclass, K=9)


op <- par(mfrow = c(2,1))
plot(cvtree$size,cvtree$dev, xlab = "Tree Size", ylab = "Deviance", main = "Tree Size Deviance")
par(op)
```




```r
# best size is 23
trntree.pruned <- prune.tree(trntree,best=23)
```

```r
predictions <- predict(trntree.pruned, trn.cl, type="vector")
for (i in 1:2456){
  d = predictions[i,7]
  predictions[i,7] <- ifelse(d < 0.5,'Hillary Clinton','Donald Trump')
}

c = list()
for (i in 1:2456){
  c[[length(c)+1]] <- predictions[i,7]
}
```

```
tree.train.error <- calc_error_rate(c, trn.cl$candidate)

predictions2 <- predict(trntree.pruned, tst.cl, type="vector")
for (i in 1:614){
  d = predictions2[i,7]
  predictions2[i,7] <- ifelse(d < 0.5,'Hillary Clinton','Donald Trump')
}

d = list()
for (i in 1:614){
  d[[length(d)+1]] <- predictions2[i,7]
}

tree.test.error <- calc_error_rate(d, tst.cl$candidate)

records[1,] <- c(tree.train.error, tree.test.error)
records
```

```
##      train.error test.error
## tree  0.04845277 0.09446254
## knn           NA         NA
## lda           NA         NA
```

Question 14

```
set.seed(20)
nfold <- 10
folds <- sample(cut(1:nrow(trn.cl), breaks=nfold, labels=FALSE))

kvec <- c(1, seq(10, 50, length.out=9))

do.chunk <- function(chunkid, folddef, Xdat, Ydat, k){
  train = (folddef!=chunkid)
  Xtr = Xdat[train,]
  Ytr = Ydat[train]
  Xvl = Xdat[!train,]
  Yvl = Ydat[!train]

  predYtr = knn(train = Xtr, test = Xtr, cl = Ytr, k = k)
  predYvl = knn(train = Xtr, test = Xvl, cl = Ytr, k = k)

  data.frame(train.error = calc_error_rate(predYtr, Ytr),
             val.error = calc_error_rate(predYvl, Yvl))
}

avg.train.error.vec <- c()
avg.test.error.vec <- c()
for(i in 1:10) {
  a = ldply(1:9,
            do.chunk,
            folddef=folds,
            Xdat= dplyr::select(trn.cl,-candidate),
```

```
            Ydat= trn.cl$candidate,
            k=kvec[i])
  avg.train.error = (a[1,1]+a[2,1]+a[3,1]+a[4,1]+a[5,1]+a[6,1]+a[7,1]+a[8,1]+a[9,1])/9
  avg.test.error = (a[1,2]+a[2,2]+a[3,2]+a[4,2]+a[5,2]+a[6,2]+a[7,2]+a[8,2]+a[9,2])/9

  avg.train.error.vec = c(avg.train.error.vec, avg.train.error)
  avg.test.error.vec = c(avg.test.error.vec, avg.test.error)
}

knn.test.error <- min(avg.test.error.vec)
knn.train.error <- avg.train.error.vec[which.min(avg.test.error.vec)]
records[2,] <- c(knn.train.error, knn.test.error)
records
```

```
##      train.error test.error
## tree  0.04845277 0.09446254
## knn   0.11400409 0.11628413
## lda           NA         NA
```

Question 15

```
pca.records <- matrix(NA, nrow=3, ncol=2)
colnames(pca.records) <- c("train.error","test.error")
rownames(pca.records) <- c("tree","knn","lda")
election.cl2 <- select(election.cl,-candidate)
election.cl2$TotalPop <- as.numeric(election.cl2$TotalPop)
election.cl2 <- scale(election.cl2)
election.cl.pca <- prcomp(election.cl2)
str(election.cl.pca)
```

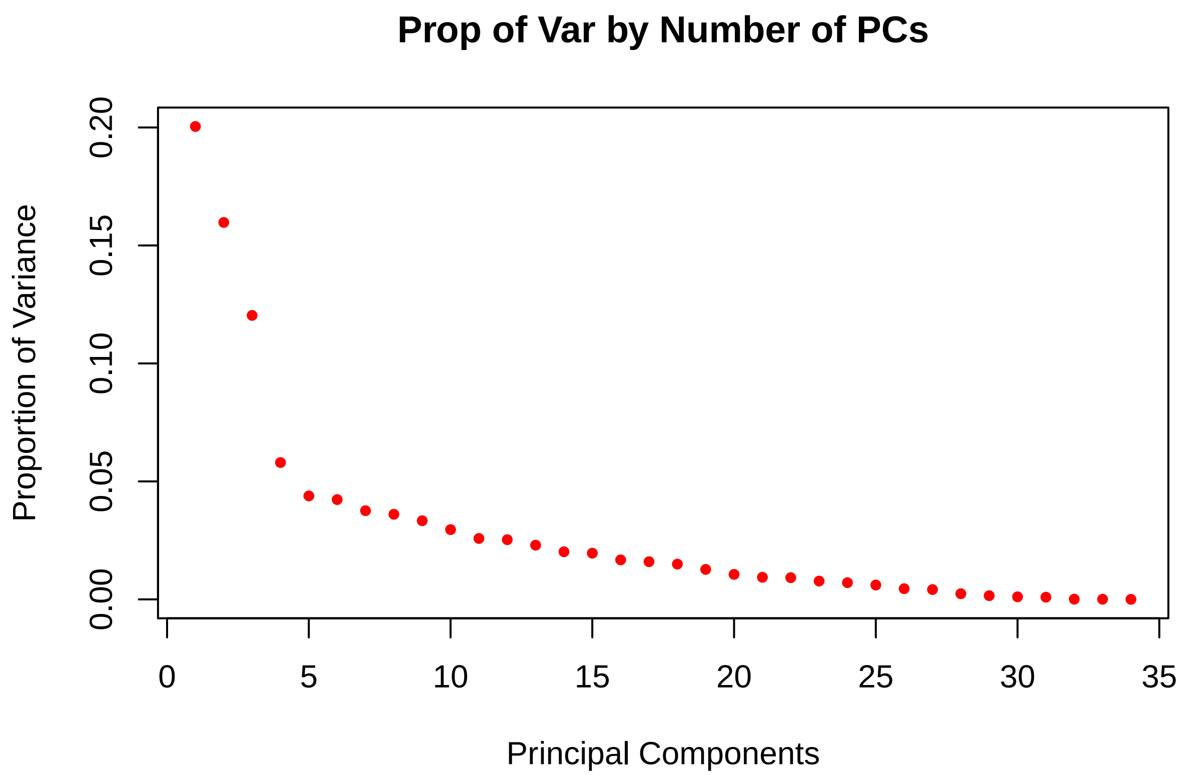
```
## List of 5
##  $ sdev    : num [1:34] 2.61 2.33 2.02 1.4 1.22 ...
##  $ rotation: num [1:34, 1:34] -0.0496 0.0261 -0.0484 -0.041 0.1841 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : chr [1:34] "TotalPop" "Men" "Women" "Hispanic" ...
##   .. ..$ : chr [1:34] "PC1" "PC2" "PC3" "PC4" ...
##  $ center  : Named num [1:34] 5.15e-17 -1.55e-15 -1.33e-16 -2.13e-17 -1.28e-18 ...
##   ..- attr(*, "names")= chr [1:34] "TotalPop" "Men" "Women" "Hispanic" ...
##  $ scale   : Named num [1:34] 1213.38 2.06 614.12 13.31 19.39 ...
##   ..- attr(*, "names")= chr [1:34] "TotalPop" "Men" "Women" "Hispanic" ...
##  $ x       : num [1:3070, 1:34] -0.535 1.924 1.682 -0.545 0.271 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : NULL
##   .. ..$ : chr [1:34] "PC1" "PC2" "PC3" "PC4" ...
##  - attr(*, "class")= chr "prcomp"
```

```
sum.ele.pca <- summary(election.cl.pca)
sum.var <- 0
for (i in 1:34){
  sum.var = sum.ele.pca$importance[2,i] + sum.var
  if (sum.var>0.9){
    print(i)
    break
```

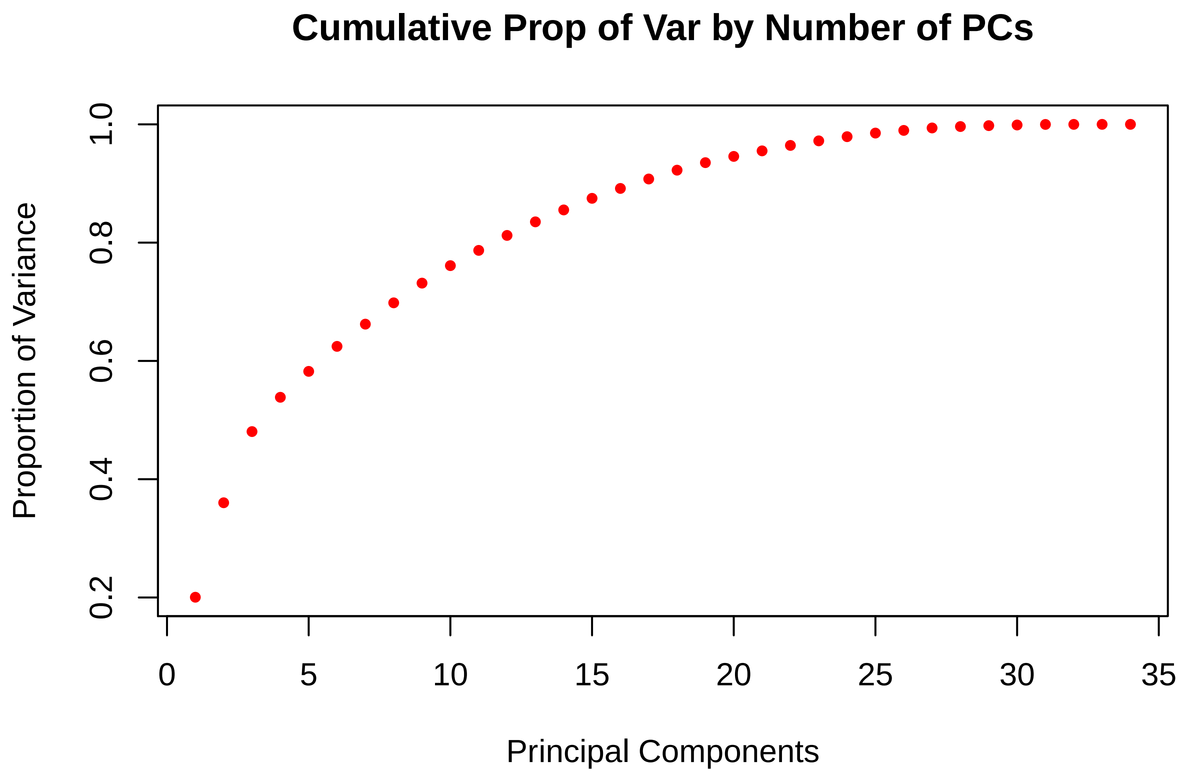
```
  }
}
```

```
## [1] 17
```

```
var.pca <- election.cl.pca$sdev^2
proportion <- var.pca/sum(var.pca)
plot(proportion,
     main = "Prop of Var by Number of PCs",
     xlab = "Principal Components",
     ylab = "Proportion of Variance",
     pch = 16,
     cex = .75,
     col='red')
```




```
Cproportion <- cumsum(var.pca)/sum(var.pca)
plot(Cproportion,
     main = "Cumulative Prop of Var by Number of PCs",
     xlab = "Principal Components",
     ylab = "Proportion of Variance",
     pch=16,
     cex = .75,
     col='red')
```



Looking at the summary of the PCA matrix we can see that the minimum number of Principal Components in order to accout for 90% of the total variance is 17.

Question 16

```r
df.election.pca.x <- as.data.frame(election.cl.pca$x)
data.pca <- cbind(election.cl$candidate, df.election.pca.x)
data.pca <- as.data.frame(data.pca)
names(data.pca)[1] <- "candidate"

set.seed(10)
n <- nrow(data.pca)
in.trn <- sample.int(n, 0.8*n)
tr.pca <- data.pca[in.trn,]
test.pca <- data.pca[-in.trn,]

head(tr.pca)
```

```
##          candidate        PC1        PC2        PC3        PC4         PC5
## 1558 Donald Trump -5.3041389 -0.8597815 -3.9675468 -3.1957783 -0.08047004
## 942  Donald Trump -0.9662062  0.5149412  0.9165879  1.3590921  0.31156203
## 1310 Donald Trump  1.1819360  0.6409613  2.6441019 -1.5220651 -0.15204084
## 2126 Donald Trump -2.1549878  3.6317331 -8.4591013  1.7221043  1.02262677
## 262  Donald Trump -2.1833761  0.3852831 -0.5765413  1.0240136 -0.86382767
## 691  Donald Trump  1.1790534  0.2997215  1.3729761  0.4051871  0.63029585
```

```
##              PC6        PC7       PC8        PC9       PC10        PC11
## 1558 -0.2419150 -0.3030275  1.115791  0.7736474  1.0817168 -0.39551352
## 942  -0.3275572  0.1044763  2.039305  0.1564399  0.9356193 -0.22259964
## 1310  0.9224057 -0.3148161 -0.471737 -0.2689322  0.2316760  0.53176979
## 2126 -1.9265475 -3.6522067 -3.339325  0.2123036  2.3671493  4.48949413
## 262   0.2127275  0.2271466  0.120101  0.5118856 -0.6627565  0.89293295
## 691  -1.1040825  1.0835762 -1.094231 -0.4025593 -0.1493833 -0.01278583
##             PC12       PC13        PC14       PC15        PC16       PC17
## 1558 -1.77444240  0.8582194 -0.70865405 -1.0232428  0.40936073 -0.4087346
## 942  -0.73864743  0.3383973 -0.07839124  1.0049390 -0.50942075 -0.2598862
## 1310 -0.23294167  0.6923908  0.42935992  0.6265949 -0.18616729 -0.3350760
## 2126 -0.00855891 -1.6603832  1.76443313 -0.1192481 -0.71515157 -0.7027052
## 262  -0.99548418  1.7350957  0.07506511  0.5497652 -0.03162557  0.3862599
## 691  -0.76271657  0.3433607  0.17632125 -1.0376911  0.80906514  0.3186242
##              PC18         PC19       PC20       PC21        PC22
## 1558  0.002812804 -1.664468408 -0.7154521 -0.3564166  0.48047263
## 942  -0.368365808 -0.164644357 -0.1666981 -0.3187658 -0.23004946
## 1310  0.055973856 -0.003147345 -0.1458947  0.1047890  0.08386894
## 2126 -0.136925736  0.786282585  0.2018207  0.8106516  0.15414294
## 262  -0.547978292  0.166219594 -0.3907554 -0.4438206  0.07119740
## 691   0.011660411  0.914789105 -1.1496228  0.4823506  0.45302731
##             PC23         PC24       PC25        PC26        PC27
## 1558 -0.41503309  0.263220433  0.2602436 -0.05394346  0.21312365
## 942   0.08747367 -0.009176133  0.1261544  0.36653039  0.07260519
## 1310 -0.16532222  0.077449451 -0.2906918 -0.28797890  0.10535407
## 2126  0.14626976 -0.485658615  0.5781911 -0.15602897 -0.22124949
## 262  -0.07536905 -0.186171309 -0.8846307 -1.04682961 -0.02571918
## 691  -0.02006442 -0.569668153 -0.3606024  0.26014074  0.74593694
##             PC28         PC29        PC30        PC31        PC32
## 1558  0.00285480  0.001691172 -0.33878030 -0.17564180  0.04216274
## 942   0.11781239 -0.017207230  0.07759823 -0.02462265  0.01164107
## 1310 -0.10839219  0.058813920 -0.06422602  0.28790264  0.02831389
## 2126  0.42390910 -0.175789454  0.05821555  0.83455592 -0.35894525
## 262   0.49980727  0.252205358  0.06805872 -0.06007699  0.04862443
## 691  -0.07562252 -0.247336258 -0.19282318 -0.10559412  0.01556989
##               PC33          PC34
## 1558  1.224719e-02 -1.729597e-15
## 942   2.264389e-03  1.254097e-17
## 1310  6.987968e-04 -3.072648e-16
## 2126 -1.042742e-01  3.833156e-15
## 262  -6.063129e-03  8.298596e-16
## 691  -7.313651e-05 -3.858342e-16
```

```r
head(test.pca)
```

```
##          candidate       PC1        PC2        PC3        PC4         PC5
## 1     Donald Trump -0.534608 -2.3383326  0.2488554  0.4221960 -0.01942941
## 8     Donald Trump -1.482219 -1.6541146  2.7819220 -1.1469909  1.39109379
## 24    Donald Trump -1.259255  0.7768294  0.9297121  0.1162212 -1.60271706
## 34    Donald Trump -2.193477  0.3694519  0.8698010  0.3009422  0.78962702
## 35 Hillary Clinton -9.220720 -0.6106496 -2.2871919  3.2130451 -2.37419723
## 37    Donald Trump -1.657118  0.5538342  0.4970016  0.6161101 -1.42656026
##           PC6        PC7         PC8        PC9        PC10         PC11
## 1  -1.0554443 -0.4119244 -0.44915407 -0.1309234 -0.36891586 -0.365332423
## 8  -0.7317522  0.7456810 -1.84320054 -0.5888855  0.03107667 -0.483867285
```

```
## 24  1.3129312  0.3974195 -1.12809833 -1.1501539 -0.60363483  1.047091176
## 34  0.2449074  0.2819398 -0.57793586  0.3376414 -0.38067382  0.004649763
## 35  0.8040246  0.4303297 -1.28042346 -0.3190843 -1.95498817 -0.208457566
## 37  0.8700131  0.2550706 -0.05052385 -0.1925798 -0.56565791  0.309364315
##           PC12       PC13         PC14       PC15       PC16       PC17
## 1   0.01837778  0.5065415 -0.670332846  0.6026566  0.3507514  0.4045562
## 8  -0.43760553 -0.6952835 -0.006475763 -0.5209882  0.3940848 -0.2780899
## 24 -0.37358661 -0.1730428  1.524862626  0.7606203 -0.1111132 -0.1194390
## 34 -0.45302687 -0.5387033 -0.140455059 -0.4545016 -0.4091450 -0.1558792
## 35  0.54185562  0.8891842  1.016156580  0.6622763  0.5864148 -0.1791978
## 37  1.10918134 -0.3378709 -0.123521692  0.2876865  0.2889031  1.1630388
##           PC18        PC19        PC20         PC21       PC22
## 1  -0.04984370 -0.01705522  0.03733451 -0.188834103 -0.3771647
## 8  -0.51886480  0.01518771  0.80496933  0.026239732  0.1134704
## 24 -0.17209014 -0.41639130 -0.33008711 -0.009002311 -0.7544363
## 34 -0.21327084 -0.08703761  0.16812438  0.167856222  0.1345852
## 35  1.01201319 -0.68062543 -0.24100951  0.324914722  0.9032762
## 37 -0.02347463 -0.21575240  0.06280952  0.435414955 -0.1232914
##            PC23        PC24       PC25        PC26         PC27
## 1   0.004485094  0.05360134 -0.4982141  0.32723724 -0.174026207
## 8  -0.305563425  0.48877267 -0.2187110 -0.25112894  0.004143709
## 24 -0.175981569  0.33041991 -0.5829753  0.19211016  0.577306596
## 34  0.387025135 -0.05823013  0.5411182 -0.21239206  0.047183145
## 35  0.542351239 -0.45664442  0.8461351 -0.01638456 -0.244888291
## 37 -0.440186732  0.11990347 -0.3963533 -0.48818492  0.085524028
##           PC28        PC29        PC30        PC31         PC32
## 1  -0.18401304  0.22742544  0.08696954 -0.02140527 -0.066203625
## 8   0.12607318  0.13438655  0.12600460 -0.16445072  0.001628442
## 24  0.05605443 -0.04894544 -0.09708482 -0.19245413 -0.017649050
## 34 -0.73141723  0.37489333 -0.08352975  0.08256474 -0.004104669
## 35 -0.09832920  0.01925777  0.42139174 -0.08306825  0.013255520
## 37  0.28956941  0.22907976 -0.00322449 -0.07698337  0.016311475
##           PC33          PC34
## 1   0.03416442  1.350302e-15
## 8   0.03108867 -1.306638e-15
## 24 -0.03324883  9.351918e-16
## 34  0.01243862 -2.030878e-16
## 35  0.01951005  5.714843e-15
## 37  0.03460244  1.490226e-15
```

Some Setups

```r
set.seed(20)
nfold <- 10
folds.pca <- sample(cut(1:nrow(trn.cl), breaks=nfold, labels=FALSE))
calc_error_rate <- function(predicted.value, true.value){
  return(mean(true.value!=predicted.value))
}
records.pca <- matrix(NA, nrow=3, ncol=2)
colnames(records.pca) = c("train.error.pca","test.error.pca")
rownames(records.pca) = c("tree","knn","glm")
```

Question 17

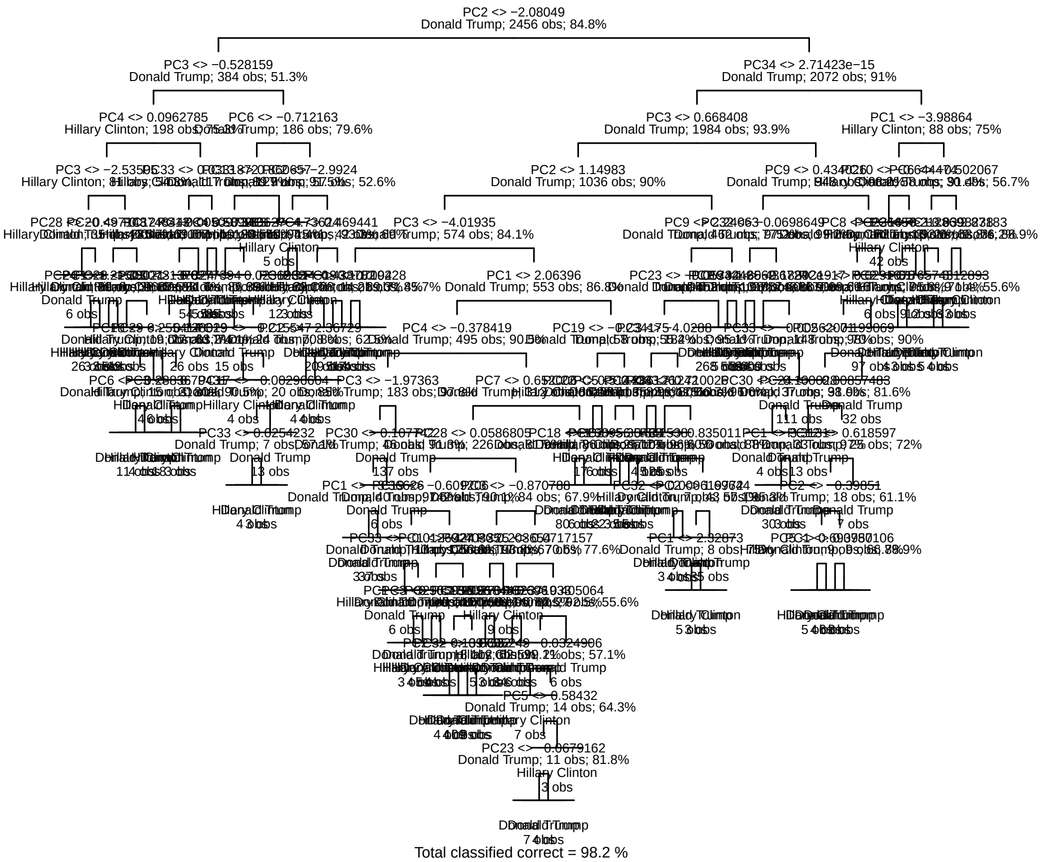
```
tree.train.pca <- tree(candidate~., data=tr.pca)
tree.train.control.pca <- tree.control(nobs=3070, minsize = 7, mindev = 1e-6)
tree.train.pca <- tree(candidate~., data = tr.pca, control = tree.train.control.pca)
summary(tree.train.pca)
```

```
##
## Classification tree:
## tree(formula = candidate ~ ., data = tr.pca, control = tree.train.control.pca)
## Variables actually used in tree construction:
##  [1] "PC2"  "PC3"  "PC4"  "PC28" "PC24" "PC20" "PC1"  "PC18" "PC6"  "PC33"
## [11] "PC12" "PC30" "PC29" "PC13" "PC17" "PC14" "PC34" "PC19" "PC7"  "PC32"
## [21] "PC25" "PC27" "PC5"  "PC23" "PC22" "PC9"  "PC8"  "PC10" "PC16"
## Number of terminal nodes:  99
## Residual mean deviance:  0.07418 = 174.8 / 2357
## Misclassification error rate: 0.01832 = 45 / 2456
```

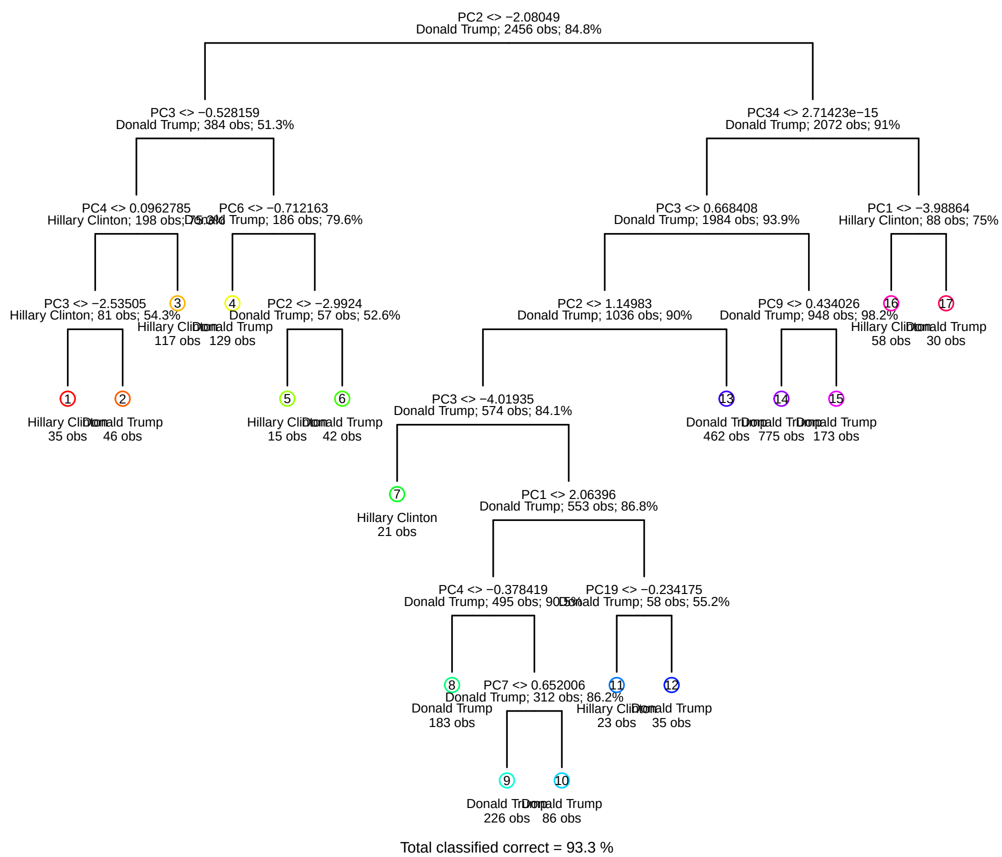
```
cv.tree.pca <- cv.tree(tree.train.pca, rand=folds, prune.misclass)
cv.tree.pca
```

```
## $size
##  [1] 99 81 79 73 59 55 49 43 35 23 19 17 15  8  6  4  3  1
##
## $dev
##  [1] 253 249 244 244 237 237 238 238 233 233 233 233 234 235 236 250 317
## [18] 374
##
## $k
##  [1]       -Inf  0.0000000  0.5000000  0.8333333  1.0000000  1.2500000
##  [7]  1.3333333  1.5000000  2.0000000  2.2500000  2.5000000  3.0000000
## [13]  5.5000000  6.0000000  6.5000000  9.0000000 44.0000000 50.0000000
##
## $method
## [1] "misclass"
##
## attr(,"class")
## [1] "prune"         "tree.sequence"
```

```
draw.tree(tree.train.pca, nodeinfo=TRUE, cex=.5, size = 0)
```




```r
prunedtree.pca <- prune.tree(tree.train.pca, best=17)
draw.tree(prunedtree.pca, cex=0.5, nodeinfo=TRUE)
```




```
prediction.pca <- predict(prunedtree.pca, tr.pca, type="vector")
for (i in 1:nrow(tr.pca)){
  d = prediction.pca[i,7]
  prediction.pca[i,7] <- ifelse(d < 0.5,'Hillary Clinton','Donald Trump')
}

c = list()
for (i in 1:nrow(tr.pca)){
  c[[length(c)+1]] <- prediction.pca[i,7]
}
tree.train.error <- calc_error_rate(c, tr.pca$candidate)

prediction.pca2 <- predict(prunedtree.pca, test.pca, type="vector")
for (i in 1:nrow(test.pca)){
```

```
  d = prediction.pca2[i,7]
  prediction.pca2[i,7] <- ifelse(d < 0.5,'Hillary Clinton','Donald Trump')
}

d = list()
for (i in 1:nrow(test.pca)){
  d[[length(d)+1]] <- prediction.pca2[i,7]
}
tree.test.error <- calc_error_rate(d, test.pca$candidate)

records.pca[1,] <- c(tree.train.error, tree.test.error)
records.pca
```

```
##      train.error.pca test.error.pca
## tree      0.06718241     0.07980456
## knn               NA             NA
## glm               NA             NA
```

Question 18

```
set.seed(20)
nfold <- 10
folds.pca <- sample(cut(1:nrow(trn.cl), breaks=nfold, labels=FALSE))

kvec <- c(1, seq(10, 50, length.out=9))

do.chunk <- function(chunkid, folddef, Xdat, Ydat, k){
  train = (folddef!=chunkid)
  Xtr = Xdat[train,]
  Ytr = Ydat[train]
  Xvl = Xdat[!train,]
  Yvl = Ydat[!train]

  predYtr = knn(train = Xtr, test = Xtr, cl = Ytr, k = k)
  predYvl = knn(train = Xtr, test = Xvl, cl = Ytr, k = k)

  data.frame(train.error = calc_error_rate(predYtr, Ytr),
             val.error = calc_error_rate(predYvl, Yvl))
}

avg.train.error.pca.vec <- c()
avg.test.error.pca.vec <- c()
for(i in 1:10) {
  a = ldply(1:9,
            do.chunk,
            folddef=folds,
            Xdat = dplyr::select(tr.pca,-candidate),
            Ydat = tr.pca$candidate,
            k=kvec[i])
  avg.train.error.pca = (a[1,1]+a[2,1]+a[3,1]+a[4,1]+a[5,1]+a[6,1]+a[7,1]+a[8,1]+a[9,1])/9
  avg.test.error.pca = (a[1,2]+a[2,2]+a[3,2]+a[4,2]+a[5,2]+a[6,2]+a[7,2]+a[8,2]+a[9,2])/9
```

```
  avg.train.error.pca.vec = c(avg.train.error.pca.vec, avg.train.error.pca)
  avg.test.error.pca.vec = c(avg.test.error.pca.vec, avg.test.error.pca)
}

knn.test.error.pca <- min(avg.test.error.pca.vec)
knn.train.error.pca <- avg.train.error.pca.vec[which.min(avg.test.error.pca.vec)]

records.pca[2,] <- c(knn.train.error.pca, knn.test.error.pca)
records.pca
```

```
##      train.error.pca test.error.pca
## tree      0.06718241     0.07980456
## knn       0.06660318     0.07512674
## glm               NA             NA
```

Question 19

If our goal is to predict the election, we could use time series analysis to forecast the election results. In the dataset we used, the data was collected after the election was done. Typically, prediction of elections and forecasting is done using multiple data points over set periods of time. Many pollsters and data analysts interested in forecasting use many different polls and sampling techniques to model voter behavior. For example, a certain county may come out with a poll predicting voter behavior in their county, depending on how reliable it is we can add it to a model. If we had the resources to get data on polls leading up to the election we could build models on voter behavior relating to other statistics such as approval ratings, how well the economy is doing, stances on big issues, etc.

Although it would be practically difficult to acquire the data, knowing voter preferences based on demographics could lead to more interesting analysis knowing the actual results. For example, if we had more data on race and preferred candidate, we could figure out how much of each demographic voted for Hillary and how much of each demographic voted for Trump with more accuracy. We could figure out how much of each demographic is worth in terms of electoral votes based on demographic size and voter turnout rate per demographic. Using our techniques we can infer voter preference, but it is hard to determine total votes based on a specific demographic. It is difficult to find datasets online in regards to voter demographics when connecting them to actual behavior. If we had our own resources then exit polls and surveys would likely be the best way to connect demographics with voter behavior. Having more detailed data connecting demographics directly to voter beahvior could lead to better model testing.

Question 20

Additional Classification Methods: Logistic regression

In this question, we are going to use logistic regression method for testing the training error and test error with the original dataset first, and then use the datasets containing PCs to compare the errors.

```
train.glm.data <- trn.cl

train.glm.data[,1] <- as.character(train.glm.data$candidate)
train.glm.data[which(train.glm.data$candidate=="Donald Trump"),1] <- 1
train.glm.data[which(train.glm.data$candidate=="Hillary Clinton"),1] <- 0
train.glm.data[,1] <- as.numeric(train.glm.data$candidate)
train.glm.data[,2] <- as.numeric(train.glm.data$TotalPop)

model <- glm(candidate~., data=train.glm.data, family=binomial)
```

```r
pred.train <- predict(model, data=train.glm.data, type="response")

pred.train <- ifelse(pred.train > 0.5, 1, 0)

train.error.glm <- calc_error_rate(pred.train, train.glm.data$candidate)
train.error.glm
```

```
## [1] 0.07288274
```

First of all, we get our training error for logistic regression which is approximately 7%.

```r
test.glm.data <- tst.cl

test.glm.data[,1] <- as.character(test.glm.data$candidate)
test.glm.data[which(test.glm.data$candidate=="Donald Trump"),1] <- 1
test.glm.data[which(test.glm.data$candidate=="Hillary Clinton"),1] <- 0
test.glm.data[,1] <- as.numeric(test.glm.data$candidate)
test.glm.data[,2] <- as.numeric(test.glm.data$TotalPop)

pred.test <- predict(model, data=test.glm.data, type="response")

pred.test <- ifelse(pred.test > 0.5, 1, 0)

test.error.glm <- calc_error_rate(pred.test, test.glm.data$candidate)
test.error.glm
```

```
## [1] 0.2390065
```

Then, we got our test error for logistic regression which is approximately 24%. Below is the completed records matrix.

```r
records[3,] <- c(train.error.glm, test.error.glm)
records
```

```
##      train.error test.error
## tree  0.04845277 0.09446254
## knn   0.11400409 0.11628413
## lda   0.07288274 0.23900651
```

Next, we want to find the training and test error with full principle components.

```r
train.glm.pca <- tr.pca

train.glm.pca[,1] <- as.character(train.glm.pca[,1])
train.glm.pca[which(train.glm.pca$candidate=="Donald Trump"),1] <- 1
train.glm.pca[which(train.glm.pca$candidate=="Hillary Clinton"),1] <- 0
train.glm.pca[,1] <- as.numeric(train.glm.pca$candidate)

model.pca <- glm(candidate~., data=train.glm.pca, family=binomial)
pred.train.pca <- predict(model.pca, data=train.glm.pca, type="response")
pred.train.pca <- ifelse(pred.train.pca > 0.5, 1, 0)

train.error.glm.pca <- calc_error_rate(pred.train.pca, train.glm.pca$candidate)
train.error.glm.pca
```

```
## [1] 0.07288274
```

```r
test.glm.pca <- test.pca

test.glm.pca[,1] <- as.character(test.glm.pca[,1])
test.glm.pca[which(test.glm.pca$candidate=="Donald Trump"),1] <- 1
test.glm.pca[which(test.glm.pca$candidate=="Hillary Clinton"),1] <- 0
test.glm.pca[,1] <- as.numeric(test.glm.pca$candidate)

pred.test.pca <- predict(model.pca, data=test.glm.pca, type="response")
pred.test.pca <- ifelse(pred.train.pca > 0.5, 1, 0)

test.error.glm.pca <- calc_error_rate(pred.test.pca, test.glm.pca$candidate)
test.error.glm.pca
```

```
## [1] 0.2390065
```

According to our result, we get the same training error and test error as we use the original dataset, because we are using the full PCs, we have captured 100% of the variance. Below is the completed records.pca matrix.

```r
records.pca[3,] <- c(train.error.glm.pca, test.error.glm.pca)
records.pca
```

```
##      train.error.pca test.error.pca
## tree      0.06718241     0.07980456
## knn       0.06660318     0.07512674
## glm       0.07288274     0.23900651
```