131Final

Jenny Zhang (4304531) and Annie Ma (4354379)

December 12, 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.

Background

The presidential election in 2012 did not come as a surprise. Some correctly predicted the outcome of the election correctly including Nate Silver (https://en.wikipedia.org/wiki/Nate_Silver), and many speculated his approach (https://www.theguardian.com/science/grrlscientist/2012/nov/08/nate-sliver-predict-us-election).

Despite the success in 2012, the 2016 presidential election came as a big surprise (https://fivethirtyeight.com/features/the-polls-missed-trump-we-asked-pollsters-why/) 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?

Answer: Voter behavior prediction is a hard problem because what people change their minds constantly. We can use variables such as race and wealth to predict voter behavior. So, for example, we can predict that a larger proportion of black voters would vote for Obama. Although we can use old data to predict new data, it fails to miss out context because voter behavior changes over time. For example, if the unemployment rate falls, the incumbent tends to become more popular and expected votes increases. Voters can last minute change their votes leading up to the election. This is an inherent weakness of models used to predict election results. It is difficult to forecast election because there are so many small changes in voter behavior that can cause dramatic changes in election results.

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

Answer: Silver looked at the full range of probabilities instead of the maximum probability. For each date, he calculated the probability of support. For the following date, he used the model for the actual support to calculate the probability that support has shifted. This prediction model is based on Bayes' Theorem. Silver made use of the full range of probabilities to calculate a probability for each percentage support for Obama in each state. In other words, he calculate the probability that Obama would win in each state if the election day was called on that day.

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

Answer: In 2016, both candidates were controversial causing it to be difficult to predict the election outcomes. Many election models seriously underestimated Trump's chances of victory. Voters preferences are strongly influenced by news events. For example, the news about the Comey Letter prompted many late-deciding voters to vote for Trump. The media gave disproportionate attention to the Comey Letter, causing a sharp decline in Clinton's polling with a large enough magnitude to change the election results. However, many models were too slow to self-correct when new polling results contradicts pre-existing overconfidence that Clinton would win the election. Demographics also gave Trump a huge advantage in the Electoral College because his voters were disportionality concentrated in the swing states. To make future predictions better, we think models need to quickly recognize meaningful shifts in the polls. Voter demographic information and the uncertainty of the large amount of third-party and undecided voters should be taken into account of these models.

Data

```
## read data and convert candidate from string to factor
election.raw <- read_delim("data/election/election.csv", delim = ",") %>% mutate(candidate=as.factor(candidate))
```

```
## Parsed with column specification:
## cols(
##   county = col_character(),
##   fips = col_character(),
##   candidate = col_character(),
##   state = col_character(),
##   votes = col_double()
## )
```

```
census_meta <- read_delim("data/census/metadata.csv", delim = ";", col_names = FALSE)
```

```
## Parsed with column specification:
## cols(
##   X1 = col_character(),
##   X2 = col_character(),
##   X3 = col_character()
## )
```

```
census <- read_delim("data/census/census.csv", delim = ",")
```

```
## Parsed with column specification:
## cols(
##   .default = col_double(),
##   State = col_character(),
##   County = col_character()
## )
```

```
## See spec(...) for full column specifications.
```

Election data

The meaning of each column in `election.raw` is clear except `fips`. The acronym is short for Federal Information Processing Standard (https://en.wikipedia.org/wiki/FIPS_county_code).

In our dataset, `fips` values denote the area (US, state, or county) that each row of data represent. For example, `fips` value of 6037 denotes Los Angeles County.

```
kable(election.raw %>% filter(county == "Los Angeles County"), table.attr = "style = \"color: black;\"")  %>% kab
le_styling(bootstrap_options = c("striped", "hover", "condensed", "responsive"), full_width=FALSE)
```

county	fips	candidate	state	votes
Los Angeles County	6037	Hillary Clinton	CA	2464364
Los Angeles County	6037	Donald Trump	CA	769743
Los Angeles County	6037	Gary Johnson	CA	88968
Los Angeles County	6037	Jill Stein	CA	76465
Los Angeles County	6037	Gloria La Riva	CA	21993

Some rows in `election.raw` are summary rows and 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.

4. Report the dimension of `election.raw` after removing rows with `fips=2000`. Provide a reason for excluding them. Please make sure to use the same name `election.raw` before and after removing those observations.

Answer: The dimension of the `election.raw` after removing rows with `fips=2000` is 18345 by 5. Looking at the data, it looks like the fips value of 2000 is already represented by Alaska, so we want to remove it to prevent duplicates.

```
#4
dim(election.raw)
```

```
## [1] 18351     5
```

```
election.raw <- election.raw %>%
  filter(fips != 2000)
dim(election.raw)
```

```
## [1] 18345     5
```

Census data

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

State	County	TotalPop	Men	Women	Hispanic	White	Black	Native	Asian	Pacific	Citizen	Income	IncomeErr	IncomePerCap	IncomeP
Alabama	Autauga	1948	940	1008	0.9	87.4	7.7	0.3	0.6	0.0	1503	61838	11900	25713	
Alabama	Autauga	2156	1059	1097	0.8	40.4	53.3	0.0	2.3	0.0	1662	32303	13538	18021	
Alabama	Autauga	2968	1364	1604	0.0	74.5	18.6	0.5	1.4	0.3	2335	44922	5629	20689	
Alabama	Autauga	4423	2172	2251	10.5	82.8	3.7	1.6	0.0	0.0	3306	54329	7003	24125	
Alabama	Autauga	10763	4922	5841	0.7	68.5	24.8	0.0	3.8	0.0	7666	51965	6935	27526	
Alabama	Autauga	3851	1787	2064	13.1	72.9	11.9	0.0	0.0	0.0	2642	63092	9585	30480	

Census data: column metadata

Column information is given in `metadata`.

Data wrangling

5. 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`.

```
#5
election_federal <- election.raw %>%
  filter(fips == "US")

election_state <- election.raw %>%
  filter(fips != "US" & fips == state)

election <- election.raw %>%
  filter(!is.na(county))
```

6. How many named presidential candidates were there in the 2016 election? Draw a bar chart of all votes received by each candidate.You can split this into multiple plots or may prefer to plot the results on a log scale. Either way, the results should be clear and legible!

Answer: There are 31 named presidential candidates in the 2016 election.

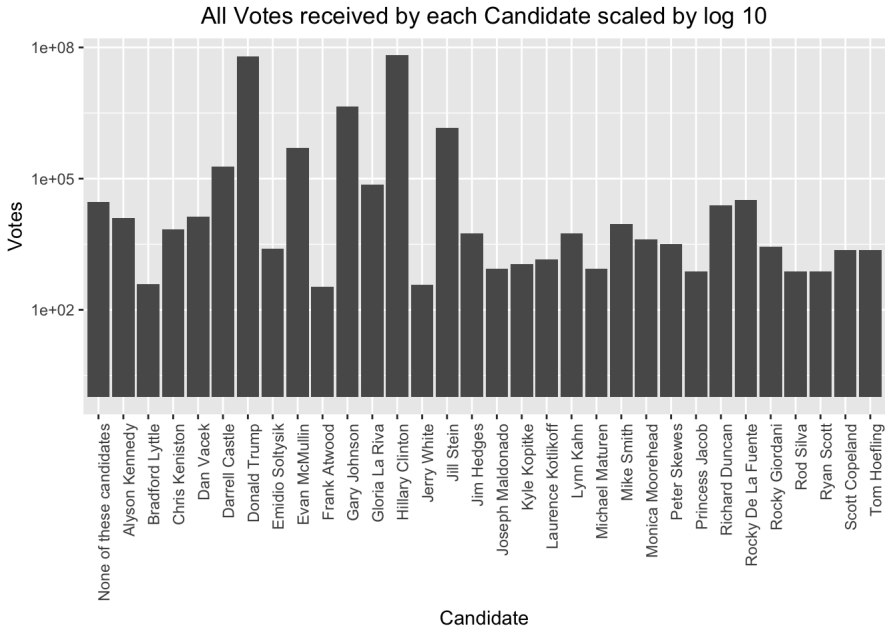
```
#6
unique(election_federal$candidate) %>% kable(col.names = 'Candidates', table.attr = "style = \"color: black;\"")
 %>% kable_styling(bootstrap_options = c("striped", "hover", "condensed", "responsive"), full_width=FALSE)
```

Candidates
Donald Trump
Hillary Clinton
Gary Johnson
Jill Stein
Evan McMullin
Darrell Castle
Gloria La Riva
Rocky De La Fuente
None of these candidates
Richard Duncan
Dan Vacek
Alyson Kennedy
Mike Smith
Chris Keniston
Lynn Kahn
Jim Hedges
Monica Moorehead
Peter Skewes
Rocky Giordani
Emidio Soltysik
Scott Copeland
Tom Hoefling
Laurence Kotlikoff
Kyle Kopitke
Joseph Maldonado
Michael Maturen
Rod Silva
Princess Jacob
Ryan Scott
Bradford Lyttle
Jerry White
Frank Atwood

```
length(unique(election_federal$candidate))
```

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

```
ggplot(data = election_federal) +
  geom_col(mapping = aes(x = candidate, y = votes)) +
  scale_y_log10() +
  ggtitle("All Votes received by each Candidate scaled by log 10") +
  ylab("Votes") + xlab("Candidate") +
  theme(axis.text.x = element_text(angle = 90, hjust = 1), plot.title = element_text(hjust = 0.5))
```



7. 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).

```
#7
county_winner <- election %>% group_by(fips, add = TRUE) %>% mutate(total = sum(votes)) %>% mutate(pct = votes/to
tal) %>% top_n(1, pct)

state_winner <- election_state %>% group_by(fips, add = TRUE) %>% mutate(total = sum(votes)) %>% mutate(pct = vot
es/total) %>% top_n(1, pct)

county_winner
```

```
## # A tibble: 3,112 x 7
## # Groups:   fips [3,112]
##    county              fips  candidate       state   votes   total   pct
##    <chr>               <chr> <fct>           <chr>   <dbl>   <dbl> <dbl>
##  1 Los Angeles County 6037  Hillary Clinton CA    2464364 3421533 0.720
##  2 Cook County        17031 Hillary Clinton IL    1611946 2156395 0.748
##  3 Maricopa County    4013  Donald Trump    AZ     747361 1536743 0.486
##  4 Harris County      48201 Hillary Clinton TX     707914 1305434 0.542
##  5 San Diego County   6073  Hillary Clinton CA     735476 1291078 0.570
##  6 Orange County      6059  Hillary Clinton CA     609961 1186203 0.514
##  7 King County        53033 Hillary Clinton WA     718322  998499 0.719
##  8 Miami-Dade County  12086 Hillary Clinton FL     624146  980204 0.637
##  9 Broward County     12011 Hillary Clinton FL     553320  831950 0.665
## 10 Kings County       36047 Hillary Clinton NY     640553  800393 0.800
## # … with 3,102 more rows
```

```
state_winner
```

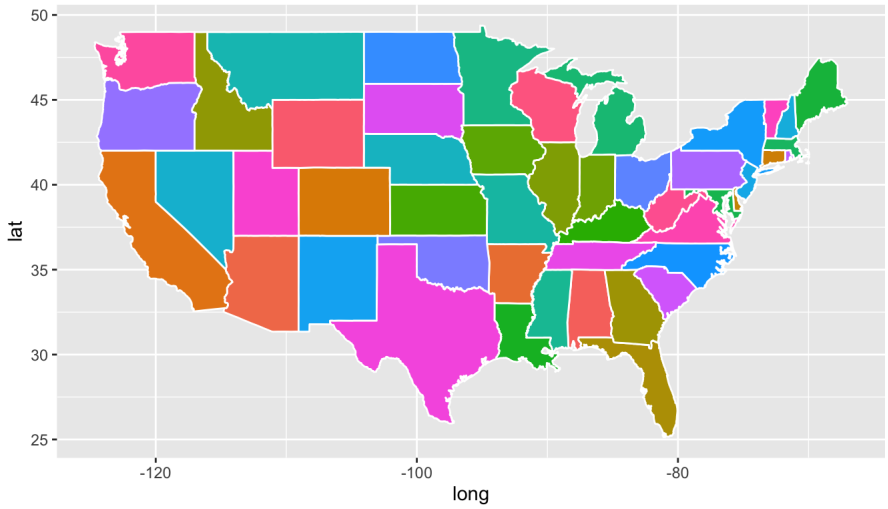
```
## # A tibble: 51 x 7
## # Groups:   fips [51]
##    county fips  candidate       state   votes    total   pct
##    <chr>  <chr> <fct>           <chr>   <dbl>    <dbl> <dbl>
##  1 <NA>   CA    Hillary Clinton CA    8753788 14060856 0.623
##  2 <NA>   FL    Donald Trump    FL    4617886  9419886 0.490
##  3 <NA>   TX    Donald Trump    TX    4685047  8917965 0.525
##  4 <NA>   NY    Hillary Clinton NY    4556124  7660190 0.595
##  5 <NA>   PA    Donald Trump    PA    2970733  6115402 0.486
##  6 <NA>   IL    Hillary Clinton IL    3090729  5523142 0.560
##  7 <NA>   OH    Donald Trump    OH    2841005  5480173 0.518
##  8 <NA>   MI    Donald Trump    MI    2279543  4790329 0.476
##  9 <NA>   NC    Donald Trump    NC    2362631  4682073 0.505
## 10 <NA>   GA    Donald Trump    GA    2089104  4092373 0.510
## # … with 41 more rows
```

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.

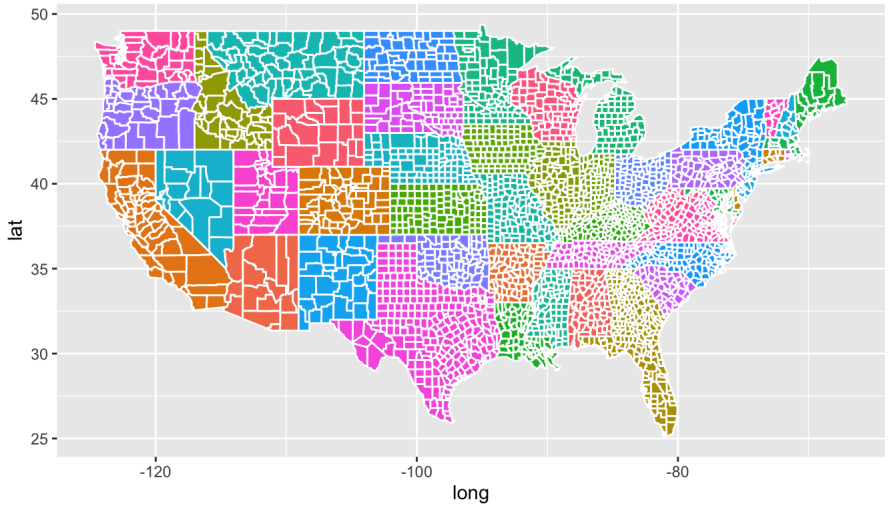
```
states <- map_data("state")
ggplot(data = states) +
  geom_polygon(aes(x = long, y = lat, fill = region, group = group), color = "white") +
  coord_fixed(1.3) +
  guides(fill=FALSE)  # color legend is unnecessary and takes too long
```



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

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

```
#8
counties = map_data("county")
ggplot(data = counties) +
  geom_polygon(aes(x = long, y = lat, fill = region, group = group), color = "white") +
  coord_fixed(1.3) +
  guides(fill=FALSE)  # color legend is unnecessary and takes too long
```

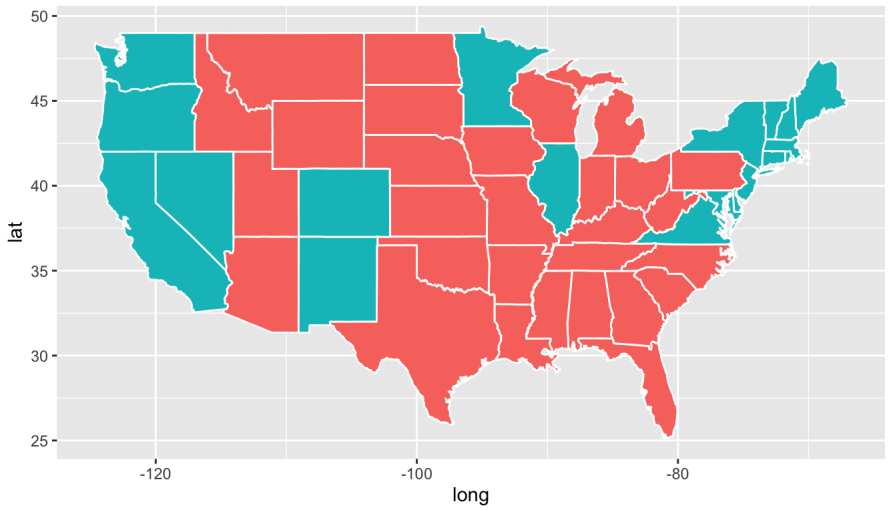


9. Now color the map by the winning candidate for each state. First, combine `states` variable and `state_winner` we created earlier using `left_join()`. Note that `left_join()` needs to match up values of states to join the tables. A call to `left_join()` takes all the values from the first table and looks for matches in the second table. If it finds a match, it adds the data from the second table; if not, it adds missing values. Here, we'll be combing the two datasets based on state name. However, the state names are in different formats in the two tables: 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 (https://www.nytimes.com/elections/results/president).

```
#9
states <- states %>%
  mutate(fips = state.abb[match(region, tolower(state.name))])
states <- left_join(states, state_winner)
```

```
## Joining, by = "fips"
```

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



10. 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 (https://www.nytimes.com/elections/results/president).

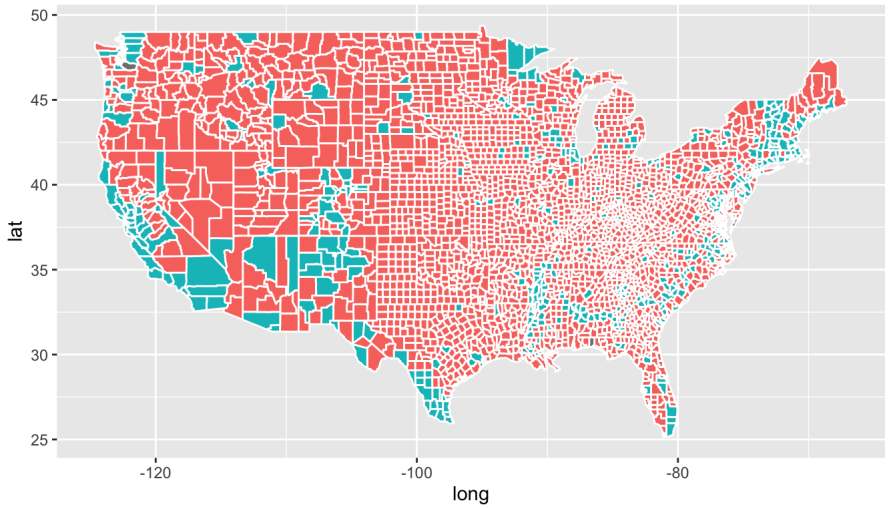
```
#10
county.fips <- maps::county.fips %>% separate(polyname, c("region", "subregion"), sep = ",")
counties <- left_join(counties, county.fips)
```

```
## Joining, by = c("region", "subregion")
```

```
counties$fips<-as.character(counties$fips)
counties <- left_join(counties, county_winner)
```

```
## Joining, by = "fips"
```

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

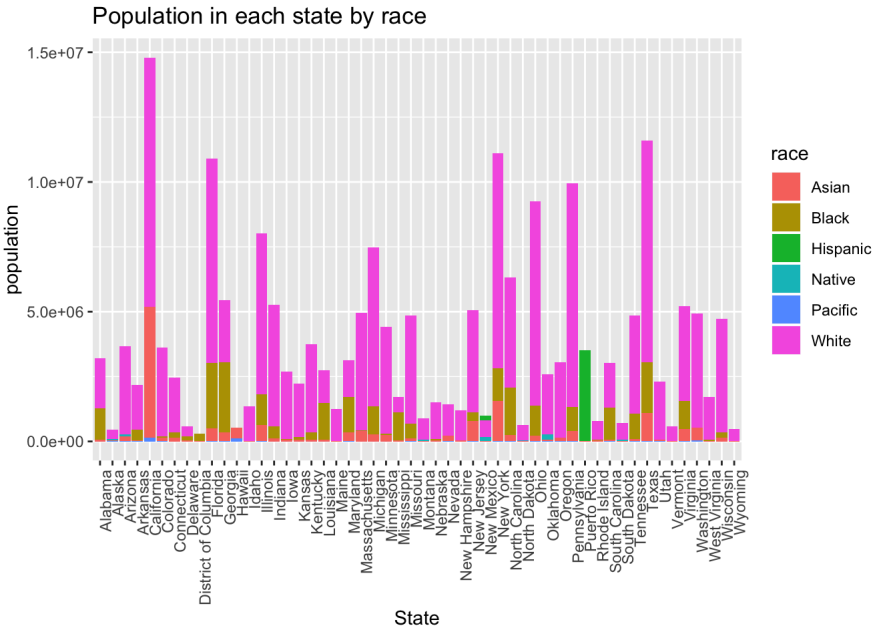


11. Create a visualization of your choice using `census` data. Many exit polls noted that demographics played a big role in the election (https://fivethirtyeight.com/features/demographics-not-hacking-explain-the-election-results/). Use this Washington Post article (https://www.washingtonpost.com/graphics/politics/2016-election/exit-polls/) and this R graph gallery (https://www.r-graph-gallery.com/) for ideas and inspiration.

```
#11
census.race <- census %>%
  drop_na %>%
  mutate(Hispanic.num = Hispanic/100 * TotalPop,
              White.num = White/100 * TotalPop,
            Black.num = Black/100 * TotalPop,
            Native.num = Native/100 * TotalPop,
            Asian.num = Asian/100 * TotalPop,
            Pacific.num = Pacific/100 *TotalPop) %>%
  select(State:Women, Hispanic.num:Pacific.num) %>%
  group_by(State) %>%
  summarize_at(vars(TotalPop:Pacific.num), funs(sum(.))) %>%
  gather(key = 'race_population', value = 'population', Hispanic.num:Pacific.num) %>%
  separate(race_population, c("race", "num")) %>%
  select(-num)
```

```
## Warning: funs() is soft deprecated as of dplyr 0.8.0
## Please use a list of either functions or lambdas:
##
##   # Simple named list:
##   list(mean = mean, median = median)
##
##   # Auto named with `tibble::lst()`:
##   tibble::lst(mean, median)
##
##   # Using lambdas
##   list(~ mean(., trim = .2), ~ median(., na.rm = TRUE))
## This warning is displayed once per session.
```

```
ggplot(census.race, aes(State, population, fill = race)) +
  geom_bar(position = "identity", stat = "identity") +
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  ggtitle("Population in each state by race")
```



12. 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.
    - *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`*** :

```
#12
census.del = census %>%
  drop_na %>%
  mutate(Men = (Men/TotalPop)*100,
         Employed = (Employed/TotalPop)*100,
         Citizen = (Citizen/TotalPop)*100) %>%
  mutate(Minority = Hispanic + Black + Native + Asian + Pacific) %>%
  select(-Hispanic, -Black, -Native, -Asian, -Pacific) %>%
  select(-Walk, -PublicWork, -Construction)

census.subct = census.del %>%
  group_by(State, County) %>%
  add_tally(TotalPop, name = "CountyTotal") %>%
  mutate(weight = TotalPop/CountyTotal)

census.ct = census.subct %>%
  summarize_at(vars(Men:Minority), funs(sum(. * weight)))
```

```
## Warning: funs() is soft deprecated as of dplyr 0.8.0
## Please use a list of either functions or lambdas:
##
##   # Simple named list:
##   list(mean = mean, median = median)
##
##   # Auto named with `tibble::lst()`:
##   tibble::lst(mean, median)
##
##   # Using lambdas
##   list(~ mean(., trim = .2), ~ median(., na.rm = TRUE))
## This warning is displayed once per session.
```

```
kable(head(census.ct),table.attr = "style = \"color: black;\"") %>% kable_styling(bootstrap_options = c("striped"
, "hover", "condensed", "responsive"), full_width=FALSE)
```

State	County	Men	Women	White	Citizen	Income	IncomeErr	IncomePerCap	IncomePerCapErr	Poverty	ChildPoverty	Profe
Alabama	Autauga	48.43266	3348.805	75.78823	73.74912	51696.29	7771.009	24974.50	3433.674	12.91231	18.70758	3
Alabama	Baldwin	48.84866	3934.167	83.10262	75.69406	51074.36	8745.050	27316.84	3803.718	13.42423	19.48431	3
Alabama	Barbour	53.82816	1491.941	46.23159	76.91222	32959.30	6031.065	16824.22	2430.189	26.50563	43.55962	2
Alabama	Bibb	53.41090	2930.106	74.49989	77.39781	38886.63	5662.358	18430.99	3073.599	16.60375	27.19708	2

State	County	Men	Women	White	Citizen	Income	IncomeErr	IncomePerCap	IncomePerCapErr	Poverty	ChildPoverty	Profe
Alabama	Blount	49.40565	3562.081	87.85385	73.37550	46237.97	8695.786	20532.27	2052.055	16.72152	26.85738	2
Alabama	Bullock	53.00618	1968.034	22.19918	75.45420	33292.69	9000.345	17579.57	3110.645	24.50260	37.29116	1

Dimensionality reduction

13. 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. Discuss whether you chose to center and scale the features before running PCA and the reasons for your choice. What are the three features with the largest absolute values of the first principal component? Which features have opposite signs and what does that mean about the correaltion between these features?

Answer: I chose to scale and center the features before running PCA because scaling puts all the variables on the same scale and we don't have to worry about the units of the variables. This is good practice when the predictor variables are of mixed types. The 3 features with the largest absolute values for the first principal component for county are IncomePerCap. ChildPoverty, and Poverty. The 3 features with the largest absolute values for the first principal component for sub-county are IncomePerCap. Professional, and Poverty. We will look at the 3 PC1 features with the largest absolute value and compare their signs. Same sign indicates that these features are positively or negatively correlated. In county, ChildPoverty and Poverty share the same (positive) sign but IncomePerCap has an opposite (negative) sign. This means there is a positive correlation between ChildPoverty and Poverty. This means the higher the child poverty rate, the higher the poverty rate. In sub-county, IncomePerCap and Professional share the same (positive) sign but poverty has an opposite (negative) sign. This means there is a positive correlation between IncomePerCap and Professional. This means the higher the Income per capita, the higher the number of professionals,

```
#13
ct.pr.out=prcomp(census.ct[c(-1,-2)], center = TRUE, scale=TRUE)
ct.pc = ct.pr.out$x[,1:2]
subct.pr.out=prcomp(census.subct[c(-1,-2)], center = TRUE, scale=TRUE)
subct.pc = subct.pr.out$x[,1:2]

ct.loadings = ct.pr.out$rotation[,1]
ct.loadings
```

```
##             Men           Women           White         Citizen          Income
##    -0.008040684     0.020520051    -0.224008909    -0.006142070    -0.318412310
##       IncomeErr    IncomePerCap IncomePerCapErr         Poverty    ChildPoverty
##    -0.168174737    -0.350707737    -0.193815519     0.342177906     0.343376381
##    Professional         Service          Office      Production           Drive
##    -0.249711502     0.181627540     0.017201815     0.118433735     0.095470004
##         Carpool         Transit     OtherTransp      WorkAtHome     MeanCommute
##     0.076510547    -0.069882623     0.009626341    -0.176797870     0.060376412
##        Employed     PrivateWork    SelfEmployed      FamilyWork    Unemployment
##    -0.327605325    -0.054175228    -0.100441841    -0.050313668     0.291420523
##        Minority
##     0.227494070
```

```
ct.absloadings = abs(ct.loadings)
print(head(sort(ct.absloadings, decreasing = TRUE)))
```

```
## IncomePerCap ChildPoverty      Poverty     Employed       Income Unemployment
##    0.3507077    0.3433764    0.3421779    0.3276053    0.3184123    0.2914205
```

```
subct.loadings = subct.pr.out$rotation[,1]
subct.loadings
```

```
##        TotalPop             Men           Women           White         Citizen
##      0.03821891      0.01642198      0.03684653      0.23953202      0.15909760
##          Income       IncomeErr    IncomePerCap IncomePerCapErr         Poverty
##      0.30301762      0.19852612      0.31768263      0.21129408     -0.30506843
##    ChildPoverty    Professional         Service          Office      Production
##     -0.29821720      0.30629553     -0.26896690     -0.01310184     -0.20692162
##           Drive         Carpool         Transit     OtherTransp      WorkAtHome
##      0.07952107     -0.16218331     -0.05756426     -0.04559322      0.17231486
##     MeanCommute        Employed     PrivateWork    SelfEmployed      FamilyWork
##      0.01080909      0.22134344     -0.04145183      0.06885062      0.01483659
##    Unemployment        Minority     CountyTotal          weight
##     -0.25305991     -0.24112912     -0.02113679     -0.01209501
```

```
subct.absloadings = abs(subct.loadings)
print(head(sort(subct.absloadings, decreasing = TRUE)))
```

```
## IncomePerCap Professional      Poverty       Income ChildPoverty      Service
##    0.3176826    0.3062955    0.3050684    0.3030176    0.2982172    0.2689669
```

14. Determine the number of minimum number of PCs needed to capture 90% of the variance for both the county and sub-county analyses. Plot proportion of variance explained (PVE) and cumulative PVE for both county and sub-county analyses.

Answer: For county analyses, 13 principal components are needed to capture 90% of the variance. For sub-county analyses, 16 principal components are needed to capture 90% of the variance.

```
#14
subct.pr.var = subct.pr.out$sdev^2
subct.pr.var
```

```
##  [1] 7.372470286 3.486572752 2.471012941 1.800739411 1.428424954 1.301623730
##  [7] 1.209528287 1.019968170 0.953482667 0.909687889 0.856919589 0.788565175
## [13] 0.718898346 0.686089685 0.578392315 0.564988020 0.518985575 0.450005461
## [19] 0.437948293 0.383430418 0.346343259 0.269411167 0.209020280 0.089451990
## [25] 0.053452098 0.050373574 0.037320510 0.003611628 0.003281528
```

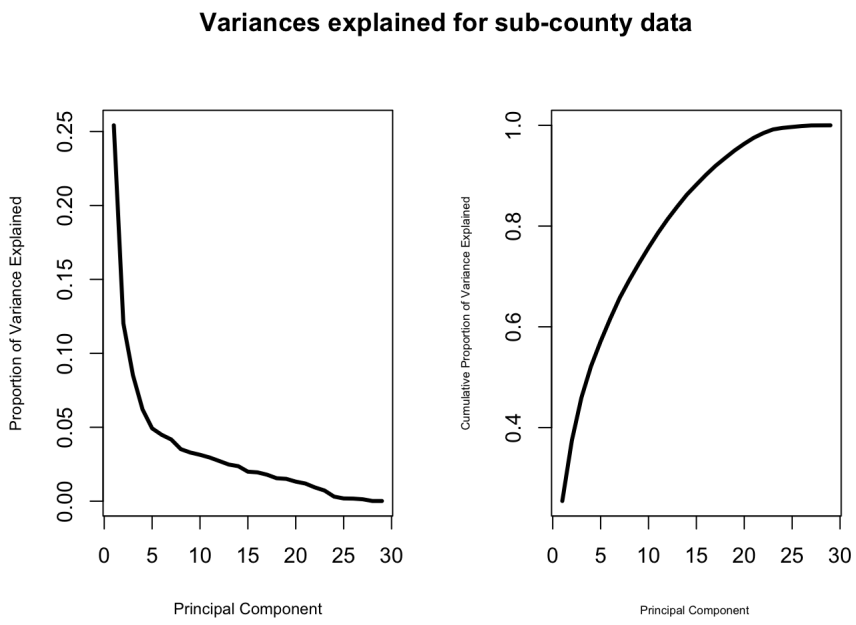
```
subct.pve = subct.pr.var/sum(subct.pr.var)
subct.pve
```

```
##  [1] 0.2542231133 0.1202266466 0.0852073428 0.0620944625 0.0492560329
##  [6] 0.0448835769 0.0417078720 0.0351713162 0.0328787126 0.0313685479
## [11] 0.0295489513 0.0271919026 0.0247895981 0.0236582650 0.0199445626
## [16] 0.0194823455 0.0178960543 0.0155174297 0.0151016653 0.0132217385
## [21] 0.0119428710 0.0092900403 0.0072075959 0.0030845514 0.0018431758
## [26] 0.0017370198 0.0012869141 0.0001245389 0.0001131561
```

```
subct.cumulative_pve = cumsum(subct.pve)
subct.cumulative_pve
```

```
##  [1] 0.2542231 0.3744498 0.4596571 0.5217516 0.5710076 0.6158912 0.6575990
##  [8] 0.6927704 0.7256491 0.7570176 0.7865666 0.8137585 0.8385481 0.8622063
## [15] 0.8821509 0.9016332 0.9195293 0.9350467 0.9501484 0.9633701 0.9753130
## [22] 0.9846030 0.9918106 0.9948952 0.9967384 0.9984754 0.9997623 0.9998868
## [29] 1.0000000
```

```
par(mfrow=c(1, 2))
plot(subct.pve, type="l", lwd=3, xlab="Principal Component",
ylab="Proportion of Variance Explained", cex.lab = 0.75)
plot(subct.cumulative_pve, type="l", lwd=3, xlab="Principal Component ",
ylab="Cumulative Proportion of Variance Explained", cex.lab = 0.55)
title("Variances explained for sub-county data", line = -1, outer=TRUE)
```




```
#subct.cumulative_pve
#min(which(subct.cumulative_pve >= 0.90))

ct.pr.var = ct.pr.out$sdev^2
ct.pr.var
```

```
##  [1] 6.673847046 3.746866063 3.299049480 1.704167271 1.210901689 1.152395297
##  [7] 1.133213578 0.971232549 0.858457538 0.782681764 0.748400311 0.623818807
## [13] 0.501798956 0.463492617 0.397716716 0.353520713 0.313696451 0.299269426
## [19] 0.216285895 0.178355514 0.135264017 0.096948732 0.061793407 0.044736070
## [25] 0.029669638 0.002420455
```

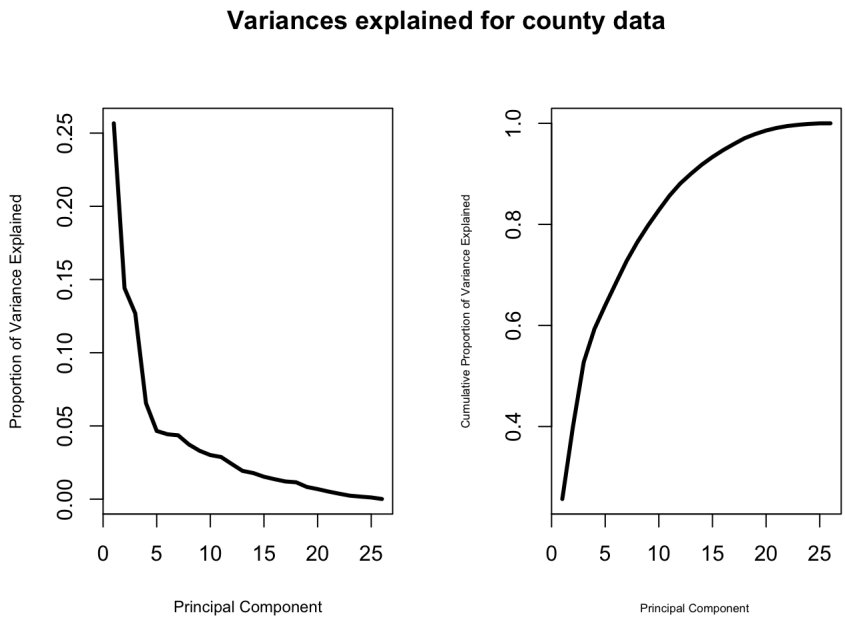
```
ct.pve = ct.pr.var/sum(ct.pr.var)
ct.pve
```

```
##  [1] 2.566864e-01 1.441102e-01 1.268865e-01 6.554490e-02 4.657314e-02
##  [6] 4.432290e-02 4.358514e-02 3.735510e-02 3.301760e-02 3.010314e-02
## [11] 2.878463e-02 2.399303e-02 1.929996e-02 1.782664e-02 1.529680e-02
## [16] 1.359695e-02 1.206525e-02 1.151036e-02 8.318688e-03 6.859827e-03
## [21] 5.202462e-03 3.728797e-03 2.376669e-03 1.720618e-03 1.141140e-03
## [26] 9.309442e-05
```

```
ct.cumulative_pve = cumsum(ct.pve)
ct.cumulative_pve
```

```
##  [1] 0.2566864 0.4007967 0.5276832 0.5932281 0.6398012 0.6841241 0.7277092
##  [8] 0.7650643 0.7980819 0.8281851 0.8569697 0.8809627 0.9002627 0.9180893
## [15] 0.9333861 0.9469831 0.9590483 0.9705587 0.9788774 0.9857372 0.9909397
## [22] 0.9946685 0.9970451 0.9987658 0.9999069 1.0000000
```

```
par(mfrow=c(1, 2))
plot(ct.pve, type="l", lwd=3, xlab="Principal Component",
ylab="Proportion of Variance Explained", cex.lab = 0.75)
plot(ct.cumulative_pve, type="l", lwd=3, xlab="Principal Component ",
ylab="Cumulative Proportion of Variance Explained", cex.lab = 0.55)
title("Variances explained for county data", line = -1, outer=TRUE)
```




```
#ct.cumulative_pve

min(which(subct.cumulative_pve >= 0.90))
```

```
## [1] 16
```

```
min(which(ct.cumulative_pve >= 0.90))
```

```
## [1] 13
```

Clustering

15. With `census.ct` , perform hierarchical clustering with complete linkage. Cut the tree to partition the observations into 10 clusters. Re-run the hierarchical clustering algorithm using the first 2 principal components of ct.pc as inputs instead of the originald features. Compare and contrast the results. For both approaches investigate the cluster that contains San Mateo County. Which approach seemed to put San Mateo County in a more appropriate clusters? Comment on what you observe and discuss possible explanations for these observations.

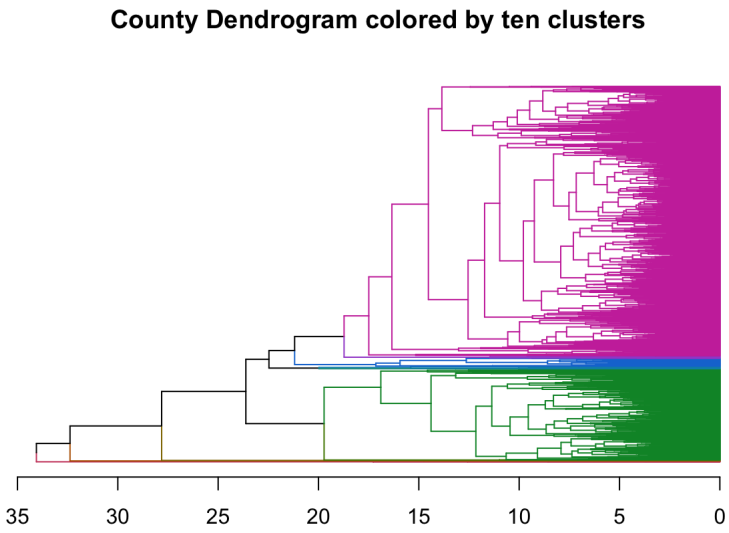
Answer: When using the first 5 principal components, San Mateo county is Democratic county but it was placed into cluster 5 along with many Republican counties. This method misclassified the the county of San Mateo, possibly because the first principle component is income per capita. Trump voters tend to be more influenced by income per capita as opposed the Clinton voters. In other words, San Mateo and other democratic counties could be misclassified because this method placed more importance on income per capita.

```
#15
set.seed(1)
census.scaled = as.data.frame(scale(census.ct[c(-1,-2)]))
dis = dist(census.scaled, method="euclidean")
census.hc = hclust(dis, method="complete")
#plot(census.hc, hang=-1, labels=, main='Cluster Dendrogram', cex=0.25)
#rect.hclust(census.hc, k=10, border = 2:4)
hc.cut = cutree(census.hc, k=10)

dend1 = as.dendrogram(census.hc)
dend1 = color_branches(dend1, k=10)
dend1 = color_labels(dend1, k=10)
dend1 = set(dend1, "labels_cex", 0.3)
dend1 = set_labels(dend1, labels=census.hc$County[order.dendrogram(dend1)])
```

```
## Warning in `labels<-.dendrogram`(`*tmp*`, value = labels): The lengths of the
## new labels is shorter than the number of leaves in the dendrogram - labels are
## recycled.
```

```
## Warning in rep(new_labels, length.out = leaves_length): 'x' is NULL so the
## result will be NULL
```

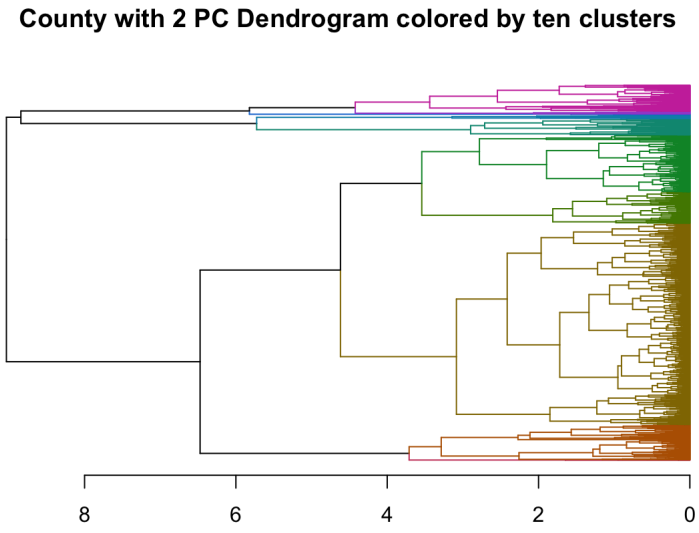
```
plot(dend1, horiz=T, main = "County Dendrogram colored by ten clusters")
```




```
ct.pc.scaled = as.data.frame(scale(ct.pc))
dis = dist(ct.pc.scaled, method="euclidean")
ct.pc.hc = hclust(dis, method="complete")
#plot(census.hc, hang=-1, labels=, main='Cluster Dendrogram', cex=0.25)
#rect.hclust(census.hc, k=10, border = 2:4)
ct.pc.hc.cut = cutree(ct.pc.hc, k=10)

dend2 = as.dendrogram(ct.pc.hc)
dend2 = color_branches(dend2, k=10)
dend2 = color_labels(dend2, k=10)
dend2 = set(dend2, "labels_cex", 0.3)
dend2 = set_labels(dend2, labels=census.hc$County[order.dendrogram(dend2)])
```

```
## Warning in `labels<-.dendrogram`(`*tmp*`, value = labels): The lengths of the
## new labels is shorter than the number of leaves in the dendrogram - labels are
## recycled.

## Warning in `labels<-.dendrogram`(`*tmp*`, value = labels): 'x' is NULL so the
## result will be NULL
```

```
plot(dend2, horiz=T, main = "County with 2 PC Dendrogram colored by ten clusters")
```



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.

```
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 %>%
  ungroup %>%
  mutate_at(vars(State, County), tolower)

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

## save meta information
election.meta <- election.cl %>% select(c(county, fips, state, votes, pct, total))

## save predictors and class labels
election.cl = election.cl %>% select(-c(county, fips, state, votes, pct, total))
```

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:

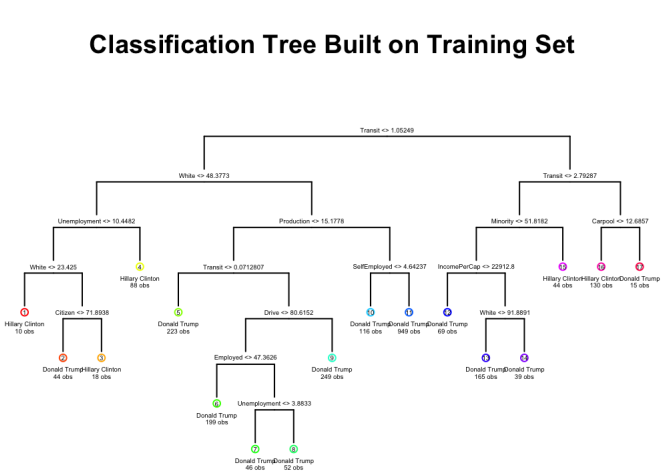
```
calc_error_rate = function(predicted.value, true.value){
  return(mean(true.value!=predicted.value))
}
records = matrix(NA, nrow=3, ncol=2)
colnames(records) = c("train.error","test.error")
rownames(records) = c("tree","logistic","lasso")
```

Classification

16. Decision tree: train a decision tree by `cv.tree()`. Prune tree to minimize misclassification error. 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. Intepret and discuss the results of the decision tree analysis. Use this plot to tell a story about voting behavior in the US (remember the NYT infographic? (https://archive.nytimes.com/www.nytimes.com/imagepages/2008/04/16/us/20080416_OBAMA_GRAPHIC.html))

Answer: In both decision trees, it selected Transit as the first variable. Urban cities tend to have more transit than suburban and rural areas. From our decision trees, we see that Transit with higher value and would vote for Hillary. This is probably because urban cities hold a more liberal and democratic affiliation. We also see more White voters tend to vote for Trump regardless of where they live.

```
#16
electiontree = tree(candidate~., data = trn.cl)

draw.tree(electiontree, nodeinfo=FALSE, cex=0.3)
title("Classification Tree Built on Training Set")
```




```
set.seed(1)
cv <- cv.tree(electiontree, rand = folds, FUN=prune.misclass)
cv
```

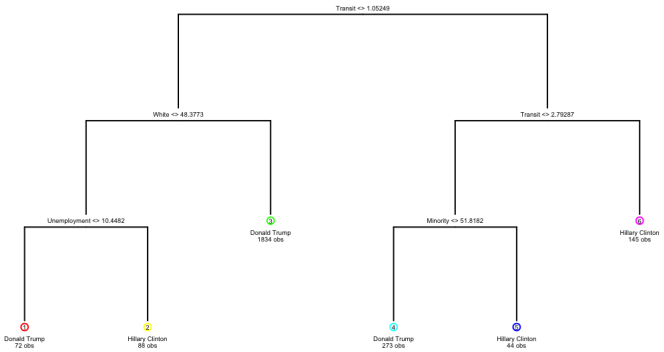
```
## $size
## [1] 17  9  8  7  6  5  3  1
##
## $dev
## [1] 217 217 211 214 208 235 298 375
##
## $k
## [1] -Inf  0.0  2.0  7.0  8.0 26.0 34.0 46.5
##
## $method
## [1] "misclass"
##
## attr(,"class")
## [1] "prune"         "tree.sequence"
```

```
best.size.cv = min(cv$size[which(cv$dev==min(cv$dev))])
best.size.cv
```

```
## [1] 6
```

```
#best size is 6
electiontree.pruned = prune.misclass(electiontree, best=best.size.cv)
draw.tree(electiontree.pruned, nodeinfo=FALSE, cex=0.3)
title("Classification Pruned Tree Built on Training Set")
```

Classification Pruned Tree Built on Training Set




```
electiontree.pruned.predict.train <- predict(electiontree.pruned, trn.cl, type = "class")
tree.train.error <- calc_error_rate(electiontree.pruned.predict.train, trn.cl$candidate)
tree.train.error
```

```
## [1] 0.07899023
```

```
electiontree.pruned.predict.test <- predict(electiontree.pruned, tst.cl, type = "class")
tree.test.error <- calc_error_rate(electiontree.pruned.predict.test, tst.cl$candidate)
tree.test.error
```

```
## [1] 0.07154472
```

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

```
##          train.error test.error
## tree      0.07899023 0.07154472
## logistic          NA         NA
## lasso             NA         NA
```

17. Run a logistic regression to predict the winning candidate in each county. Save training and test errors to `records` variable. What are the significant variables? Are the consistent with what you saw in decision tree analysis? Interpret the meaning of a couple of the significant coefficients in terms of a unit change in the variables.

Answer: White, Citizen, IncomePerCap, Professional, Service, Production, Drive, Carpool, Employed, PrivateWork, FamilyWork and Unemployment are statistically significant at alpha level 0.001. While holding other variables constant, a unit change in the variables would change the log odds of the outcome. The variable White has coefficient of -0.2148. For every unit increase in White, the log odds of voting for Hillary decreases by 21.48% while holding other variables constant. The variable Unemployment has coefficient of 0.1593. For every unit increase in White, the log odds of voting for Hillary increases by 15.93% while holding other variables constant. The results from our logistic regression model is consistent with the results from out decision tree model above. White voters are more likely to vote for Trump. Unemployed voters are more likely to vote for Hillary. It is interesting to note that Transit was not a significantly variable in our logistic regression model but it was an important feature used in our decision tree.

```
#17
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
## -4.0546  -0.2608  -0.1094  -0.0393   3.5453
##
## Coefficients:
##                   Estimate Std. Error z value Pr(>|z|)
## (Intercept)     -2.486e+01  9.451e+00  -2.630  0.00854 **
## Men              9.484e-02  4.897e-02   1.937  0.05278 .
## Women           -2.835e-05  1.679e-04  -0.169  0.86592
## White           -1.760e-01  6.666e-02  -2.640  0.00830 **
## Citizen          1.266e-01  2.820e-02   4.491 7.08e-06 ***
## Income          -8.586e-05  2.731e-05  -3.144  0.00166 **
## IncomeErr       -5.363e-06  6.380e-05  -0.084  0.93301
## IncomePerCap     2.643e-04  6.717e-05   3.934 8.34e-05 ***
## IncomePerCapErr -3.626e-04  1.720e-04  -2.108  0.03500 *
## Poverty          4.770e-02  4.128e-02   1.156  0.24785
## ChildPoverty    -1.690e-02  2.531e-02  -0.668  0.50432
## Professional     2.880e-01  3.819e-02   7.541 4.66e-14 ***
## Service          3.307e-01  4.714e-02   7.015 2.30e-12 ***
## Office           8.428e-02  4.407e-02   1.912  0.05581 .
## Production       1.687e-01  4.096e-02   4.117 3.84e-05 ***
## Drive           -2.097e-01  4.671e-02  -4.490 7.12e-06 ***
## Carpool         -1.733e-01  6.012e-02  -2.882  0.00395 **
## Transit          8.530e-02  9.349e-02   0.912  0.36160
## OtherTransp     -5.726e-02  9.679e-02  -0.592  0.55415
## WorkAtHome      -1.654e-01  7.229e-02  -2.288  0.02212 *
## MeanCommute      5.797e-02  2.453e-02   2.363  0.01812 *
## Employed         2.058e-01  3.362e-02   6.122 9.27e-10 ***
## PrivateWork      1.066e-01  2.143e-02   4.975 6.52e-07 ***
## SelfEmployed     2.385e-02  4.657e-02   0.512  0.60861
## FamilyWork      -8.887e-01  3.785e-01  -2.348  0.01888 *
## Unemployment     2.096e-01  3.968e-02   5.282 1.28e-07 ***
## Minority        -3.627e-02  6.427e-02  -0.564  0.57254
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 2119.56  on 2455  degrees of freedom
## Residual deviance:  855.19  on 2429  degrees of freedom
## AIC: 909.19
##
## Number of Fisher Scoring iterations: 7
```

```
prob.training = predict(glm.fit, trn.cl, type="response")
prob.test = predict(glm.fit, tst.cl, type="response")
prob.training = round(prob.training, digits=2)
prob.test = round(prob.test, digits=2)

# Save the predicted labels using 0.5 as a threshold
prob.training <- ifelse(prob.training<= 0.5, "Donald Trump", "Hillary Clinton")
prob.test <- ifelse(prob.test<= 0.5, "Donald Trump", "Hillary Clinton")

logres.train.error <-
  calc_error_rate(prob.training, trn.cl$candidate)
# testing error
logres.test.error <-
  calc_error_rate(prob.test, tst.cl$candidate)

logres.train.error
```

```
## [1] 0.07003257
```

```
logres.test.error
```

```
## [1] 0.06341463
```

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

```
##          train.error test.error
## tree      0.07899023 0.07154472
## logistic  0.07003257 0.06341463
## lasso             NA         NA
```

18. You may notice that you get a warning `glm.fit: fitted probabilities numerically 0 or 1 occurred`. As we discussed in class, this is an indication that we have perfect separation (some linear combination of variables perfectly predicts the winner). This is usually a

sign that we are overfitting. One way to control overfitting in logistic regression is through regularization. Use the `cv.glmnet` function from the 'glmnet' library to run K-fold cross validation and select the best regularization parameter for the logistic regression with LASSO penalty. Reminder: set 'alpha=1' to run LASSO regression, set `lambda = c(1, 5, 10, 50) * 1e-4` in `cv.glmnet()` function to set pre-defined candidate values for the tuning parameter λ. This is because the default candidate values of λ in `cv.glmnet()` is relatively too large for our dataset thus we use pre-defined candidate values. What is the optimal value of λ in cross validation? What are the non-zero coefficients in the LASSO regression for the optimal value of λ? How do they compare to the unpenalized logistic regression? Save training and test errors to the `records` variable.

Answer: The optimal value of λ is 0.001. The non-zero coefficients in the LASSO regression are all variables except for Men, ChildPoverty, OtherTransp, and SelfEmployed. The absolute values of most of the variables in LASSO regression are half of the absolute values in logistic regression.

```
#18
set.seed(1)
xtrain.lasso = trn.cl %>%
  select(-candidate) %>%
  as.matrix
ytrain.lasso = trn.cl$candidate %>%
                          droplevels(except = c("Donald Trump", "Hillary Clinton"))
cv.lasso = cv.glmnet(xtrain.lasso, ytrain.lasso, alpha = 1, lambda = c(1, 5, 10, 50) * 1e-4, family = "binomial")

bestlam = cv.lasso$lambda.min
bestlam
```

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

```
lasso.mod = glmnet(xtrain.lasso, ytrain.lasso, alpha = 1, lambda = bestlam, family = "binomial")

xtest.lasso = tst.cl %>%
  select(-candidate) %>%
  as.matrix

lasso.predict.train = predict(lasso.mod, newx = xtrain.lasso, type = "class")
lasso.predict.test = predict(lasso.mod, newx = xtest.lasso, type = "class")

lasso.train.error = calc_error_rate(lasso.predict.train, trn.cl$candidate)
lasso.test.error = calc_error_rate(lasso.predict.test, tst.cl$candidate)
lasso.train.error
```

```
## [1] 0.06881107
```

```
lasso.test.error
```

```
## [1] 0.06666667
```

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

```
##          train.error test.error
## tree      0.07899023 0.07154472
## logistic  0.07003257 0.06341463
## lasso     0.06881107 0.06666667
```

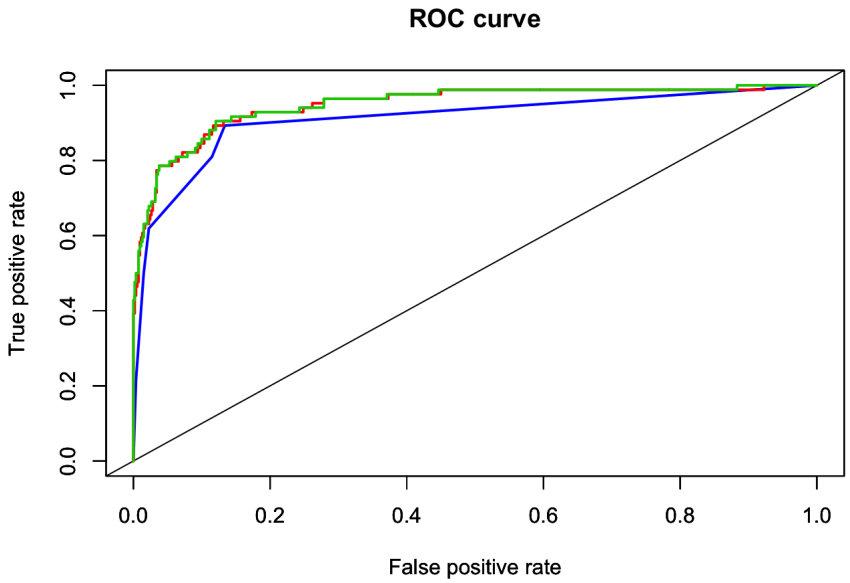
```
lasso.coef <- predict(lasso.mod, type="coefficients")[1:26,]
lasso.coef
```

```
##     (Intercept)             Men           Women           White         Citizen
##   -2.691561e+01    6.997042e-02   -2.460544e-06   -1.308120e-01    1.315770e-01
##          Income       IncomeErr    IncomePerCap IncomePerCapErr         Poverty
##   -6.071980e-05   -1.612193e-05    2.034207e-04   -2.566706e-04    3.242331e-02
##    ChildPoverty    Professional         Service          Office      Production
##   -1.637612e-03    2.606474e-01    2.989287e-01    6.188643e-02    1.378877e-01
##          Drive         Carpool         Transit     OtherTransp      WorkAtHome
##   -1.809434e-01   -1.437560e-01    1.064818e-01   -1.782729e-02   -1.218047e-01
##     MeanCommute        Employed     PrivateWork    SelfEmployed      FamilyWork
##    4.095099e-02    1.944322e-01    9.780724e-02    8.973790e-04   -7.442034e-01
##    Unemployment
##    1.972514e-01
```

19. Compute ROC curves for the decision tree, logistic regression and LASSO logistic regression using predictions on the test data. Display them on the same plot. Based on your classification results, discuss the pros and cons of the various methods. Are the different classifiers more appropriate for answering different kinds of questions about the election?

Answer: Decision trees are easier to interpret the results and would be better for those who do not have a background in statistics. However, decision trees tend to overfit the data and its accuracy is not as competitive in comparison to other classification methods. In our analysis, logistic regression and lasso regularized methods hold better testing results than the decision tree. Logistic regression model is great in understanding the percent change in predicted chance in victory given the values of important predictors. A downside to logistic regression is that we cannot solve the nonlinear problems. Logistic regression also tends to overfit as well.

```
#19
prob.tree.pruned.test = predict(electiontree.pruned, tst.cl)
prob.log.test = predict(glm.fit, tst.cl, type="response")
prob.lasso.test = predict(lasso.mod, newx = xtest.lasso)

prediction.tree = prediction(prob.tree.pruned.test[,13], droplevels(tst.cl$candidate, except = c("Donald Trump",
"Hillary Clinton")))

prediction.log = prediction(prob.log.test, droplevels(tst.cl$candidate, except = c("Donald Trump", "Hillary Clint
on")))

prediction.lasso = prediction(prob.lasso.test, droplevels(tst.cl$candidate, except = c("Donald Trump", "Hillary C
linton")))

perf.tree = performance(prediction.tree, measure = "tpr", x.measure = "fpr")
perf.log = performance(prediction.log, measure="tpr", x.measure="fpr")
perf.lasso = performance(prediction.lasso, measure = "tpr", x.measure = "fpr")

plot(perf.tree, col=4, lwd=2, main="ROC curve")
abline(0,1)
par(new=TRUE)
plot(perf.log, col=2, lwd=2)
par(new=TRUE)
plot(perf.lasso,col=3,lwd=2)
```




```
auc.tree = performance(prediction.tree, "auc")@y.values
auc.tree
```

```
## [[1]]
## [1] 0.9104789
```

```
auc.log = performance(prediction.log, "auc")@y.values
auc.log
```

```
## [[1]]
## [1] 0.9479419
```

```
auc.lasso = performance(prediction.lasso, "auc")@y.values
auc.lasso
```

```
## [[1]]
## [1] 0.9490629
```

Taking it further

20. This is an open question. Interpret and discuss any overall insights gained in this analysis 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). In addition, propose and tackle at least one more interesting question. Creative and thoughtful analyses will be rewarded! This part will be worth up to a 20% of your final project grade!

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?
    - Exploring additional classification methods: KNN, LDA, QDA, SVM, random forest, boosting etc. (You may research and use methods beyond those covered in this course). How do these compare to logistic regression and the tree method?
    - Bootstrap: Perform boostrap to generate plots similar to ISLR Figure 4.10/4.11. Discuss the results.
    - Use linear regression models to predict the total vote for each candidate by county. Compare and contrast these results with the classification models. Which do you prefer and why? How might they complement one another?
    - Conduct an exploratory analysis of the “purple” counties– the counties which the models predict Clinton and Trump were roughly equally likely to win. What is it about these counties that make them hard to predict?
    - Instead of using the native attributes (the original features), we can use principal components to create new (and lower dimensional) set of features with which to train a classification model. This sometimes improves classification performance. Compare classifiers trained on the original features with those trained on PCA features.

#K-nearest neighbor We will train a KNN model for classification

```
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))
}
```

```
# 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 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))
```

```
## Warning: funs() is soft deprecated as of dplyr 0.8.0
## Please use a list of either functions or lambdas:
##
##   # Simple named list:
##   list(mean = mean, median = median)
##
##   # Auto named with `tibble::lst()`:
##   tibble::lst(mean, median)
##
##   # Using lambdas
##   list(~ mean(., trim = .2), ~ median(., na.rm = TRUE))
## This warning is displayed once per session.
```

```
# picking the best k
min.error <- val.error.means %>%
filter(error==min(error))
bestk <- max(min.error$neighbors)
paste( "best number of neighbors:", bestk)
```

```
## [1] "best number of neighbors: 15"
```

```
#Setting up new records matrix
records2 = matrix(NA, nrow=3, ncol=2)
colnames(records2) = c("train.error","test.error")
rownames(records2) = c("knn","random forest","svm")
print(records2)
```

```
##               train.error test.error
## knn                    NA         NA
## random forest          NA         NA
## svm                    NA         NA
```

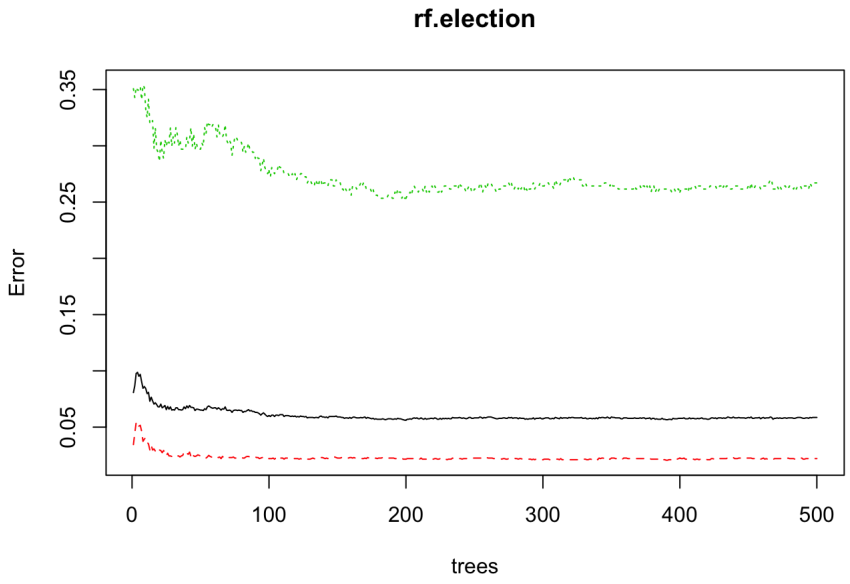
```
# calculating training errors at each k
# (by taking mean of each cv result)
train.error.means <- errors %>%
                     filter(variable=="train.error") %>%
                     group_by(neighbors) %>%
                     summarise_at(vars(error),funs(mean))
```

```
# 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
records2[1,1] <- train.errork
records2[1,2] <- test.errork
records2
```

```
##               train.error test.error
## knn             0.1343648  0.1317073
## random forest          NA         NA
## svm                    NA         NA
```

#Random Forest

```
# Sample 250 observations as training data
set.seed(3)
train = sample(1:nrow(election.cl), 2456)
train.election = election.cl[train,]
# The rest as test data
test.election = election.cl[-train,]
train.election$candidate <- droplevels(train.election$candidate)
#Random Forest
rf.election = randomForest(candidate~ ., data=train.election, ntree=500, importance=TRUE)
plot(rf.election)
```



```
#train.rf.err = 0.0012215

#yhat.rf = predict (rf.election, newdata = test.election)

pred.rf.train <- predict (rf.election, newdata = trn.cl)
pred.rf.test <- predict (rf.election, newdata = tst.cl)

rf.train.error = calc_error_rate(pred.rf.train, droplevels(trn.cl$candidate, except = c("Donald Trump", "Hillary
 Clinton")))
rf.test.error = calc_error_rate(pred.rf.test, droplevels(tst.cl$candidate, except = c("Donald Trump", "Hillary Cl
inton")))
rf.train.error
```

```
## [1] 0.01140065
```

```
rf.test.error
```

```
## [1] 0.01138211
```

```
# Confusion matrix
#rf.err = table(pred = yhat.rf, truth = test.election$candidate)
#test.rf.err = 1 - sum(diag(rf.err))/sum(rf.err)
#test.rf.err

# adding to records
records2[2,1] <- rf.train.error
records2[2,2] <- rf.test.error
records2
```

```
##               train.error test.error
## knn            0.13436482 0.13170732
## random forest  0.01140065 0.01138211
## svm                   NA         NA
```

#SVM

```
svmfit=svm(candidate~., data=trn.cl, kernel="linear", cost=1,scale=TRUE)

svm.predict.train = predict(svmfit, trn.cl, type = "class")
svm.predict.test = predict(svmfit, tst.cl, type = "class")

svm.train.error = calc_error_rate(svm.predict.train, trn.cl$candidate)
svm.train.error
```

```
## [1] 0.07003257
```

```
svm.test.error = calc_error_rate(svm.predict.test, tst.cl$candidate)
svm.test.error
```

```
## [1] 0.06829268
```

```
records2[3,1] <- svm.train.error
records2[3,2] <- svm.test.error
records2
```

```
##               train.error test.error
## knn            0.13436482 0.13170732
## random forest  0.01140065 0.01138211
## svm            0.07003257 0.06829268
```

```
#Purple America
library(Hmisc)
```

```
## Loading required package: lattice
```

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

```
## Loading required package: Formula
```

```
##
## Attaching package: 'Hmisc'
```

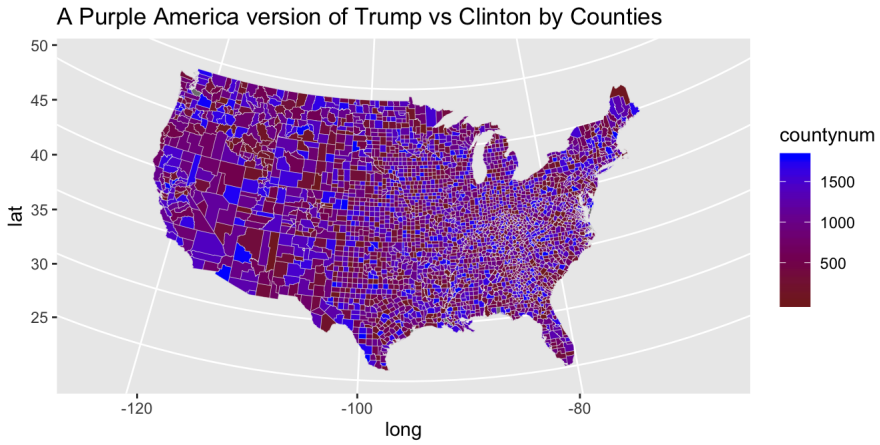
```
## The following object is masked from 'package:e1071':
##
##     impute
```

```
## The following objects are masked from 'package:dplyr':
##
##     src, summarize
```

```
## The following objects are masked from 'package:base':
##
##     format.pval, units
```

```
counties$countynum <- as.numeric(as.factor(counties$county))
ggplot(data =counties,
             aes(x = long, y = lat, group = group, fill = countynum, group = group)) +
             geom_polygon(color = "gray90", size = 0.1) +
             coord_map(projection = "albers", lat0 = 39, lat1 = 45) +
             labs(title = "A Purple America version of Trump vs Clinton by Counties") +
             scale_fill_gradient2(low = "violet",
                                  mid = scales::muted("red"),
                                  high = "blue")
```

```
## Warning: Duplicated aesthetics after name standardisation: group
```



Exploratory Analysis of “Purple America” “Purple America” is the belief that if we analyze the recent election results, we can see that America falls between a spectrum of “Red” (Republican) and “Blue” (Democrat). America is not as split between “Red” and “Blue” as the news portray it to be. America is not a nation divided. It looks like a sizable amount of counties is purple. Purple indicates that election results has favored both parties equally. So what makes these counties so hard to predict? It falls back on human behavior. People will vote for whoever can address their concerns despite party affiliation. It is hard to measure and predict human behavior because However, the map above is misleading because sparsely populated areas are represented the same way as densely populated areas. For example, Los Angeles has a population of 4 million people but is only represented by a small area in the map whereas Montana has a population of less than 1 million people but it gets a large area of the map. In other words, this map is showing votes spatially but what really matters is how many people in each area voted. Although communities can shift their allegiance over time, it seems like communities like San Mateo are far from purple. In 2016, a vast majority of San Mateo voted in favor of Clinton. This raises concern when younger generations would not have exposure to alternative political views.