2016 Election Prediction

Ashley Radford and Samyuktha Sridhar

3/23/2018

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?

   Voter behavior and election forecasting are difficult to predict due to many factors. One reason could be differential voter turnout; 2016's polls were based on the assumption that relatively equal numbers of Democrats and Republicans would vote, but in reality there was a significantly higher republican voter turnout than Democratic voter turnout. Another reason could be that there were last minute changes in voter decisions; post-election polls found that many voters changed their vote in the week leading up to the election. A third reason could be that poll results were interpreted as corresponding to real changes in voter preference, but a spike in candidate votes in the polls might have just come from a change in nonresponse patterns; for example, if Trump voters spiked in the polls in a given week, it might have been because more Trump voters responded to the polls during that week.

2. What was unique to Nate Silver's approach in 2012 that allowed him to achieve good predictions?

   Usually, the outcome with the maximum probability is taken to be the most likely outcome. Silver's approach was to look at a full range of probabilities rather than just the maximum probability. In the election setting, for example, he calculated a range of probabilities of support for different dates. For the following date, he could use the model for actual support to predict the probability that support has shifted from one number to another; then if the actual polling numbers are higher for example, the probability of support is likely to be in the higher end of the range. This prediction model is based on the Bayes' Theorem.

3. What went wrong in 2016? What do you think should be done to make future predictions better?

   In 2016, the candidates were extremely controversial, and thus made it really hard to accurately predict the outcome of the votes. News and current events played a huge part in swaying voters during this election, thus prompting many voters to make last minute decisions and causing most predictions to be wrong. To make future predictions more accurate, voter demographic information should be taken into account at a federal, state, and county level and supervised learning models should be applied to better predict which factors are most influential in voter choice, and categorize voters into candidate groups.

Data

```
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 (and 11 columns) 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 acronym 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
1001020100	Alabama	Autauga	1948	940	1008	0.9	87.4	7.7	0.3	0.6
1001020200	Alabama	Autauga	2156	1059	1097	0.8	40.4	53.3	0.0	2.3
1001020300	Alabama	Autauga	2968	1364	1604	0.0	74.5	18.6	0.5	1.4
1001020400	Alabama	Autauga	4423	2172	2251	10.5	82.8	3.7	1.6	0.0
1001020500	Alabama	Autauga	10763	4922	5841	0.7	68.5	24.8	0.0	3.8
1001020600	Alabama	Autauga	3851	1787	2064	13.1	72.9	11.9	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

CensusTract	Census.tract.ID	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
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$fips) != as.character(election.raw$state))

# checking if all the observations are present / not overlapped with eachother
dim(election_federal)[1] + dim(election_state)[1] + dim(election)[1] == dim(election.raw)[1]
```

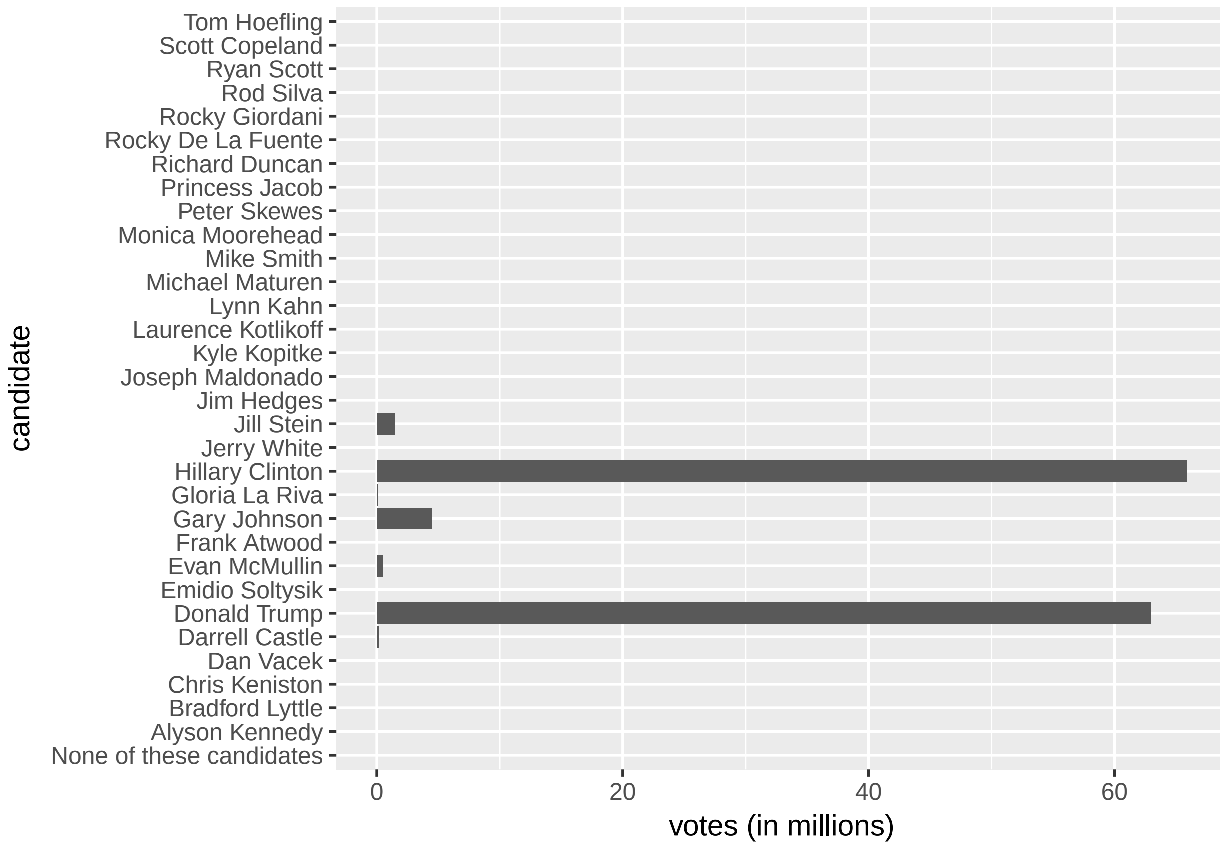
```
## [1] TRUE
```

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

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

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

```r
ggplot(data = election_federal, mapping = aes(x = candidate,y = votes/1000000)) +
    geom_bar(stat="identity") +
    scale_y_continuous() +
    ylab("votes (in millions)") +
    coord_flip()
```



There were 32 presidential candidates in the 2016 election.

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)
```

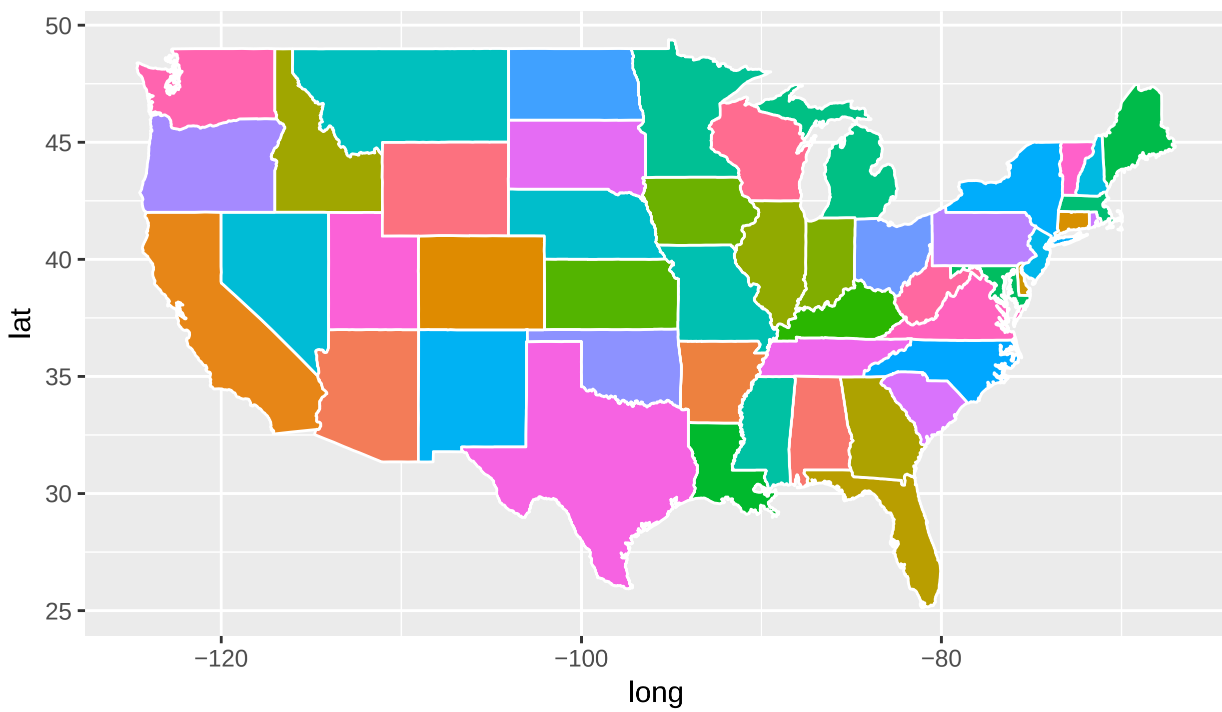
```
## Selecting by pct
```

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)
```

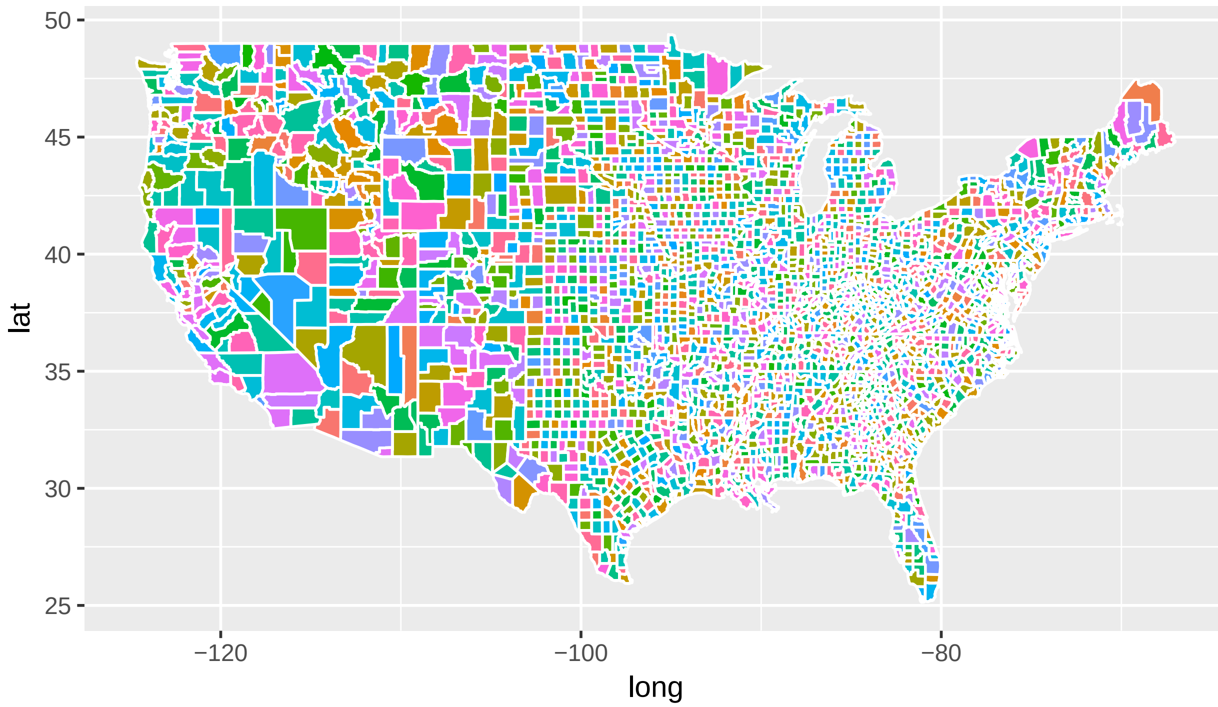



```r
    # color legend is unnecessary and takes too long
```

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.

```r
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)
```

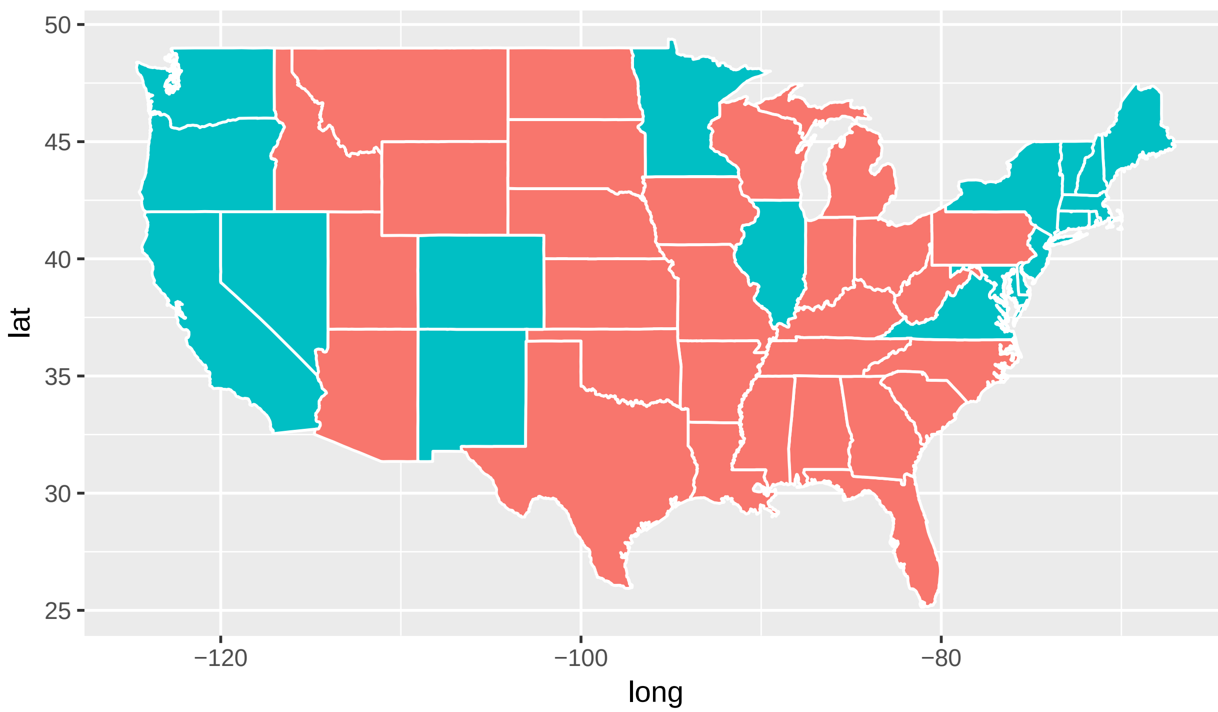


8. Now color the map by the winning candidate for each state. First, combine `states` variable and `state_winner` we created earlier using `left_join()`. Note that `left_join()` needs to match up values of states to join the tables; however, they are in different formats: e.g. `AZ` vs. `arizona`. Before 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.

```
fips = state.abb[match(states$region, tolower(state.name))]
states <- states %>% mutate(fips=fips)
combined_states <- left_join(states, state_winner, by="fips")
```

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

```
# plotting 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.

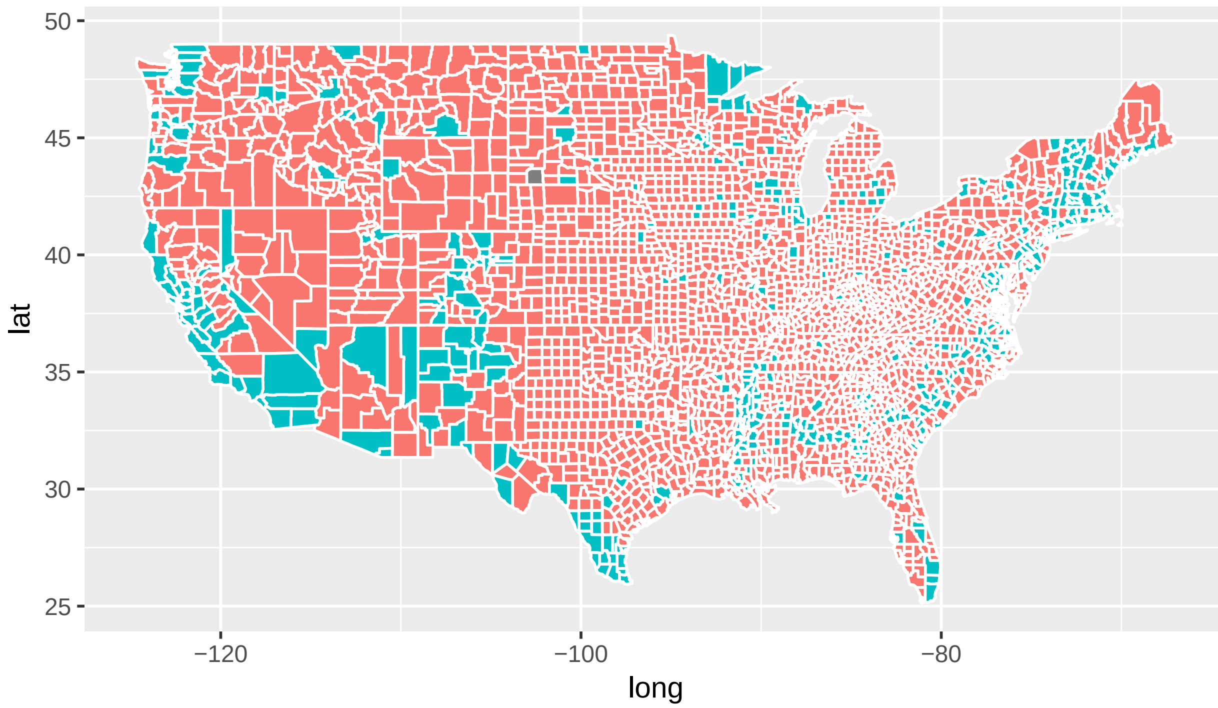
```
county_prepa <- maps::county.fips %>%
    separate(polyname, c("region","subregion"), sep=",")
# note that some counties are named with a specific "sub-subgroup""
# ie accomack:main and accomack:chincoteague
# these will not join with their proper subgroups (like accomack)
# when using left_join and so the following code splits those with
# specific sub-subgroups and eliminates rest of the string after ":"
county_prepb <- county_prepa %>%
    separate(subregion, c("subregion","extra"), sep=":")
```

```
## Warning: Too few values at 3069 locations: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
## 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, ...
```

```
county_fips <- county_prepb[-4] # get rid of extra

# changing fips column from numeric to factor
county_fips <- county_fips %>% mutate(fips=as.factor(fips))

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
# plotting the winning candidate for each county
# note county "oglala" (fips = 46102) is greyed out because it is
# not in maps::county.fips (and thus county_fips) or counties data
ggplot(data = combined_countiesb) +
  geom_polygon(aes(x = long, y = lat, fill = candidate, group = group),
               color = "white") + coord_fixed(1.3) + guides(fill=FALSE)
```




```r
county_winner$county[-which(county_winner$fips %in% county_prepa$fips)]
```

```
##  [1] <NA>                   Honolulu County       Chesapeake city
##  [4] Richmond city          Alexandria city       Hawaii County
##  [7] Maui County            Portsmouth city       Roanoke city
## [10] Lynchburg city         Kauai County          Charlottesville city
## [13] Danville city          Harrisonburg city     Manassas city
## [16] Petersburg city        Salem city            Fairfax city
## [19] Fredericksburg city    Staunton city         Winchester city
## [22] Waynesboro city        Hopewell city         Colonial Heights city
## [25] Falls Church city      Williamsburg city     Poquoson city
## [28] Bristol city           Radford city          Martinsville city
## [31] Manassas Park city     Franklin city         <NA>
## [34] Lexington city         Buena Vista city      Covington city
## [37] Galax city             Emporia city          Norton city
## 1846 Levels: Abbeville County Acadia Parish Accomack County ... Ziebach County
```

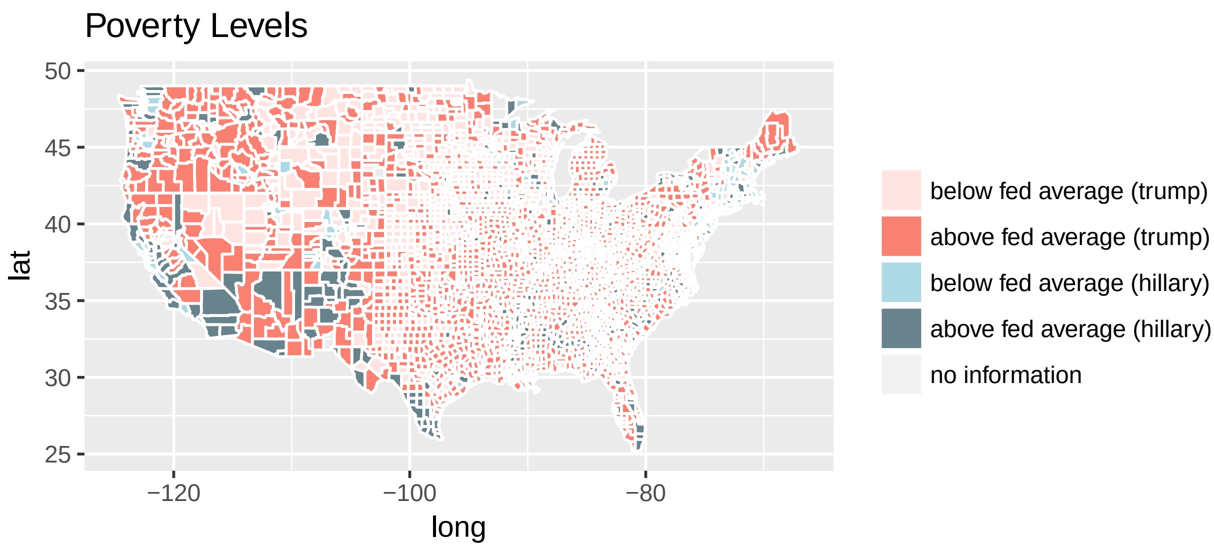
10. Create a visualization of your choice using `census` data. Many exit polls noted that demographics played a big role in the election. Use this Washington Post article and this R graph gallery for ideas and inspiration.

    The following map visualizes the average poverty level of each county, grouped by who they voted for. The pink/orange base color represents those who voted for Donald Trump while the blue base color represents those who voted for Hillary Clinton. The darker color of each group represents those counties that are above the federal poverty level while the lighter color of each group represents those counties that are below the federal poverty level. Looking at this visualization, one can see that even though Hillary Clinton has fewer counties, on average her counties have a lower rate of poverty than Donald Trump's counties.

```
census_pov_mean <- census %>% group_by(State, County) %>%
    mutate(avg_pov = mean(Poverty, na.rm=TRUE)) %>%
    ungroup()

census_pm_lowera <- census_pov_mean %>%
    mutate(region = tolower(census_pov_mean$State),
           subregion = tolower(census_pov_mean$County))
census_pm_lowerb <- census_pm_lowera[38:40] %>%
    group_by(region, subregion) %>% distinct()


poverty_countiesa <- left_join(county_fips, census_pm_lowerb,
                               by = c("subregion", "region"))
poverty_countiesb <- left_join(combined_countiesb, poverty_countiesa,
                               by = c("fips","subregion", "region"))
```

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

```
poverty_countiesc <- poverty_countiesb %>%
    mutate(avg_povl=as.factor(ifelse(avg_pov > 12.7 &
                                         poverty_countiesb$candidate == "Donald Trump","1",
                                    ifelse(poverty_countiesb$candidate == "Donald Trump","0",
                                    ifelse(avg_pov > 12.7,"3","2")))))
                                    # federal average poverty rate in 2016 = 12.7

ggplot() +
    geom_polygon(data=poverty_countiesc, aes(x=long, y=lat, fill=avg_povl, group=group),
                 color = "white") +
    scale_fill_manual("",labels=c("below fed average (trump)","above fed average (trump)",
                                  "below fed average (hillary)", "above fed average (hillary)",
                                  "no information"),
                      values=c("mistyrose","salmon","lightblue","lightblue4")) +
    ggtitle("Poverty Levels") +
    coord_fixed(1.3)
```



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 should be deleted.*
   - *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`.
   - *County census data, `census.ct`*: start with `census.subct`, use `summarize_at()` to compute weighted sum
   - *Print few rows of `census.ct`*:

```
# filter out any rows with NA
census.del <- na.omit(census) %>%
    mutate(Men = Men/TotalPop*100,
           Employed = Employed/TotalPop*100,
           Citizen = Citizen/TotalPop*100,
           Minority = 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)] # reordering (want minority next to white)

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

```
    mutate(CountyTotal = n) %>%
    mutate(Weight = TotalPop/CountyTotal) %>%
    select(-n)

census.ct <- census.subct %>%
    summarise_at(vars(Men:CountyTotal), funs(weighted.mean(.,Weight)))
census.ct <- data.frame(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 principal components data frames, call them `ct.pc` and `subct.pc`, respectively. What are the most prominent loadings of the first two principal components PC1 and PC2?

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

```r
# getting the principal components
ct.pc <- data.frame(ct.pca$rotation)
subct.pc <- data.frame(subct.pca$rotation)

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

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

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

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

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

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

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

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

The most prominent loadings at the county level of PC1 and PC2 are income per capita and income error, respectively. The most prominent loadings at the subcounty level of PC1 and PC2 are income per capita and the percentage of population that commute via 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?

```r
# using the entire data set
scale.census.ct <- scale(census.ct[3:28])
dista <- dist(scale.census.ct, method="euclidean")
hc.census.ct <- hclust(dista, method="complete")
clustersa <- cutree(hc.census.ct, k=10)
table(clustersa)
```

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

```r
# using the first 5 principal components
ct.pc.scores <- data.frame(ct.pca$x[,1:5])
scale.ct.pc <- scale(ct.pc.scores)
distb <- dist(scale.ct.pc, method="euclidean")
hc.ct.pc <- hclust(distb, method="complete")
clustersb <- cutree(hc.ct.pc, k=10)
table(clustersb)
```

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

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

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

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

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

```
dataclustersa <- census.ct %>% mutate(Cluster=clustersa)
dataclustersb <- census.ct %>% mutate(Cluster=clustersb)
```

When using census.ct, the county San Mateo is placed into cluster 2. But when using the first five principal components, San Mateo is placed into cluster 1. Furthermore, when looking at the cluster assignments attached to the original data (in the dataframes dataclustersa and dataclustersb) we see that when San Mateo is placed in cluster 2, it appears to be more in line with cluster guidelines (we want the elements in the clusters to be as similar as possible); there are less Alabama counties inside cluster 2 with San Mateo for example (which we would expect since San Mateo is a county from California). But when San Mateo is placed into cluster 1 there are way more differing counties in its cluster (most of Alabama counties are in this cluster for example). This is most likely due to the fact that the first five principal components do not describe most of the variance in census.ct, thus there are disagreements in the clustering.

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. The following code makes necessary changes to merge them into `election.cl` for classification.

```
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:

```
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:

```
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=3, ncol=2)
colnames(records) = c("train.error","test.error")
rownames(records) = c("tree","knn","logistic")
```

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.

```r
trn.cl <- trn.cl %>% select(-total) # getting rid of constant variable
tst.cl <- tst.cl %>% select(-total) # getting rid of constant variable

# setting up the X and Y variables
trn.clX <- trn.cl %>% select(-candidate)
trn.clY <- trn.cl$candidate
tst.clX <- tst.cl %>% select(-candidate)
tst.clY <- tst.cl$candidate

# creating the original tree
cantree <- tree(candidate~.,trn.cl)
summary(cantree)
```

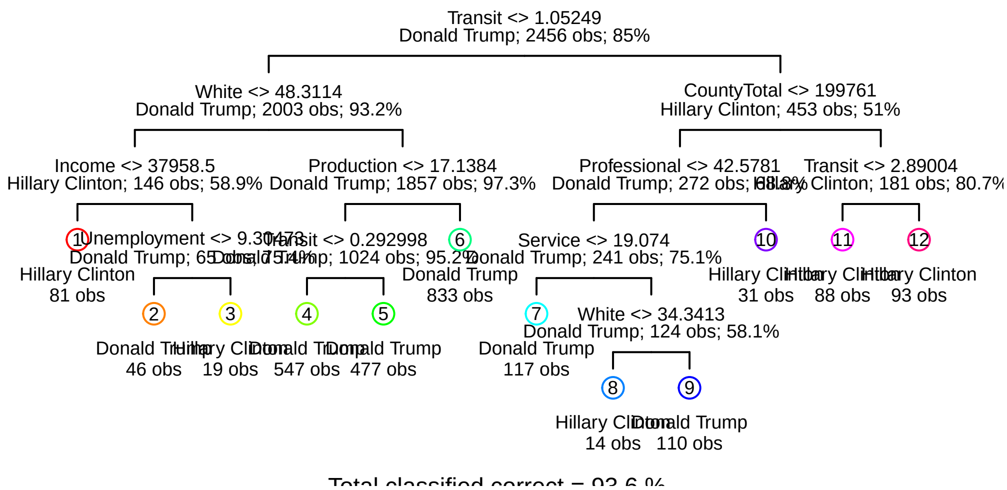
```
##
## Classification tree:
## tree(formula = candidate ~ ., data = trn.cl)
## Variables actually used in tree construction:
## [1] "Transit"      "White"        "Income"       "Unemployment"
## [5] "Production"   "CountyTotal"  "Professional" "Service"
## Number of terminal nodes:  12
## Residual mean deviance:  0.3598 = 879.3 / 2444
## Misclassification error rate: 0.06433 = 158 / 2456
```

```r
# using cross validation to find best size
cvtree <- cv.tree(cantree, rand=folds, FUN=prune.misclass)
best.size.cv <- min(cvtree$size[which(cvtree$dev==min(cvtree$dev))])
best.size.cv
```

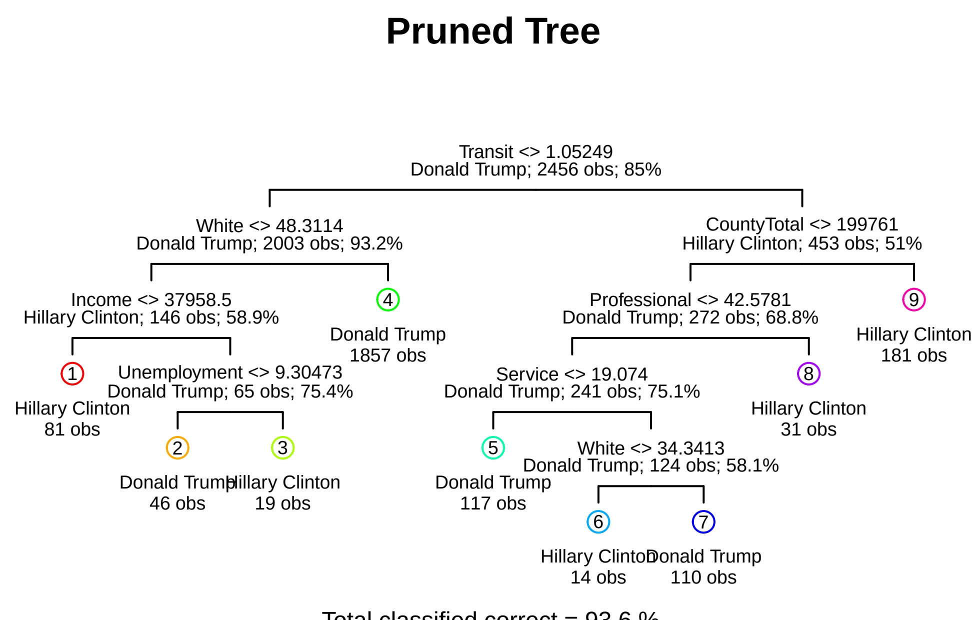
```
## [1] 9
```

```r
# pruning the tree based on cv size
prunedtree <- prune.tree(cantree, best=best.size.cv, method="misclass")

# plotting the two trees before and after pruning
draw.tree(cantree, nodeinfo=TRUE, cex=0.6)
title("Unpruned Tree")
```

Unpruned Tree




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




```
# training error
pred.cantree.train <- predict(prunedtree, trn.clX, type="class")
train.errort <- calc_error_rate(pred.cantree.train, trn.clY)

# test error
pred.cantree.test <- predict(prunedtree, tst.clX, type="class")
test.errort <- calc_error_rate(pred.cantree.test, tst.clY)

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

```
##          train.error test.error
## tree      0.06433225 0.08143322
## knn               NA         NA
## logistic          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`.

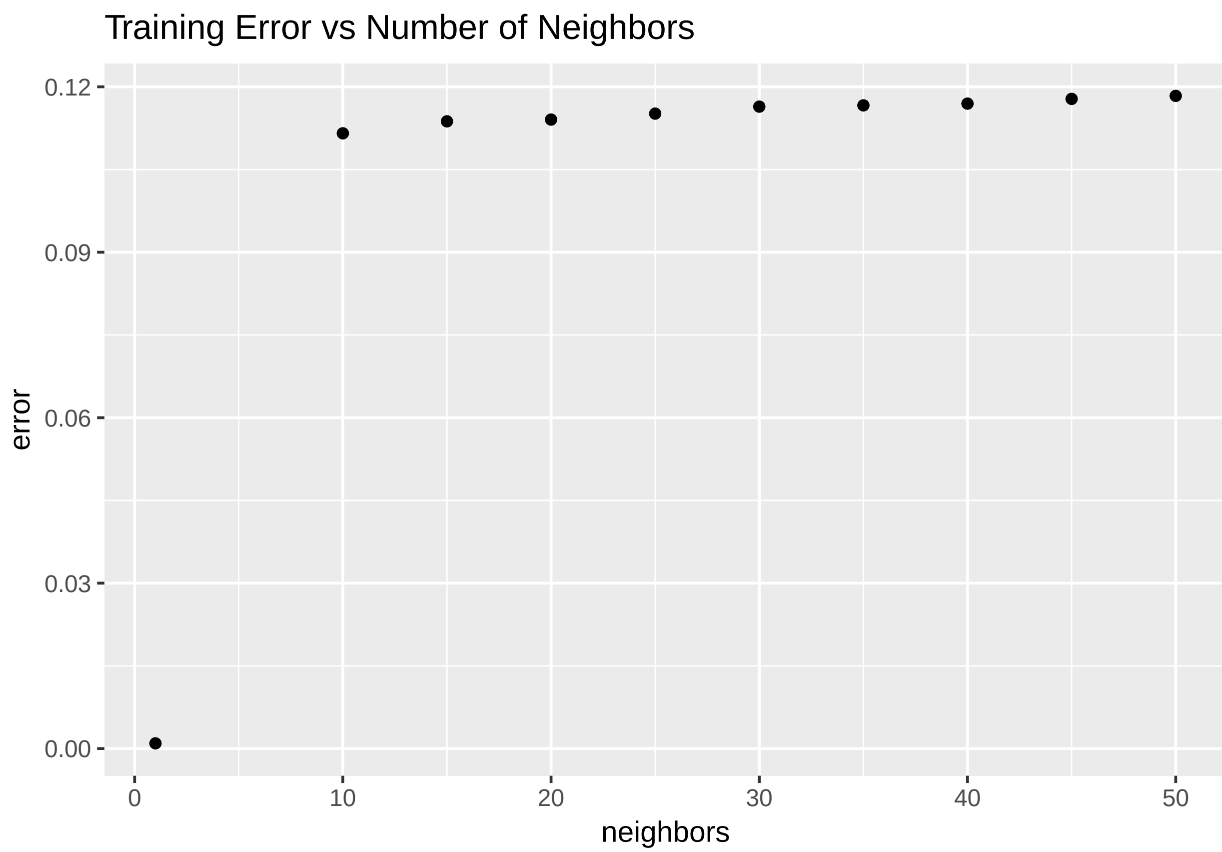
```
do.chunk <- function(chunkid, folddef, Xdat, Ydat, k){
    train = (folddef!=chunkid)

    Xtr = Xdat[train,]
    Ytr = Ydat[train]

    Xvl = Xdat[!train,]
    Yvl = Ydat[!train]
```

```r
    ## get classifications for current training chunks
    predYtr = knn(train = Xtr, test = Xtr, cl = Ytr, k = k)

    ## get classifications for current test chunk
    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))
}
```

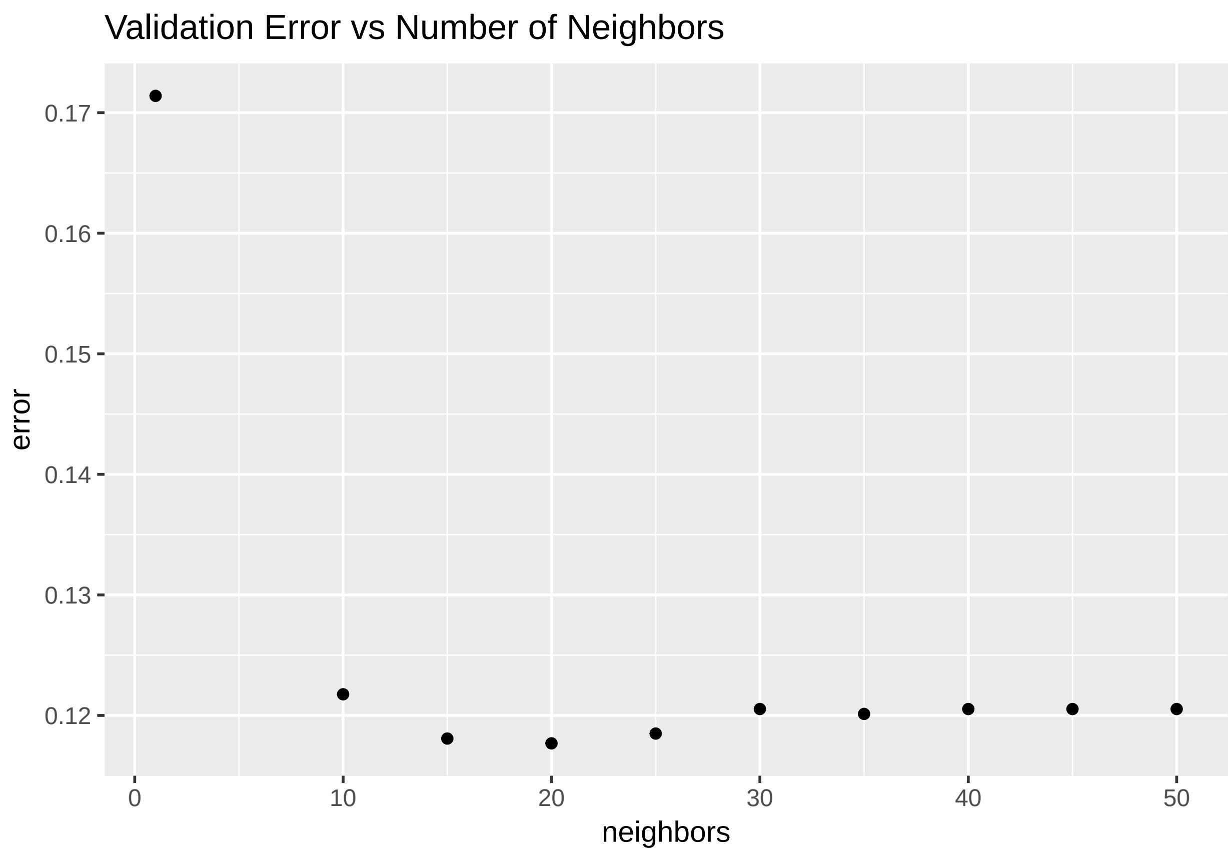
```r
# creating a vector of possible k values
kvec <- c(1, seq(10, 50, length.out=9))
kerrors <- NULL

# going through each possible k value
# and performing cross validaiton
for (j in kvec) {
    tve <- plyr::ldply(1:nfold, do.chunk, folddef=folds,
                 Xdat=trn.clX, Ydat=trn.clY, k=j)
    tve$neighbors <- j
    kerrors <- rbind(kerrors, tve)
}

# calculating test errors at each k
# (by taking mean of each cv result)
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))

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

bestk <- max(min.error$neighbors)
bestk
```

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

```r
# calculating training errors at each k
# (by taking mean of each cv result)
train.error.means <- errors %>%
    filter(variable=="train.error") %>%
    group_by(neighbors) %>%
    summarise_at(vars(error),funs(mean))

# plotting
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.clX, test=trn.clX, cl=trn.clY, k=bestk)
train.errork <- calc_error_rate(pred.knn.train, trn.clY)

# test errors
pred.knn.test <- knn(train=trn.clX, test=tst.clX, cl=trn.clY, k=bestk)
test.errork <- calc_error_rate(pred.knn.test, tst.clY)

# adding to records
records[2,1] <- train.errork
records[2,2] <- test.errork
records
```

```
##          train.error test.error
## tree      0.06433225 0.08143322
## knn       0.11074919 0.12377850
## logistic          NA         NA
```

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=3, ncol=2)
colnames(pca.records) = c("train.error","test.error")
rownames(pca.records) = c("tree","knn","lda")
```

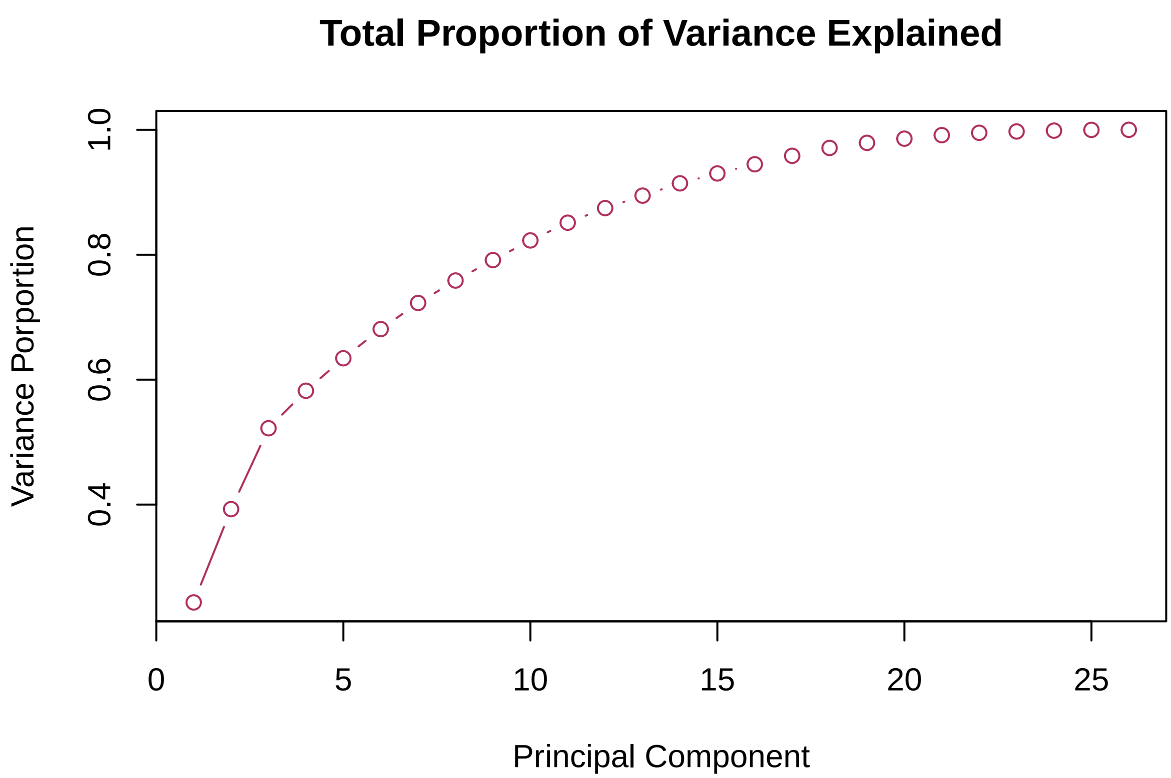
15. Compute principal components from the independent variables in training data. Then, determine the number

of minimum number of PCs needed to capture 90% of the variance. Plot proportion of variance explained.

```
trn.pca <- prcomp(trn.clX, 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
```

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



Fourteen principal components are needed to captures 90% of the variance.

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`.

```
# getting all the obsevations of the pca
trn.pc <- data.frame(trn.pca$x)
tr.pca <- trn.pc %>% mutate(candidate=trn.cl$candidate)

tst.pca <- prcomp(tst.clX, scale=TRUE)
tst.pc <- data.frame(tst.pca$x)
test.pca <- tst.pc %>% mutate(candidate=tst.cl$candidate)
```

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

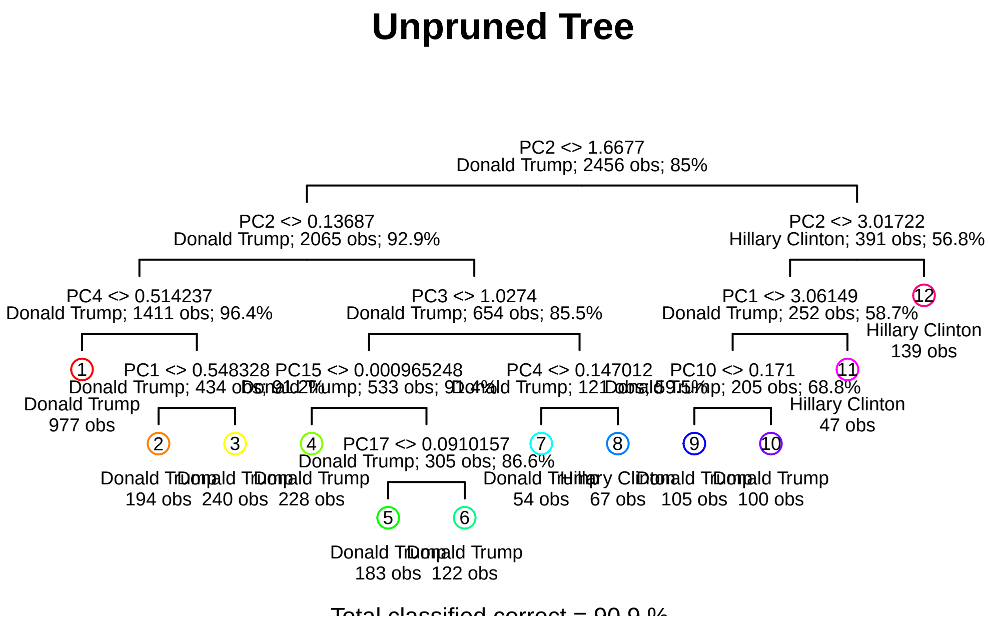
```r
# setting up the X and Y variables
# (for clarity in variable name)
tr.pcaX <- trn.pc
tr.pcaY <- tr.pca$candidate
test.pcaX <- tst.pc
test.pcaY <- test.pca$candidate

# creating the original tree
pcatree <- tree(candidate~.,tr.pca)
summary(pcatree)
```

```
##
## Classification tree:
## tree(formula = candidate ~ ., data = tr.pca)
## Variables actually used in tree construction:
## [1] "PC2"  "PC4"  "PC1"  "PC3"  "PC15" "PC17" "PC10"
## Number of terminal nodes:  12
## Residual mean deviance:  0.4685 = 1145 / 2444
## Misclassification error rate: 0.0908 = 223 / 2456
```

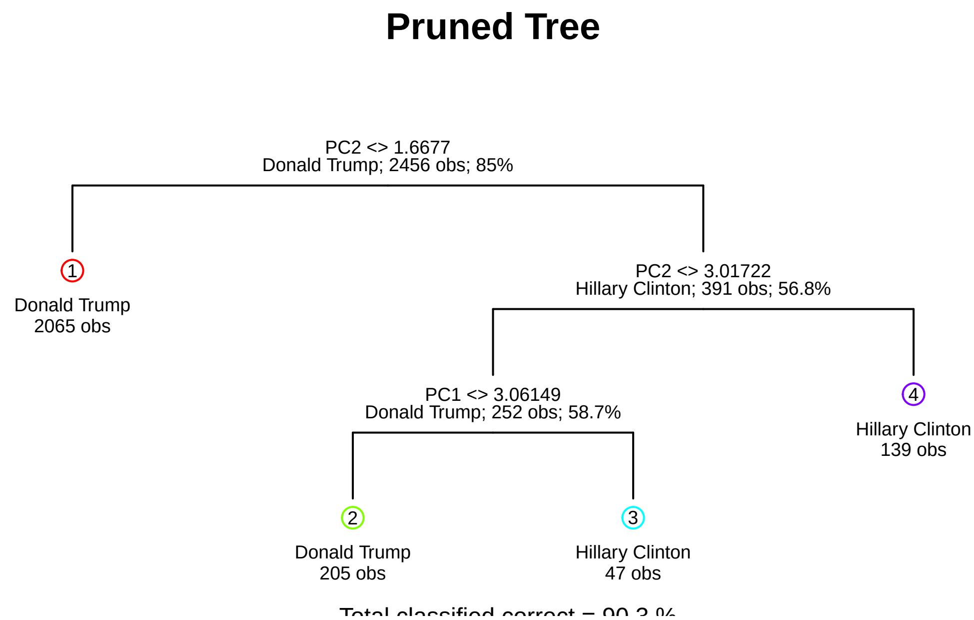
```r
# using cross validation to find best size
cvpcatree <- cv.tree(pcatree, rand=folds, FUN=prune.misclass)
best.size.cvpca <- min(cvpcatree$size[which(cvpcatree$dev==min(cvpcatree$dev))])
best.size.cvpca
```

```
## [1] 4
```

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

# plotting the two trees before and after pruning
draw.tree(pcatree, nodeinfo=TRUE, cex=0.6)
title("Unpruned Tree")
```




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




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

# training error
pred.pcatree.train <- predict(prunedpcatree, tr.pcaX, type="class")
train.errorpt <- calc_error_rate(pred.pcatree.train, tr.pcaY)

# test error
pred.pcatree.test <- predict(prunedpcatree, test.pcaX, type="class")
test.errorpt <- calc_error_rate(pred.pcatree.test, test.pcaY)

# putting errors into records
pcarecords[1,1] <- train.errorpt
pcarecords[1,2] <- test.errorpt
pcarecords
```

```
##      train.error test.error
## tree  0.09690554  0.1009772
## knn           NA         NA
```

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

```
# creating a vector of possible k values
kvecpca <- c(1, seq(10, 50, length.out=9))
kerrorspca <- NULL

# going through each possible k value
```

```r
# and performing cross validaiton
for (j in kvecpca) {
    tve <- plyr::ldply(1:nfold, do.chunk, folddef=folds,
                 Xdat=tr.pcaX, Ydat=tr.pcaY, k=j)
    tve$neighbors <- j
    kerrorspca <- rbind(kerrorspca, tve)
}

# calculating test errors at each k
# (by taking mean of each cv result)
errorspca <- melt(kerrorspca, id.vars=c("fold","neighbors"), value.name="error")
val.error.meanspca <- errorspca %>%
    filter(variable=="val.error") %>%
    group_by(neighbors) %>%
    summarise_at(vars(error),funs(mean))

# picking the best k
min.errorpca <- val.error.meanspca %>%
    filter(error==min(error))

bestk <- max(min.errorpca$neighbors)
bestk
```

```
## [1] 15
```

```r
# calculating training errors at each k
# (by taking mean of each cv result)
train.error.meanspca <- errorspca %>%
    filter(variable=="train.error") %>%
    group_by(neighbors) %>%
    summarise_at(vars(error),funs(mean))

# training errors
pred.pcaknn.train <- knn(train=tr.pcaX, test=tr.pcaX, cl=tr.pcaY, k=bestk)
train.errorpk <- calc_error_rate(pred.pcaknn.train, tr.pcaY)

# test errors
pred.pcaknn.test <- knn(train=tr.pcaX, test=test.pcaX, cl=tr.pcaY, k=bestk)
test.errorpk <- calc_error_rate(pred.pcaknn.test, test.pcaY)

# adding to records
pcarecords[2,1] <- train.errorpk
pcarecords[2,2] <- test.errorpk
pcarecords
```

```
##      train.error test.error
## tree  0.09690554  0.1009772
## knn   0.06636808  0.1123779
```

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).

Because it is so difficult to predict election outcomes due to the number of factors involved, this project showed us that you must determine the most influential factors in order to form the most accurate predictions.

The data itself had some discrepancies, where some counties were split into 2 subcounties, some cities were classified as counties, and a few counties had no data in the “county name” field. These discrepancies made it difficult to identify the voting outcome of those counties, possibly skewing the data.

In our poverty levels visualization, we found that although Hillary had fewer counties vote in her favor, her counties on average had a lower rate of poverty than the counties that voted in Trump’s favor. This is consistent with our analyses because it tells us that Trump’s voters on average had a lower income, and we found in our PCA results that income per capita was the most influential factor on voting outcome.

The PCA analysis showed us that the most influential variables in the census data on voting outcomes were income per capita and income error on the county level, and income per capita and method of transportation on the subcounty level. On the subcounty level, we were surprised to find that the percentage of the population that commuted via public transportation was so influential; one reason for this could be that voters who take public transportation are in a lower income bracket, and therefore this variable was related to income per capita.

There were also some discrepancies in the cluster analysis, when we looked at San Mateo county as an example. San Mateo county, which we know to be a historically Democrat-voting county was placed into cluster 1 with many Trump counties, when using the first 5 principal components from the PCA analysis. This method appears to incorrectly classify the county of San Mateo, possibly due to the fact that income per capita was the first principal component. This could be because income per capita is more influential with Trump voters than with Clinton voters, so it misclassified some of the counties that voted for Clinton by placing more importance on income per capita.

If possible, we could collect more data on how each county voted in the 2012 election, to see how many counties switched parties from 2012 to 2016. The 2016 election was relatively unusual, so it would be interesting to compare these results to the results of a similar analysis on the 2012 election, to see if the most influential factors are the same across each election, or if it is dependent on the candidates.

Taking it further

20. Propose and tackle at least one interesting question. Be creative! We will use logistic regression as a classification method for candidate.

```
# using logistic regression on full data set (2 classes)
glm.fit <- glm(candidate~., data = trn.cl, family = binomial)
```

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

```
summary(glm.fit)
```

```
##
## Call:
## glm(formula = candidate ~ ., family = binomial, data = trn.cl)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -3.7362  -0.2705  -0.1133  -0.0407   3.5782
##
## Coefficients:
##                    Estimate Std. Error z value Pr(>|z|)
## (Intercept)      -1.158e+01  9.701e+00  -1.193 0.232686
## Men               8.346e-02  5.386e-02   1.550 0.121229
## White            -2.147e-01  6.550e-02  -3.278 0.001047 **
## Minority         -8.369e-02  6.274e-02  -1.334 0.182229
```

```
## Citizen          1.069e-01  3.049e-02   3.508 0.000452 ***
## Income          -7.558e-05  2.752e-05  -2.747 0.006016 **
## IncomeErr       -3.703e-05  6.269e-05  -0.591 0.554786
## IncomePerCap     2.669e-04  6.717e-05   3.974 7.07e-05 ***
## IncomePerCapErr -2.759e-04  1.308e-04  -2.109 0.034904 *
## Poverty          2.083e-02  4.110e-02   0.507 0.612267
## ChildPoverty    -7.147e-03  2.551e-02  -0.280 0.779357
## Professional     2.739e-01  3.972e-02   6.897 5.32e-12 ***
## Service          3.590e-01  4.953e-02   7.248 4.23e-13 ***
## Office           9.549e-02  4.801e-02   1.989 0.046688 *
## Production       1.811e-01  4.317e-02   4.196 2.72e-05 ***
## Drive           -2.542e-01  5.393e-02  -4.714 2.43e-06 ***
## Carpool         -2.441e-01  6.681e-02  -3.653 0.000259 ***
## Transit         -1.745e-02  1.022e-01  -0.171 0.864480
## OtherTransp     -9.864e-02  1.010e-01  -0.976 0.328820
## WorkAtHome      -2.093e-01  7.920e-02  -2.642 0.008238 **
## MeanCommute      6.120e-02  2.494e-02   2.454 0.014133 *
## Employed         1.650e-01  3.287e-02   5.021 5.14e-07 ***
## PrivateWork      8.717e-02  2.216e-02   3.934 8.36e-05 ***
## SelfEmployed     7.917e-03  4.674e-02   0.169 0.865516
## FamilyWork      -1.189e+00  4.080e-01  -2.914 0.003564 **
## Unemployment     1.813e-01  3.811e-02   4.758 1.96e-06 ***
## CountyTotal      3.666e-07  4.036e-07   0.908 0.363716
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 2074.96  on 2455  degrees of freedom
## Residual deviance:  853.13  on 2429  degrees of freedom
## AIC: 907.13
##
## Number of Fisher Scoring iterations: 7
```

```r
#training error
glm.probs.train <- predict(glm.fit, trn.clX, type="response")
glm.pred.train <- rep("Donald Trump", length(trn.clY))
glm.pred.train[glm.probs.train > 0.5]="Hillary Clinton"
train.errorl <- calc_error_rate(glm.pred.train, trn.clY)

#test error
glm.probs.test <- predict(glm.fit, tst.clX, type="response")
glm.pred.test <- rep("Donald Trump", length(tst.clY))
glm.pred.test[glm.probs.test > 0.5]="Hillary Clinton"
test.errorl <- calc_error_rate(glm.pred.test, tst.clY)

# adding to records
records[3,1] <- train.errorl
records[3,2] <- test.errorl
records
```

```
##          train.error test.error
## tree      0.06433225 0.08143322
## knn       0.11074919 0.12377850
## logistic  0.06392508 0.07654723
```

Logistic regression has the lowest misclassification error on the test set. This indicates that the decision boundary for the candidates is probably on the linear side. So since KNN is a completely nonparametric approach, and this data appears to have a linear decision boundary based on our results, we expect KNN to not perform as well as logistic regression (it is a too flexible approach). This is the same for classification trees. If the relationship between the variables and the response is well approximated by a linear model then an approach such as logistic regression is expected to outperform the decision tree method, and this is precisely what happens. But, the test errors between these two methods are not too different, and so we may prefer to use the decision tree method because of its interpretability and visualization.