PSTAT 131 Final Project: 2020 Election Analysis

Xilin Huang (5562194) and Danming Wang (5833587)

December 17, 2020

Data

We will essentially start the analysis with two data sets. The first one is the election data. The data contains county-level election results. Note that this is not the final election results, as recounting are still taking place in many states.
The second dataset is the 2017 United States county-level census data.
The following code load in these two data sets: `election.raw` and `census`.

```
## read data and convert candidate names and party names from string to factor
election.raw <- read_csv("candidates_county.csv", col_names = TRUE) %>%
  mutate(candidate = as.factor(candidate), party = as.factor(party))

## remove the word "County" from the county names
words.to.remove = c("County")
remove.words <- function(str, words.to.remove){
  sapply(str, function(str){
    x <- unlist(strsplit(str, " "))
    x <- x[!x %in% words.to.remove]
    return(paste(x, collapse = " "))
  }, simplify = "array", USE.NAMES = FALSE)
}
election.raw$county <- remove.words(election.raw$county, words.to.remove)

## read census data
census <- read_csv("census_county.csv")
```

Election data

1. Report the dimension of `election.raw`. Are there missing values in the data set? Compute the total number of distinct values in `state` in `election.raw` to verify that the data contains all states and a federal district.

```
# Find the dimension of the election.raw data set
dim(election.raw)
```

```
## [1] 31167     5
```

```
#To find if there is any missing value in the election.raw data set
sum(is.na(election.raw))
```

```
## [1] 0
```

```
#Compute the total number of distinct values in each columns
apply(election.raw, 2, function(x) length(unique(x)))
```

```
##     state    county candidate     party     votes
```

```
##         51       2825          38          26       6703
```

`election.raw` has 31167 rows and 5 columns. There is no missing value in this data set. There are 51 distinct values in `state` in `election.raw`, which corresponds to one federal district and all of the 50 states in the U.S. Thus, the data contains all states and a federal district.

Census data

2. Report the dimension of `census`. Are there missing values in the data set? Compute the total number of distinct values in `county` in `census`. Compare the values of total number of distinct county in `census` with that in `election.raw`. Comment on your findings.

```r
# Find the dimension of the census data set
dim(census)
```

```
## [1] 3220   37
```

```r
#To find is there any missing value in the census data set
sum(is.na(census))
```

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

```r
#Compute the total number of distinct values in each columns
apply(census, 2, function(x) length(unique(x)))
```

```
##         CountyId            State           County         TotalPop
##             3220               52             1955             3175
##              Men            Women         Hispanic            White
##             3112             3087              481              728
##            Black           Native            Asian          Pacific
##              487              184              130               28
## VotingAgeCitizen           Income        IncomeErr     IncomePerCap
##             3139             3074             2399             2982
##  IncomePerCapErr          Poverty     ChildPoverty     Professional
##             1880              394              543              356
##          Service           Office     Construction       Production
##              233              200              238              301
##            Drive          Carpool          Transit             Walk
##              361              199              119              177
##       OtherTransp       WorkAtHome      MeanCommute         Employed
##               95              184              309             3069
##      PrivateWork       PublicWork     SelfEmployed       FamilyWork
##              376              317              216               42
##     Unemployment
##              223
```

`census` has 3220 rows and 37 columns. There is one missing value in the data set. The total number of distinct values in `county` in `census` is 1955, which is lower than 2825, the total number of distinct values in `county` in `election.raw`. Since the census data is collected in 2017, the reason for the difference might be that there are more counties or county equivalent in 2020 than in 2017. Or it could be that some counties changed their county names after 2017 since there are overlaps of county names between different states.

3. **Construct aggregated data sets from election.raw data:**

```r
#Keep the county-level data as it is in election.raw
head(election.raw)
```

```
## # A tibble: 6 x 5
```

```
##   state    county     candidate     party  votes
##   <chr>    <chr>      <fct>         <fct>  <dbl>
## 1 Delaware Kent       Joe Biden     DEM    44518
## 2 Delaware Kent       Donald Trump  REP    40976
## 3 Delaware Kent       Jo Jorgensen  LIB     1044
## 4 Delaware Kent       Howie Hawkins GRN      420
## 5 Delaware Kent       Write-ins     WRI        0
## 6 Delaware New Castle Joe Biden     DEM   194238
```

```r
#Create a state-level summary into a election.state
election.state = election.raw %>%
  group_by(state,candidate,party) %>%
  summarise(votes=sum(votes))
```

```
## `summarise()` regrouping output by 'state', 'candidate' (override with `.groups` argument)
```

```r
head(election.state)
```

```
## # A tibble: 6 x 4
## # Groups:   state, candidate [6]
##   state   candidate       party   votes
##   <chr>   <fct>           <fct>   <dbl>
## 1 Alabama Donald Trump    REP   1434159
## 2 Alabama Jo Jorgensen    LIB     24994
## 3 Alabama Joe Biden       DEM    843473
## 4 Alabama Write-ins       WRI      7274
## 5 Alaska  Brock Pierce    IND       297
## 6 Alaska  Don Blankenship CST       348
```

```r
#Create a federal-level summary into a election.total
election.total<-election.raw %>%
  group_by(candidate,party) %>%
  summarise(votes=sum(votes))
```

```
## `summarise()` regrouping output by 'candidate' (override with `.groups` argument)
```

```r
head(election.total)
```

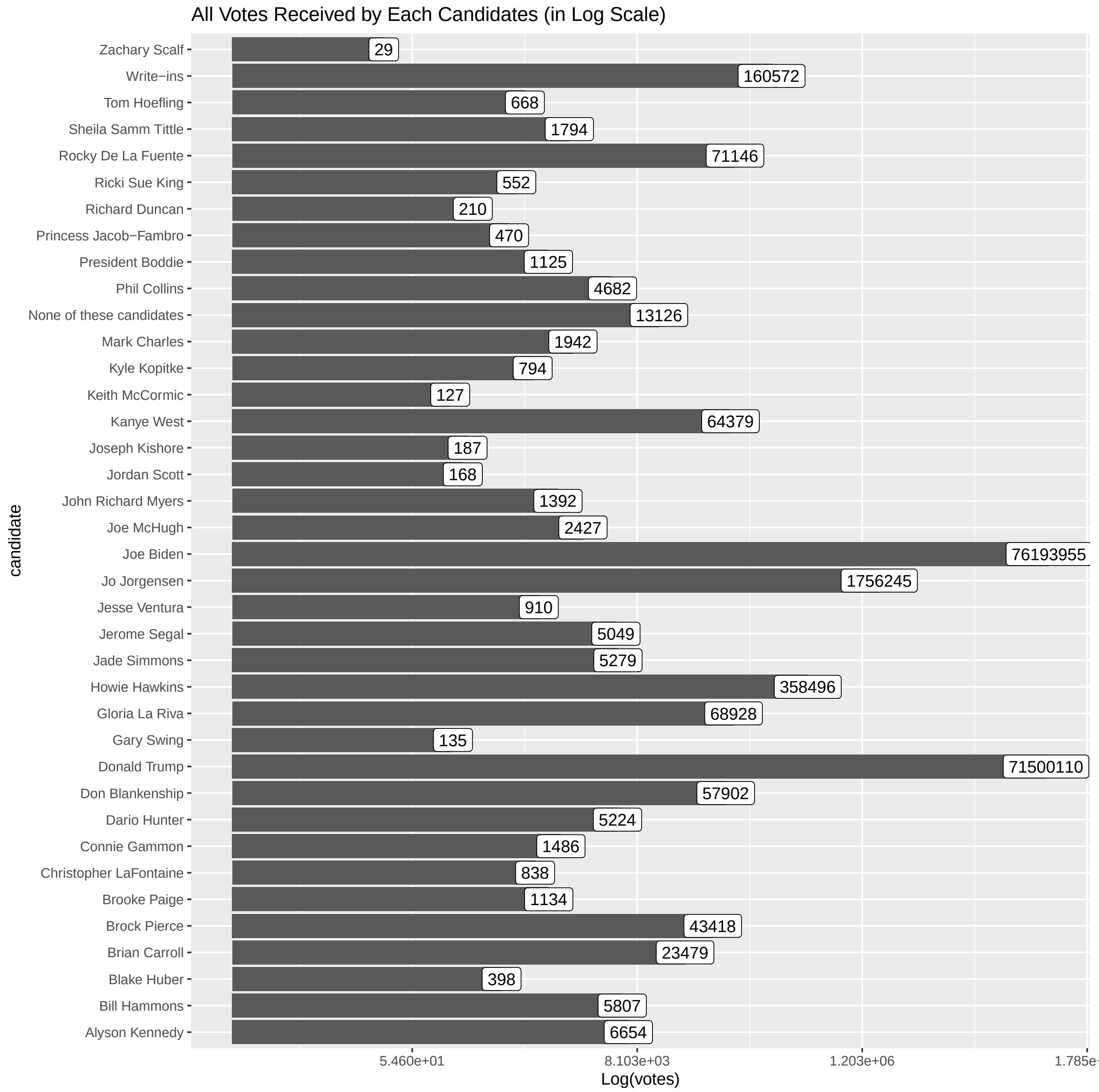
```
## # A tibble: 6 x 3
## # Groups:   candidate [6]
##   candidate      party votes
##   <fct>          <fct> <dbl>
## 1 Alyson Kennedy SWP    6654
## 2 Bill Hammons   UTY    5807
## 3 Blake Huber    APV     398
## 4 Brian Carroll  ASP   23479
## 5 Brock Pierce   IND   43418
## 6 Brooke Paige   GOP    1134
```

4. How many named presidential candidates were there in the 2020 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! (For fun: spot Kanye West among the presidential candidates!)

```r
#Find the number of named presidential candidates in the 2020 election
nrow(election.total)
```

```
## [1] 38
```

```r
# Draw the bar chart
ggplot(data=election.total, mapping=aes(x=candidate, y=votes,label=votes))+
  geom_bar(stat="identity")+
  scale_y_continuous(trans = "log")+
  ylab("Log(votes)")+
  ggtitle("All Votes Received by Each Candidates (in Log Scale)")+
  geom_label()+
  coord_flip()
```



There were 38 named presidential candidates in the 2020 election. The bar chart above displays all votes received by each candidate on a log scale with labels of the exact votes for each candidate. From the plot, we can see that Kanye West won 64379 votes, which is the eighth highest votes among the 38 named presidential candidates in the 2020 election.

5. **Create data sets `county.winner` and `state.winner` by taking the candidate with the highest proportion of votes in both county level and state level.** Hint: to create `county.winner`, start with `election.raw`, group by `county`, compute `total` votes, and `pct=votes/total` as the proportion

of votes. Then choose the highest row using `top_n` (variable `state.winner` is similar).

```
#Create the county.winner
county.winner<-election.raw %>%
  group_by(county) %>%
  mutate(total=sum(votes),pct=votes/total)%>%
  top_n(1,wt=pct)
#show the first six county.winners
head(county.winner)
```

```
## # A tibble: 6 x 7
## # Groups:   county [6]
##   state                county               candidate   party  votes  total   pct
##   <chr>                <chr>                <fct>       <fct>  <dbl>  <dbl> <dbl>
## 1 Delaware             New Castle           Joe Biden   DEM   194238 287047 0.677
## 2 Delaware             Sussex               Donald Tru~ REP    71196 202727 0.351
## 3 District of Columbia District of Columb~ Joe Biden   DEM    29509  31260 0.944
## 4 District of Columbia Ward 2               Joe Biden   DEM    24247  27259 0.890
## 5 District of Columbia Ward 3               Joe Biden   DEM    33584  37377 0.899
## 6 District of Columbia Ward 4               Joe Biden   DEM    35117  37223 0.943
```

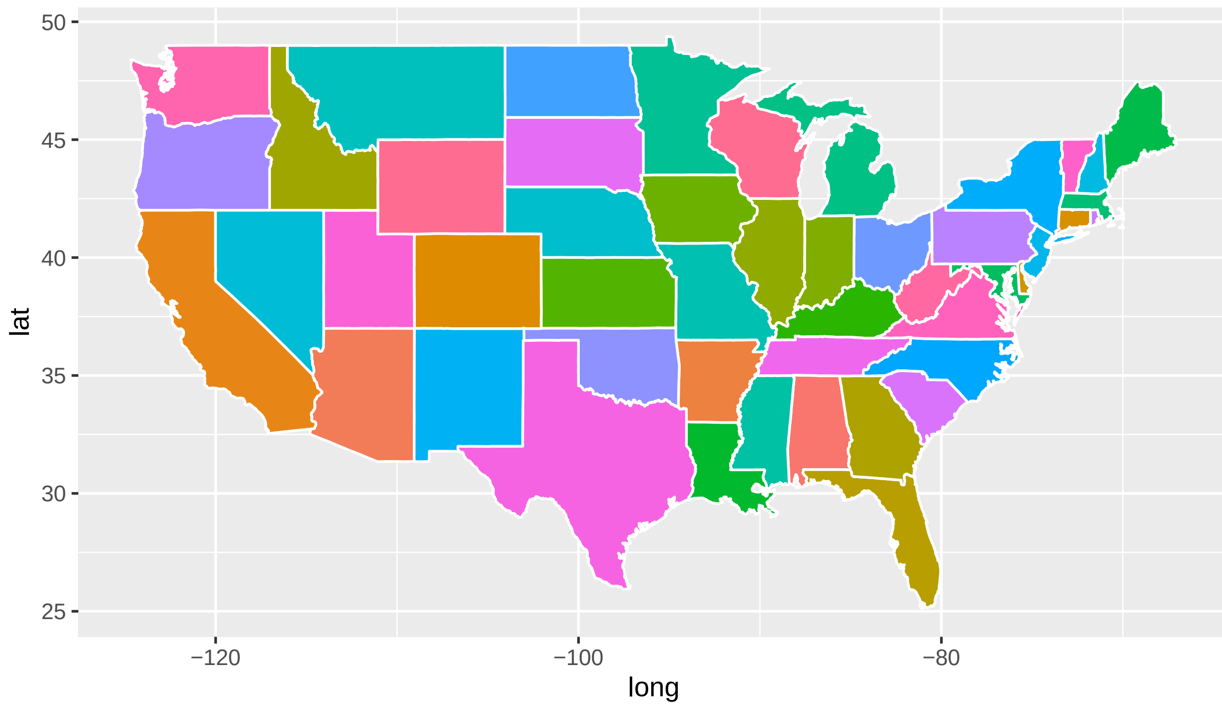
```
#Create the state.winner
state.winner<-election.state %>%
  group_by(state) %>%
  mutate(total=sum(votes),pct=votes/total)%>%
  top_n(1,wt=pct)
#show the first six state.winners
head(state.winner)
```

```
## # A tibble: 6 x 6
## # Groups:   state [6]
##   state      candidate    party   votes    total   pct
##   <chr>      <fct>        <fct>   <dbl>    <dbl> <dbl>
## 1 Alabama    Donald Trump REP   1434159  2309900 0.621
## 2 Alaska     Donald Trump REP     80999   131885 0.614
## 3 Arizona    Joe Biden    DEM   1643664  3322535 0.495
## 4 Arkansas   Donald Trump REP    761251  1216818 0.626
## 5 California Joe Biden    DEM   9315259 14414296 0.646
## 6 Colorado   Joe Biden    DEM   1753416  3173127 0.553
```

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.

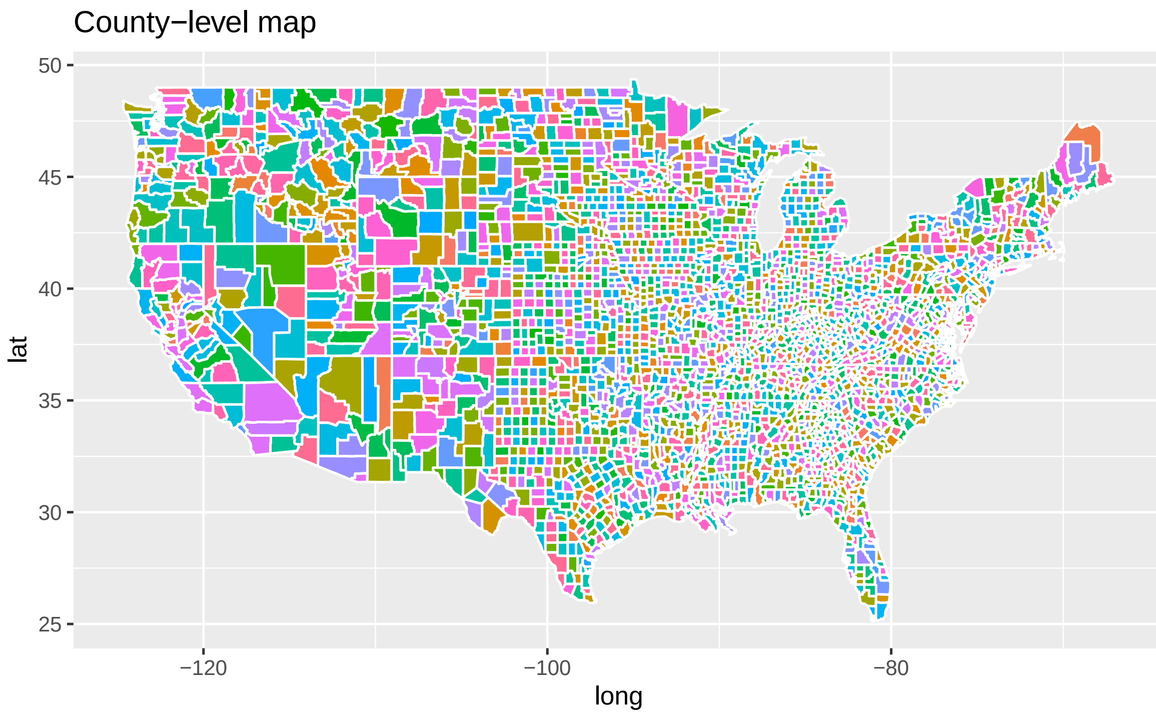
```
states<-map_data("state")

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`.

6. **Use similar code to above to draw county-level map by creating `counties=map_data("county")`. Color by county.**

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

ggplot(data=counties)+
  geom_polygon(aes(x=long,y=lat,fill=subregion,group=group),
               color="white")+
  coord_fixed(1.3)+
  guides(fill=FALSE) + #color legend is unnecessary and takes too long
  ggtitle("County-level map")
```



7. **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 marches 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 data sets based on state name. However, the state names in `states` and `state.winner` can be in different formats: check them! Before using `left_join()`, use certain transform to make sure the state names in the two data sets: `states` (for map drawing) and `state.winner` (for coloring) are in the same formats. Then `left_join()`. Your figure will look similar to New York Times map.

   We first check formats of state names in `states` and `state.winner`.

```
head(states$region)
```

```
## [1] "alabama" "alabama" "alabama" "alabama" "alabama" "alabama"
```

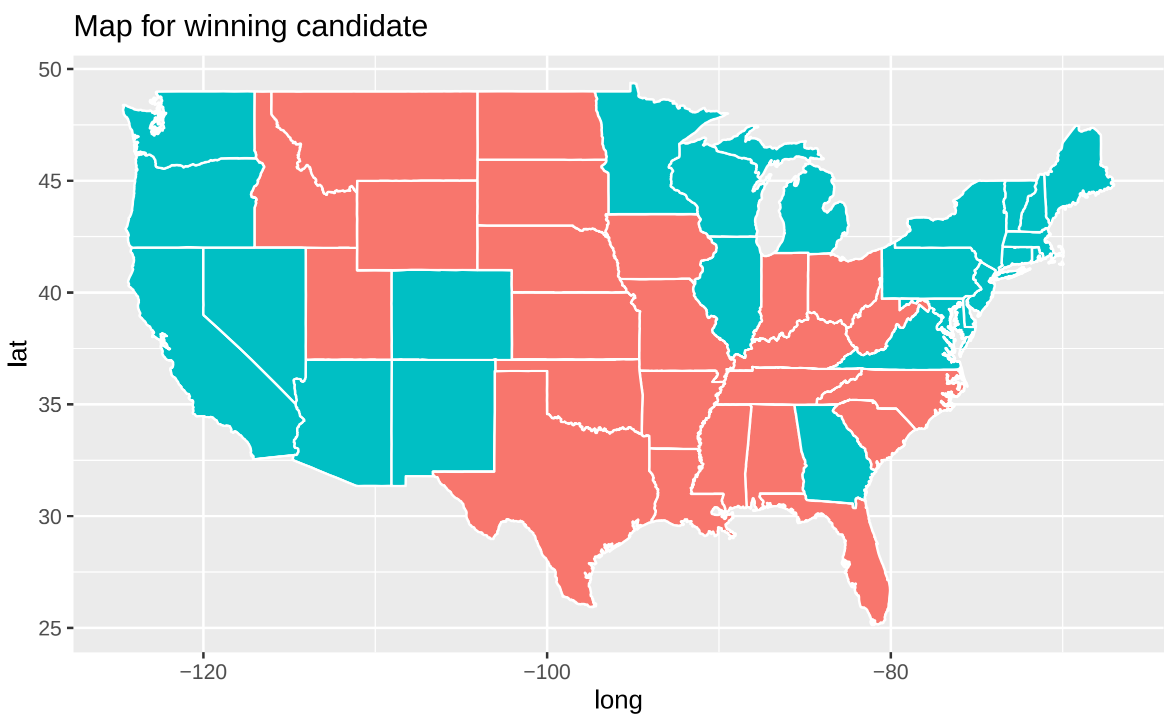
```
head(state.winner$state)
```

```
## [1] "Alabama"    "Alaska"     "Arizona"    "Arkansas"   "California"
## [6] "Colorado"
```

   By printing out the first six state names in `states` and `state.winner`, we can see that state names in 'states' are not capitalized while state names in 'state.winner' are. So we need to transform them into one format to get our desired plot.

```
# Transform format of state names in states
states$region=str_to_title(states$region)

states.join<-left_join(state.winner,states,by=c("state"="region"))
```

```r
ggplot(data = states.join)+
  geom_polygon(aes(x = long, y=lat,fill=candidate,group=group),
                color="white")+
  coord_fixed(1.3)+
  guides(fill=FALSE)+
  ggtitle("Map for winning candidate")
```



8. Color the map of the state of California by the winning candidate for each county. Note that some county have not finished counting the votes, and thus do not have a winner. Leave these counties uncolored.

   Again, We will first check formats of state names and county names in `counties` and `county.winner`. Note than in `counties` the variable `subregion` represents county names.

```r
head(counties)
```

```
##     long   lat group order  region subregion
## 1 -86.51 32.35     1     1 alabama   autauga
## 2 -86.53 32.35     1     2 alabama   autauga
## 3 -86.55 32.37     1     3 alabama   autauga
## 4 -86.56 32.38     1     4 alabama   autauga
## 5 -86.58 32.38     1     5 alabama   autauga
## 6 -86.59 32.38     1     6 alabama   autauga
```

```r
head(county.winner)
```

```
## # A tibble: 6 x 7
## # Groups:   county [6]
##   state                    county                  candidate    party  votes  total   pct
```

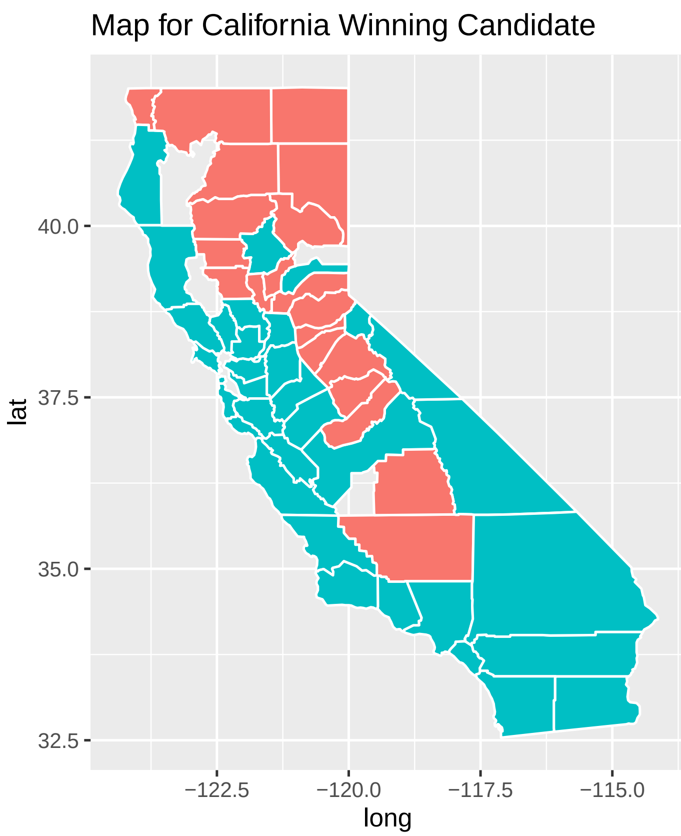
```
##   <chr>                <chr>                <fct>       <fct>  <dbl>  <dbl> <dbl>
## 1 Delaware             New Castle           Joe Biden   DEM   194238 287047 0.677
## 2 Delaware             Sussex               Donald Tru~ REP    71196 202727 0.351
## 3 District of Columbia District of Columb~ Joe Biden   DEM    29509  31260 0.944
## 4 District of Columbia Ward 2               Joe Biden   DEM    24247  27259 0.890
## 5 District of Columbia Ward 3               Joe Biden   DEM    33584  37377 0.899
## 6 District of Columbia Ward 4               Joe Biden   DEM    35117  37223 0.943
```

By printing out the first rows in `counties` and `couny.winner`, we can see that state names and county names in 'counties' are not capitalized while state names and county names in 'county.winner' are. So we need to transform them into one format to get our desired plot.

```
CA.county<-filter(counties,region=="california")
CA.winner<-filter(county.winner,state=="California")
CA.county$region=str_to_title(CA.county$region)
CA.county$subregion=str_to_title(CA.county$subregion)
CA.county<-left_join(CA.winner,CA.county,by=c("county"="subregion"))
head(CA.county)
```

```
## # A tibble: 6 x 12
## # Groups:   county [1]
##   state county candidate party  votes  total   pct  long   lat group order
##   <chr> <chr>  <fct>     <fct>  <dbl>  <dbl> <dbl> <dbl> <dbl> <dbl> <int>
## 1 Cali~ Fresno Joe Biden DEM   170694 323481 0.528 -121.  36.3   166  7374
## 2 Cali~ Fresno Joe Biden DEM   170694 323481 0.528 -121.  36.3   166  7375
## 3 Cali~ Fresno Joe Biden DEM   170694 323481 0.528 -121.  36.3   166  7376
## 4 Cali~ Fresno Joe Biden DEM   170694 323481 0.528 -121.  36.3   166  7377
## 5 Cali~ Fresno Joe Biden DEM   170694 323481 0.528 -121.  36.5   166  7378
## 6 Cali~ Fresno Joe Biden DEM   170694 323481 0.528 -121.  36.6   166  7379
## # ... with 1 more variable: region <chr>
```

```
ggplot(data = CA.county)+
  geom_polygon(aes(x = long, y=lat,fill=candidate,group=group),
               color="white")+
  coord_fixed(1.3)+
  guides(fill=FALSE)+
  ggtitle("Map for California Winning Candidate")
```



9. **(Open-ended) 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 graph visualizes the state winner result by sex.

```r
# The sex variable here means that major population of the state is in this gender
state.sex = census %>%
  group_by(State)%>%
  summarise(Men=sum(Men),Women=sum(Women))%>%
  mutate(sex=ifelse(Men>Women,"Men","Women"))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```r
# remove Purto Rico from state.sex since this state is not included in state.winner
state.sex=state.sex[-which(!(state.sex$State %in% state.winner$state)),]
state.sex.result = left_join(state.sex,state.winner,by=c("State"="state"))%>%
  group_by(sex)%>%
  summarise(Trump=length(which(candidate=="Donald Trump")),
            Biden=length(which(candidate=="Joe Biden"))) %>%
  pivot_longer(cols=2:3,names_to="Candidate",values_to="Nstates")
```

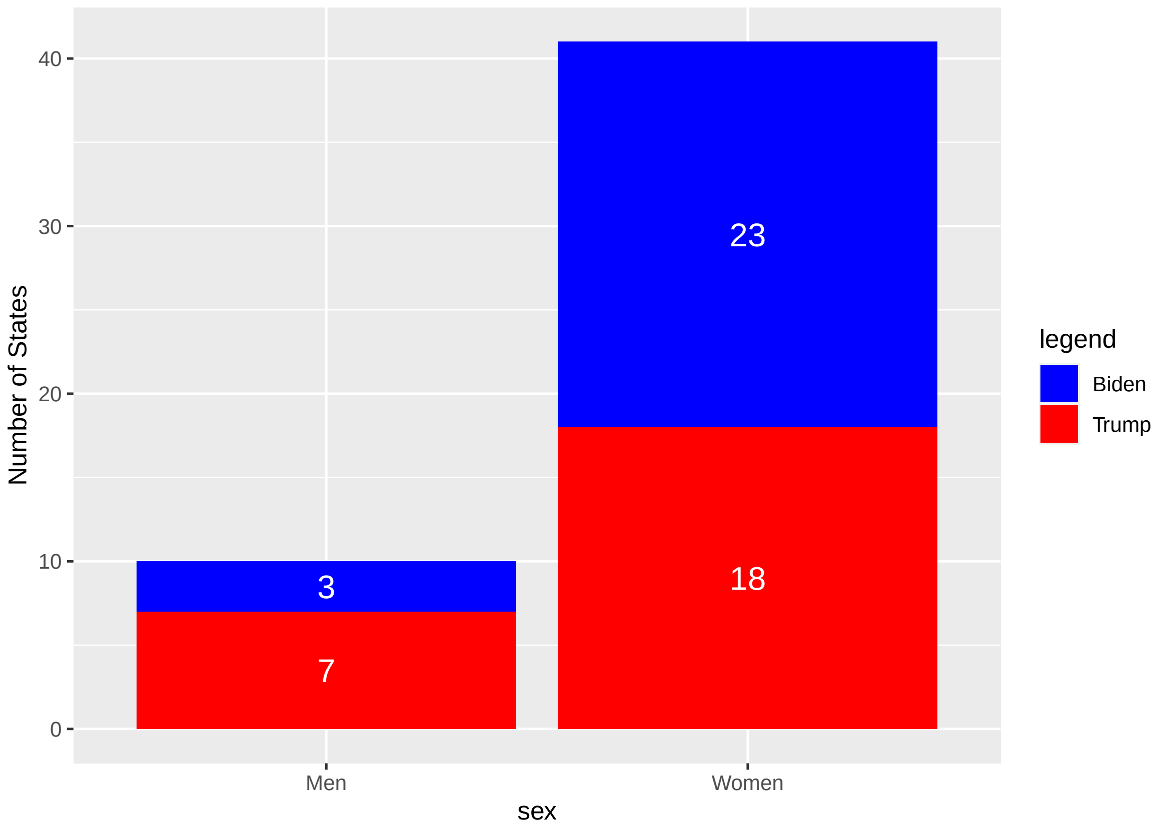
```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```r
# the last step reshapes the data for plot

state.sex.result
```

```
## # A tibble: 4 x 3
##   sex   Candidate Nstates
```

```
##    <chr> <chr>       <int>
## 1 Men   Trump           7
## 2 Men   Biden           3
## 3 Women Trump          18
## 4 Women Biden          23
```

```r
ggplot(state.sex.result, aes(fill=Candidate,y=Nstates, x=sex,label=Nstates))+
  geom_bar(position = "stack", stat="identity")+ylab("Number of States")+
  geom_text(size = 5, position = position_stack(vjust = 0.5),col="white")+
  scale_fill_manual("legend", values=c("Trump"="red","Biden"="blue"))
```



The x-axis denotes the dominant gender group of a state, men or women. If a state has more male population than female population from the `census` data, then we categorize the state as dominant by male group and vice versa. The y-axis denotes the number of states that are dominant by each gender group. The red color means the state winner is Donald Trump while the blue color means the state winner is Joe Biden.
From this graph, we can see that in the 51 states of the United States, $3 + 7 = 10$ states have more male population while $23 + 18 = 41$ states have more female population. Among the 10 states whose major gender group are men, 3 of them had Joe Biden as the state winner and 7 of them had Donald Trump as the state winner. Among the 41 states whose major gender group are women, 23 of them had Joe Biden as the state winner and 18 of them had Donald Trump as the state winner. These show that state with more women is slightly more likely to support Biden than Trump, while state with more men is much more likely to support Trump than Biden.

10. **The census data contains county-level census information. In this problem, we clean and aggregate the information as follows.**

    - Clean county-level census data `census.clean`: start with `census`, filter out any rows with missing values, convert {`Men`, `Employed`, `VotingAgeCitizen`} attributes to percentages, compute `Minority`

attribute by combining {Hispanic, Black, Native, Asian, Pacific}, remove these variables after creating `Minority`, remove {`IncomeErr`, `IncomePerCap`, `IncomePerCapErr`, `Walk`, `PublicWork`, `Construction`}.
Many columns are perfectly colineared, in which case one column should be deleted.

```
census.clean = census %>%
  drop_na() %>%
  mutate_at(vars(Men, Employed, VotingAgeCitizen),list(~./TotalPop*100)) %>%
  mutate(Minority = Hispanic+Black+Native+Asian+Pacific)%>%
  dplyr::select(-c(Hispanic, Black, Native, Asian, Pacific,IncomeErr,
            IncomePerCap, IncomePerCapErr, Walk, PublicWork, Construction))
# change the order of columns so that 'Minority' is next to 'White'
census.clean = census.clean[,c(1:6,27,7:26)]

# Find perfectly colineared columns
correlation = cor(census.clean[-c(1:3)],method = "pearson")
index <- which(abs(correlation) > .9 & abs(correlation) != 1,
               # criteria for perfect colinearity
               arr.ind = T)
# the result of the which function is now in rows & columns
cbind.data.frame(col1 = rownames(correlation)[index[,1]], # get the row name
                 col2 = colnames(correlation)[index[,2]]) # get the column name
```

```
##          col1        col2
## 1       Women    TotalPop
## 2    TotalPop       Women
## 3       White    Minority
## 4    Minority       White
## 5 ChildPoverty     Poverty
## 6     Poverty ChildPoverty
```

We find the perfectly colineared columns by using the correlation matrix. If two different columns are perfectly colineared, then the correlation between them should be very close to either 1 or -1. Here we consider a correlation whose absolute value is larger than 0.9 as a symbol of perfect colinearity. Then we detect the perfect colinear relationship between column `Women` and `TotalPop`, `Minority` and `White`, and `Poverty` and `ChildPoverty`. Thus, to avoid perfect colinearity, we remove columns `Women`, `White`, and `Poverty` from `census.clean`.

```
census.clean=census.clean %>%
  dplyr::select(-c(Women,White,Poverty))
```

- Print the first 5 rows of `census.clean`:

```
head(census.clean,5)
```

```
## # A tibble: 5 x 24
##   CountyId State County TotalPop   Men Minority VotingAgeCitizen Income
##      <dbl> <chr> <chr>     <dbl> <dbl>    <dbl>            <dbl>  <dbl>
## 1     1001 Alab~ Autau~    55036  48.9     22.8             74.5  55317
## 2     1003 Alab~ Baldw~   203360  48.9     15.4             76.4  52562
## 3     1005 Alab~ Barbo~    26201  53.3     52.8             77.4  33368
## 4     1007 Alab~ Bibb ~    22580  54.3     24.8             78.2  43404
## 5     1009 Alab~ Bloun~    57667  49.4     10.9             73.7  47412
## # ... with 16 more variables: ChildPoverty <dbl>, Professional <dbl>,
## #   Service <dbl>, Office <dbl>, Production <dbl>, Drive <dbl>, Carpool <dbl>,
## #   Transit <dbl>, OtherTransp <dbl>, WorkAtHome <dbl>, MeanCommute <dbl>,
## #   Employed <dbl>, PrivateWork <dbl>, SelfEmployed <dbl>, FamilyWork <dbl>,
```

```
## #   Unemployment <dbl>
```

Dimensionality reduction

11. **Run PCA for the cleaned county level census data (with `State` and `County` excluded).**

    - Discuss whether you chose to center and scale the features before running PCA and the reasons for your choice.

First, we should also exclude the `CountyId` variable because it is just a numerical indicator of County name, which has the same function as `County`. When running PCA on `census.clean`, we are only interested in the unsupervised learning regime for now, where our focus is on reducing the dimension of the census data rather than County names. So we can exclude `CountyId` when doing PCA.
Second, we need to center and scale the features before running PCA because centering is required before performing PCA and the and the features are recorded on different scales. Variable `TotalPop` measures the exact number of total population in the county and variable `Income` measures the median household income ($), which are incomparable with other variables that measure the percentage of the total population with specific features. This can also be reflected from the obviously larger mean and variance for `TotalPop` and `Income` than for other other variables as shown below.

```r
# mean of all the variables
apply(census.clean[-c(1:3)],2,mean)
```

```
##        TotalPop             Men        Minority VotingAgeCitizen
##       1.008e+05       5.004e+01       2.311e+01        7.501e+01
##          Income    ChildPoverty    Professional          Service
##       4.899e+04       2.304e+01       3.148e+01        1.821e+01
##          Office      Production           Drive          Carpool
##       2.188e+01       1.583e+01       7.965e+01        9.852e+00
##         Transit     OtherTransp      WorkAtHome      MeanCommute
##       9.393e-01       1.596e+00       4.736e+00        2.348e+01
##        Employed     PrivateWork     SelfEmployed      FamilyWork
##       4.343e+01       7.488e+01       7.774e+00        2.789e-01
##    Unemployment
##       6.668e+00
```

```r
# variance or all the variables
apply(census.clean[-c(1:3)],2,var)
```

```
##        TotalPop             Men        Minority VotingAgeCitizen
##       1.053e+11       5.837e+00       5.308e+02        2.758e+01
##          Income    ChildPoverty    Professional          Service
##       1.926e+08       1.414e+02       4.255e+01        1.389e+01
##          Office      Production           Drive          Carpool
##       1.003e+01       3.374e+01       5.807e+01        8.782e+00
##         Transit     OtherTransp      WorkAtHome      MeanCommute
##       9.443e+00       2.790e+00       9.448e+00        3.227e+01
##        Employed     PrivateWork     SelfEmployed      FamilyWork
##       4.835e+01       5.801e+01       1.486e+01        2.008e-01
##    Unemployment
##       1.422e+01
```

If we failed to center and scale the variables before performing PCA, then most of the principal components that we observed would be driven by the `TotalPop` variable since it has by far the largest mean and variance among all the variables as shown above.

- Save the first two principle components PC1 and PC2 into a two-column data frame, call it `pc.county`. 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 correlation between these features?

```r
pr.out=prcomp(census.clean[-c(1:3)],scale=TRUE, center = TRUE)
PC1 = pr.out$rotation[,1] # first PC
PC2 = pr.out$rotation[,2] # second PC
pc.county = data.frame(cbind(PC1,PC2))
pc.county
```

```
##                         PC1      PC2
## TotalPop           0.069520 -0.12171
## Men                0.024362  0.18339
## Minority          -0.248420  0.12468
## VotingAgeCitizen  -0.024651  0.04199
## Income             0.366184 -0.19563
## ChildPoverty      -0.395330  0.14344
## Professional       0.321414  0.01847
## Service           -0.209813  0.18473
## Office            -0.085058 -0.22276
## Production        -0.166453 -0.16792
## Drive             -0.194421 -0.34954
## Carpool           -0.072239  0.10644
## Transit            0.097196 -0.01928
## OtherTransp        0.015780  0.18876
## WorkAtHome         0.282258  0.30356
## MeanCommute       -0.109073 -0.19422
## Employed           0.390723 -0.17011
## PrivateWork        0.004528 -0.48041
## SelfEmployed       0.178168  0.35984
## FamilyWork         0.100474  0.24948
## Unemployment      -0.347566  0.11081
```

```r
# Find the three features with the largest abs values of PC1
rownames(pc.county)[order(abs(pc.county[,1]),decreasing = TRUE)[c(1:3)]]
```

```
## [1] "ChildPoverty" "Employed"     "Income"
```

```r
# Find features with opposite signs
rownames(pc.county)[which(PC1*PC2<0)]
```

```
##  [1] "TotalPop"         "Minority"         "VotingAgeCitizen" "Income"
##  [5] "ChildPoverty"     "Service"          "Carpool"          "Transit"
##  [9] "Employed"         "PrivateWork"      "Unemployment"
```

Three features with the largest absolute values of the first principal component, listed in the order of large to small, are `ChildPoverty`, `Employed`, and `Income`.

Features `TotalPop`, `Minority`, `VotingAgeCitizen`, `Income`, `ChildPoverty`, `Service`, `Carpool`, `Transit`, `Employed`, `Privatework`, and `Unemployment` have opposite signs. Among these features with oppostive signs, we can divide them into two groups: features with negative PC1 and positive PC2, and features with positive PC1 and negative PC2.

```r
# Group 1: features with negative PC1 and positive PC2
rownames(pc.county)[which(PC1<0&PC2>0)]
```

```
## [1] "Minority"         "VotingAgeCitizen" "ChildPoverty"     "Service"
## [5] "Carpool"          "Unemployment"
```

```r
# Group 2: features with positive PC1 and negative PC2
rownames(pc.county)[which(PC1>0&PC2<0)]
```

```
## [1] "TotalPop"    "Income"      "Transit"     "Employed"    "PrivateWork"
```

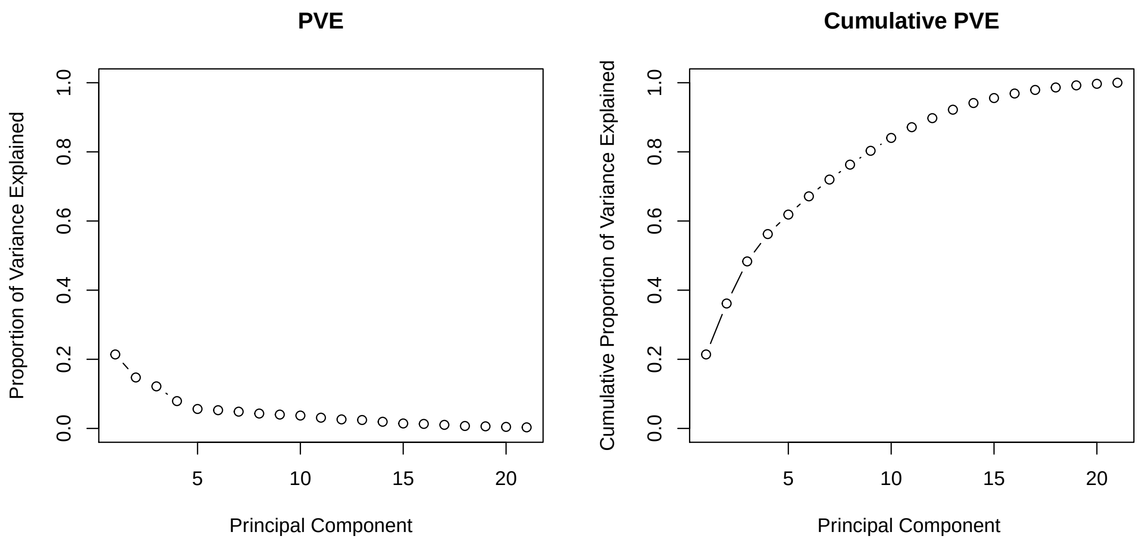
```r
# Check the correlation between features with opposite signs
cor(census.clean[c(rownames(pc.county)[which(PC1*PC2<0)])])
```

```
##                   TotalPop Minority VotingAgeCitizen  Income ChildPoverty
## TotalPop          1.000000  0.18290         -0.25101  0.2434     -0.06360
## Minority          0.182898  1.00000         -0.41495 -0.2857      0.59375
## VotingAgeCitizen -0.251011 -0.41495          1.00000 -0.2312      0.01333
## Income            0.243441 -0.28575         -0.23122  1.0000     -0.75309
## ChildPoverty     -0.063595  0.59375          0.01333 -0.7531      1.00000
## Service          -0.004349  0.30360          0.13439 -0.3624      0.37131
## Carpool          -0.066870  0.08825         -0.21418 -0.1233      0.08396
## Transit           0.401644  0.16547         -0.17544  0.2583     -0.05019
## Employed          0.147697 -0.43456         -0.11733  0.7210     -0.74558
## PrivateWork       0.197212 -0.22545         -0.09059  0.2480     -0.19399
## Unemployment      0.007689  0.57466          0.01650 -0.5065      0.68577
##                    Service  Carpool  Transit Employed PrivateWork Unemployment
## TotalPop         -0.004349 -0.06687  0.40164   0.1477     0.19721     0.007689
## Minority          0.303601  0.08825  0.16547  -0.4346    -0.22545     0.574659
## VotingAgeCitizen  0.134391 -0.21418 -0.17544  -0.1173    -0.09059     0.016496
## Income           -0.362393 -0.12327  0.25829   0.7210     0.24798    -0.506545
## ChildPoverty      0.371314  0.08396 -0.05019  -0.7456    -0.19399     0.685765
## Service           1.000000  0.07648  0.04479  -0.3801    -0.21787     0.348809
## Carpool           0.076475  1.00000 -0.09935  -0.1224    -0.07425     0.047816
## Transit           0.044789 -0.09935  1.00000   0.1648     0.07969     0.016050
## Employed         -0.380081 -0.12240  0.16476   1.0000     0.29033    -0.672256
## PrivateWork      -0.217873 -0.07425  0.07969   0.2903     1.00000    -0.182100
## Unemployment      0.348809  0.04782  0.01605  -0.6723    -0.18210     1.000000
```

Features within the each group are mostly positively correlated while features of different group are mostly negatively correlated with each other.

12. **Determine the number of minimum number of PCs needed to capture 90% of the variance for the analysis.** Plot proportion of variance explained (PVE) and cumulative PVE.

```r
pr.var=pr.out$sdev^2
pve=pr.var/sum(pr.var)
op <- par(mfrow=c(1,2))
# Plot the proportion of variance explained by each principal component (PVE)
plot(pve, xlab="Principal Component", ylab="Proportion of Variance Explained",
     ylim=c(0,1),type='b',main="PVE")

# Plot the cumulative PVE
plot(cumsum(pve), xlab="Principal Component ",
     ylab=" Cumulative Proportion of Variance Explained ", ylim=c(0,1), type='b',
     main="Cumulative PVE")
```




```
par(op)
# find number of PCs to capture 90% of the total variation
which(cumsum(pve)>=0.9)[1]
```

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

```
cumsum(pve)[which(cumsum(pve)>=0.9)[1]]
```

```
## [1] 0.9219
```

We need 13 PCs to capture 90% of the total variation.

Clustering

13. **With `census.clean` (with `State` and `County` excluded), 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 from `pc.county` as inputs instead of the original features. Compare the results and comment on your observations.

    As we have discussed in part 11, we will also exclude `CountyId` when performing hierarchical clustering. And we need to scale our data since the features are recorded on different scales.

```
# First approach: hierarchical clustering with complete linkage on census.clean
## Compute the euclidean distance matrix
census.dist = dist(scale(census.clean[-c(1:3)]), method = "euclidean")
## Agglomerative Hierarchical clustering using complete linkage
set.seed(1)
census.hclust = hclust(census.dist,method="complete")
## cut tree to partition the observations into 10 clusters
clus = cutree(census.hclust,k=10)

# Second approach: use the first 2 principal components from pc.county for HC
pc.county.scores = data.frame(pr.out$x[,c(1,2)])
census.pc.dist = dist(pc.county.scores, method = "euclidean")
census.pc.hclust = hclust(census.pc.dist, method = "complete")
clus.pc = cutree(census.pc.hclust,k=10)
```

```r
# compare
table(clus)
```

```
## clus
##    1    2    3    4    5    6    7    8    9   10
## 2975   24    6  145   13    1   14    7   30    4
```

```r
table(clus.pc)
```

```
## clus.pc
##    1    2    3    4    5    6    7    8    9   10
## 1441  891  249  118   20  339   31    1  119   10
```

For simplicity, we denote the hierarchical clustering on `census.clean` with original features as the HC approach, and hierarchical clustering algorithm using the first 2 principal components from `pc.county` as inputs as HCPC approach. By using HC approach, the cluster 1 we obtainied contains 2975 counties, which are far more than the number of counties that each of the other 9 clusters contains. However, by performing the HCPC approach, the counties are distributed more evenly into the 10 clusters than the first approach. But most of counties are still placed into cluster 1. Clusters do not spread out too much.

- For both approaches investigate the cluster that contains *Santa Barbara County*. Which approach seemed to put Santa Barbara County in a more appropriate clusters? Comment on what you observe and discuss possible explanations for these observations.

First, we try to find which cluster contains *Santa Barbara County*.

```r
# Find index of Santa Barbara County
sb.idx = which(census.clean$County=="Santa Barbara County")

# Find which cluster contains SB county, HC
clus[sb.idx]
```

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

```r
sb.hc.cluster = census.clean[which(clus==clus[sb.idx]),]
#length(which(sb.hc.cluster$State=="California"))

# Find which cluster contains SB county, HCPC
clus.pc[sb.idx]
```

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

```r
sb.hcpc.cluster=census.clean[which(clus.pc==clus.pc[sb.idx]),]
#length(which(sb.hcpc.cluster$State=="California"))
```

We find that HC approach placed Santa Barbara County into cluster 1 of 2975 counties while HCPC approach placed Santa Barbara County into cluster 6 of 339 counties.
Next, we will evaluate which approach seemed to put Santa Barbara County in a more appropriate cluster by using internal cluster validation. We will apply two commonly used indices for assessing the goodness of clustering: the silhouette width S_i and the Dunn index.
The silhouette width measures how well an observation is clustered and it estimates the average distance between clusters. Observations with a large S_i (almost 1) are very well clustered while observations with a negative S_i are probably placed in the wrong cluster. (Source: https://www.datanovia.com/en/lessons/cluster-validation-statistics-must-know-methods/#silhouette-coefficient)
The Dunn Index is the ratio of the smallest distance between observations not in the same cluster to the largest intra-cluster distance. The Dunn Index has a value between zero and infinity, and should be maximized. (Source: http://finzi.psych.upenn.edu/R/library/clValid/html/dunn.html)

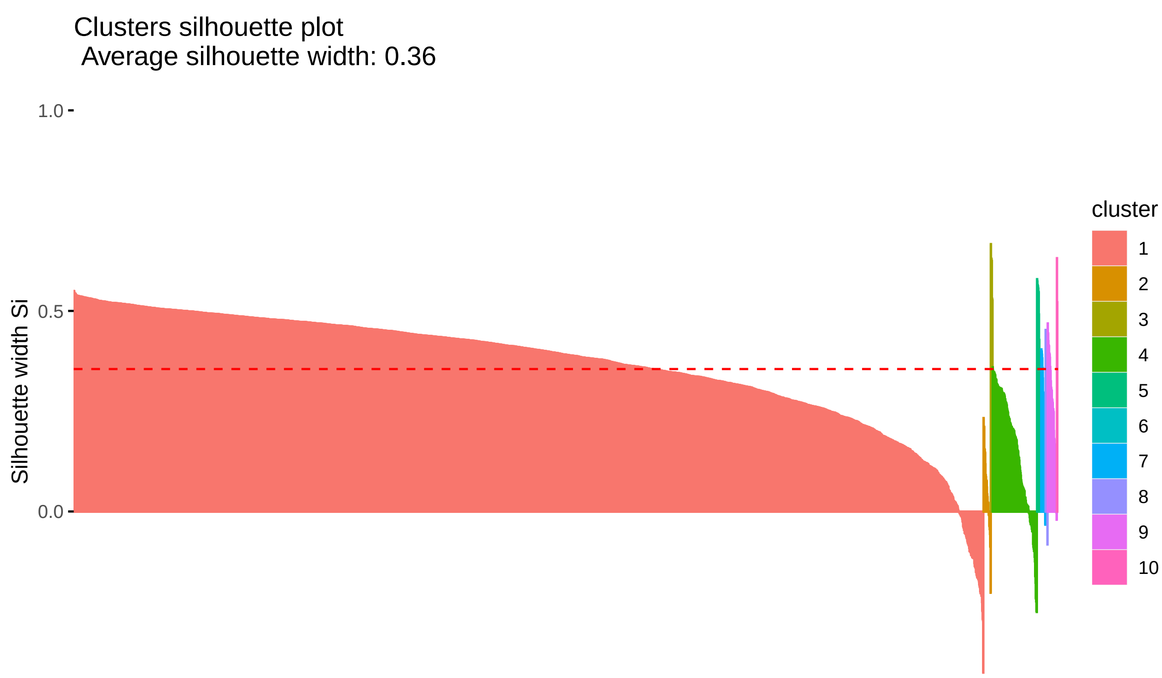
```r
sil.hc = silhouette(clus,census.dist)
sil.hc[sb.idx,]
```

```
##   cluster  neighbor sil_width
##    1.0000    4.0000    0.3013
```

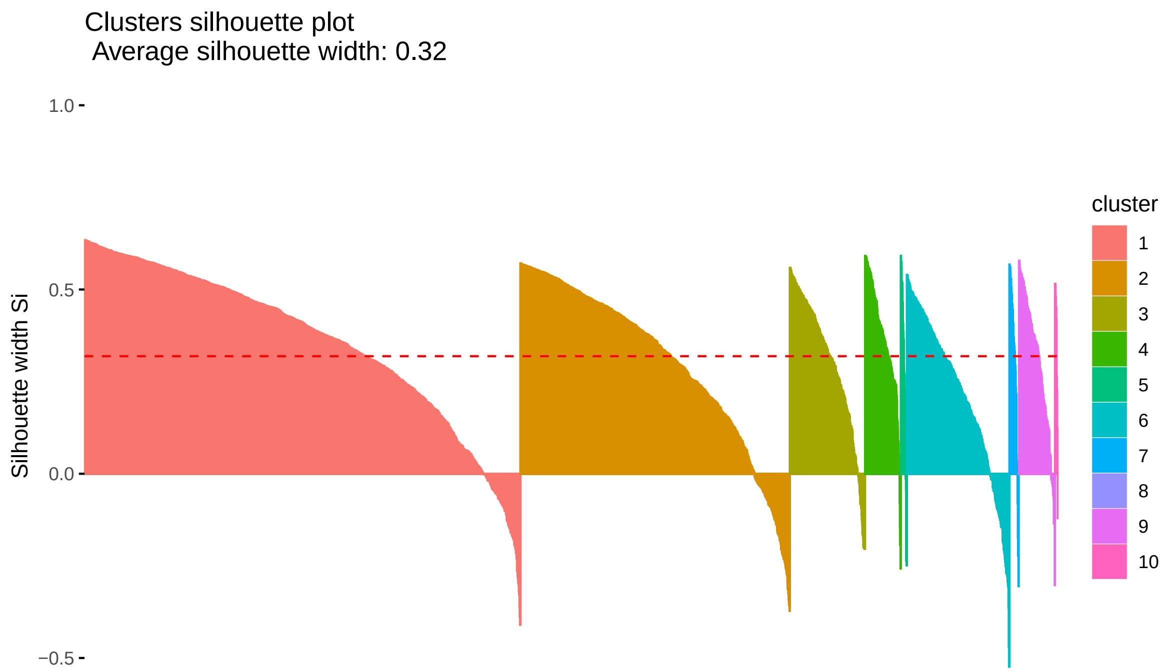
```r
sil.hcpc = silhouette(clus.pc, census.pc.dist)
sil.hcpc[sb.idx,]
```

```
##   cluster  neighbor sil_width
##     6.000     4.000     0.456
```

```r
fviz_silhouette(sil.hc, print.summary = FALSE)
```




```r
fviz_silhouette(sil.hcpc, print.summary = FALSE)
```



By examining silhouette width for *Santa Barbara County* with different approaches, we find that $S_{\text{Santa Barbara County}}$ with HC approach (0.3013) is slightly smaller than $S_{\text{Santa Barbara County}}$ with HCPC approach (0.456). It seems to imply that HCPC approach placed *Santa Barbara County* in a more appropriate cluster. However, by comparing the silhouette plot for HC and HCPC approaches, we find that far less fraction of counties in cluster 1 from HC approach contains have negative S_i than cluster 6 from HCPC approach does. It means that larger fraction of counties in cluster 1 from HC approach are placed correctly than in cluster 6 from HCPC approach. So the HC approach might perform better than HCPC approach. We will check our conclusion next by using Dunn index.

```
dunn(census.dist,clus)
```

```
## [1] 0.1217
```

```
dunn(census.pc.dist,clus.pc)
```

```
## [1] 0.007661
```

We can see that the Dunn index of HC approach (0.1217) is much larger than that of HCPC approach (0.007661). Thus, HC approach contains more compact and well-separated clusters than HCPC approach does.

Finally, we check the within-cluster variation of clusters containing *Santa Barbara County* from different appraoch.

```
apply(sb.hc.cluster[-c(1:3)],2,var)<apply(sb.hcpc.cluster[-c(1:3)],2,var)
```

```
##        TotalPop             Men        Minority VotingAgeCitizen
##           FALSE            TRUE           FALSE             TRUE
##          Income    ChildPoverty    Professional          Service
##           FALSE           FALSE           FALSE             TRUE
##          Office      Production           Drive          Carpool
##            TRUE           FALSE            TRUE             TRUE
##         Transit     OtherTransp       WorkAtHome      MeanCommute
```

```
##               TRUE             TRUE             TRUE            FALSE
##           Employed      PrivateWork     SelfEmployed       FamilyWork
##              FALSE             TRUE             TRUE             TRUE
##       Unemployment
##              FALSE
```

```r
sum(apply(sb.hc.cluster[-c(1:3)],2,var)<apply(sb.hcpc.cluster[-c(1:3)],2,var))
```

```
## [1] 12
```

By comparing the within-cluster variation of the cluster containing *Santa Barbara County* between HC and HCPC approach, we can see that 12 out of the 21 variances of each feature for counties in cluster 1 from HC approach are smaller than variances of the features for counties in cluster 6 from HCPC approach. This implies that counties in the same cluster with *Santa Barbara County* from HC approach are more similar to each other than counties in the same cluster with *Santa Barbara County* from HCPC approach are.
In general, we can conclude that HC approach placed *Santa Barbara County* in a more appropriate cluster than the HCPC approach did. The reason might be that the first two principal components do not describe most of the variance in `census.clean` and hence there are more disagreements inside a cluster.

Classification

We start considering supervised learning tasks now. The most interesting/important question to ask is: *can we use census information in a county to predict the winner in that county?*
In order to build classification models, we first need to combine `county.winner` and `census.clean` data. This seemingly straightforward task is harder than it sounds. For simplicity, the following code makes necessary changes to merge them into `election.cl` for classification.

```r
# we move all state and county names into lower-case
tmpwinner <- county.winner %>% ungroup %>%
  mutate_at(vars(state, county), tolower)

# we move all state and county names into lower-case
# we further remove suffixes of "county" and "parish"
tmpcensus <- census.clean %>% mutate_at(vars(State, County), tolower) %>%
  mutate(County = gsub(" county|  parish", "", County))

# we join the two datasets
election.cl <- tmpwinner %>%
  left_join(tmpcensus, by = c("state"="State", "county"="County")) %>%
  na.omit

# drop levels of county winners if you haven't done so in previous parts
election.cl$candidate <- droplevels(election.cl$candidate)

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

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

14. **Understand the code above. Why do we need to exclude the predictor party from `election.cl`?**

We exclude the predictor `party` from `election.cl` because it is not useful for predicting the winner in a county. Note that the two candidates, Donald Trump and Joe Bider, belong to different party. Thus, to predict the winner in a county is equivalent to predict the party that the winner represents for. In supervised learning we only need one response, so we need to exclude the predictor `party` from `election.cl`.

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

```
set.seed(10)
n <- nrow(election.cl)
idx.tr <- sample.int(n, 0.8*n)
election.tr <- election.cl[idx.tr, ]
election.te <- election.cl[-idx.tr, ]
```

Use the following code to define 10 cross-validation folds:

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

Using the following error rate function. And the object `records` is used to record the classification performance of each method in the subsequent problems.

```
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

15. **Decision tree: train a decision tree by `cv.tree()`.**

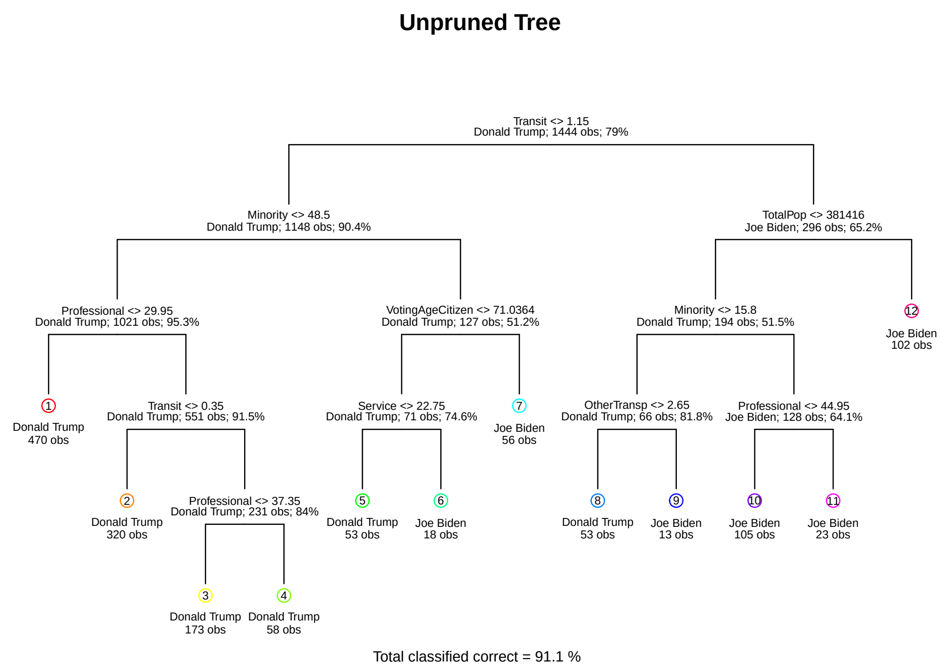
- Prune tree to minimize misclassification error. Be sure to use the folds from above for cross-validation. Visualize the trees before and after pruning.

```
# Setting up the X and Y variables for convenience
x.tr = election.tr[-1]
y.tr = election.tr$candidate
x.te = election.te[-1]
y.te = election.te$candidate

set.seed(3)

# Construct a single decision tree
tree.election = tree(candidate~., data=election.tr)

# Using CV to find best size with the folds defined above
cv=cv.tree(tree.election,FUN = prune.misclass,rand=folds)
best_size = min(cv$size[cv$dev == min(cv$dev)])
best_size
```

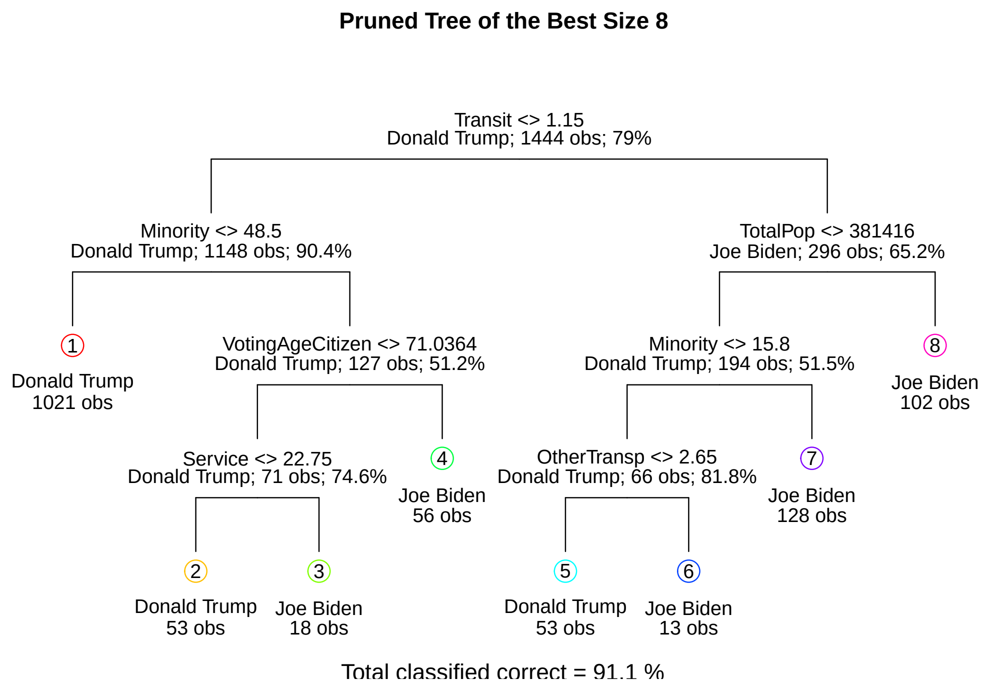
```
## [1] 8
```

```
# Prune tree.election
pt.election = prune.misclass(tree.election, best=best_size)
```

```r
# visualize tree before pruning
draw.tree(tree.election, nodeinfo = TRUE, cex=0.6)
title("Unpruned Tree")
```




```r
# Visualize pruned tree
draw.tree(pt.election, nodeinfo = TRUE)
title("Pruned Tree of the Best Size 8")
```



- Save training and test errors to `records` object.

```
# training error
tree.tr.pred = predict(pt.election,newdata = x.tr, type="class")
tree.tr.err = calc_error_rate(tree.tr.pred, y.tr)

# test error
tree.te.pred = predict(pt.election,newdata = x.te, type="class")
tree.te.err = calc_error_rate(tree.te.pred, y.te)

# save to records
records[1,1] = tree.tr.err
records[1,2] = tree.te.err
records
```

```
##          train.error test.error
## tree         0.08864     0.1357
## logistic          NA         NA
## lasso             NA         NA
```

- Interpret and discuss the results of the decision tree analysis. Use this plot to tell a story about voting behavior.

  The best size for the decision tree that minimizes misclassification error is 8. After pruning the tree, our training error is 0.08864 and our test error is 0.1357. The test error is much larger than the training error, so the decision tree for election result has low bias and high variance.

  From the plot of the pruned tree, we can say that a county whose percentage of population commuting on public transportation is less than 1.15 and percentage of population that is minority is less than 48.5 is most likely to vote for Donald Trump.

16. **Run a logistic regression to predict the winning candidate in each county.**

- Run a logistic regression to predict the winning candidate in each county. Save training and test errors to `records` variable.

  In this part, we will simply use the "majority rule". If the predicted probability is larger than 50%, then we classify the predicted winner as Joe Biden. Otherwise, we classify the predicted winner as Donald Trump.

```
glm.fit = glm(candidate~., data=election.tr, family = binomial)
```

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

```
# Get the estimated probabilities
glm.tr.prob = predict(glm.fit,newdata = x.tr, type="response")
glm.te.prob = predict(glm.fit, newdata = x.te,type="response")
# Assign label using majority rule
glm.tr.pred = ifelse(glm.tr.prob<=0.5, "Donald Trump", "Joe Biden")
glm.te.pred = ifelse(glm.te.prob<=0.5, "Donald Trump", "Joe Biden")

# Calculate error
glm.tr.err = calc_error_rate(glm.tr.pred, y.tr)
glm.te.err = calc_error_rate(glm.te.pred, y.te)
cat("\n")
```

```
# save to records
records[2,1] = glm.tr.err
records[2,2] = glm.te.err
records
```

```
##          train.error test.error
## tree         0.08864    0.13573
## logistic     0.06787    0.08033
## lasso             NA         NA
```

- What are the significant variables? Are they 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.

```
# Find significant variables
summary(glm.fit)
```

```
##
## Call:
## glm(formula = candidate ~ ., family = binomial, data = election.tr)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -4.023  -0.257  -0.100  -0.024   3.468
##
## Coefficients:
##                    Estimate Std. Error z value Pr(>|z|)
## (Intercept)       -4.05e+01   8.39e+00   -4.83  1.4e-06 ***
## TotalPop           2.19e-06   7.60e-07    2.88  0.00393 **
## Men               -7.92e-03   5.97e-02   -0.13  0.89450
## Minority           1.38e-01   1.27e-02   10.85  < 2e-16 ***
## VotingAgeCitizen   1.70e-01   3.04e-02    5.58  2.4e-08 ***
## Income            -9.80e-06   1.81e-05   -0.54  0.58823
## ChildPoverty       1.18e-02   2.11e-02    0.56  0.57663
```

```
## Professional      3.21e-01   4.92e-02    6.52  7.2e-11 ***
## Service           3.46e-01   6.11e-02    5.65  1.6e-08 ***
## Office            1.91e-01   6.17e-02    3.09  0.00197 **
## Production        2.43e-01   5.29e-02    4.60  4.3e-06 ***
## Drive            -2.06e-01   5.44e-02   -3.78  0.00015 ***
## Carpool          -1.96e-01   7.18e-02   -2.73  0.00629 **
## Transit           8.58e-02   1.14e-01    0.75  0.45185
## OtherTransp       1.54e-01   1.13e-01    1.36  0.17350
## WorkAtHome       -1.10e-01   8.48e-02   -1.30  0.19311
## MeanCommute       2.94e-02   3.17e-02    0.93  0.35422
## Employed          2.45e-01   4.13e-02    5.92  3.1e-09 ***
## PrivateWork       4.88e-02   2.58e-02    1.89  0.05851 .
## SelfEmployed     -1.57e-03   5.74e-02   -0.03  0.97817
## FamilyWork       -3.61e-01   3.31e-01   -1.09  0.27586
## Unemployment      1.52e-01   5.09e-02    2.98  0.00284 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1483.67  on 1443  degrees of freedom
## Residual deviance:  529.79  on 1422  degrees of freedom
## AIC: 573.8
##
## Number of Fisher Scoring iterations: 7
```

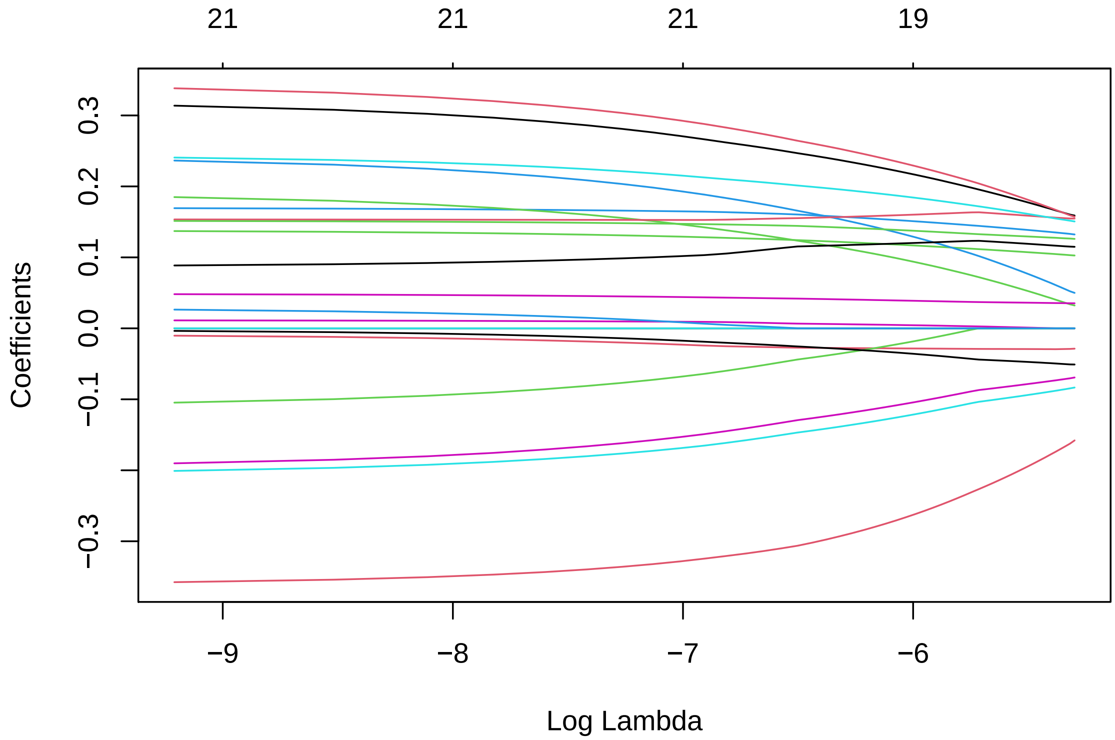
From the above results from `summary()` function, we can see that variables `TotalPop`, `Minority`, `VotingAgeCitizen`, `Professional`, `Service`, `Office`, `Production`, `Drive`, `Carpool`, `Employed`, and `Unemployment` are significant at levels 0.05 since their p-values are less than 0.05. The significant variables are slightly different from those in decision tree analysis. Only variable `Minority`, `TotalPop`, `VotingAgeCitizen`, and `Service` are significant for both methods.
The logistic regression coefficients, if logit link function is used, give the change in the log odds of the outcome for a one unit increase in a predictor variable, while others being held constant. Therefore:
* The variable `TotalPop` has a coefficient $2.19e-06$. For every one person change in `TotalPop`, the log odds of county winner being Joe Biden (versus county winner being Donald Trump) increases by $2.19e-06$, holding other variables fixed.
* The variable `Minority` has a coefficient $1.38e-01 = 0.138$. For every one percent change in `Minorty`, the log odds of county winner being Joe Biden (versus county winner being Donald Trump) increases by 0.138, holding other variables fixed.
* The variable `VotingAgeCitizen` has a coefficient $1.70e-01 = 0.17$. For every one percent change in `VotingAgeCitizen`, the log odds of county winner being Joe Biden (versus county winner being Donald Trump) increases by 0.17, holding other variables fixed.
* The variable `Professional` has a coefficient $3.21e-01 = 0.321$. For a one percent increase in `Professional`, the log odds of county winner being Joe Biden (versus county winner being Donald Trump) increases by 0.321, holding other variables fixed.
* The variable `Drive` has a coefficient $-2.06e-01 = -0.206$. For a one percent increase in `Drive`, the log odds of county winner being Joe Biden (versus county winner being Donald Trump) decrease by 0.206, holding other variables fixed.
* The same logic goes for other significant predictor variables.

17. 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.

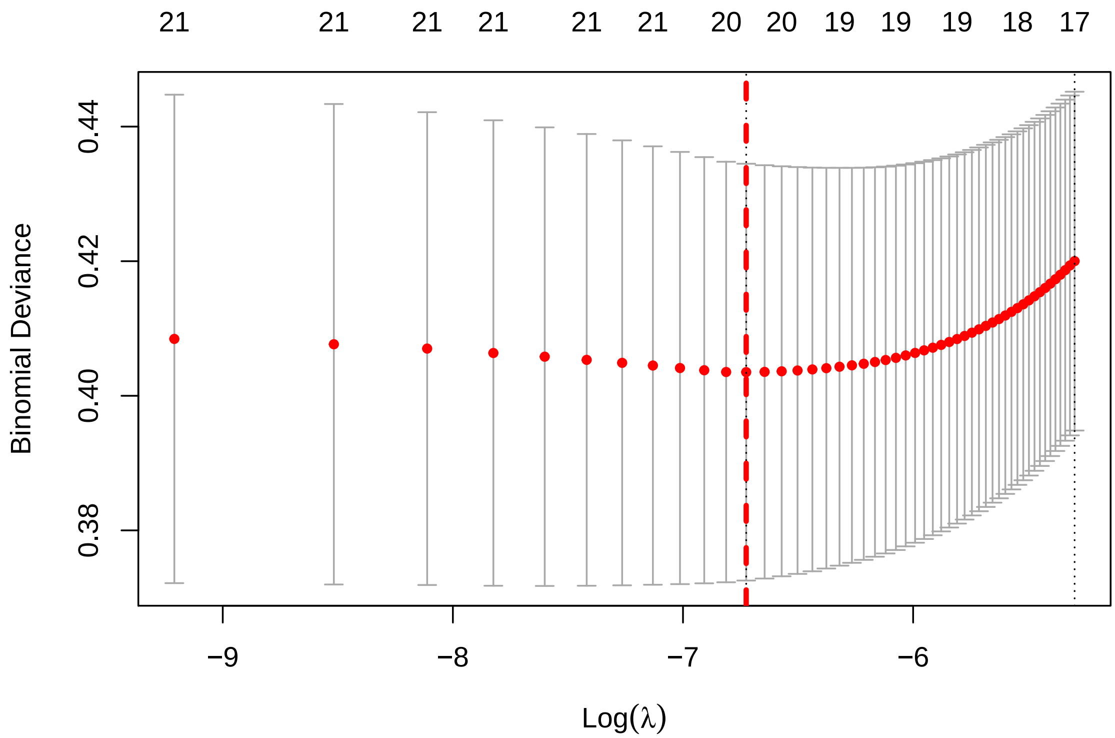
- Use the `cv.glmnet` function from the `glmnet` library to run a 10-fold cross validation and select the best regularization parameter for the logistic regression with LASSO penalty. Set `lambda = seq(1, 50) * 1e-4` in `cv.glmnet()` function to set pre-defined candidate values for the tuning parameter λ.

  Here we transform categorical variable `candidate` into numerical variables. We use '0' for "Donald Trump" and '1' for "Joe Biden".

```
x.tr.lasso=model.matrix(candidate~., election.tr)[,-1]
y.tr.lasso=ifelse(y.tr=="Joe Biden",1,0)
# fit the lasso regression model on the training dataset
lasso.mod = glmnet(x.tr.lasso, y.tr.lasso,alpha = 1, lambda = seq(1, 50) * 1e-4,
                   family = "binomial")
plot(lasso.mod, xvar="lambda")
```




```
# use cross-validation to choose optimal lambda from the list above
set.seed(1)
cv.out.lasso=cv.glmnet(x.tr.lasso,y.tr.lasso,alpha = 1, lambda = seq(1, 50) * 1e-4,
                       foldid=folds, family="binomial")
plot(cv.out.lasso)
abline(v = log(cv.out.lasso$lambda.min), col="red", lwd=3, lty=2)
```



- 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? Comment on the comparison.

```
# The optimal value of lambda from the list
bestlam = cv.out.lasso$lambda.min
bestlam
```

```
## [1] 0.0012
```

```
# examine the coefficient estimates corresponding to the optimal value of lambda
lasso.mod = glmnet(x.tr.lasso, y.tr.lasso,alpha = 1, lambda = seq(1, 50) * 1e-4,
                   family = "binomial")
lasso.coef=predict(lasso.mod,type="coefficients",s=bestlam)[1:22,]
lasso.coef
```

```
##       (Intercept)          TotalPop               Men          Minority
##        -3.543e+01         2.132e-06        -2.579e-02         1.267e-01
## VotingAgeCitizen            Income      ChildPoverty      Professional
##         1.629e-01         0.000e+00         8.238e-03         2.580e-01
##           Service            Office        Production             Drive
##         2.781e-01         1.344e-01         1.790e-01        -1.576e-01
##           Carpool           Transit       OtherTransp        WorkAtHome
##        -1.408e-01         1.080e-01         1.538e-01        -5.567e-02
##       MeanCommute          Employed       PrivateWork      SelfEmployed
##         3.624e-03         2.078e-01         4.288e-02        -2.169e-02
##        FamilyWork      Unemployment
##        -3.172e-01         1.458e-01
```

```r
# Compare with unpenalized logistic regression
coef(glm.fit)
```

```
##      (Intercept)        TotalPop             Men        Minority
##       -4.054e+01       2.191e-06      -7.918e-03       1.382e-01
## VotingAgeCitizen          Income    ChildPoverty    Professional
##        1.698e-01      -9.799e-06       1.176e-02       3.206e-01
##          Service          Office      Production           Drive
##        3.456e-01       1.910e-01       2.431e-01      -2.059e-01
##          Carpool         Transit      OtherTransp      WorkAtHome
##       -1.960e-01       8.580e-02       1.537e-01      -1.104e-01
##      MeanCommute        Employed     PrivateWork     SelfEmployed
##        2.935e-02       2.447e-01       4.882e-02      -1.571e-03
##       FamilyWork    Unemployment
##       -3.609e-01       1.518e-01
```

The optimal value of λ in cross validation is 0.0012. The non-zero coefficients in the LASSO regreesion for the optimal value of λ are: `TotalPop`, `Men`, `Minority`, `VotingAgeCitizen`, `ChildPoverty`, `Professional`, `Service`, `Office`, `Production`, `Drive`, `Carpoop`, `Transit`, `OtherTransp`, `WorkAtHome`, `MeanCommute`, `Employed`, `PrivateWork`, `SelfEmployed`, `FamilyWork`, and `Unemployment`.

Compared to the unpenalized logistic regression, we find that coefficient of variable `Income` is set to zero. So the logistic regression with LASSO penalty and λ chosen by 10-fold cross validation did variable selection. The `Income` variable might not be associated with the response `candidate`.

- Save training and test errors to the `records` variable:

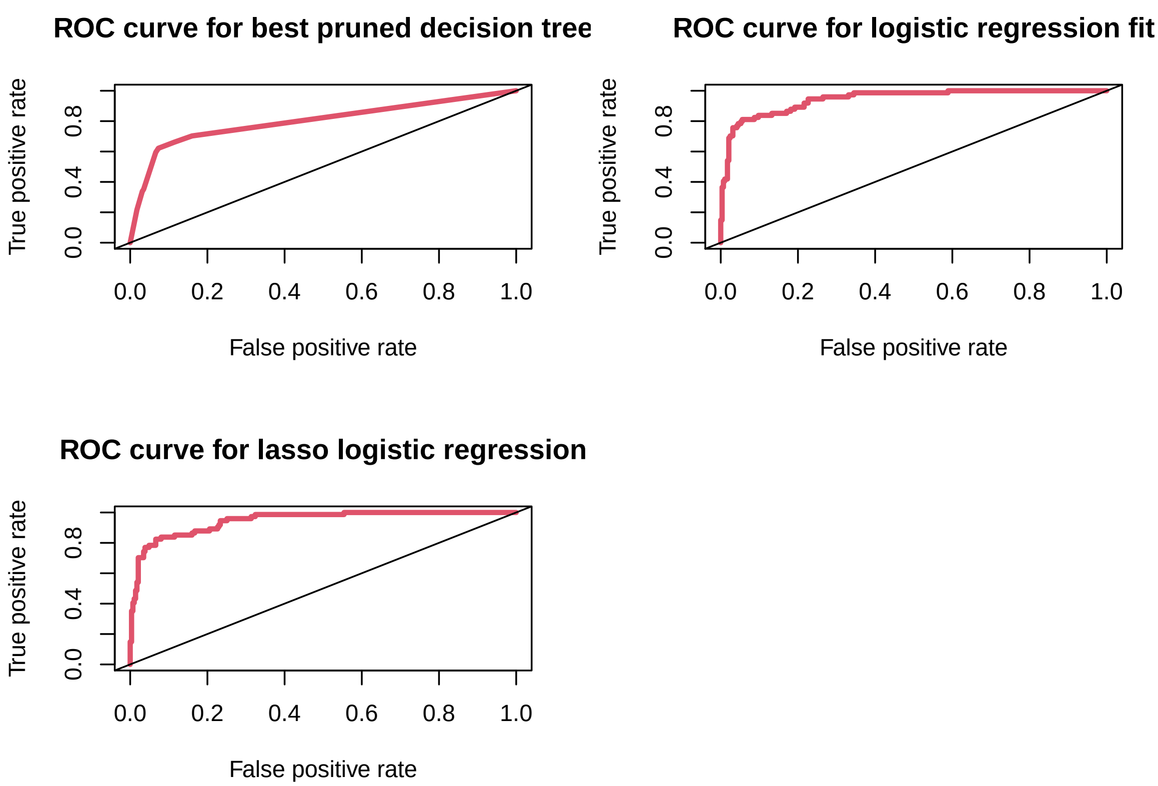
```r
# training error
lasso.tr.prob = predict(lasso.mod,s=bestlam,newx = x.tr.lasso, type = "response")
lasso.tr.pred = ifelse(lasso.tr.prob<=0.5, "Donald Trump", "Joe Biden")
lasso.tr.err = calc_error_rate(lasso.tr.pred, y.tr)

x.te.lasso=model.matrix(candidate~., election.te)[,-1]
# test error
lasso.te.prob = predict(lasso.mod,s=bestlam, newx=x.te.lasso, type = "response")
lasso.te.pred = ifelse(lasso.te.prob<=0.5, "Donald Trump", "Joe Biden")
lasso.te.err = calc_error_rate(lasso.te.pred, y.te)

records[3,1] = lasso.tr.err
records[3,2] = lasso.te.err
records
```

```
##          train.error test.error
## tree         0.08864    0.13573
## logistic     0.06787    0.08033
## lasso        0.06856    0.08310
```

18. **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.

```r
# prediction using decision tree
prob.tree=predict(pt.election, newdata = election.te)[,"Joe Biden"]
pred.tree = prediction(prob.tree,y.te)
# We want TPR on the y axis and FPR on the x axis
perf.tree = performance(pred.tree, measure="tpr", x.measure="fpr")

# prediction using logistic regression
```

```r
pred.glm = prediction(glm.te.prob, y.te)
perf.glm = performance(pred.glm, measure="tpr", x.measure="fpr")

# prediction using LASSO logistic regression
pred.lasso = prediction(lasso.te.prob, y.te)
perf.lasso = performance(pred.lasso, measure="tpr", x.measure="fpr")

op <- par(mfrow = c(2,2))
plot(perf.tree, col=2, lwd=3, main="ROC curve for best pruned decision tree")
abline(0,1)
plot(perf.glm, col=2, lwd=3, main="ROC curve for logistic regression fit")
abline(0,1)
plot(perf.lasso, col=2, lwd=3, main="ROC curve for lasso logistic regression")
abline(0,1)
par(op)
```



- 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?

```r
auc.tree = performance(pred.tree, "auc")@y.values
print(auc.tree)
```

```
## [[1]]
## [1] 0.7987
```

```r
auc.glm = performance(pred.glm, "auc")@y.values
print(auc.glm)
```

```
## [[1]]
## [1] 0.9447
```

```r
auc.lasso = performance(pred.lasso, "auc")@y.values
print(auc.lasso)
```

```
## [[1]]
## [1] 0.9459
```

Different classifiers are more appropriate for answering different kinds of questions about the election.

By calculating the AUCs from ROC curves, we find that the decision tree analysis has the lowest class separation capacity compared with the logistic regression and LASSO regression. And it also has low predictive accuracy as shown from the training error and test error. However, decision tree analysis is easy to explain and interpret. We can visualize the result. If we are interested in different possible outcomes of county winner depending on demographic data of a county, then the decision tree analysis is better than logistic regression and LASSO regression since it gives a visualization of how different voting behaviors lead to different possible outcomes straightforwardly. It is helpful to determine targeted voter groups when doing political compaigns.

Logistic regression gives higher class separation capacity and prediction accuracy and hence is suitable for analyzing and predciting the election outcome. And we can know which predictors are important by looking at the coefficients of the variables. But logistic regression may overfit the model and leads to the problem of perfect separation (some linear combination of variables perfectly predicts the winner). So logistic regression has to be used when predictors are not correlated.

Logistic regression with LASSO penalty slightly improves the class separation capacity of logistic regression. Even though the training error and test error of LASSO regression in our case is slightly greater than those of logistic regression, it selects variables and avoids overfitting. LASSO regression is appropriate for problems on providing reliable predictions of possible presidential winner with correlated predictors. However, LASSO regression cannot deal with situations where the number of features (p) is greater than the number of observations (n). Also, LASSO cannot do group selection in the process of variable selection. It will arbitrarily select only one feature from a group of correlated features.

Taking it further

19. **Explore additional classification methods.** Consider applying additional *two* classification methods from KNN, LDA, QDA, SVM, random forest, boosting, neural networks etc. How do these compare to the tree method, logistic regression, and the lasso logistic regression?

    For this problem, we will apply KNN and SVM methods.

- K-nearest Neighbor (KNN) with K chosen by cross validation:

  We first using 10-fold cross-validation to find the optimal k for KNN method. To do so, we define the `do.chunk()` function as we did in lab 4.

```r
# do.chunk() for k-fold Cross-validation
do.chunk <- function(chunkid, folddef, Xdat, Ydat, ...){
  # Get training index
  train = (folddef!=chunkid)
  # Get training set by the above index
  Xtr = Xdat[train,]
  # Get responses in training set
  Ytr = Ydat[train]
  # Get validation set
  Xvl = Xdat[!train,]
  # Get responses in validation set
  Yvl = Ydat[!train]
```

```r
  # Predict training labels
  predYtr = knn(train=Xtr, test=Xtr, cl=Ytr, ...)
  # Predict validation labels
  predYvl = knn(train=Xtr, test=Xvl, cl=Ytr, ...)
  data.frame(fold = chunkid,
             train.error = mean(predYtr != Ytr), # Training error for each fold
             val.error = mean(predYvl != Yvl)) # Validation error for each fold
}
```

Then we will carry out 10-fold cross-validation with the `folds` defind previously.

```r
# Set error.folds (a vector) to save validation errors in future
error.folds = NULL
# Give possible number of nearest neighbours to be considered
allK = 1:50
# Set seed since do.chunk() contains a random component induced by knn()
set.seed(888)

# Loop through different number of neighbors
for (k in allK){
  # Loop through different chunk id
  for (j in seq(nfold)){
    tmp = do.chunk(chunkid=j, folddef=folds, Xdat=x.tr, Ydat=y.tr, k=k)
    tmp$neighbors = k # Record the last number of neighbor
    error.folds = rbind(error.folds, tmp) # combine results
  }
}

# Transform the format of error.folds for further convenience
errors = melt(error.folds, id.vars=c('fold', 'neighbors'), value.name='error')
# Choose the number of neighbors which minimizes validation error
val.error.means = errors %>%
  # Select all rows of validation errors
  filter(variable=='val.error') %>%
  # Group the selected data frame by neighbors
  group_by(neighbors, variable) %>%
  # Calculate CV error rate for each k
  summarise_each(funs(mean), error) %>%
  # Remove existing group
  ungroup() %>%
  filter(error==min(error))
```

```
## Warning: `summarise_each_()` is deprecated as of dplyr 0.7.0.
## Please use `across()` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_warnings()` to see where this warning was generated.
```

```
## Warning: `funs()` is 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 every 8 hours.
## Call `lifecycle::last_warnings()` to see where this warning was generated.
```

```r
cat("\n")
```

```r
# Best number of neighbors
# if there is a tie, pick larger number of neighbors for simpler model
numneighbor = max(val.error.means$neighbors)
numneighbor
```

```
## [1] 26
```

Next, we can train a 26-NN classifier and add its training and test error to `records2`.

```r
set.seed(99)
# training error
knn.tr.pred = knn(train=x.tr, test=x.tr, cl=y.tr, k=numneighbor)
knn.tr.err = calc_error_rate(knn.tr.pred, y.tr)

# test error
knn.te.pred = knn(train=x.tr, test=x.te, cl=y.tr, k=numneighbor)
knn.te.err = calc_error_rate(knn.te.pred, y.te)


records2= matrix(NA,nrow = 2, ncol=2)
colnames(records2) = c("train.error","test.error")
rownames(records2) = c("KNN","SVM")
records2[1,1] = knn.tr.err
records2[1,2] = knn.te.err
records2
```

```
##     train.error test.error
## KNN      0.1427     0.1828
## SVM          NA         NA
```

- Support Vector Machine (SVM) with radial kernel with optimal cost from the list `c(0.001, 0.01, 0.1, 1, 10, 100)`:

```r
#find the cost of best SVM model
set.seed(1)
tune.out = tune(svm,candidate~., data=election.tr,kernel='radial',
                ranges=list(cost=c(0.001, 0.01, 0.1, 1, 10, 100)))

opt.cost = tune.out$best.parameters # optimal cost
opt.cost
```

```
##   cost
## 4    1
```

```r
# The best SVM model with the optimal cost
svmfit.best = tune.out$best.model

# training error
svm.tr.pred = predict(svmfit.best,newdata = election.tr)
svm.tr.err = calc_error_rate(svm.tr.pred, y.tr)
```

```r
# test error
svm.te.pred = predict(svmfit.best,newdata = election.te)
svm.te.err = calc_error_rate(svm.te.pred, y.te)


records2[2,1] = svm.tr.err
records2[2,2] = svm.te.err
records2
```

```
##     train.error test.error
## KNN     0.14266    0.18283
## SVM     0.04294    0.09141
```

Then we combine `records2` with `records`:

```r
rbind(records,records2)
```

```
##          train.error test.error
## tree         0.08864    0.13573
## logistic     0.06787    0.08033
## lasso        0.06856    0.08310
## KNN          0.14266    0.18283
## SVM          0.04294    0.09141
```

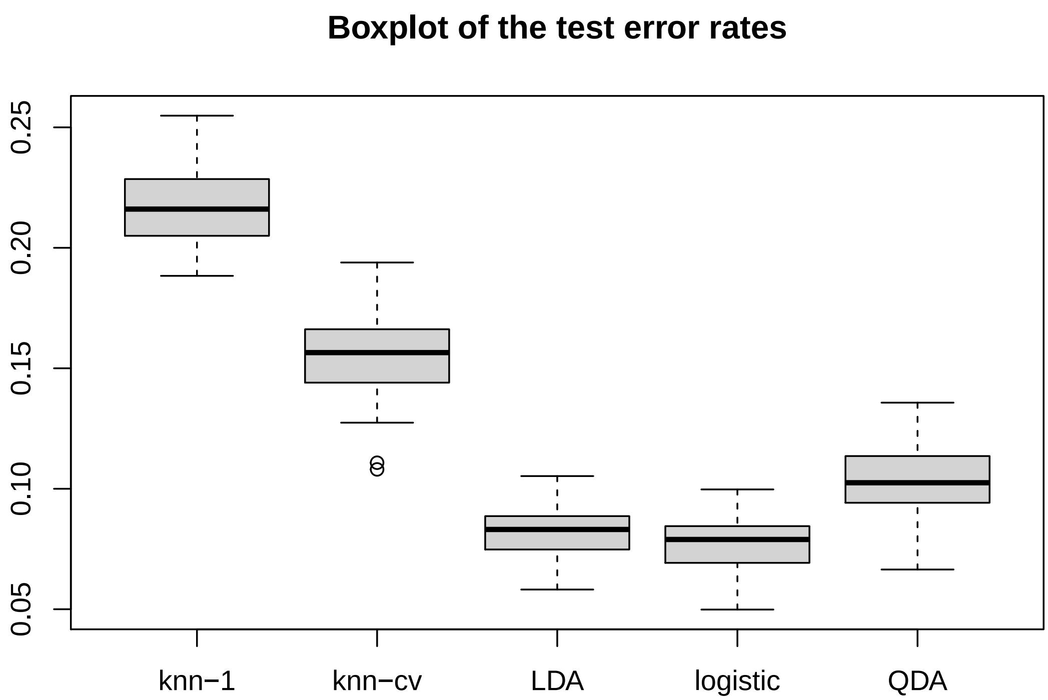
From the above results, we can see that two methods with the lowest misclassification error on the test set among the five methods are: logistic regression and logistic regression with LASSO penalty, which are all linear methods in classification. This indicates that the decision boundary for the candidates is probably linear. Thus, even though SVM method with radial kernal has the lowest training error, we expect SVM not to perform as well as the linear methods on the test data.
Also, it is understandable that the KNN method has the largest misclassification error since we have a high-dimensional dataset ($p = 21 > 4$). And it is reasonable that the decision tree analysis has large misclassification errors on test data since it generally does not have good predictive accuracy.

20. **Tackle at least one more interesting question. Creative and thoughtful analysis will be rewarded!**

    - Bootstrap: Perform bootstrap to generate plots similar to ISLR Figure 4.10/4.11. Discuss the results.

For this question, we consider four classification methods: K-nearest neighbors (KNN), Linear Discriminant Analysis (LDA), Logistic regression, and Quadratic Discriminant Analysis (QDA). we perform bootstrapping to generate 100 random training data sets. On each of these training sets, we fit each method to the data and computed the resulting test error rate. Then we visualize the error rates in boxcharts. Note that we performed KNN with two values of K: $K = 1$ and a value of $K = 26$ chosen by cross validation.

```r
test.error.df=data.frame(knn1.test.err,knn.cv.test.err,
                         lda.test.err,logistic.test.err,
                         qda.test.err)

boxplot(test.error.df,names=c("knn-1","knn-cv","LDA","logistic","QDA"))
title("Boxplot of the test error rates")
```



From the plot, we can see that three methods having the smallest test error rates are: LDA, logistic regression, and QDA. Logistic regression performed the best in this setting since the decision boundary is very likely to be linear by our previous anlysis. KNN performed poorly because it is a non-linear and flexible method which paid a price in terms of variance that was not offset by a reduction in bias. QDA also performed worse than LDA and logistic regression since it fits a more flexible classifier than necessary. The results of LDA is slightly inferior to those of logistic regression since the observations might not be drawn from a normal distribution.

21. **(Open ended) 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).

    First, the election data and census data have some discrepancies. The census data was collected in 2017 while the election data was collected in 2020. Thus, there are changes in counties that were not captured by the census data and may influence our cluster, map, and prediction results.

    Second, we build state-level and county-level maps to visualize the 2020 presidential election dataset. We found that most of the states chose Joe Biden and most of the counties in California chose Joe Biden. Then a visualization of the state winner result by sex showed that states with more male population are more likely to choose Donald Trump compared to states with more female population. We could further investigate the election result by sex if information about the gender of each voter are collected. And we should also add transgender into the gender identity to make our analysis more comprehensive.

    We create a new county-level census data by aggregating certain columns and remove certain variables to avoid perfect colinearity. The detection of perfect colinearity is important since it could affect the PCA process for choosing the most influential features. The PCA analysis shows that three of the most influential variables are child poverty percentage, employed percentage, and median household income, which are related to a measurement of a county's economy. When running cluster analysis of Santa

Barbara County, we find that hierarchical clustering on the first two principal components fails to place Santa Barbara County into a better cluster than hierarchical clustering on the original features does. This might attribute to the fact that the first two principal components are not enough to cover most of the variance. We may improve our cluster results by performing hierarchical cluster algorithm on more principal components.

To predict the election winner with classification methods, we combine the Untied States county-level census data with the election data. We applied several methods for prediction and found that logistic regression with LASSO penalty is the most suitable method by looking at the ROC curves and the table for training and test errors. Logistic regression with LASSO penalty has highest class separation capacity indicated by the largest AUC and produces relatively accurate predictions on the test data without overfitting problem. But we can also use decision tree analysis and logistic regression to answer different kinds of questions about election. Besides, compared with classification methods such as KNN, LDA, and QDA under the condition that the decision boundary is linear, logistic regression performs the best and LDA is slightly inferior.

A possible direction for future analysis is to collect previous presidential election data, conduct similar analysis on previous data, and then compare the results. Previous data can help us know historically what kind of features will affect the voting behavior of citizens.