PSTAT 131 - 2016 Election Prediction

Jaeyun Lee (7916406) and Wentian Sun(4950614)

5/21/2019

Predicting voter behavior is complicated for many reasons despite the tremendous effort in collecting, analyzing, and understanding many available datasets. For our final project, we will analyze the 2016 presidential election dataset, but, first, some background.

Background

The presidential election in 2012 did not come as a surprise. Some correctly predicted the outcome of the election correctly including Nate Silver, and many speculated his approach.

Despite the success in 2012, the 2016 presidential election came as a big surprise to many, and it was a clear example that even the current state-of-the-art technology can surprise us.

Answer the following questions in one paragraph for each.

1. What makes voter behavior prediction (and thus election forecasting) a hard problem? The voter behavior prediction is a hard problem because election is affected by the voter's demographic, beliefs, education, gender, economic standing. In addition, not all Americans vote. The data that is collected could potentially be biased because not all of voters are willing to share who they voted. Also, note that American voting system is complicated: Popular votes and Electoral votes for the presidential election. With the given data, there are questions that can be answer and cannot be answered. It is also a hard problem because the credibility is questionable.

2. What was unique to Nate Silver's approach in 2012 that allowed him to achieve good predictions? Nate Silver's approach is based on the Bayes' Theorem. His method accounts the full scope of possibilities and considers baseline factors such as range of different dates, shift in support from one candidate to another. His approach of considering important variables that affect voter's decision allowed him to achieve good predictions.

3. What went wrong in 2016? What do you think should be done to make future predictions better? The 2016 election was controversial as media focused on two leading candidates Hillary and Trump. Their radical ideas and reformity and their promises to the public renewed public's support in U.S. government. The media's focus on FBI investigation on Hillary, Russias involvement, Trump's unethical behaviors, and many more controversial topics are televised. This caused many voters to change their view on candidates which led to unpredicatable voter shifts. The data we have is based on the poll and the sampling error is a big issue. To make future predictions better, we should utilize more variables that show how voters feel about candidates in correspondance to their social status. In addition, treat time as sensitive as voters perspective on candidates change over time.

Data

```r
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)
```

Election data

Following is the first few rows of the `election.raw` data:

county	fips	candidate	state	votes
NA	US	Donald Trump	US	62984825
NA	US	Hillary Clinton	US	65853516
NA	US	Gary Johnson	US	4489221
NA	US	Jill Stein	US	1429596
NA	US	Evan McMullin	US	510002
NA	US	Darrell Castle	US	186545

The meaning of each column in `election.raw` is clear except `fips`. The accronym is short for Federal Information Processing Standard.

In our dataset, `fips` values denote the area (US, state, or county) that each row of data represent: i.e., some rows in `election.raw` are summary rows. These rows have `county` value of `NA`. There are two kinds of summary rows:

- Federal-level summary rows have `fips` value of `US`.
- State-level summary rows have names of each states as `fips` value.

Census data

Following is the first few rows of the `census` data:

CensusTract	State	County	TotalPop	Men	Women	Hispanic	White	Black	Native	Asian	Pacific	C
1001020100	Alabama	Autauga	1948	940	1008	0.9	87.4	7.7	0.3	0.6	0.0	
1001020200	Alabama	Autauga	2156	1059	1097	0.8	40.4	53.3	0.0	2.3	0.0	
1001020300	Alabama	Autauga	2968	1364	1604	0.0	74.5	18.6	0.5	1.4	0.3	
1001020400	Alabama	Autauga	4423	2172	2251	10.5	82.8	3.7	1.6	0.0	0.0	
1001020500	Alabama	Autauga	10763	4922	5841	0.7	68.5	24.8	0.0	3.8	0.0	
1001020600	Alabama	Autauga	3851	1787	2064	13.1	72.9	11.9	0.0	0.0	0.0	

Census data: column metadata

Column information is given in `metadata`.

CensusTract	Census.tract.ID	numeric
State	State, DC, or Puerto Rico	string
County	County or county equivalent	string
TotalPop	Total population	numeric
Men	Number of men	numeric
Women	Number of women	numeric
Hispanic	% of population that is Hispanic/Latino	numeric
White	% of population that is white	numeric
Black	% of population that is black	numeric
Native	% of population that is Native American or Native Alaskan	numeric
Asian	% of population that is Asian	numeric
Pacific	% of population that is Native Hawaiian or Pacific Islander	numeric

CensusTract	Census.tract.ID	numeric
Citizen	Number of citizens	numeric
Income	Median household income ($)	numeric
IncomeErr	Median household income error ($)	numeric
IncomePerCap	Income per capita ($)	numeric
IncomePerCapErr	Income per capita error ($)	numeric
Poverty	% under poverty level	numeric
ChildPoverty	% of children under poverty level	numeric
Professional	% employed in management, business, science, and arts	numeric
Service	% employed in service jobs	numeric
Office	% employed in sales and office jobs	numeric
Construction	% employed in natural resources, construction, and maintenance	numeric
Production	% employed in production, transportation, and material movement	numeric
Drive	% commuting alone in a car, van, or truck	numeric
Carpool	% carpooling in a car, van, or truck	numeric
Transit	% commuting on public transportation	numeric
Walk	% walking to work	numeric
OtherTransp	% commuting via other means	numeric
WorkAtHome	% working at home	numeric
MeanCommute	Mean commute time (minutes)	numeric
Employed	% employed (16+)	numeric
PrivateWork	% employed in private industry	numeric
PublicWork	% employed in public jobs	numeric
SelfEmployed	% self-employed	numeric
FamilyWork	% in unpaid family work	numeric
Unemployment	% unemployed	numeric

Data wrangling

4. Remove summary rows from `election.raw` data: i.e.,

   - Federal-level summary into a `election_federal`.
   - State-level summary into a `election_state`.
   - Only county-level data is to be in `election`.

```
election_federal <- filter(election.raw, is.na(county) & fips=="US")
election_state <- filter(election.raw, is.na(county) & election.raw$fips != "US" &
                            as.character(election.raw$fips) == as.character(election.raw$state))
election <- filter(election.raw, election.raw$fips != "US" & as.character(election.raw$fips) != as.chara
```

5. How many named presidential candidates were there in the 2016 election? Draw a bar chart of all votes received by each candidate

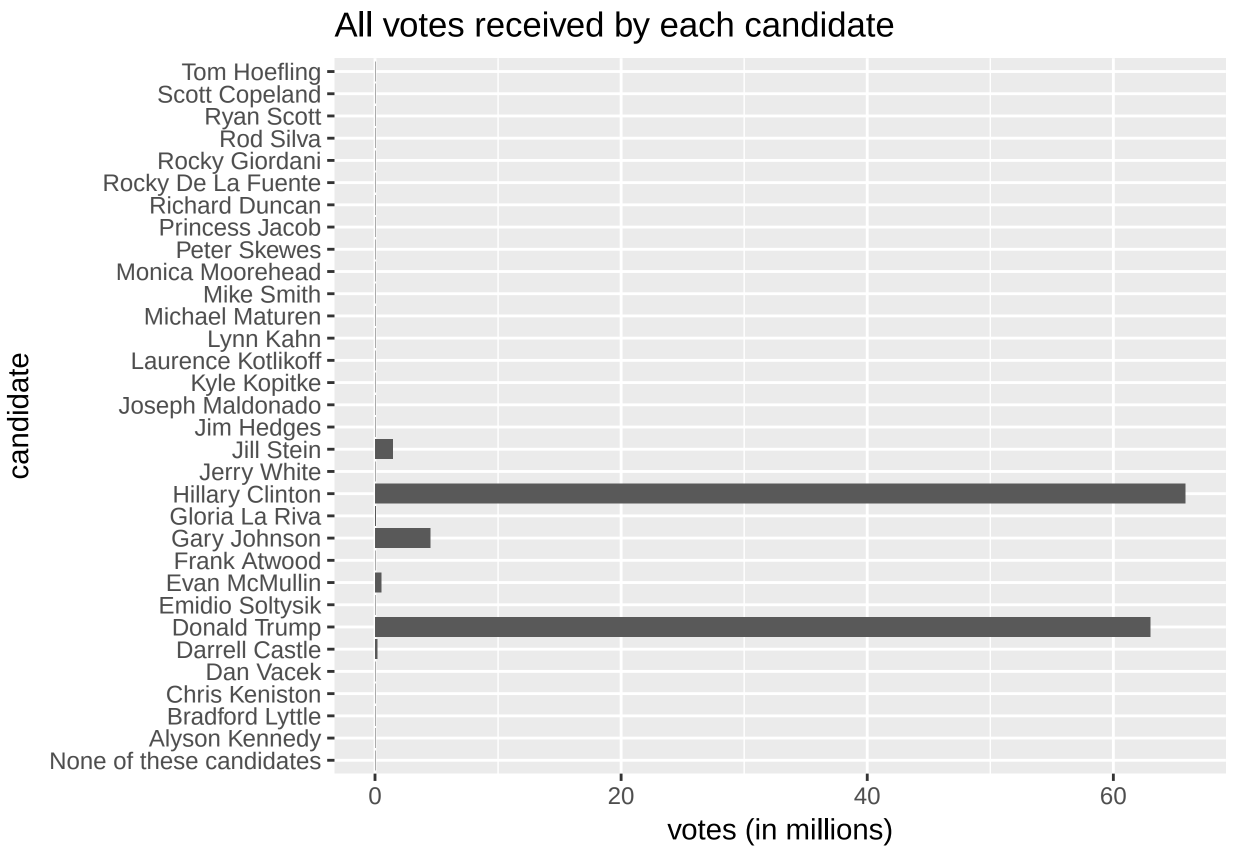
```
dim(election_federal)[1]
```

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

```
named_president <- unique(election$candidate)
num_named <- length(named_president)
named_president
```

```
##  [1] Hillary Clinton          Donald Trump
##  [3] Gary Johnson             Jill Stein
##  [5] Gloria La Riva           Darrell Castle
##  [7] Alyson Kennedy           Rocky De La Fuente
##  [9] Emidio Soltysik           None of these candidates
## [11] Evan McMullin            Dan Vacek
## [13] Richard Duncan           Monica Moorehead
## [15] Rocky Giordani           Mike Smith
## [17] Chris Keniston           Michael Maturen
## [19] Kyle Kopitke             Joseph Maldonado
## [21] Rod Silva                Tom Hoefling
## [23] Ryan Scott               Frank Atwood
## [25] Bradford Lyttle          Laurence Kotlikoff
## [27] Jim Hedges               Lynn Kahn
## [29] Peter Skewes             Princess Jacob
## [31] Jerry White              Scott Copeland
## 32 Levels:  None of these candidates Alyson Kennedy ... Tom Hoefling
```

```
ggplot(data = election_federal, aes(x=candidate,y = votes/1000000)) +
  geom_bar(stat="identity") +
  scale_y_continuous() +
  ylab("votes (in millions)") +
  ggtitle("All votes received by each candidate") +
  coord_flip()
```



```
## change plot colors, re-order syntax
```

6. Create variables `county_winner` and `state_winner` by taking the candidate with the highest proportion of votes. Hint: to create `county_winner`, start with `election`, group by `fips`, compute `total` votes, and `pct = votes/total`. Then choose the highest row using `top_n` (variable `state_winner` is similar).

```r
county_winner <- election %>%
  group_by(fips) %>%
  mutate(total = sum(votes),pct = votes/total) %>%
  top_n(1)
```

```
## Selecting by pct
```

```r
state_winner <- election_state %>%
  group_by(fips) %>%
  mutate(total = sum(votes),pct = votes/total) %>%
  top_n(1)
```

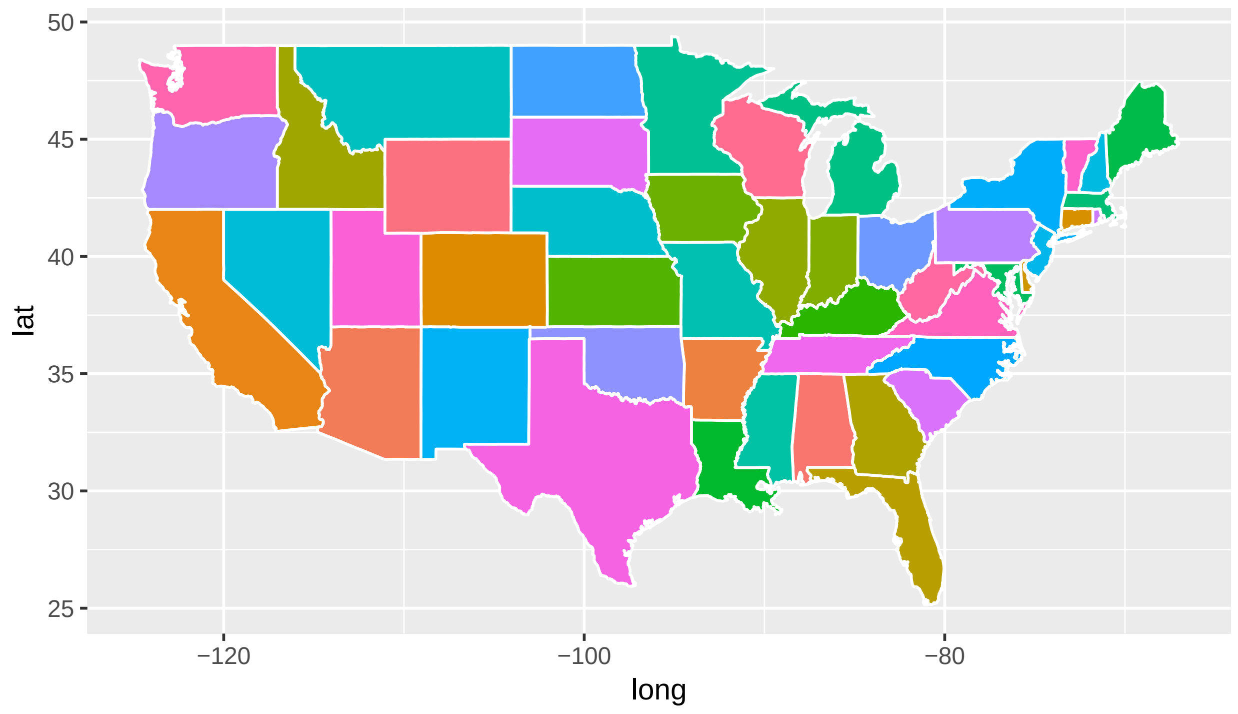
```
## Selecting by pct
```

Visualization

Visualization is crucial for gaining insight and intuition during data mining. We will map our data onto maps.

The R package `ggplot2` can be used to draw maps. Consider the following code.

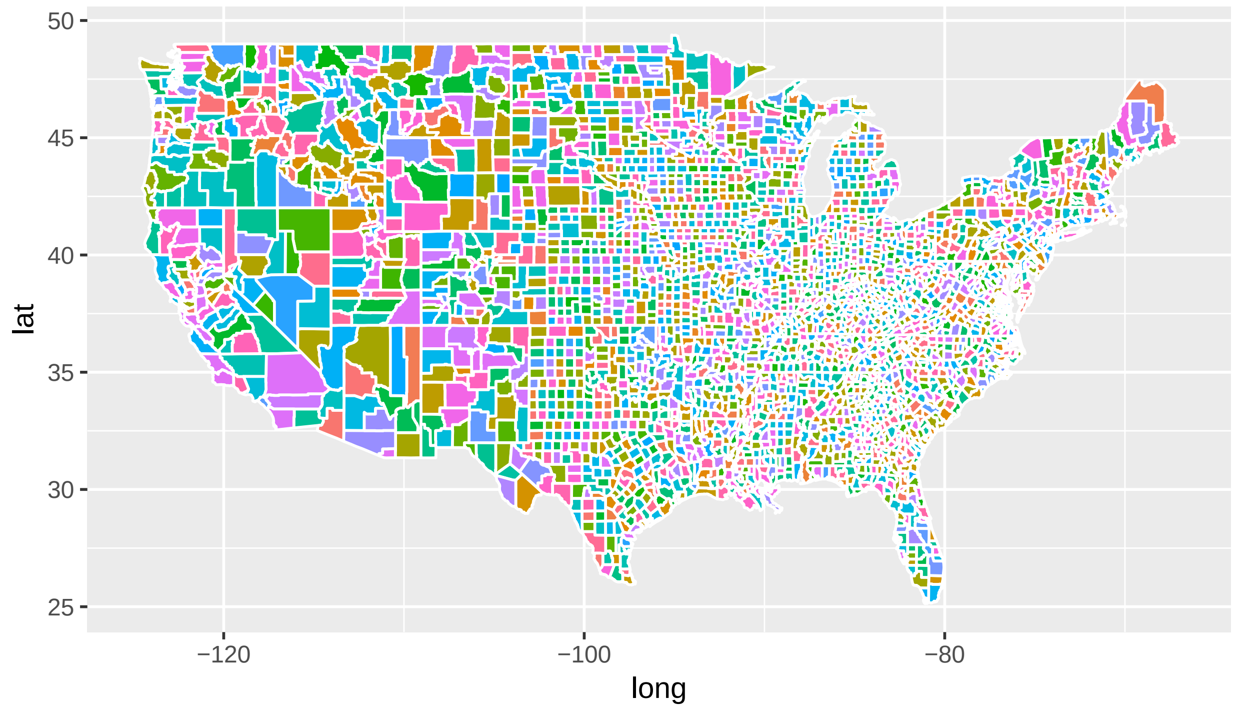
```r
states = map_data("state")

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



The variable `states` contain information to draw white polygons, and fill-colors are determined by `region`.

7. Draw county-level map by creating `counties = map_data("county")`. Color by county

```
counties = map_data("county")

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

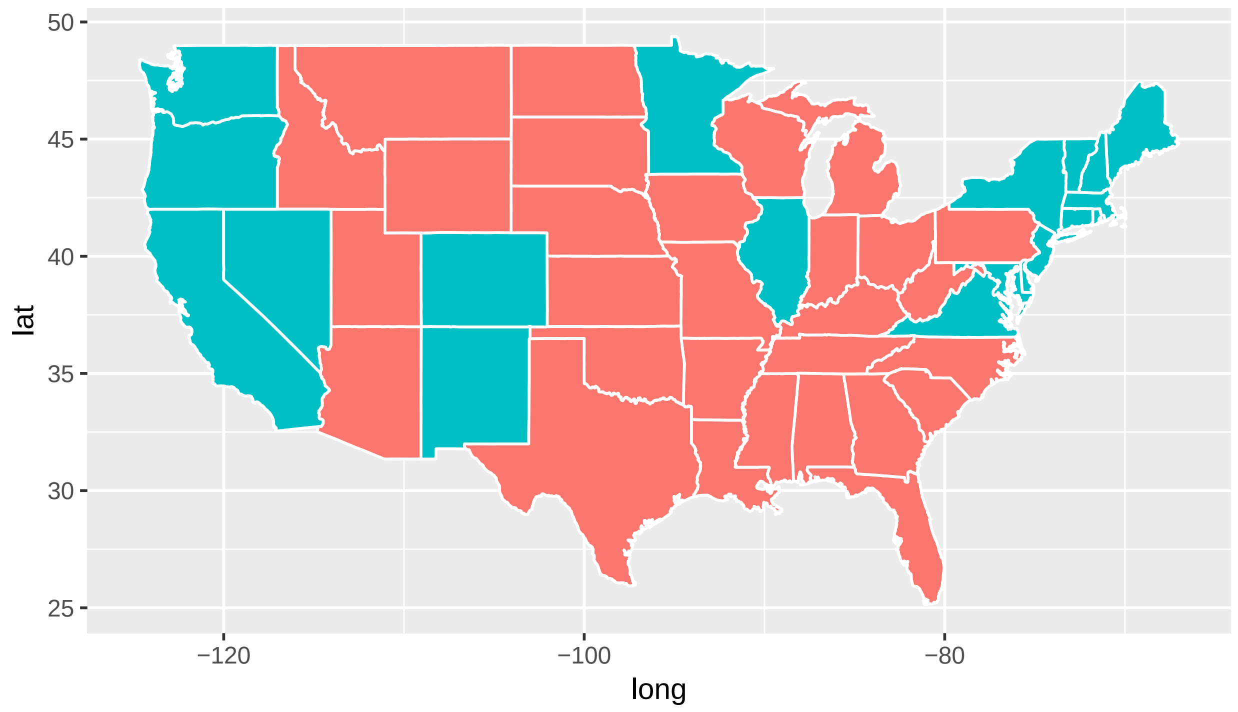


8. Now color the map by the winning candidate for each state. First, combine `states` variable and `state_winner` we created earlier using `left_join()`. Note that `left_join()` needs to match up values of states to join the tables; however, they are in different formats: e.g. `AZ` vs. `arizona`. Before using `left_join()`, create a common column by creating a new column for `states` named `fips = state.abb[match(some_column, some_function(state.name))]`. Replace `some_column` and `some_function` to complete creation of this new column. Then `left_join()`. Your figure will look similar to state_level New York Times map.

```r
# new column for states named fips
fips = state.abb[match(states$region, tolower(state.name))]
states <- states %>% mutate(fips=fips)
# match up values of states to join the tables, combine state_winner with states
combined_states <- left_join(states, state_winner, by="fips")
```

```
## Warning: Column `fips` joining character vector and factor, coercing into
## character vector
```

```r
# map by the winning candidate for each state
ggplot(data = combined_states) +
  geom_polygon(aes(x = long, y = lat, fill = candidate, group = group), color = "white") +
  coord_fixed(1.3) +
  guides(fill=FALSE)
```



9. The variable `county` does not have `fips` column. So we will create one by pooling information from `maps::county.fips`. Split the `polyname` column to `region` and `subregion`. Use `left_join()` combine `county.fips` into `county`. Also, `left_join()` previously created variable `county_winner`. Your figure will look similar to county-level New York Times map.

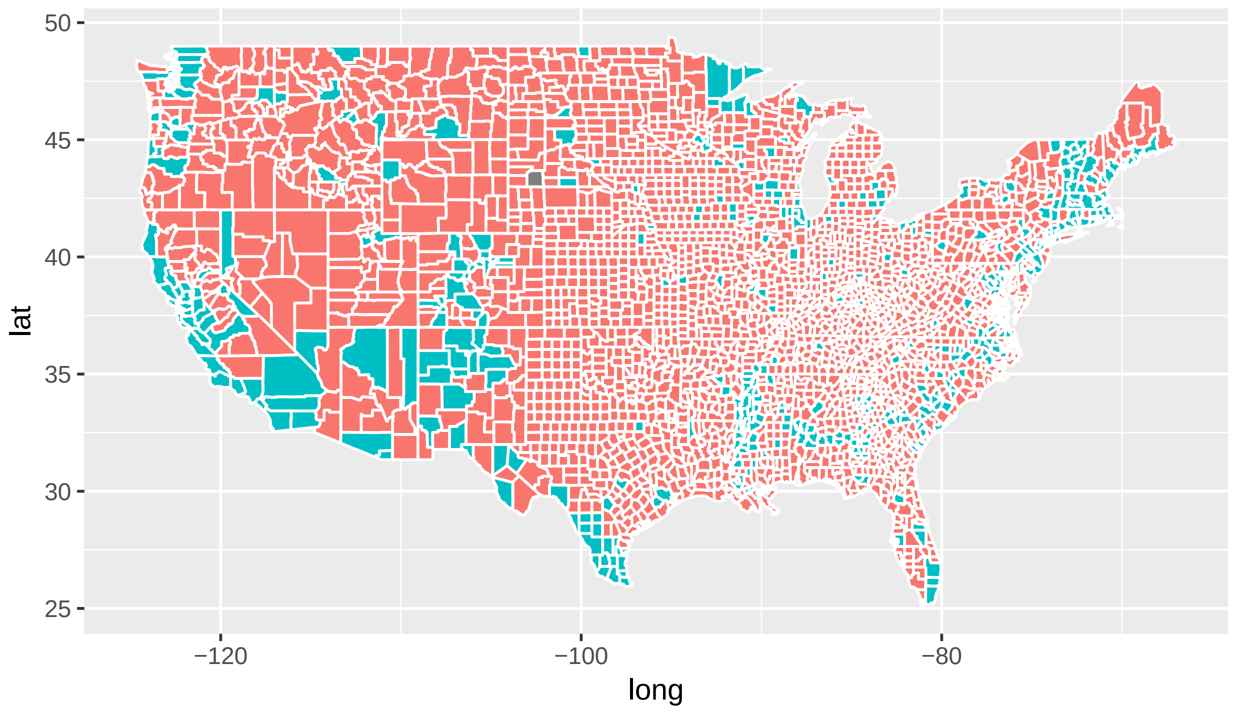
```
# split polyname column to region and subregion
county_prepa <- maps::county.fips %>%
  separate(polyname, c("region","subregion"), sep=",")
county_prepb <- county_prepa %>%
  separate(subregion, c("subregion","extra"), sep=":")
```

```
## Warning: Expected 2 pieces. Missing pieces filled with `NA` in 3069
## rows [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19,
## 20, ...].
```

```
# create fips column for variable county
county_fips <- county_prepb[-4]
county_fips <- county_fips %>% mutate(fips=as.factor(fips))
# combine
combined_countiesa <- left_join(counties, county_fips, by= c("subregion","region"))
combined_countiesb <- left_join(combined_countiesa, county_winner, by="fips")
```

```
## Warning: Column `fips` joining factors with different levels, coercing to
## character vector
```

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

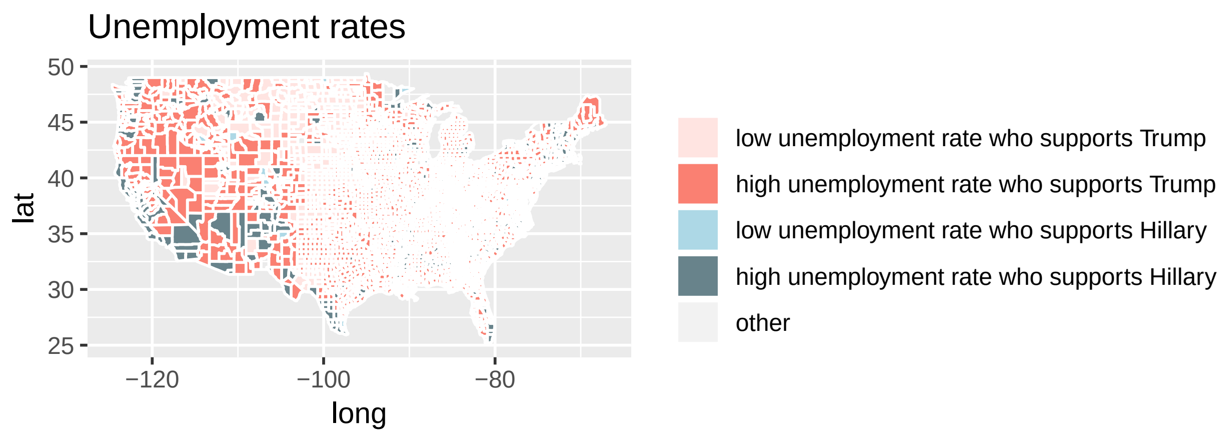


10. Create a visualization of your choice using **census** data. Many exit polls noted that demographics played a big role in the election. Use this Washington Post article and this R graph gallery for ideas and inspiration.

```r
# unemployment rate and how it affects voters
census.unemp.mean <- census %>% group_by(State, County) %>%
  mutate(avg_unemp = mean(Unemployment, na.rm=TRUE)) %>%
  ungroup()

census.unemp.1b <- census.unemp.mean %>%
  mutate(region = tolower(census.unemp.mean$State), subregion = tolower(census.unemp.mean$County))
census.unemp.2b <- census.unemp.1b[38:40] %>%
  group_by(region, subregion) %>%
  distinct()

# combining the columns, unemployment rate in 2016 = 4.7
county.unemp.1 <- left_join(county_fips, census.unemp.2b, by = c("subregion", "region"))
county.unemp.2 <- left_join(combined_countiesb, county.unemp.1, by = c("fips", "subregion", "region"))
```

```
## Warning: Column `fips` joining character vector and factor, coercing into
## character vector
```

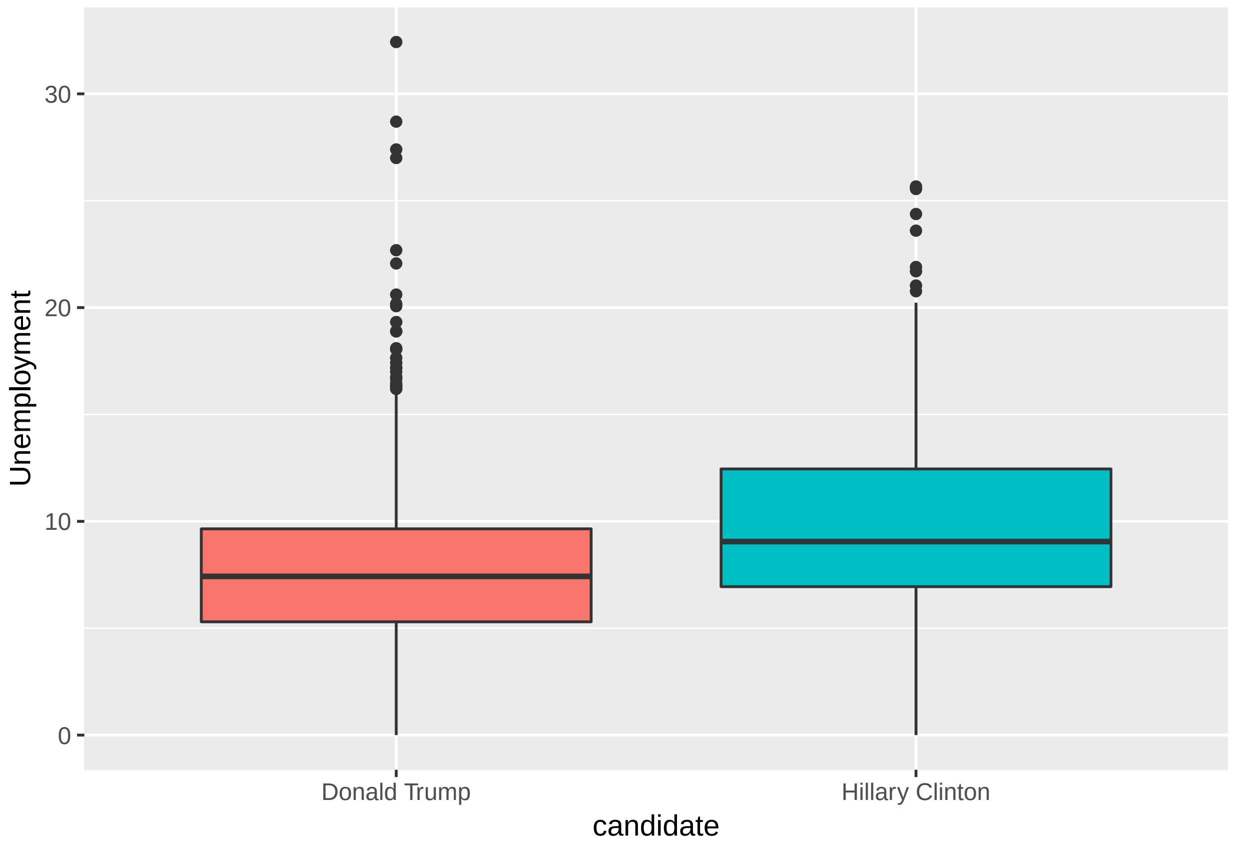
```
county.unemp.3 <- county.unemp.2 %>%
  mutate(avg_unemp = as.factor(ifelse(avg_unemp > 4.7 & county.unemp.2$candidate == "Donald Trump", "1"
                                    ifelse(county.unemp.2$candidate == "Donald Trump", "0",
                                           ifelse (avg_unemp > 4.7, "3", "2")))))

# plot
p <- ggplot()
p +
  geom_polygon(data = county.unemp.3, aes(x = long, y = lat, fill = avg_unemp, group = group),
  color = "white") +
  scale_fill_manual("",
                    labels = c("low unemployment rate who supports Trump", "high unemployment rate who s
                               "low unemployment rate who supports Hillary", "high unemployment rate who
                               "other"),
                    values = c("mistyrose", "salmon", "lightblue", "lightblue4")) +
  coord_fixed(1.3) +
  ggtitle("Unemployment rates")
```


Unemployment rates
50
45
40
35
30
25
lat
−120
−100
−80
long
low unemployment rate who supports Trump
high unemployment rate who supports Trump
low unemployment rate who supports Hillary
high unemployment rate who supports Hillary
other

```
county.fips2=county_fips
counties = counties %>% left_join(county.fips2,by=c("region","subregion"))
county_winner$fips=as.factor(as.character(county_winner$fips))
counties = counties %>% left_join(county_winner,by="fips")
```

```
## Warning: Column `fips` joining factors with different levels, coercing to
## character vector
```

```r
Census_Unemp = census %>% dplyr::select(State,County,Unemployment) %>%
  drop_na()%>%mutate_at(vars(State,County),tolower)
colnames(Census_Unemp) = c("region","subregion","Unemployment")
Census_Unemp = Census_Unemp %>% group_by(region,subregion) %>%
  dplyr::summarise(Unemploy=mean(Unemployment))
counties_Unemp = counties %>% select(region,subregion,candidate) %>% unique
counties_Unemp = counties_Unemp %>%
  left_join(Census_Unemp,by=c("region","subregion"))%>%drop_na()
ggplot(counties_Unemp,aes(x=candidate,y=Unemploy,fill=candidate))+
  geom_boxplot() + labs(y="Unemployment") +
  guides(fill=F)
```



We decided to focus on the unemployment rate in each county from the census data. Trump with his radical idea of deporting illegal immigrants to provide jobs for Americans prompted my partner and I to focus how unemployment rate affects the voter turn out. From the map, we see that it is mostly red indicating that the unemployment rate across the United States seems to support Trump in his policy of providing job to Americans. However, according to the box plot, the county unemployment rate does not seem to affect the election results.

11. The `census` data contains high resolution information (more fine-grained than county-level). In this problem, we aggregate the information into county-level data by computing `TotalPop`-weighted average of each attributes for each county. Create the following variables:

- *Clean census data* ***census.del***: start with `census`, filter out any rows with missing values, convert {`Men`, `Employed`, `Citizen`} attributes to a percentages (meta data seems to be inaccurate), compute `Minority` attribute by combining {Hispanic, Black, Native, Asian, Pacific}, remove {`Walk`, `PublicWork`, `Construction`}.
  Many columns seem to be related, and, if a set that adds up to 100%, one column will be deleted.

```
# clean census data, filter out rows with missing values
census.del<-na.omit(census)%>%
  mutate(Men=Men/TotalPop*100,
         Employed=Employed/TotalPop*100,
         Citizen=Citizen/TotalPop*100,
         Minority=rowSums(.[c("Hispanic", "Black", "Native", "Asian", "Pacific")]))%>%
  select(-Women, -Hispanic, -Native, -Black, -Asian, -Pacific, -Construction,-Walk, -PublicWork)
census.del <- census.del[,c(1:6,29,7:28)]
```

```
* _Sub-county census data, `census.subct`_:
  start with `census.del` from above, `group_by()` two attributes {`State`, `County`},
  use `add_tally()` to compute `CountyTotal`. Also, compute the weight by `TotalPop/CountyTotal`.
```

```
census.subct <- census.del %>%
  group_by(State, County) %>%
  add_tally(TotalPop) %>%
  mutate(CountyTotal = n) %>%
  mutate(weight = TotalPop/CountyTotal) %>%
  select(-n)
```

```
* _County census data, `census.ct`_:
  start with `census.subct`, use `summarize_at()` to compute weighted sum
```

```
census.ct <- census.subct %>%
  summarise_at(vars(Men:CountyTotal), funs(weighted.mean(., weight)))
```

```
## Warning: funs() is soft deprecated as of dplyr 0.8.0
## please use list() instead
##
## # Before:
## funs(name = f(.)
##
## # After:
## list(name = ~f(.))
## This warning is displayed once per session.
```

```
census.ct <- data.frame(census.ct)
```

```
* _Print few rows of `census.ct`_:
```

```
print(head(census.ct))
```

```
##     State  County      Men    White Minority  Citizen   Income IncomeErr
## 1 Alabama Autauga 48.43266 75.78823 22.53687 73.74912 51696.29  7771.009
## 2 Alabama Baldwin 48.84866 83.10262 15.21426 75.69406 51074.36  8745.050
## 3 Alabama Barbour 53.82816 46.23159 51.94382 76.91222 32959.30  6031.065
## 4 Alabama    Bibb 53.41090 74.49989 24.16597 77.39781 38886.63  5662.358
## 5 Alabama  Blount 49.40565 87.85385 10.59474 73.37550 46237.97  8695.786
## 6 Alabama Bullock 53.00618 22.19918 76.53587 75.45420 33292.69  9000.345
##   IncomePerCap IncomePerCapErr  Poverty ChildPoverty Professional  Service
## 1     24974.50        3433.674 12.91231     18.70758     32.79097 17.17044
## 2     27316.84        3803.718 13.42423     19.48431     32.72994 17.95092
## 3     16824.22        2430.189 26.50563     43.55962     26.12404 16.46343
## 4     18430.99        3073.599 16.60375     27.19708     21.59010 17.95545
## 5     20532.27        2052.055 16.72152     26.85738     28.52930 13.94252
## 6     17579.57        3110.645 24.50260     37.29116     19.55253 14.92420
##     Office Production    Drive   Carpool    Transit OtherTransp WorkAtHome
## 1 24.28243   17.15713 87.50624  8.781235 0.09525905   1.3059687  1.8356531
## 2 27.10439   11.32186 84.59861  8.959078 0.12662092   1.4438000  3.8504774
## 3 23.27878   23.31741 83.33021 11.056609 0.49540324   1.6217251  1.5019456
## 4 17.46731   23.74415 83.43488 13.153641 0.50313661   1.5620952  0.7314679
## 5 23.83692   20.10413 84.85031 11.279222 0.36263213   0.4199411  2.2654133
## 6 20.17051   25.73547 74.77277 14.839127 0.77321596   1.8238247  3.0998783
##   MeanCommute Employed PrivateWork SelfEmployed FamilyWork Unemployment
## 1    26.50016 43.43637    73.73649     5.433254 0.00000000     7.733726
## 2    26.32218 44.05113    81.28266     5.909353 0.36332686     7.589820
## 3    24.51828 31.92113    71.59426     7.149837 0.08977425    17.525557
## 4    28.71439 36.69262    76.74385     6.637936 0.39415148     8.163104
## 5    34.84489 38.44914    81.82671     4.228716 0.35649281     7.699640
## 6    28.63106 36.19592    79.09065     5.273684 0.00000000    17.890026
##   CountyTotal
## 1       55221
## 2      195121
## 3       26932
## 4       22604
## 5       57710
## 6       10678
```

Dimensionality reduction

12. Run PCA for both county & sub-county level data. Save the first two principle components PC1 and PC2 into a two-column data frame, call it `ct.pc` and `subct.pc`, respectively. What are the most prominent loadings?

```r
ct.pca <- prcomp(census.ct[,3:28], scale=TRUE)
subct.pca <- prcomp(census.subct[,4:31], scale=TRUE)

ct.pc <- data.frame(ct.pca$rotation[,1:2])
subct.pc <- data.frame(subct.pca$rotation[,1:2])

ct.pc
```

```
##                           PC1          PC2
## Men              0.0048240359 -0.135488823
## White            0.2176990922 -0.292676758
## Minority        -0.2212851691  0.288983209
## Citizen          0.0003126037 -0.239623678
## Income           0.3225865807  0.207405608
## IncomeErr        0.1738246072  0.314502186
## IncomePerCap     0.3530767161  0.138901672
## IncomePerCapErr  0.1969492637  0.207118312
## Poverty         -0.3405832434  0.056023764
## ChildPoverty    -0.3421530456  0.040582171
## Professional     0.2520238157  0.109414998
## Service         -0.1801805293  0.058927831
## Office          -0.0115397934  0.245291836
## Production      -0.1211691321 -0.143439286
## Drive           -0.0949814857  0.030319761
## Carpool         -0.0771785385 -0.036855604
## Transit          0.0765359491  0.277757814
## OtherTransp     -0.0086377486  0.059083446
## WorkAtHome       0.1724756889 -0.215721495
## MeanCommute     -0.0555820911  0.192937370
## Employed         0.3274293648  0.002977684
## PrivateWork      0.0589372390  0.181962771
## SelfEmployed     0.0938983015 -0.308799025
## FamilyWork       0.0462881560 -0.208807613
## Unemployment    -0.2876313774  0.158871955
## CountyTotal      0.0624743558  0.293195749
```

```r
subct.pc
```

```
##                         PC1          PC2
## TotalPop         0.03240180 -0.011637594
## Men              0.01730084  0.049451734
## White            0.24042491  0.308539676
## Minority        -0.24202363 -0.305445246
## Citizen          0.16084392  0.230723964
## Income           0.30251017 -0.155642964
## IncomeErr        0.19894423 -0.222428849
## IncomePerCap     0.31811986 -0.167908325
## IncomePerCapErr  0.21233534 -0.196687738
## Poverty         -0.30468855 -0.051427811
## ChildPoverty    -0.29788665 -0.027234819
## Professional     0.30643659 -0.141649910
## Service         -0.26882989 -0.057629761
## Office          -0.01383053  0.040557958
## Production      -0.20682037  0.192718992
## Drive            0.07893438  0.377206032
## Carpool         -0.16257632  0.039890691
## Transit         -0.05730847 -0.393709073
## OtherTransp     -0.04514007 -0.133109612
## WorkAtHome       0.17296089 -0.088284469
## MeanCommute      0.01001204 -0.278924637
## Employed         0.22121053 -0.060732007
## PrivateWork     -0.04201520 -0.052226242
## SelfEmployed     0.06972090 -0.009883318
## FamilyWork       0.01518467  0.044001196
## Unemployment    -0.25281744 -0.076980036
## CountyTotal     -0.02151015 -0.283842961
## weight          -0.01194567  0.228952315
```

```
rownames(ct.pc)[which(abs(ct.pc[1]) == max(abs(ct.pc[1])))]
```

```
## [1] "IncomePerCap"
```

```
rownames(ct.pc)[which(abs(ct.pc[2]) == max(abs(ct.pc[2])))]
```

```
## [1] "IncomeErr"
```

```
rownames(subct.pc)[which(abs(subct.pc[1]) == max(abs(subct.pc[1])))]
```

```
## [1] "IncomePerCap"
```

```
rownames(subct.pc)[which(abs(subct.pc[2]) == max(abs(subct.pc[2])))]
```

```
## [1] "Transit"
```

For the county, the most prominent loadings are income per capita and income error. For the sub-county, the most prominent loadings are income per capita and percentage of population that commute using public transportation. # Clustering

13. With `census.ct`, perform hierarchical clustering using Euclidean distance metric complete linkage to find 10 clusters. Repeat clustering process with the first 5 principal components of `ct.pc`. Compare and contrast clusters containing San Mateo County. Can you hypothesize why this would be the case?

```
#hierarchical clustering (Complete linkate)
distance <- dist(scale(census.ct[3:28]), method = "euclidean")
ct.hclust.complete <- hclust(distance, method = "complete")
clust.complete <- cutree(ct.hclust.complete, k=10)
table(clust.complete)
```

```
## clust.complete
##    1    2    3    4    5    6    7    8    9   10
## 2632  501    6    7    5    1   11   13   38    4
```

```
# hierarchical clustering (first 5 principal components)
ct.scores.pc <- data.frame(ct.pca$x[,1:5])
ct.scale.pc <- scale(ct.scores.pc)
distance.pc <- dist(ct.scale.pc, method="euclidean")
ct.hclust.pc <- hclust(distance.pc, method="complete")
clust.pc <- cutree(ct.hclust.pc, k=10)
table(clust.pc)
```

```
## clust.pc
##    1    2    3    4    5    6    7    8    9   10
## 2441  525   97    6    8   31    5   18    7   80
```

```r
clust.complete[which(census.ct$County == "San Mateo")]
```

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

```r
clust.pc[which(census.ct$County == "San Mateo")]
```

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

San Mateo is placed in cluster 2 in complete linkage, cluster 1 in the first 5 principal components. It seems that complete linkage cluster (cluster 2) is more appropriate because a complete link is less susceptible to noise and outliers while the other method is sensitive to noise and outliers.

Classification

In order to train classification models, we need to combine `county_winner` and `census.ct` data. This seemingly straightforward task is harder than it sounds. Following code makes necessary changes to merge them into `election.cl` for classification.

```r
tmpwinner = county_winner %>% ungroup %>%
  mutate(state = state.name[match(state, state.abb)]) %>%               ## state abbreviations
  mutate_at(vars(state, county), tolower) %>%                           ## to all lowercase
  mutate(county = gsub(" county| columbia| city| parish", "", county))  ## remove suffixes
tmpcensus = census.ct %>% mutate_at(vars(State, County), tolower)

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

## saves meta information to attributes
attr(election.cl, "location") = election.cl %>% select(c(county, fips, state, votes, pct))
election.cl = election.cl %>% select(-c(county, fips, state, votes, pct))
```

Using the following code, partition data into 80% training and 20% testing:

```r
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,]
```

Using the following code, define 10 cross-validation folds:

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

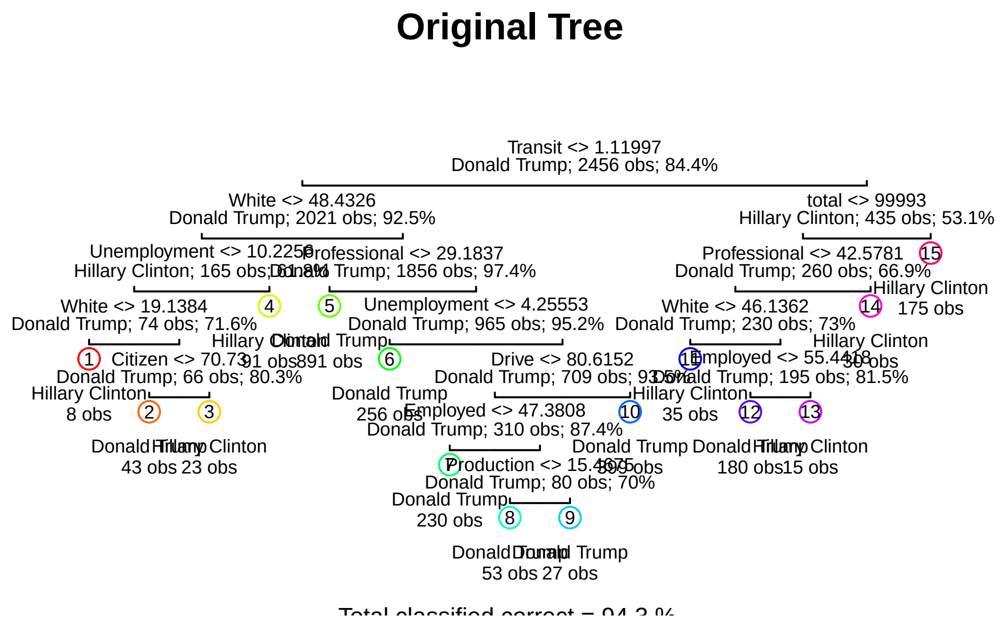
Using the following error rate function:

```
calc_error_rate = function(predicted.value, true.value){
  return(mean(true.value!=predicted.value))
}
records = matrix(NA, nrow=2, ncol=2)
colnames(records) = c("train.error","test.error")
rownames(records) = c("tree","KNN")
```

Classification: native attributes

13. Decision tree: train a decision tree by `cv.tree()`. Prune tree to minimize misclassification. Be sure to use the `folds` from above for cross-validation. Visualize the trees before and after pruning. Save training and test errors to `records` variable.

```
# X and Y for Train and Test data
trn.cl.X <- trn.cl %>% select(-candidate)
trn.cl.Y <- trn.cl$candidate
tst.cl.X <- tst.cl %>% select(-candidate)
tst.cl.Y <- tst.cl$candidate

# original tree
tree.original <- tree(candidate~.,trn.cl)
draw.tree(tree.original, nodeinfo = TRUE, cex=0.6) +
  title("Original Tree")
```



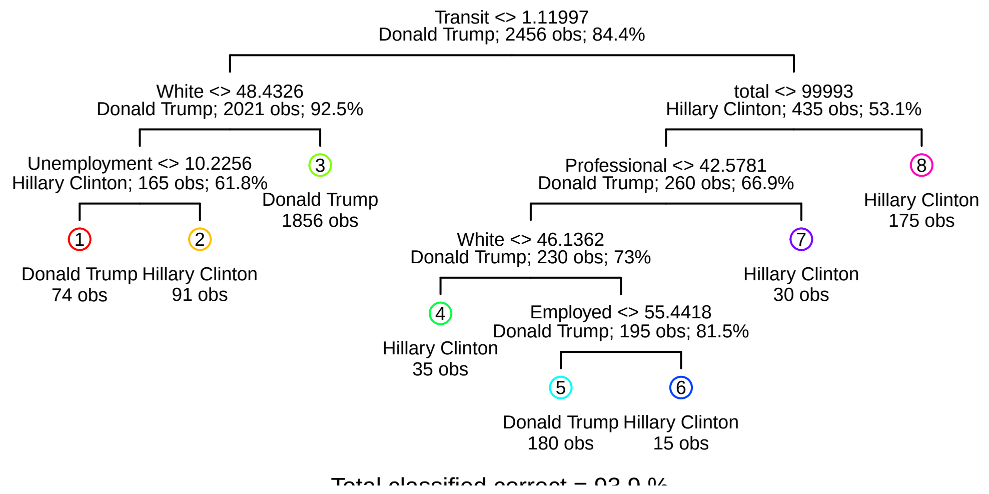
```
## integer(0)
```

```
summary(tree.original)
```

```
##
## Classification tree:
## tree(formula = candidate ~ ., data = trn.cl)
## Variables actually used in tree construction:
## [1] "Transit"      "White"        "Unemployment" "Citizen"
## [5] "Professional" "Drive"        "Employed"     "Production"
## [9] "total"
## Number of terminal nodes:  15
## Residual mean deviance:  0.3189 = 778.5 / 2441
## Misclassification error rate: 0.05741 = 141 / 2456
```

```
# cross-validation best size
cv <- cv.tree(tree.original, FUN = prune.misclass, K=sample(cut(1:nrow(trn.cl), breaks=10, labels=FALSE
best.cv <- cv$size[which.min(cv$dev)]

# pruned tree
tree.pruned <- prune.tree(tree.original, best = best.cv, method = "misclass")
draw.tree(prune.tree(tree.original, best = best.cv, method = "misclass"), cex=0.6, nodeinfo = TRUE)
title("Pruned Tree")
```

Pruned Tree



```r
# training error
pred.original.train <- predict(tree.pruned, trn.cl.X, type="class")
train.error <- calc_error_rate(pred.original.train, trn.cl.Y)
# test error
pred.original.test <- predict(tree.pruned, tst.cl.X, type="class")
test.error <- calc_error_rate(pred.original.test, tst.cl.Y)
# putting errors into records
records[1,1] <- train.error
records[1,2] <- test.error
records
```

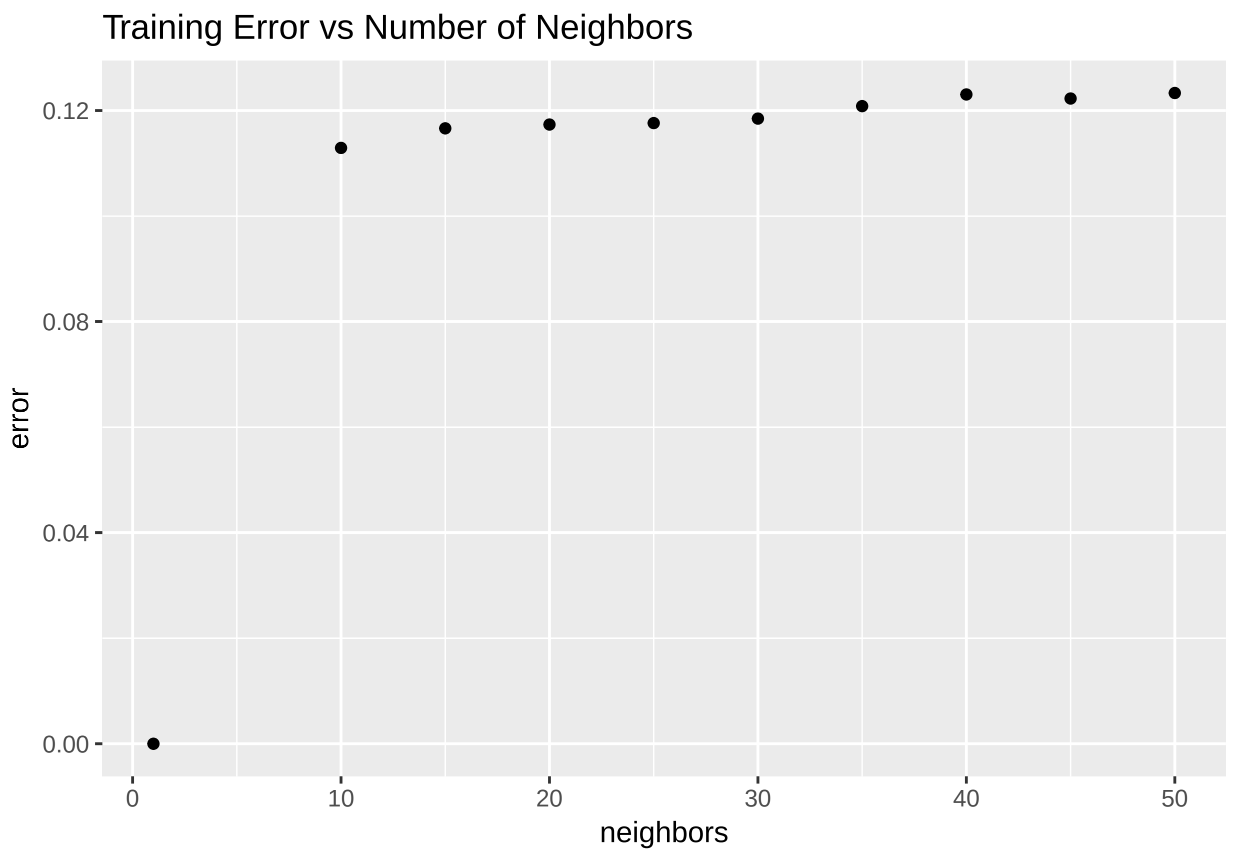
```
##      train.error test.error
## tree  0.06107492 0.07003257
## KNN           NA         NA
```

14. K-nearest neighbor: train a KNN model for classification. Use cross-validation to determine the best number of neighbors, and plot number of neighbors vs. resulting training and validation errors. Compute test error and save to `records`.

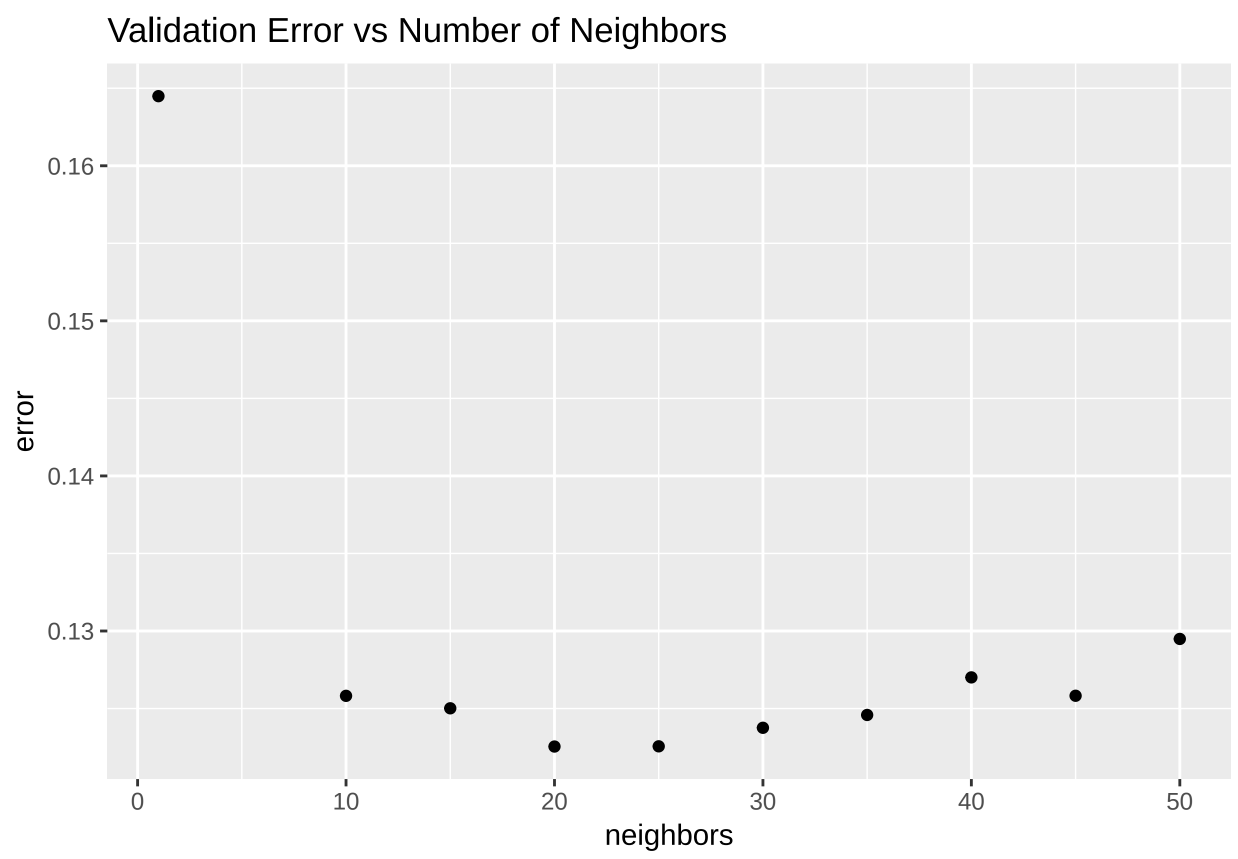
```r
do.chunk <- function(chunkid, folddef, Xdat, Ydat, k){
  train = (folddef!=chunkid)
  Xtr = Xdat[train,]
  Ytr = Ydat[train]
  Xvl = Xdat[!train,]
  Yvl = Ydat[!train]
  predYtr = knn(train = Xtr, test = Xtr, cl = Ytr, k = k)
  predYvl = knn(train = Xtr, test = Xvl, cl = Ytr, k = k)
  data.frame(fold=chunkid,
             train.error = calc_error_rate(predYtr, Ytr),
             val.error = calc_error_rate(predYvl, Yvl))
}
kvec <- c(1, seq(10, 50, length.out=9))
kerrors <- NULL
for (j in kvec) {
  tve <- plyr::ldply(1:10, do.chunk, folddef=sample(cut(1:nrow(trn.cl), breaks=10, labels=FALSE)),
  Xdat=trn.cl.X, Ydat=trn.cl.Y, k=j)
  tve$neighbors <- j
  kerrors <- rbind(kerrors, tve)
}
errors <- melt(kerrors, id.vars=c("fold","neighbors"), value.name="error")
val.error.means <- errors %>%
  filter(variable=="val.error") %>%
  group_by(neighbors) %>%
  summarise_at(vars(error),funs(mean))
min.error <- val.error.means %>%
  filter(error==min(error))
bestk <- max(min.error$neighbors)
bestk
```

```
## [1] 20
```

```r
train.error.means <- errors %>%
  filter(variable=="train.error") %>%
  group_by(neighbors) %>%
  summarise_at(vars(error),funs(mean))
ggplot(train.error.means) +
  geom_point(aes(neighbors,error)) +
  ggtitle("Training Error vs Number of Neighbors")
```



```r
ggplot(val.error.means) +
geom_point(aes(neighbors,error)) +
ggtitle("Validation Error vs Number of Neighbors")
```



```r
# training errors
pred.knn.train <- knn(train=trn.cl.X, test=trn.cl.X, cl=trn.cl.Y, k=bestk)
train.errork <- calc_error_rate(pred.knn.train, trn.cl.Y)
# test errors
pred.knn.test <- knn(train=trn.cl.X, test=tst.cl.X, cl=trn.cl.Y, k=bestk)
test.errork <- calc_error_rate(pred.knn.test, tst.cl.Y)
# adding to records
records[2,1] <- train.errork
records[2,2] <- test.errork
records
```

```
##      train.error test.error
## tree  0.06107492 0.07003257
## KNN   0.11767101 0.11237785
```

Classification: principal components

Instead of using the native attributes, we can use principal components in order to train our classification models. After this section, a comparison will be made between classification model performance between using native attributes and principal components.

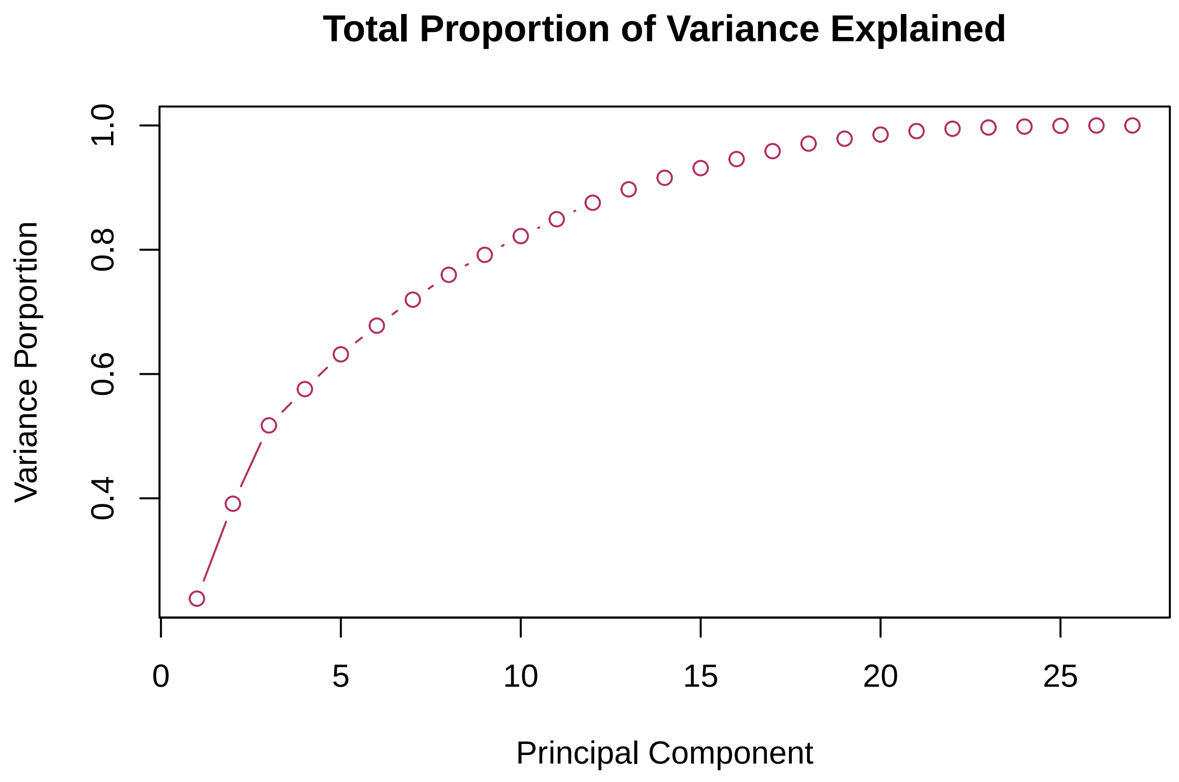
```r
pca.records = matrix(NA, nrow=2, ncol=2)
colnames(pca.records) = c("train.error","test.error")
rownames(pca.records) = c("tree","knn")
```

15. Compute principal components from the independent variables in training data. Then, determine the number of minimum number of PCs needed to capture 90% of the variance. Plot proportion of variance explained.

```r
trn.pca <- prcomp(trn.cl.X, scale=TRUE)
trn.pcavar <- trn.pca$sdev^2
trn.propvar <- trn.pcavar / sum(trn.pcavar) #pve
which(cumsum(trn.propvar) >= 0.9)[1]
```

```
## [1] 14
```

```r
plot(cumsum(trn.propvar), type="b", xlab="Principal Component",
     ylab="Variance Porportion",
     main="Total Proportion of Variance Explained", col="maroon")
```

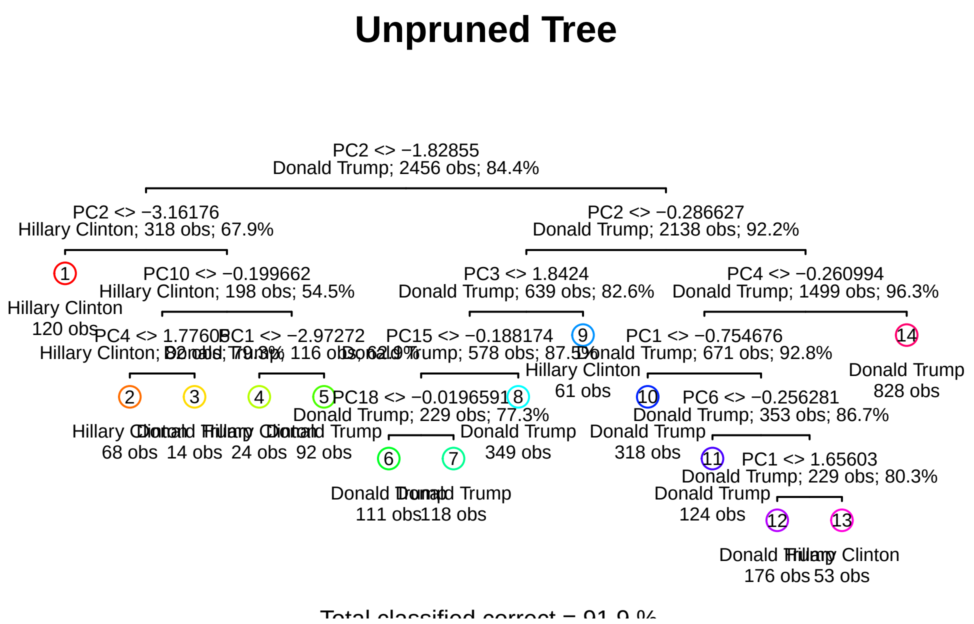


16. Create a new training data by taking class labels and principal components. Call this variable `tr.pca`. Create the test data based on principal component loadings: i.e., transforming independent variables in test data to principal components space. Call this variable `test.pca`.

```r
# training data by taking class labels and principal components
trn.pc <- data.frame(trn.pca$x)
tr.pca <- trn.pc %>%
  mutate(candidate=trn.cl$candidate)
tst.pca <- prcomp(tst.cl.X, scale=TRUE)
tst.pc <- data.frame(tst.pca$x)
# test data based on principal component loadings
test.pca <- tst.pc %>%
  mutate(candidate=tst.cl$candidate)
```

17. Decision tree: repeat training of decision tree models using principal components as independent variables. Record resulting errors.

```r
# creating the original tree
tree.pca <- tree(candidate~., tr.pca)
draw.tree(tree.pca, nodeinfo = TRUE, cex=0.6)
title ("Unpruned Tree")
```



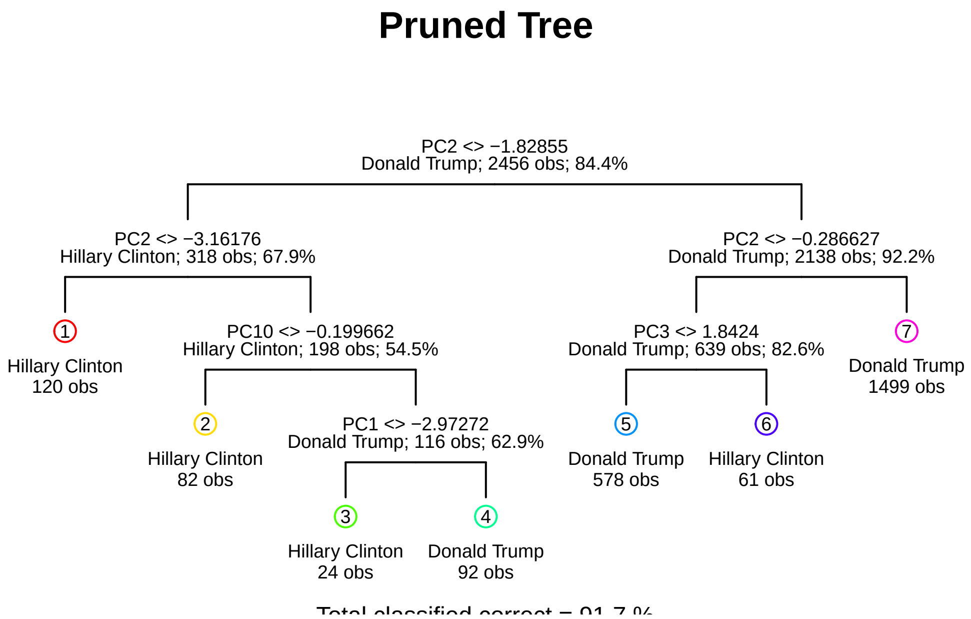
```r
summary(tree.pca)
```

```
##
## Classification tree:
## tree(formula = candidate ~ ., data = tr.pca)
## Variables actually used in tree construction:
## [1] "PC2"  "PC10" "PC4"  "PC1"  "PC3"  "PC15" "PC18" "PC6"
## Number of terminal nodes:  14
## Residual mean deviance:  0.419 = 1023 / 2442
## Misclassification error rate: 0.08062 = 198 / 2456
```

```
# using cross validation to find best size
cv.pca <- cv.tree(tree.pca, rand=sample(cut(1:nrow(trn.cl), breaks=10, labels=FALSE)), FUN=prune.miscla
best.pca <- min(cv.pca$size[which(cv.pca$dev==min(cv.pca$dev))])
best.pca
```

```
## [1] 7
```

```
# pruning the tree based on cv size
# prunedpcatree = pruned.pca
pruned.pca <- prune.tree(tree.pca, best=best.pca, method="misclass")

draw.tree(pruned.pca, nodeinfo=TRUE, cex=0.6)
title("Pruned Tree")
```



```
# training error
# pred.pcatree.train = pred.pca.train
# train.errorpt = error.train.pca
pred.pca.train <- predict(pruned.pca, trn.pc, type="class")
error.train.pca <- calc_error_rate(pred.pca.train, tr.pca$candidate)

# test error
pred.pca.test <- predict(pruned.pca, tst.pc, type="class")
error.test.pca <- calc_error_rate(pred.pca.test, test.pca$candidate)
```

```r
# putting errors into records
pca.records[1,1] <- error.train.pca
pca.records[1,2] <- error.test.pca
pca.records
```

```
##      train.error test.error
## tree  0.08346906 0.09934853
## knn           NA         NA
```

18. K-nearest neighbor: repeat training of KNN classifier using principal components as independent variables. Record resulting errors.

```r
k.pca <- c(1, seq(10, 50, length.out = 9))
k.error.pca <- c()

# cross validation
for (i in k.pca){
  temp <- plyr::ldply(1:10, do.chunk, folddef = sample(cut(1:nrow(trn.cl), breaks = 10, labels = FALSE))
                      Xdat = trn.pc, Ydat = tr.pca$candidate, k = i)
  temp$neighbors <- i
  k.error.pca <- rbind(k.error.pca, temp)
}

# Test error at each K
error.pca <- melt(k.error.pca, id.vars = c("fold", "neighbors"), value.name = "error")
val.error.means.pca <- error.pca %>%
  filter(variable == "val.error") %>%
  group_by(neighbors) %>%
  summarise_at(vars(error), funs(mean))

# best k
best.k <- max(val.error.means.pca %>%
                filter(error==min(error)))

# Train error at each K
train.error.k <- error.pca %>%
  filter(variable == "train.error") %>%
  group_by(neighbors) %>%
  summarise_at(vars(error), funs(mean))

# Train / Test errors for KNN
pred.train.knn <- knn(train = trn.pc, test = tst.pc, cl = tr.pca$candidate, k = best.k)
error.train.knn <- calc_error_rate(pred.train.knn, tr.pca$candidate)
pred.test.knn <- knn(train = trn.pc, test = tst.pc, cl = tr.pca$candidate, k = best.k)
error.test.knn <- calc_error_rate(pred.test.knn, test.pca$candidate)

# adding to records
pca.records[2,1] <- error.train.knn
pca.records[2,2] <- error.test.knn
pca.records
```

```
##      train.error test.error
## tree  0.08346906 0.09934853
## knn   0.22557003 0.07817590
```

Interpretation & Discussion

19. This is an open question. Interpret and discuss any insights gained and possible explanations. Use any tools at your disposal to make your case: visualize errors on the map, discuss what does/doesn't seems reasonable based on your understanding of these methods, propose possible directions (collecting additional data, domain knowledge, etc)

Our main goal in this project is to predict the election outcome with the given data set. We acknowledged that the data set we received require organizing, clustering, classifying, and running regressions to achieve the appropriate prediction for the result. We also obtained significant variables from principal component analysis and took pruned tree plot to describe how votes were divided between the presidential candidates.

Within the data set, we can see that there are lsted factors that have influenced the voters. For example, the residing residences, ethnicity, economic and social stability, and occupation influence voters preference toward the candidates. Although not included in the data set, we already found out that the media played a significant role swaying voter's mind by broadcasting candidate's involvement with FBI investigation, remarks about international policies.

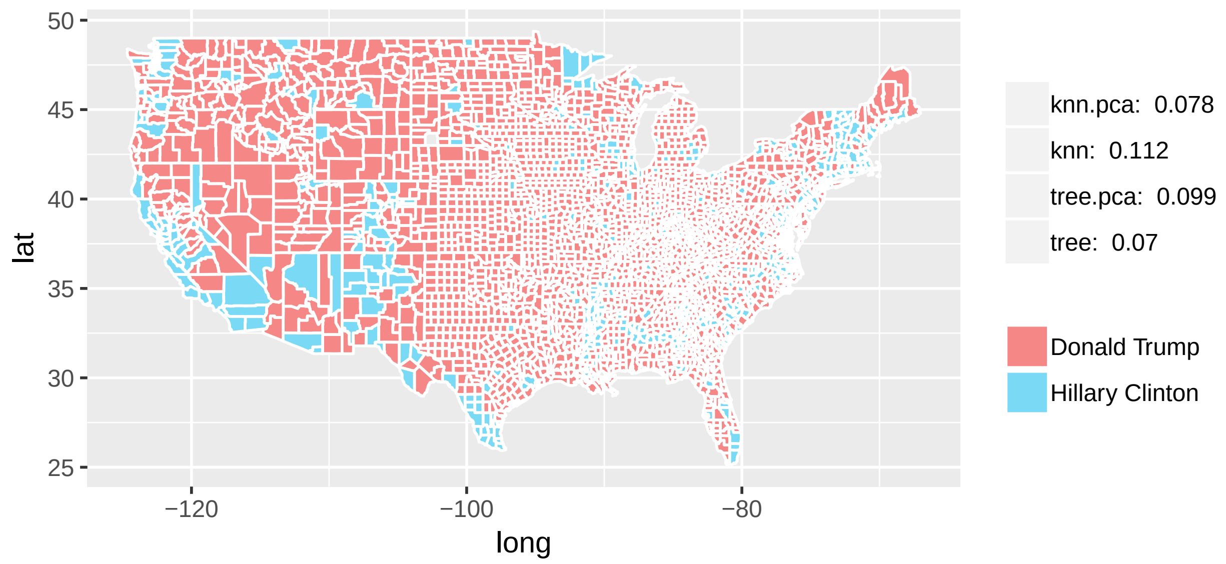
Many statisticians predicted that Clinton would win the 2016 election. This over-performing is problematic because the data used to achieve this prediction came from state's polls that were all over the place and unorganized. Silver's model is based Bayes' theorem and also on pulling historical averages that have worked and have not worked which takes consideration of all possible probabilities.

We decided to scale the features before running the principal components analysis (PCA). By using the option scale=TRUE, we make variables to have standard deviation value of one. The center and scale components corresponds to the mean and standard deviation of the variable. The largest absolute values in the loading matrix found was the "IncomePerCap" and "IncomeErr" for County and "IncomePerCap" and "Transit" for the Sub-County. These influential values translates into our reasoning that income and transit plays a role in the election results.

We also learned the difference between complete-linkage and single-link hierarchical clustering. In this project, we wrote two parts: 1) complete-linkage, and 2) first-5-principal-components. As a result of PCA, we found that San Mateo is placed in cluster 2 for complete-linkage, and placed in cluster 1 for first-5-principal-components. It is notable that complete-linkage cluster is appropriate clustering because it is less susceptible to noise and outliers.

The significant variables that resulted from logistic regression was: Citizen, IncomePerCap, Professional, Service, Production, Drive, Carpool, Employed, PrivateWork, Unemployment. As interpreted previously, economic and social influence voters. The profession, income, and social-class forge voter's preference. We found variables, Drive and Carpool, interesting because this means that voters who carpooled is likely to discuss, assert, and defend their candidate preferences.

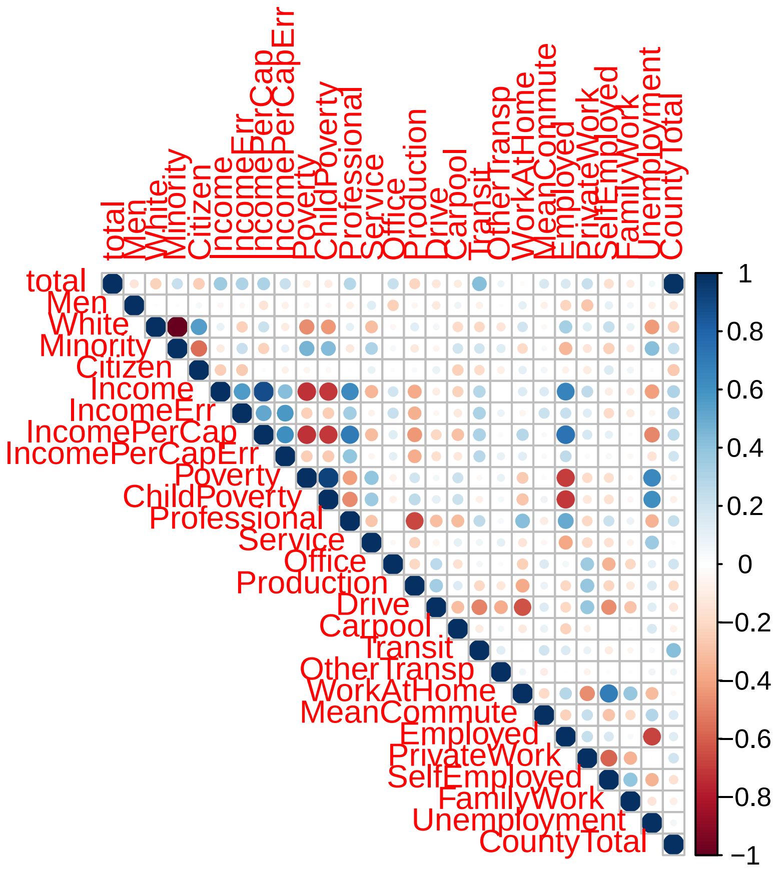
```r
rec <- round(c(records[1:2,2],pca.records[1:2,2]),3)
rec <- paste(c("tree: ","knn: ","tree.pca: ","knn.pca: "),rec)
newdf <- counties %>% na.omit
newdf$rec <- c(rep(rec,nrow(newdf)/4),rec[0])
ggplot(data = newdf) +
geom_polygon(aes(x = long, y = lat, fill = candidate, group = group), color = "white") +
geom_point(aes(x = long, y = lat,color = rec),size = 0,alpha = 0) +
coord_fixed(1.3) +
scale_fill_manual(breaks=c("Donald Trump", "Hillary Clinton"),
values = c("#f68787", "#7ad9f5")) +
theme(legend.title=element_blank())
```



From the prediction results, decision tree has lower error than knn does, and after applying pca, knn has lower error than decision tree does. As a result, decision tree performs better than knn.

Correlation matrix to view correlation between different variables.

```r
X.cl = election.cl %>% dplyr::select(-candidate)
M = cor(X.cl)
corrplot(M, type="upper")
```



According to the plot, we found that we still need to consider more factors, such as the scaling of variables, the underlying correlation between different variables, etc because these factors also affect the results significantly. For example, correlogram shows that 'White' is negatively correlated with 'Minority', and what we found in the previous results shows that 'White' people and 'Minority' tend to choose different candidates. We can also use models such as LDA/LQA, linear regression, and neural networks. we can improve the predictions by collecting more data on which party did each county vote previously, and investigating additional variables that may also affect the voters' behaviors.

Taking it further

20. Propose and tackle at least one interesting question. Be creative! Some possibilities are:

    - Data preprocessing: we aggregated sub-county level data before performing classification. Would classification at the sub-county level before determining the winner perform better? What implicit assumptions are we making?
    - Feature engineering: would a non-linear classification method perform better? Would you use native features or principal components?
    - Additional classification methods: logistic regression, LDA, QDA, SVM, random forest, etc. (You may use methods beyond this course). How do these compare to KNN and tree method?
    - Bootstrap: Perform boostrap to generate plots similar to Figure 4.10/4.11. Discuss the results.

```r
# creating a records table specifically for additional classification
records.class = matrix(NA, nrow = 4, 2)
colnames(records.class) = c("train.error", "test.error")
```

```r
rownames(records.class) = c("lasso", "logistic", "SVM", "Boosting")
records.class
```

```
##          train.error test.error
## lasso             NA         NA
## logistic          NA         NA
## SVM               NA         NA
## Boosting          NA         NA
```

Lasso

```r
x <- model.matrix(candidate~., election.cl)[,-1]
y <- factor(election.cl$candidate)
lam <- c(1, 5, 10, 50) * 1e-4
cv.lasso <- cv.glmnet(x[in.trn,], y[in.trn], alpha = 1, family = "binomial", lambda = lam, foldid = sam
best.lambda <- cv.lasso$lambda.min
best.lambda
```

```
## [1] 5e-04
```

```r
pred.train.lasso <- predict(cv.lasso, s = best.lambda, newx = x[in.trn, ], type = "class")
train.error.lasso <- calc_error_rate(pred.train.lasso, trn.cl$candidate)

pred.test.lasso <- predict(cv.lasso, s = best.lambda, newx = x[-in.trn,], type = "class")
test.error.lasso <- calc_error_rate(pred.test.lasso, tst.cl$candidate)

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

```
##          train.error test.error
## lasso     0.06758958 0.06677524
## logistic          NA         NA
## SVM               NA         NA
## Boosting          NA         NA
```

```r
# Fit logit function
elect_glm = glm(candidate~., data=trn.cl, family = binomial(link = "logit"))
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```r
# find significant variable for model
summary(elect_glm)
```

```
##
## Call:
## glm(formula = candidate ~ ., family = binomial(link = "logit"),
##     data = trn.cl)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -3.9879  -0.2515  -0.1024  -0.0350   3.5994
##
## Coefficients:
##                  Estimate Std. Error z value Pr(>|z|)
## (Intercept)     -2.545e+01  9.492e+00  -2.681 0.007345 **
## total            3.824e-06  4.142e-06   0.923 0.355992
## Men              6.848e-02  4.762e-02   1.438 0.150430
## White           -1.799e-01  6.484e-02  -2.775 0.005517 **
## Minority        -3.652e-02  6.218e-02  -0.587 0.557000
## Citizen          1.379e-01  2.825e-02   4.883 1.05e-06 ***
## Income          -7.182e-05  2.619e-05  -2.743 0.006095 **
## IncomeErr        7.489e-06  6.192e-05   0.121 0.903744
## IncomePerCap     2.211e-04  6.305e-05   3.506 0.000455 ***
## IncomePerCapErr -2.745e-04  1.320e-04  -2.080 0.037503 *
## Poverty          4.518e-02  4.067e-02   1.111 0.266649
## ChildPoverty    -1.730e-02  2.477e-02  -0.698 0.484988
## Professional     2.823e-01  3.859e-02   7.315 2.57e-13 ***
## Service          3.230e-01  4.789e-02   6.743 1.55e-11 ***
## Office           7.437e-02  4.574e-02   1.626 0.103971
## Production       1.530e-01  4.123e-02   3.711 0.000206 ***
## Drive           -1.962e-01  4.685e-02  -4.189 2.81e-05 ***
## Carpool         -1.560e-01  5.870e-02  -2.657 0.007893 **
## Transit          9.218e-02  9.461e-02   0.974 0.329867
## OtherTransp     -6.110e-02  9.519e-02  -0.642 0.520925
## WorkAtHome      -1.804e-01  7.469e-02  -2.416 0.015702 *
## MeanCommute      4.379e-02  2.396e-02   1.827 0.067661 .
## Employed         1.999e-01  3.330e-02   6.002 1.95e-09 ***
## PrivateWork      1.211e-01  2.180e-02   5.554 2.78e-08 ***
## SelfEmployed     6.724e-02  4.659e-02   1.443 0.148905
## FamilyWork      -8.552e-01  3.930e-01  -2.176 0.029565 *
## Unemployment     2.117e-01  3.982e-02   5.315 1.07e-07 ***
## CountyTotal     -1.312e-06  1.615e-06  -0.812 0.416581
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 2122.94  on 2455  degrees of freedom
## Residual deviance:  840.03  on 2428  degrees of freedom
## AIC: 896.03
##
## Number of Fisher Scoring iterations: 7
```

```r
# train error
prob.train = predict(elect_glm, type="response")
pred.train = ifelse(prob.train>0.5,
                                  "Hillary Clinton", "Donald Trump")
train.error = calc_error_rate(pred.train,trn.cl$candidate)

# test error
prob.test = predict(elect_glm, newdata=tst.cl, type="response")
pred.test = ifelse(prob.test>0.5,
                                 "Hillary Clinton", "Donald Trump")
test.error = calc_error_rate(pred.test,tst.cl$candidate)
```

```r
records.class[2,]<-c(train.error,test.error)
records.class
```

```
##          train.error test.error
## lasso     0.06758958 0.06677524
## logistic  0.06758958 0.06514658
## SVM               NA         NA
## Boosting          NA         NA
```

SVM

```r
set.seed(1)
tune.out = tune(svm, candidate ~ ., data=trn.cl, kernel="radial", ranges=list(cost=c(0.01, 0.1, 1, 10, 
bestmod=tune.out$best.model

#Calculate train error
svm.train.error = tune.out$best.performance
#Calculate test error
svm.pred.test = predict(tune.out$best.model,newdata=tst.cl)
svm.test.error = calc_error_rate(svm.pred.test, tst.cl$candidate)
records.class[3,] = c(svm.train.error,svm.test.error)
records.class
```

```
##          train.error test.error
## lasso     0.06758958 0.06677524
## logistic  0.06758958 0.06514658
## SVM       0.05904264 0.05863192
## Boosting          NA         NA
```

Boosting

```r
elect.boost <- gbm(ifelse(candidate=="Donald Trump",0,1)~ ., data=trn.cl, distribution="bernoulli", n.t

pred.train.boost <- predict(elect.boost, trn.cl, n.trees = 1000, type = "response")
pred.test.boost <- ifelse(pred.train.boost < 0.5, "Donald Trump", "Hillary Clinton")

boost.prob <- predict(elect.boost, tst.cl, n.trees=1000, type="response")
pred.test <- ifelse(boost.prob < 0.5, "Donald Trump", "Hillary Clinton")

train.error <- calc_error_rate(pred.test.boost, tst.cl$candidate)
test.error <- calc_error_rate(pred.test, tst.cl$candidate)

records.class[4,] <- c(train.error,test.error)
records.class
```

```
##          train.error test.error
## lasso     0.06758958 0.06677524
## logistic  0.06758958 0.06514658
## SVM       0.05904264 0.05863192
## Boosting  0.23045603 0.06351792
```

```
records
```

```
##      train.error test.error
## tree  0.06107492 0.07003257
## KNN   0.11767101 0.11237785
```

```
pca.records
```

```
##      train.error test.error
## tree  0.08346906 0.09934853
## knn   0.22557003 0.07817590
```

```
records.class
```

```
##          train.error test.error
## lasso     0.06758958 0.06677524
## logistic  0.06758958 0.06514658
## SVM       0.05904264 0.05863192
## Boosting  0.23045603 0.06351792
```

According to the table, SVM model performs better than KNN and tree models since it has the lowest test error. SVM has a regularization parameter so that it is able to avoid over-fitting, and it uses the kernel trick to build in expert knowledge about the problem via engineering the kernel. Compared to logistic, lasso, SVM is better for the same reason. But boosting has much smaller train error but larger test error.