PSTAT 131 Project

Bidal Orozco, Daniel Zeng, Adrian Gonzales

March 18, 2018

1.Voter prediction is difficult because the data necessary for is usually conducted through polls. As a result, polls can be a problematic source of data First, there can be a bias in the data collection process (sample being truly representative of voting population on election day). This bias can be a no response bias or it can be a result of people being polled responding to polls but not end up voting on election day. Second, people may provide false data to polls if they feel like their decision may be judged. Third, voter opinion can change from the time of the poll to election day. Lastly, decisions on variables used for analysis is crucial for forecasting and presents another difficulty for forecasters.

2. Silver is unique in his methodology in how his strong use in statistics such as time series, Bayesian probability, hierarchical clustering, and graph theory heavily within his analysis. As a result, Silver accounts for many of the problems mentioned in problem 1. He made sure to estimating the error of polls themselves to help mitigate the issue of bias in polls. He also used daily simulations to estimate the change in public voting on a day by day basis and using these simulations as predictions for possible conditions on election day. The hierarchical clustering method he used is useful because that is adjust predictions for states based on how states with similar demographics are responding to polls.

3. In 2016 polls had a high bias towards Hillary Clinton winning the elections. It is speculated that polls did not account for many of the Trump voters in their polls (either through people giving false information or being a "shy Trumper") as well as the overwhelming large number of people who would vote on election day for Trump.

```
election.raw = read.csv("~/final-project/data/election/election.csv") %>% as.tbl

census_meta = read.csv("~/final-project/data/census/metadata.csv", sep = ";") %>% as.tbl

census = read.csv("~/final-project/data/census/census.csv") %>% as.tbl

census$CensusTract = as.factor(census$CensusTract)

kable(election.raw %>% head)
```

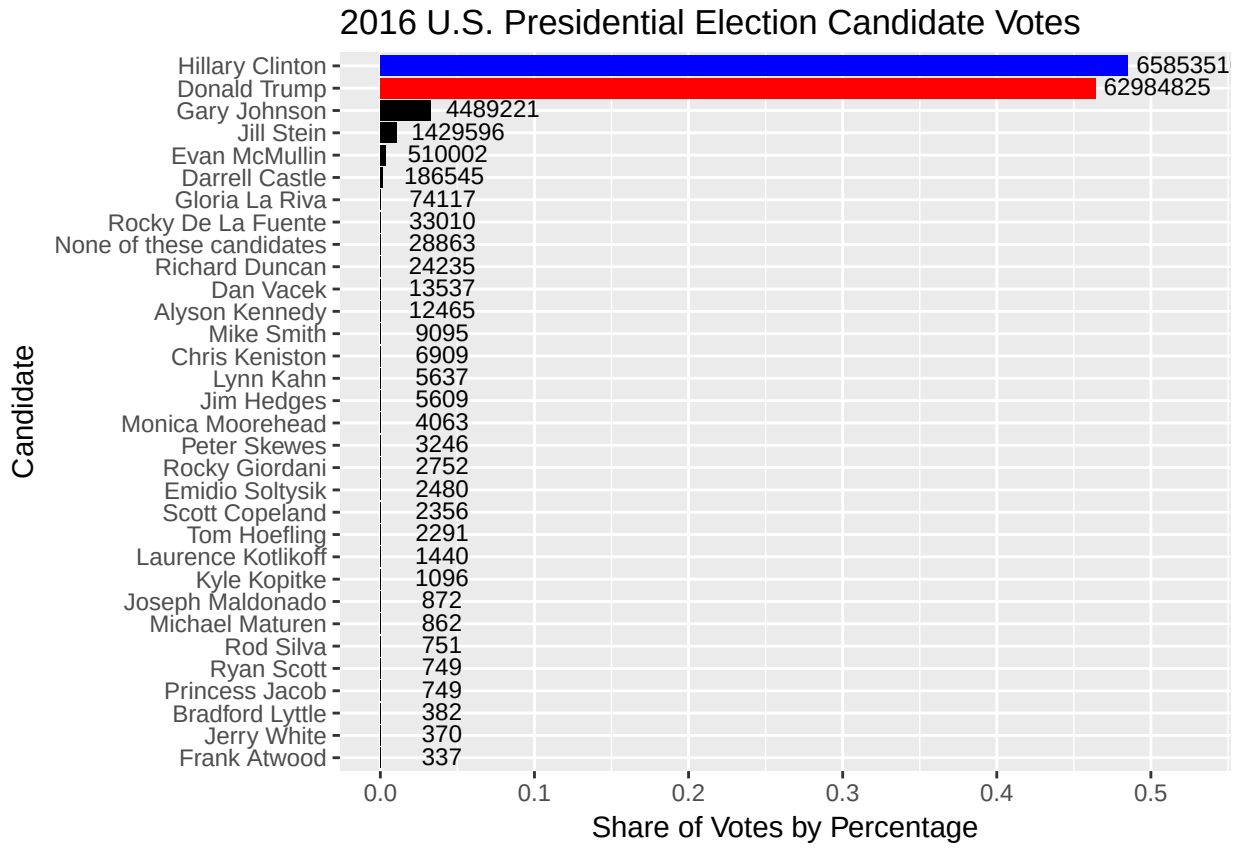
county	fips	candidate	state	votes
NA	US	Donald Trump	US	62984825
NA	US	Hillary Clinton	US	65853516
NA	US	Gary Johnson	US	4489221
NA	US	Jill Stein	US	1429596
NA	US	Evan McMullin	US	510002
NA	US	Darrell Castle	US	186545

4.

```
election_federal <- filter(election.raw, fips == "US")
election_state <- filter(election.raw, fips != "US" & is.na(county))
election <- filter(election.raw, !is.na(county))
#Adding missing county observations to election data frame
election <- rbind(election, election_state[309:312,])
#Taking out bad observations from state data frame
election_state <- filter(election_state, fips != 46102 & fips != 2000)
```

5.

```
# Creates a datafram of Candidates and how many votes they recieved
Candidate_Votes <- (election_federal %>% select(candidate, votes))

# Orders Data frame elements in descending order for
# re-leveling of factors
Candidate_Votes <- Candidate_Votes[order(Candidate_Votes$votes),]


# Re-orders the factor variable "candidate" for
# displaying the barplot in descending order
candidate.ordered <-  factor(Candidate_Votes$candidate, levels = as.vector(Candidate_Votes$candidate))

# Creates a Percentage variable to make barplot easier to read
Candidate_Votes <- Candidate_Votes %>% mutate(percentage = votes/sum(votes), candidate = candidate.ordei

ggplot(Candidate_Votes, aes(candidate, percentage)) +
  geom_col(fill = c(rep("black", times = nrow(Candidate_Votes) - 2), "red", "blue"))+coord_flip()+ labs
  geom_text(aes(label=votes), size = 3, nudge_y = 0.04, nudge_x = 0.08)+guides("Legend", nrow = 3, ncol
```



6.

```
# Grouping counties by fips
county.group <- group_by(election, fips)
# Taking the total votes per fips
total.group <- dplyr::summarize(county.group, total = sum(votes))
# Matching the total votes to their respective counties
```

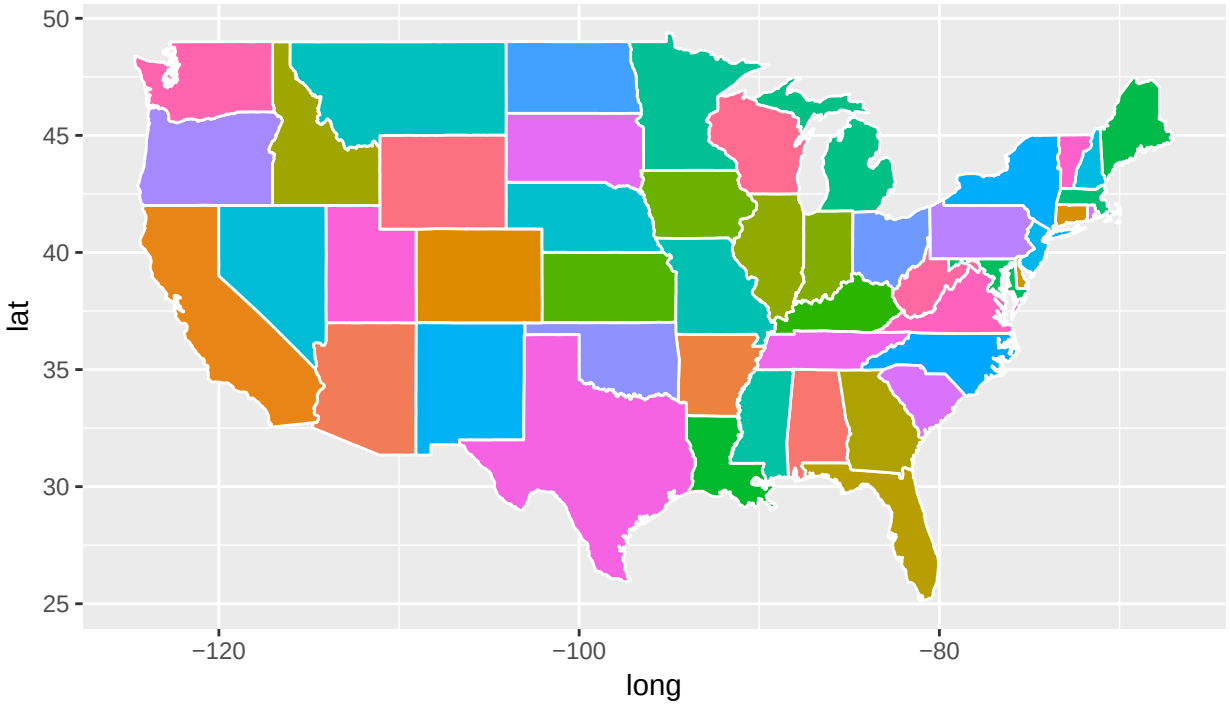
```
count.group <- left_join(county.group, total.group, by = "fips")
# Adding percent variable
county.pct <- mutate(count.group, pct = votes/total)
# Selecting winners for each county
county_winner <- top_n(county.pct, n =1)
```

```
## Selecting by pct
```

```
# Repeating above for states
state.group <- group_by(election_state, state)
total.stqte <- dplyr::summarize(state.group, total = sum(votes))
join.state <- left_join(state.group, total.stqte, by = "state")
state.pct <- mutate(join.state, pct = votes/total)
state_winner <- top_n(state.pct, n= 1)
```

```
## Selecting by pct
```

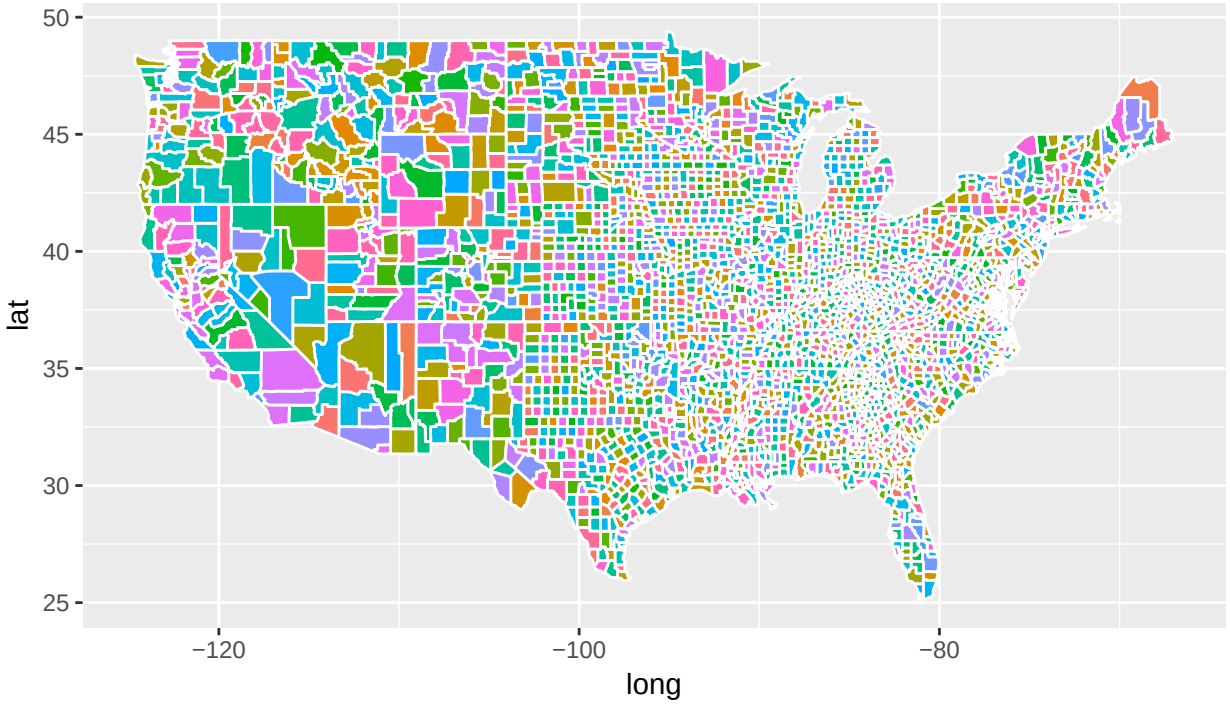
```
states = map_data("state")

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



7.

```
county = map_data("county")
```

```
ggplot(data = county) +
  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
```

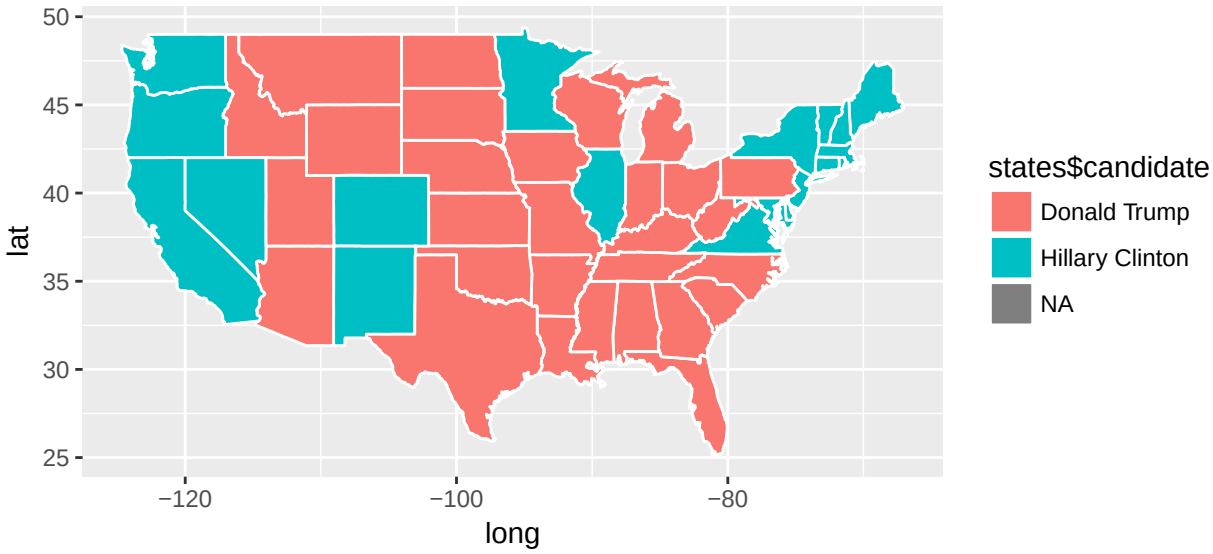


8.

```
# Adding fips variable to the states in ggmap
states=states%>%mutate(fips=state.abb[match(states$region,tolower(state.name) )])
states=left_join(states, state_winner, by="fips")
```

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

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



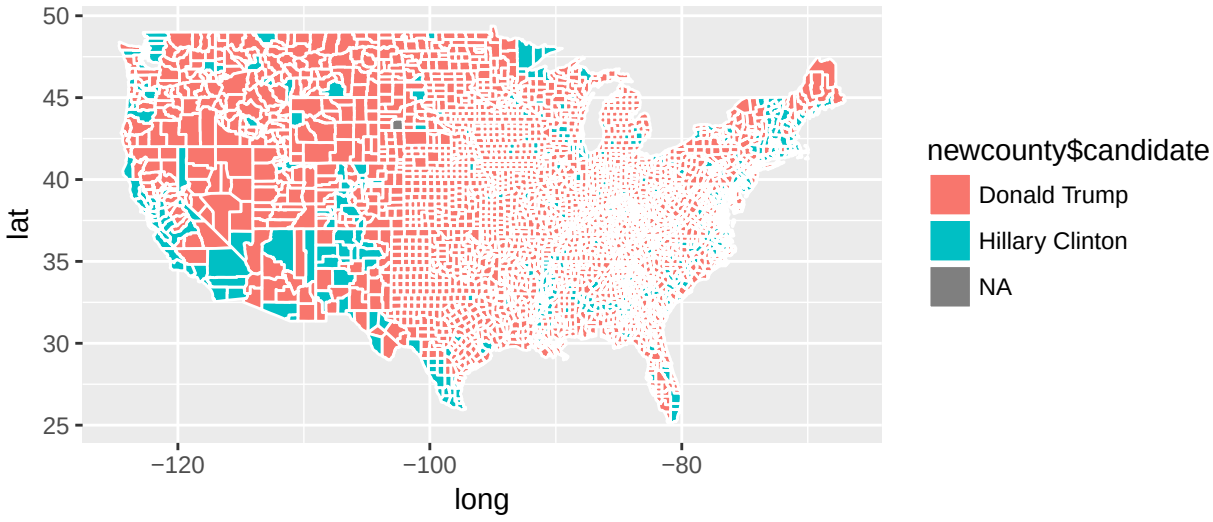
9.

```
# Matching winners onto counties in ggmap
countyseperate=separate(maps::county.fips,polyname,c("region", "subregion"),sep="," )
countyjoined=left_join(countyseperate,county,by=c("region", "subregion"))
countyjoined$fips=as.factor(countyjoined$fips)
newcounty=left_join(countyjoined,county_winner)
```

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

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

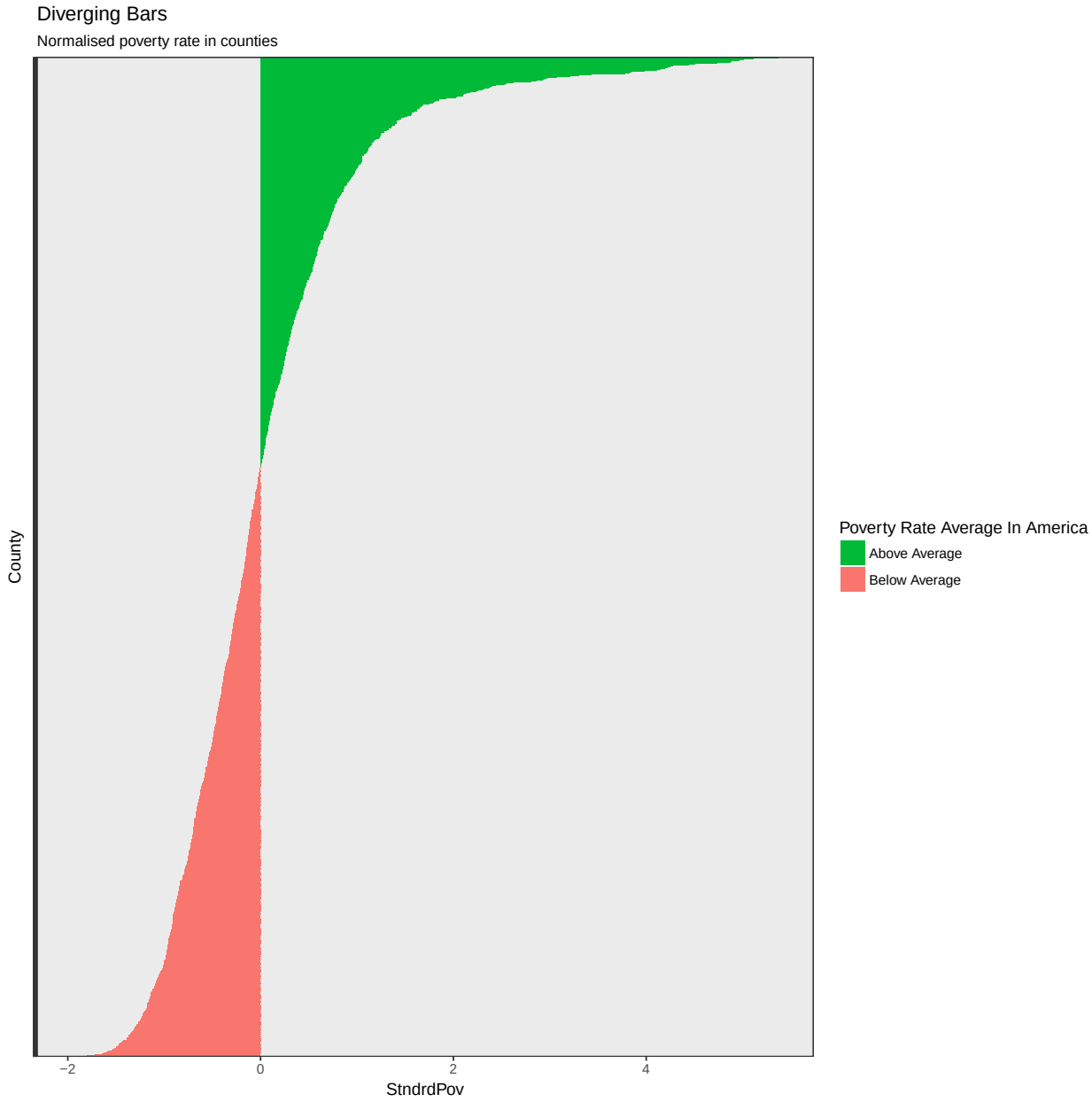
```
ggplot(data = newcounty) +
  geom_polygon(aes(x = long, y = lat, fill = newcounty$candidate, group = group),colour="white" ) +
  coord_fixed(1.3)
```



10.

```
library(ggplot2)
# Grouping by poverty
poverty.group <- group_by(census, County)
# Calculating the mean poverty per county
mean.poverty <- dplyr::summarise(poverty.group, AvgPoverty = mean(Poverty))
# Taking out the missing values
mean.poverty <- na.omit(mean.poverty)
# Standardizing mean poverty rates
mean.poverty$StndrdPov <- ((mean.poverty$AvgPoverty - mean(mean.poverty$AvgPoverty))/sd(mean.poverty$Avg
# Classifying variables into two factors
mean.poverty$PovertyLine <- ifelse(mean.poverty$StndrdPov < 0, 'below', 'above')
# Ordering data set
mean.poverty <- mean.poverty[order(mean.poverty$StndrdPov),]
# Converting into factors for graphing purposes
mean.poverty$County <- factor(mean.poverty$County, levels = mean.poverty$County)

theme_set(theme_bw())
ggplot(mean.poverty, aes(x=County, y=StndrdPov, label=StndrdPov)) +
  geom_bar(stat='identity', aes(fill=PovertyLine), width=1)  +
  scale_fill_manual(name="Poverty Rate Average In America",
                    labels = c("Above Average", "Below Average"),
                    values = c("above"="#00ba38", "below"="#f8766d")) +
  labs(subtitle="Normalised poverty rate in counties",
       title= "Diverging Bars") +
  coord_flip() +
```

```
  theme(axis.text.y = element_blank())
```



11.

```
census.del <- census

# removes rows with missing data
census.del <- census.del[complete.cases(census.del),]

# converts {'Men', 'Employed', 'Citizen'} to percentages
census.del <- census.del %>%
  mutate(Men = 100*Men/TotalPop,
         Employed = 100*Employed/TotalPop,
         Citizen = 100*Citizen/TotalPop)
```

```r
# Combines {Hispanic, Black, Native, Asian, Pacific} into ‘Minority‘ attribute and removes them.
census.del <- census.del %>% mutate(Minority = Hispanic + Black + Native + Asian + Pacific)%>% select(-

# Moves Minority attribute  to a more ergonomic index
census.del <- census.del[c(1:7, ncol(census.del), 8:(ncol(census.del)-1))]

#Removes {‘Walk‘, ‘PublicWork‘, ‘Construction‘} attributes
census.del <- select(census.del, -Walk, -PublicWork, -Construction)

# Removes "redundant" variables
census.del <- census.del %>% select(-Women,-White)
census.del
```

```r
# Creates census sub-county variable and groups tibble by
# State and County
census.subct <- group_by(census.del,State, County)

# Tallys how many subcounties are in each county of each state and
# names the column of tallies "CountyTotal"
census.subct <- add_tally(census.subct)
names(census.subct)[ncol(census.subct)] <- "CountyTotal"

# Finds weight for each subcounty defines as Population size with
# respect to how many subcounties are in the county.
census.subct <- mutate(census.subct, CountyWeight = TotalPop/CountyTotal)
census.subct
```

```r
census.ct <- census.subct

# Creates a total weight of the county weight for averaging
CountyWeightSum <- summarise_at(census.ct, .funs = funs(sum), .vars = vars("CountyWeight"))

# Renames County Weight Total for easier reading
names(CountyWeightSum)[ncol(CountyWeightSum)] <- "CountyWeightSum"

#Attaches CountyWeightSum variable to census.ct
census.ct <- left_join(census.ct,CountyWeightSum , by = c("State", "County"))

# Revalues CountyWeight to reflect its percentage of county total weight
census.ct <- mutate(census.ct, CountyWeight = CountyWeight/CountyWeightSum)

# Removes Unnecessary Variables
# information is already found in CountyWeight
census.ct <- select(census.ct, -CountyWeightSum, - CountyTotal)

# Applies Weightes to SubCounty Data
census.ct[5:28] <- census.ct[5:28]*census.ct$CountyWeight

# Aggregates population weighted subcounty data into County data
census.ct <- census.ct %>% summarise_at(vars(TotalPop:Unemployment), funs(sum))

census.ct <- ungroup(census.ct)

head(census.ct)
```

12.

```
#pca only works on numeric thus we remove state and county
# must ungroup data first then must scale

numericcensus.ct=select(ungroup(census.ct),-State,-County)
ct.pc=prcomp(scale(numericcensus.ct))

# most prominent loading for county
 ct.pc2=ct.pc$rotation[,c(1,2)]
 ct.pc2
```

```
##                            PC1          PC2
## TotalPop          0.082956993 -0.191690230
## Men               0.002298384  0.178560146
## Minority         -0.187712295 -0.074083164
## Citizen          -0.025016336  0.115116080
## Income            0.340954419 -0.162191758
## IncomeErr         0.197996494 -0.212660726
## IncomePerCap      0.367811444 -0.086233331
## IncomePerCapErr   0.216814369 -0.093489509
## Poverty          -0.336766254  0.023301745
## ChildPoverty     -0.341416433  0.008132606
## Professional      0.271781737  0.053960768
## Service          -0.175454443  0.039398090
## Office           -0.003999597 -0.286046143
## Production       -0.144894212 -0.092577866
## Drive            -0.111256279 -0.284989906
## Carpool          -0.078333721  0.060208311
## Transit           0.099178191 -0.116944274
## OtherTransp       0.003309337  0.092775111
## WorkAtHome        0.175133040  0.366615567
## MeanCommute      -0.053947922 -0.248831313
## Employed          0.332432299 -0.018674046
## PrivateWork       0.051782145 -0.403078434
## SelfEmployed      0.088996938  0.416458494
## FamilyWork        0.042840553  0.284131135
## Unemployment     -0.281027637 -0.094958643
```

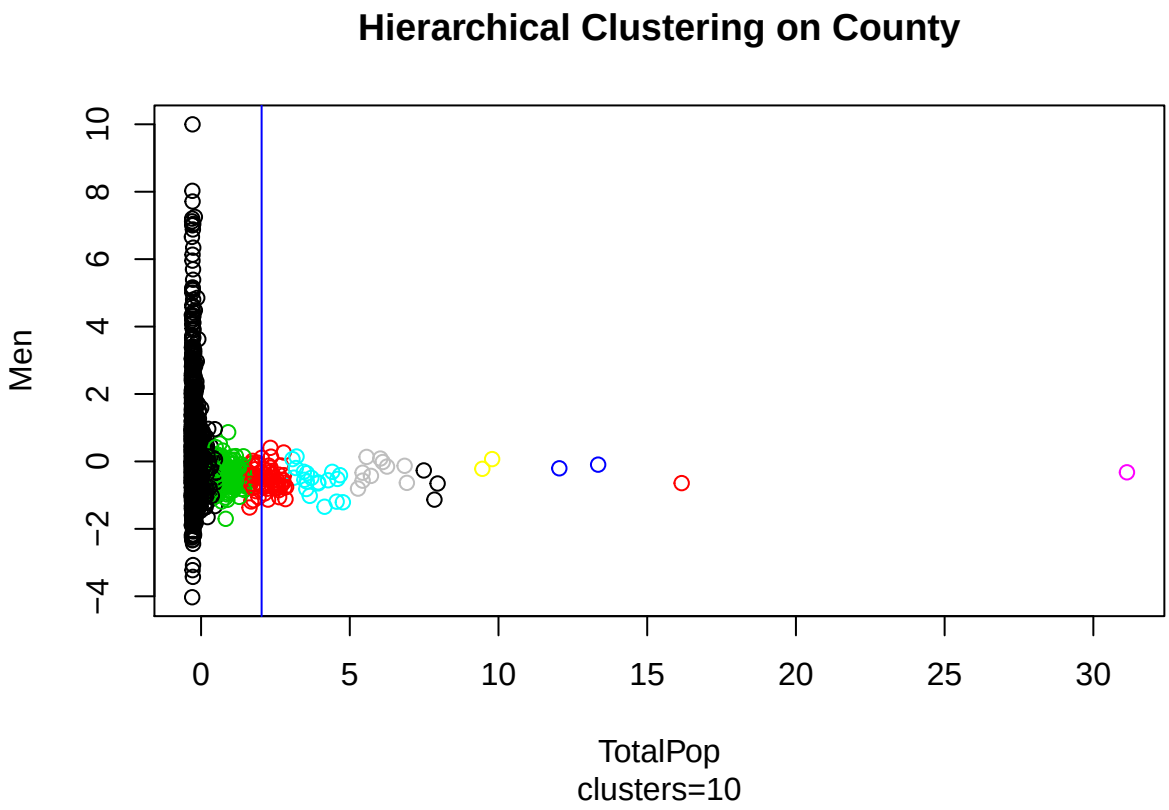
```
 # must make subcounty only have numeric
 # first must ungroup then scale
numericcensus.subct=select(ungroup(census.subct), -County , -State,-CensusTract)
subct.pc=prcomp(scale(numericcensus.subct))
# most prominent loadings
subct.pc2=ct.pc$rotation[,c(1,2)]
subct.pc2
```

```
##                            PC1          PC2
## TotalPop          0.082956993 -0.191690230
## Men               0.002298384  0.178560146
## Minority         -0.187712295 -0.074083164
## Citizen          -0.025016336  0.115116080
## Income            0.340954419 -0.162191758
## IncomeErr         0.197996494 -0.212660726
## IncomePerCap      0.367811444 -0.086233331
## IncomePerCapErr   0.216814369 -0.093489509
```

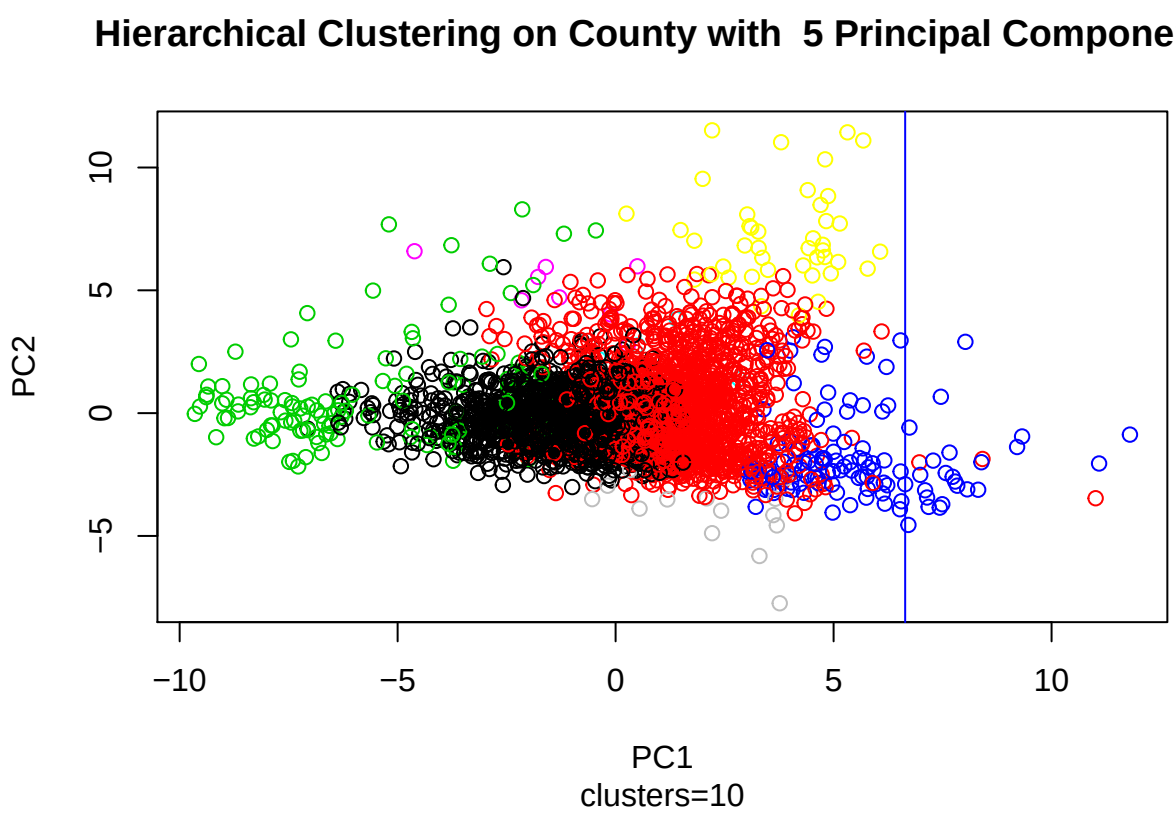
```
## Poverty         -0.336766254  0.023301745
## ChildPoverty    -0.341416433  0.008132606
## Professional     0.271781737  0.053960768
## Service         -0.175454443  0.039398090
## Office          -0.003999597 -0.286046143
## Production      -0.144894212 -0.092577866
## Drive           -0.111256279 -0.284989906
## Carpool         -0.078333721  0.060208311
## Transit          0.099178191 -0.116944274
## OtherTransp      0.003309337  0.092775111
## WorkAtHome       0.175133040  0.366615567
## MeanCommute     -0.053947922 -0.248831313
## Employed         0.332432299 -0.018674046
## PrivateWork      0.051782145 -0.403078434
## SelfEmployed     0.088996938  0.416458494
## FamilyWork       0.042840553  0.284131135
## Unemployment    -0.281027637 -0.094958643
```

13.

```
distanceCensus=dist(numericcensus.ct)
census.hcComp=hclust(distanceCensus, "complete")
census.hc10=cutree(census.hcComp,k=10)

census.pc5= ct.pc$x[,1:5]
distcensus.pc5=dist(census.pc5)
census.pcahcComp=hclust(distcensus.pc5, "complete")
census.pcahc10=cutree(census.pcahcComp,k=10)

SanMateo.Pos <- which(census.ct$County == "San Mateo")

plot( scale(numericcensus.ct), col=census.hc10,
      main="Hierarchical Clustering on County",
      sub="clusters=10")
scalednumct <- scale(numericcensus.ct)
scalednumct <- as.data.frame(scalednumct)
abline(v = scalednumct$TotalPop[SanMateo.Pos], col = "blue")
```




```
plot(ct.pc$x[,1:5],col=census.pcahc10,
     main="Hierarchical Clustering on County with  5 Principal Components",
     sub="clusters=10" )
abline(v = ct.pc$x[SanMateo.Pos,1], col = "blue")
```



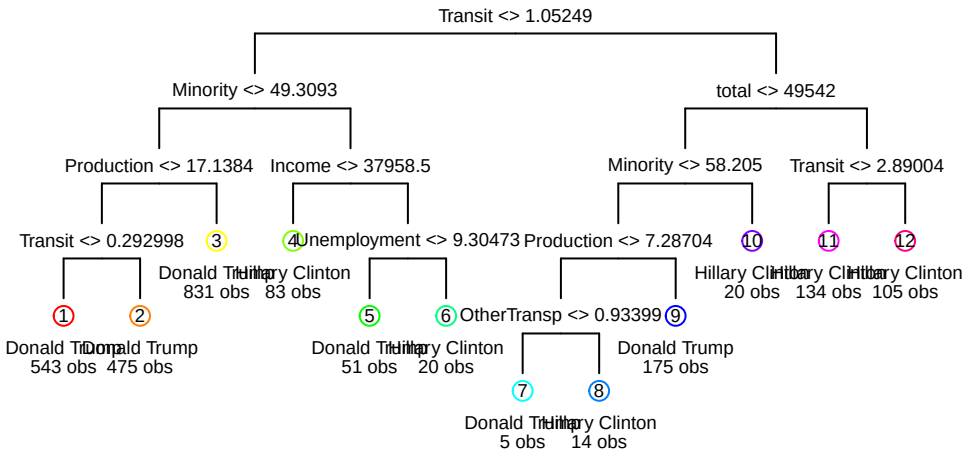
The blue line here indicates the location of San Mateo county in our scatter plot. We can see that the cluster it has been assigned to differs between the core component data and the PCA form of our data. At the same time however, in the PCA cluster graph we see that there is overlap with the cluster that San Mateo belongs to in the core component cluster. This difference in assignment can be attributed to the dimensionalty reduction caused by PCA. It is likely that the altering of distances slightly affected the San Mateo observation to be classified into a different cluster. However, the potential overlap tells us that the changes present from the transition were not extreme either.

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

tmpcensus = census.ct %>% mutate_at(vars(State, County), tolower)


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

# Creates a variable to be used in later mappings
mapping.cl <- select(election.cl, -c(pct, fips))

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

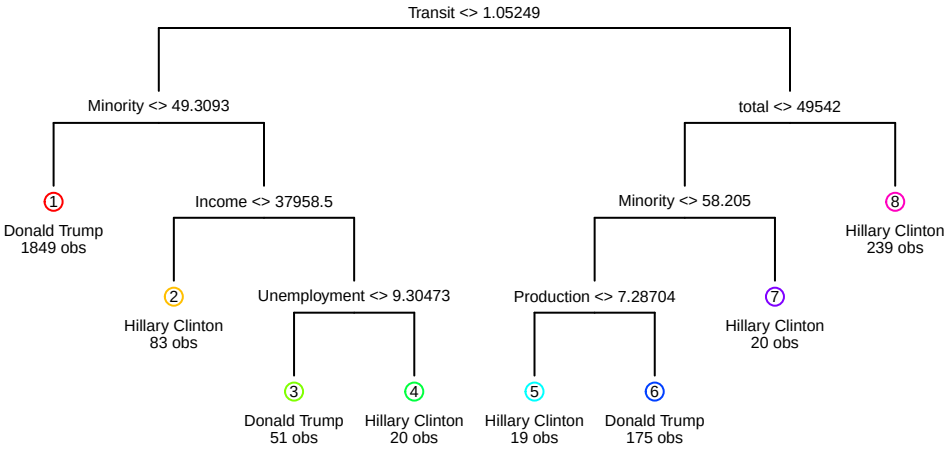
```
set.seed(10)
n = nrow(election.cl)
in.trn= sample.int(n, 0.8*n)
trn.cl = election.cl[ in.trn,]
tst.cl = election.cl[-in.trn,]

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

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

14.

```
candidate.tree <- tree(candidate ~ ., data = trn.cl)
cv <- cv.tree(candidate.tree, rand = folds, FUN = prune.misclass, K = nfold)
min.dev <- min(cv$dev)
best.size.cv <- cv$size[which(cv$dev == min.dev)]
draw.tree(candidate.tree, cex = 0.55)
```




```
tree.pruned <- prune.misclass(candidate.tree, best = best.size.cv)
draw.tree(tree.pruned, cex = 0.5)
```




```
tree.train <- predict(tree.pruned, trn.cl, type = "class")
tree.test <- predict(tree.pruned, tst.cl, type = "class")
records[1,1] <- calc_error_rate(tree.train, trn.cl$candidate)
records[1,2] <- calc_error_rate(tree.test, tst.cl$candidate)
records
```

```
##      train.error test.error
## tree  0.06636808 0.07980456
## knn           NA         NA
## lda           NA         NA
```

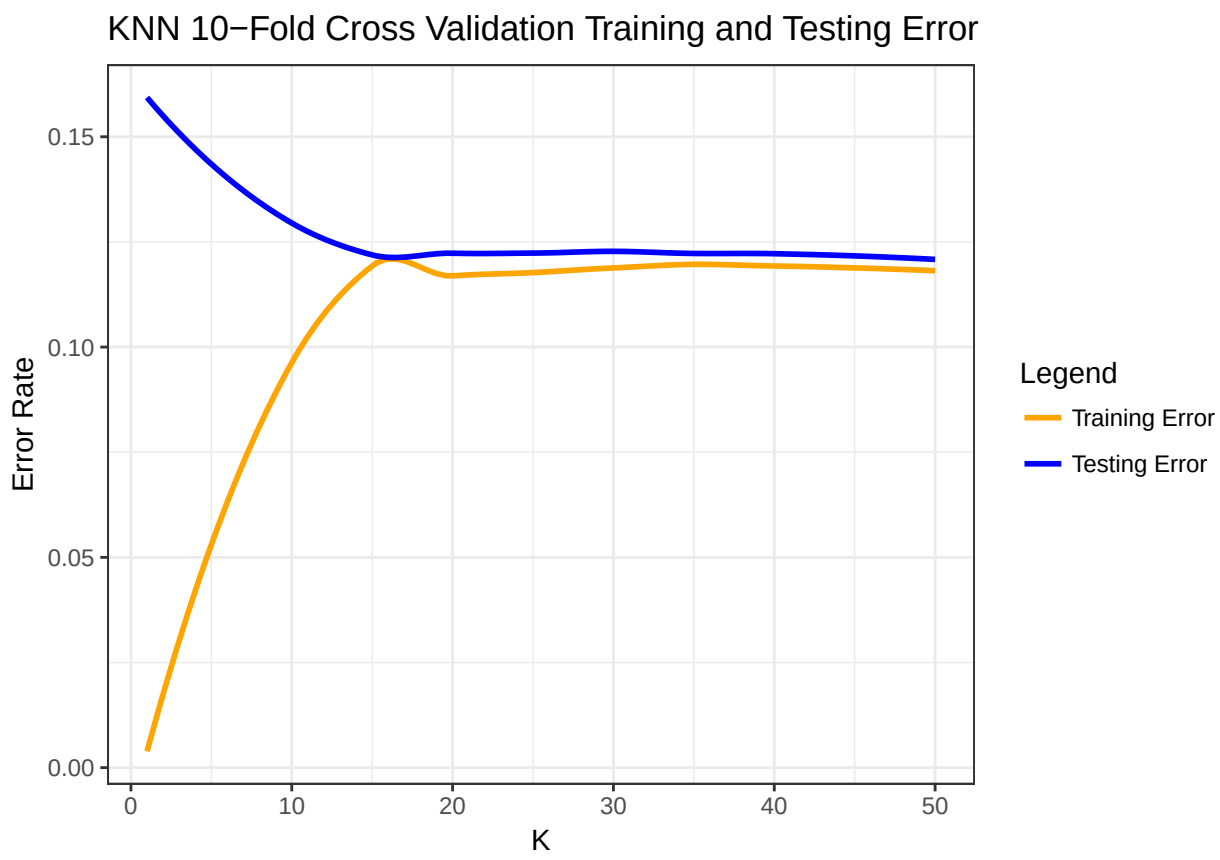
15.

```
# K values for testing
k.test = c(1, seq(10,50, length.out = 9))

# Function for CV
do.chunk <- function(chunkid, folddef, Xdat, Ydat, k){
train = (folddef!=chunkid)
Xtr = Xdat[train,]
Ytr = Ydat[train]
Xvl = Xdat[!train,]
Yvl = Ydat[!train]
## get classifications for current training chunks
predYtr = knn(train = Xtr, test = Xtr, cl = Ytr, k = k)
## get classifications for current test chunk
predYvl = knn(train = Xtr, test = Xvl, cl = Ytr, k = k)
# Returns a data fram of Training Error and Validation Error
```

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

K_Errors <- tibble("K" = k.test, "AveTrnError" = NA, "AveTstError" = NA)

predictors <- select(trn.cl, -candidate)

for(i in 1:10){

temp <- plyr::ldply(1:10, do.chunk, folds,predictors, trn.cl$candidate, K_Errors$K[i])

K_Errors$AveTrnError[i] <- mean(temp[,1])
K_Errors$AveTstError[i] <- mean(temp[,2])
}
```

```
# Melts columns for plotting
K_Errors_yax <- melt(K_Errors, id = "K")
# Renames observations for plot readability
names(K_Errors_yax)[2] <- "Legend"
levels(K_Errors_yax$Legend)<- c("Training Error", "Testing Error")


ggplot(K_Errors_yax, aes(x = K))+ ggtitle("KNN 10-Fold Cross Validation Training and Testing Error")+ yl
```

```
## `geom_smooth()` using method = 'loess'
```



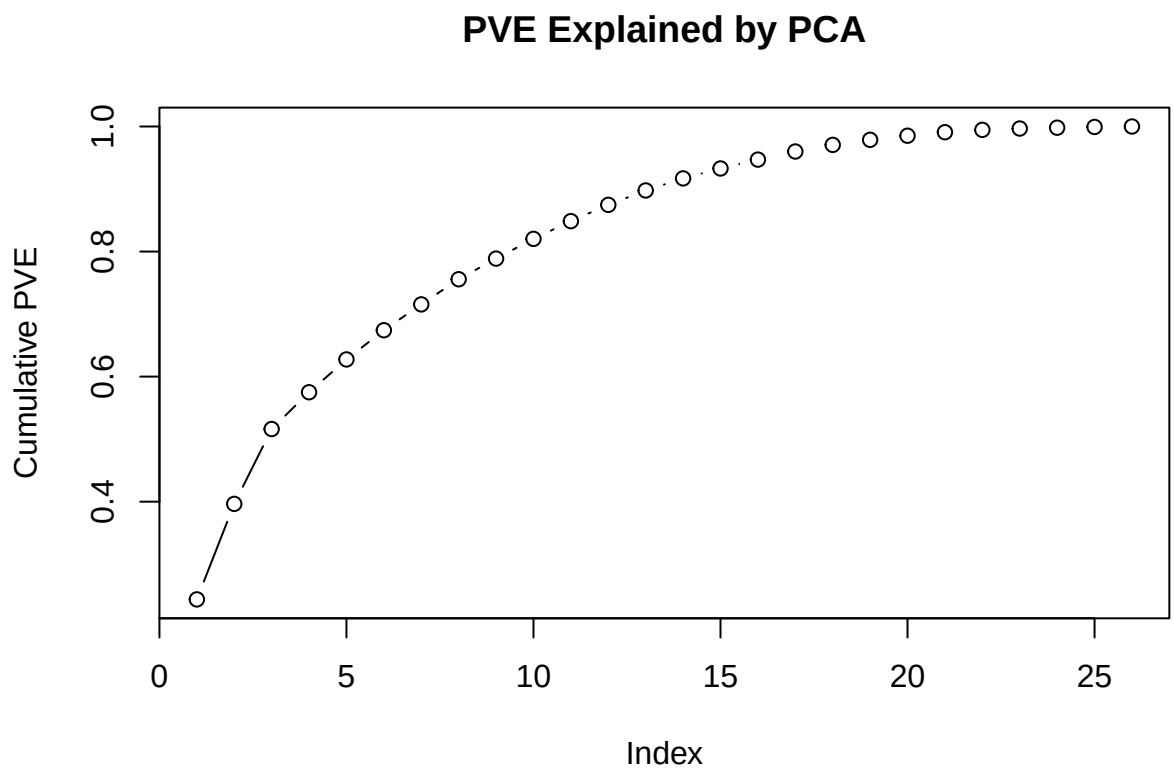
```r
# Plot shows that K = 15 is best balance of training and test error.
# Makes training set and test set predictions using k = 15
prediction.trn <- knn(train = trn.cl[2:27], test = trn.cl[2:27], cl = trn.cl$candidate, k = 15)
prediction.tst <- knn(train = trn.cl[2:27], test = tst.cl[2:27], cl = trn.cl$candidate, k = 15)

# Saves KNN results into records matrix
records[2,1] <- calc_error_rate(prediction.trn, trn.cl$candidate)
records[2,2] <- calc_error_rate(prediction.tst, tst.cl$candidate)
records
```

```
##      train.error test.error
## tree  0.06636808 0.07980456
## knn   0.11685668 0.12866450
## lda           NA         NA
```

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

16.

```r
pca.train.numeric <- select(trn.cl, -candidate)
pr.pca.train <- prcomp(scale(pca.train.numeric))
train.var <- pr.pca.train$sdev^2
train.pve <- train.var/sum(train.var)
plot(cumsum(train.pve), type = 'b', main = "PVE Explained by PCA", ylab = "Cumulative PVE")
```



```r
which.min(abs(cumsum(train.pve)-0.905))
```

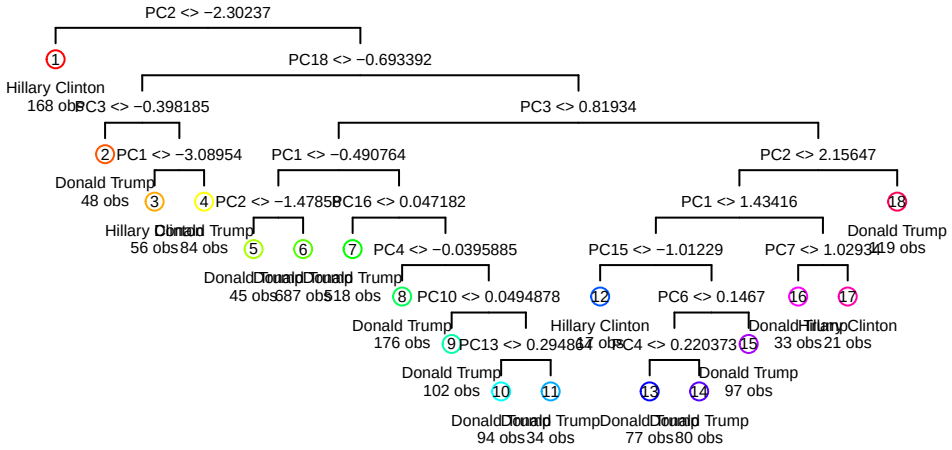
```
## [1] 13
```

17.

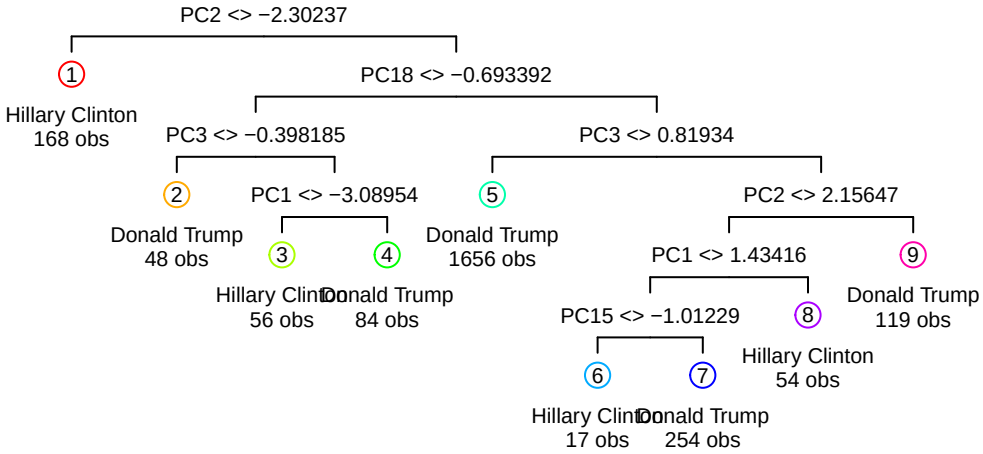
```r
# Creating PCA data frame for training data
tr.pca <- bind_cols(y = trn.cl$candidate, z = as.data.frame(pr.pca.train$x))
pca.test.numeric <- select(tst.cl, -candidate)
pr.pca.test <- prcomp(scale(pca.test.numeric))
# Creating PCA data frame for test data
test.pca <- bind_cols(y = tst.cl$candidate, z = as.data.frame(pr.pca.test$x))
```

18.

```r
pca.tree <- tree(y ~ ., data = tr.pca)
cv.pca <- cv.tree(pca.tree, rand = folds, FUN = prune.misclass, K = nfold)
pca.min.dev <- min(cv.pca$dev)
bestpca.size.cv <- cv.pca$size[which(cv.pca$dev == pca.min.dev)]
draw.tree(pca.tree, cex = 0.5)
```




```r
pca.tree.pruned <- prune.misclass(pca.tree, best = bestpca.size.cv)
draw.tree(pca.tree.pruned, cex = 0.7)
```




```
pca.tree.train <- predict(pca.tree.pruned, tr.pca, type = "class")
pca.tree.test <- predict(pca.tree.pruned, test.pca, type = "class")
pca.records[1,1] <- calc_error_rate(pca.tree.train, tr.pca$y)
pca.records[1,2] <- calc_error_rate(pca.tree.test, test.pca$y)
pca.records
```

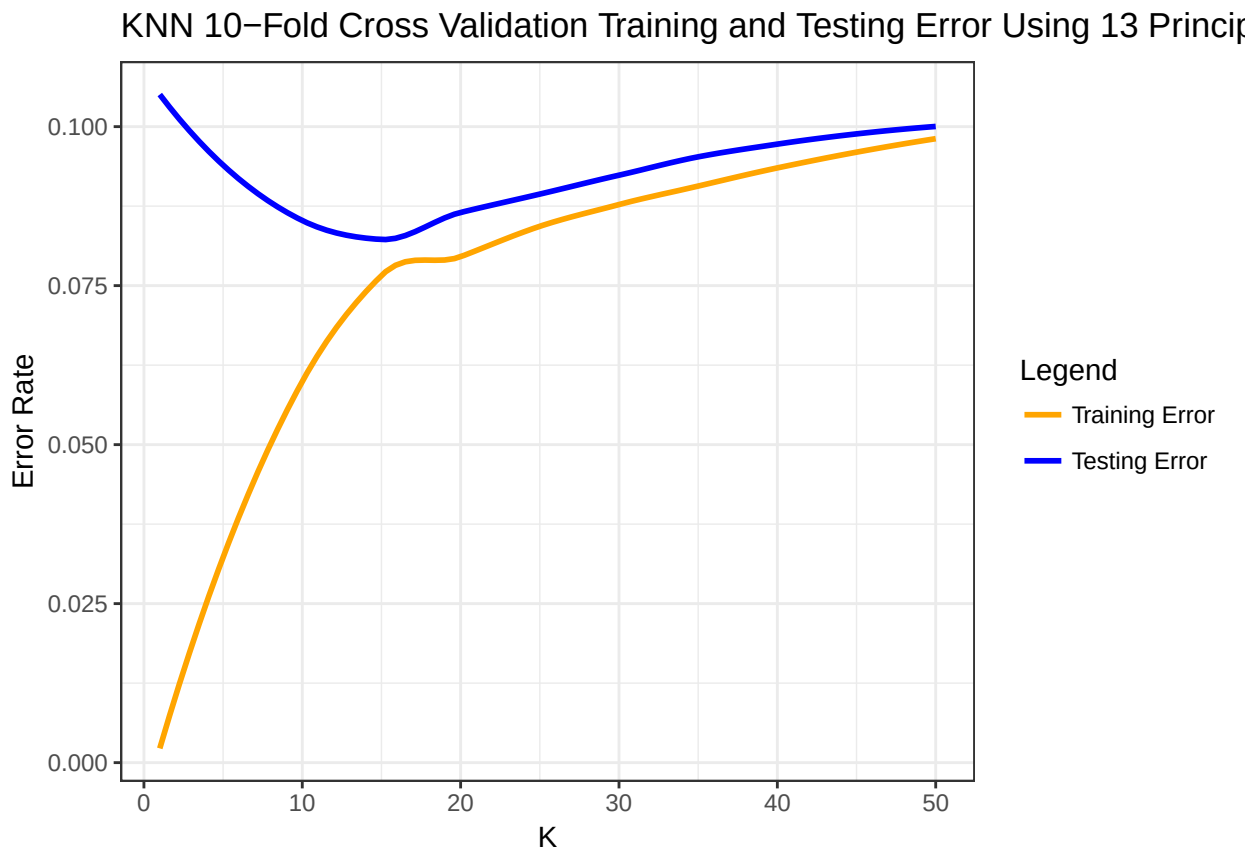
```
##      train.error test.error
## tree  0.08428339  0.3094463
## knn           NA         NA
## lda           NA         NA
```

19.

```
pca.K_Errors <- tibble("K" = k.test, "AveTrnError" = NA, "AveTstError" = NA)
pca.predictors <- select(tr.pca, -y)

for(i in 1:10){

temp <- plyr::ldply(1:10, do.chunk, folds,pca.predictors, tr.pca$y, pca.K_Errors$K[i])

pca.K_Errors$AveTrnError[i] <- mean(temp[,1])
pca.K_Errors$AveTstError[i] <- mean(temp[,2])
}

# Melts columns for plotting
pca.K_Errors_yax <- melt(pca.K_Errors, id = "K")
# Renames observations for plot readability
```

```
names(pca.K_Errors_yax)[2] <- "Legend"
levels(pca.K_Errors_yax$Legend)<- c("Training Error", "Testing Error")


ggplot(pca.K_Errors_yax, aes(x = K))+ ggtitle("KNN 10-Fold Cross Validation Training and Testing Error U
```

```
## `geom_smooth()` using method = 'loess'
```



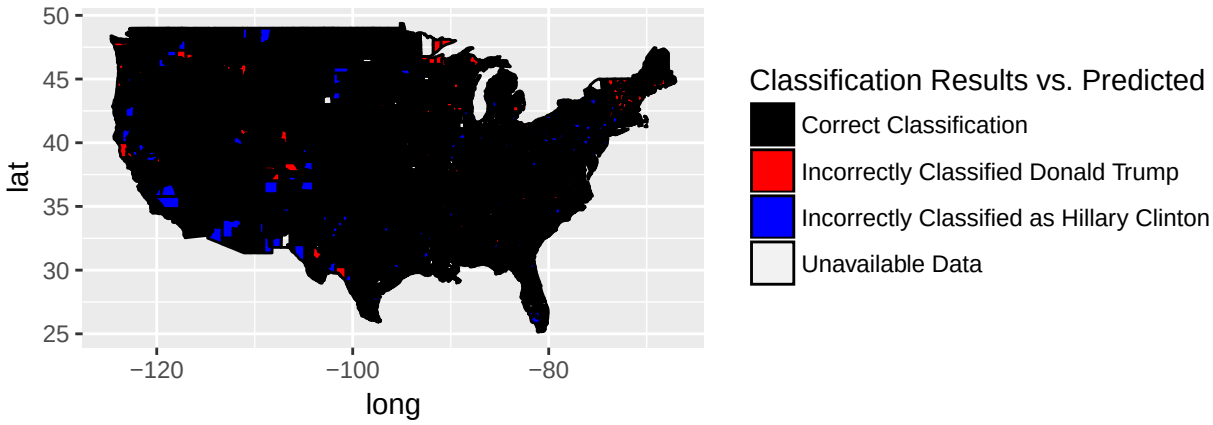

```
# Again, the plot shows that K = 15 is best balance of training and test error.

# Makes training set and test set predictions using k = 15
prediction.tr.pca <- knn(train = tr.pca[-1], test = tr.pca[-1], cl = tr.pca$y, k = 15)
prediction.ts.pca <- knn(train = tr.pca[-1], test = test.pca[-1], cl = tr.pca$y, k = 15)

# Saves KNN results into records matrix
records[2,1] <- calc_error_rate(prediction.trn, trn.cl$candidate)
records[2,2] <- calc_error_rate(prediction.tst, tst.cl$candidate)
records
```

```
##      train.error test.error
## tree  0.06636808 0.07980456
## knn   0.11685668 0.12866450
## lda           NA         NA
```

20.

```
# saves mapping.cl for modification
mapping_election <- mapping.cl
```

```
names(mapping_election)[c(1,3)] <- c("subregion", "region")

# Predicts county
predicted_candidates.tree <- predict(tree.pruned, mapping_election[4:30], type = "class")

prediction_map <- map_data("county")



prediction_mapper.tree <- bind_cols(predicted = predicted_candidates.tree,z = mapping_election)

misclass.tree <- mutate(prediction_mapper.tree, misclass = as.factor(ifelse((predicted == candidate),"bl

prediction_mappest.tree <-left_join(prediction_map,misclass.tree, by = c("region", "subregion"))


ggplot(data = prediction_mappest.tree) +
  geom_polygon(aes(x = long, y = lat, fill = misclass, group = group), color = "black") +
  coord_fixed(1.3) +
  scale_fill_manual(name = "Classification Results vs. Predicted", labels = c("Correct Classification",
```



Something that we have gathered from this data was that it is very hard to make these predictions based on just the demographics we had. If it were feasible, we would want to consider trying to take in more data about peoples political preferences such as their stance on certain issues and extrapolating those preferences out to the county level. We would also want to have a county's voting history in the past and see if those can give us better groupings within our demographics. With our given classification methods we were slightly

limited, they are fine as they are but we would like to try and get our error percentages under 10%. Another thing to try would be to do the predictions again but on the subcounty level. However, we did not have the processing power for this method.

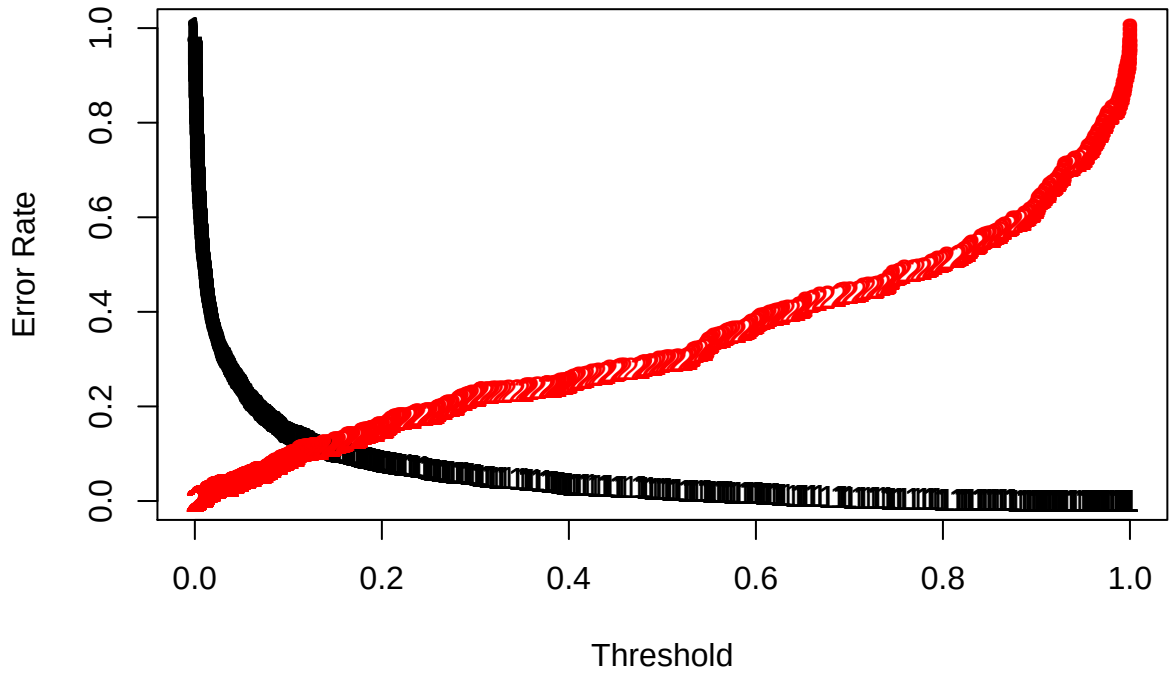
In the graph above, we have our missclassifications highlighted on a US map. We also specified what incorrect classifications were made. We can see that we definitely over predicted Hillary winning certain counties. We underestimated Trump's ability to win districts, and this was probably what happened during the election prediction before the results were made official as well.

21.

```
# Logistic Regression
logistic.model.core <- glm(candidate ~ ., data = trn.cl, family = "binomial")
```

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

```
logistic.train.predict <- predict(logistic.model.core, trn.cl, type = "response")
trn.cl <- trn.cl %>% mutate(candidate = as.factor(ifelse(candidate == "Donald Trump", "Donald Trump", "
logistic.train.prediction <- prediction(logistic.train.predict, trn.cl$candidate)
# FPR
fpr.train = performance(logistic.train.prediction, "fpr")@y.values[[1]]
cutoff.train <- performance(logistic.train.prediction, "fpr")@x.values[[1]]
# FNR
fnr.train <- performance(logistic.train.prediction, "fnr")@y.values[[1]]
library(graphics)
matplot(cutoff.train, cbind(fpr.train, fnr.train), lwd = 2, xlab = "Threshold", ylab = "Error Rate")
```




```
train.rate <- as.data.frame(cbind(Cutoff = cutoff.train, FPR = fpr.train, FNR = fnr.train))
train.rate$distance <- sqrt((train.rate[,2]^2) +(train.rate[,3])^2)
```

```
index = which.min(train.rate$distance)
best = train.rate$Cutoff[index]

# Classifying our logistic model predictions based off of our ideal threshold
trn.cl.pred <- trn.cl %>% mutate(predCandidate = as.factor(ifelse(logistic.train.predict <= best, "Dona
logistic.train.error <- calc_error_rate(trn.cl.pred$candidate, trn.cl.pred$predCandidate)
logistic.train.error
```

```
## [1] 0.1091205
```

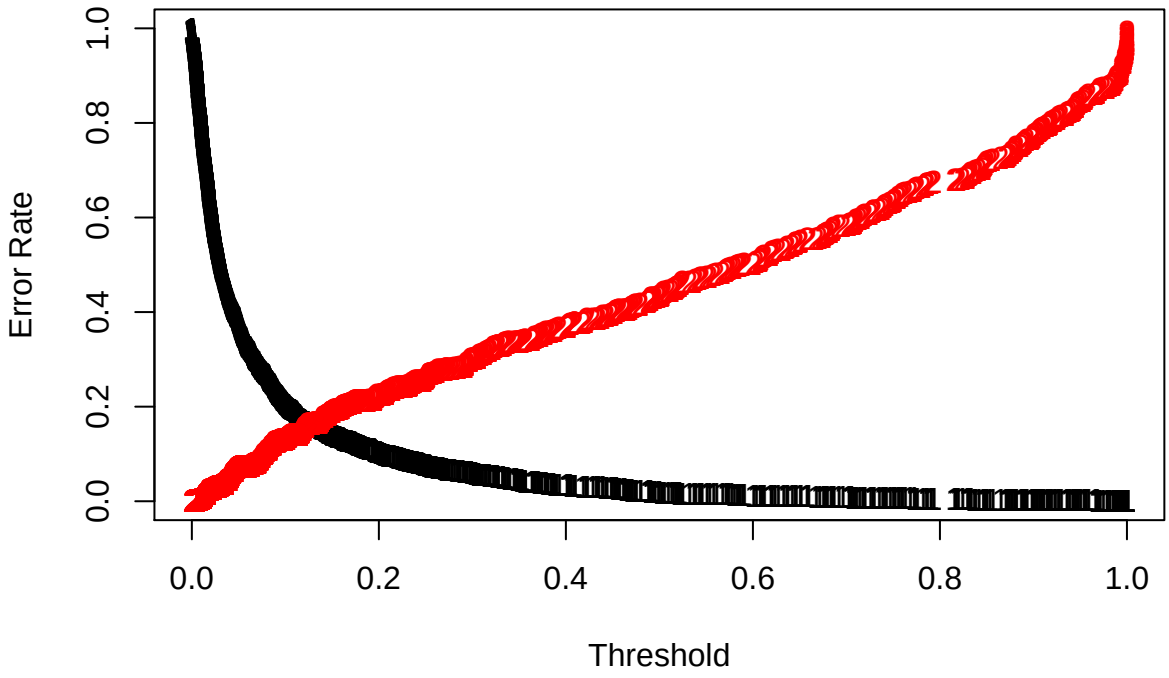
```
logistic.test.predict <- predict(logistic.model.core, tst.cl, type = "response")
tst.cl <- tst.cl %>% mutate(candidate = as.factor(ifelse(candidate == "Donald Trump", "Donald Trump", "
tst.cl.pred <- tst.cl %>% mutate(predCandidate = as.factor(ifelse(logistic.test.predict <= best, "Donal
logistic.test.error <- calc_error_rate(tst.cl.pred$candidate, tst.cl.pred$predCandidate)
logistic.test.error
```

```
## [1] 0.1237785
```

```
#Logistic Regression on PCA 13
#Used PCA 13 terms b/c min num of PCA that explains 90% of variance

tr.pca13=tr.pca[,1:14]
logistic.pcamodel.core <- glm(y ~ ., data = tr.pca13, family = "binomial")
```

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

```
# soft classifiers from equation on training
logistic.pcatrain.predict <- predict(logistic.pcamodel.core, tr.pca13, type = "response")

#
tr.pca13<- tr.pca13 %>% mutate(y = as.factor(ifelse(y == "Donald Trump", "Donald Trump", "Hillary Clint
logistic.pcatrain.prediction <- prediction(logistic.pcatrain.predict, tr.pca13$y)


# FPR
fpr.pcatrain = performance(logistic.pcatrain.prediction, "fpr")@y.values[[1]]
cutoff.pcatrain <- performance(logistic.pcatrain.prediction, "fpr")@x.values[[1]]


# FNR
fnr.pcatrain <- performance(logistic.pcatrain.prediction, "fnr")@y.values[[1]]

matplot(cutoff.pcatrain, cbind(fpr.pcatrain, fnr.pcatrain), lwd = 2, xlab = "Threshold", ylab = "Error
```




```
train.pcarate <- as.data.frame(cbind(Cutoff = cutoff.pcatrain, FPR = fpr.pcatrain, FNR = fnr.pcatrain))

train.pcarate$distance <- sqrt((train.pcarate[,2]^2) +(train.pcarate[,3])^2)
index.pca = which.min(train.pcarate$distance)
best.pca = train.pcarate$Cutoff[index.pca]

# Classifying our logistic model predictions based off of our ideal threshold
trn.pcacl.pred <- tr.pca13 %>% mutate(predCandidate = as.factor(ifelse(logistic.pcatrain.predict <= bes
logistic.pcatrain.error <- calc_error_rate(trn.pcacl.pred$y, trn.pcacl.pred$predCandidate)
logistic.pcatrain.error
```

```
## [1] 0.1706026
```

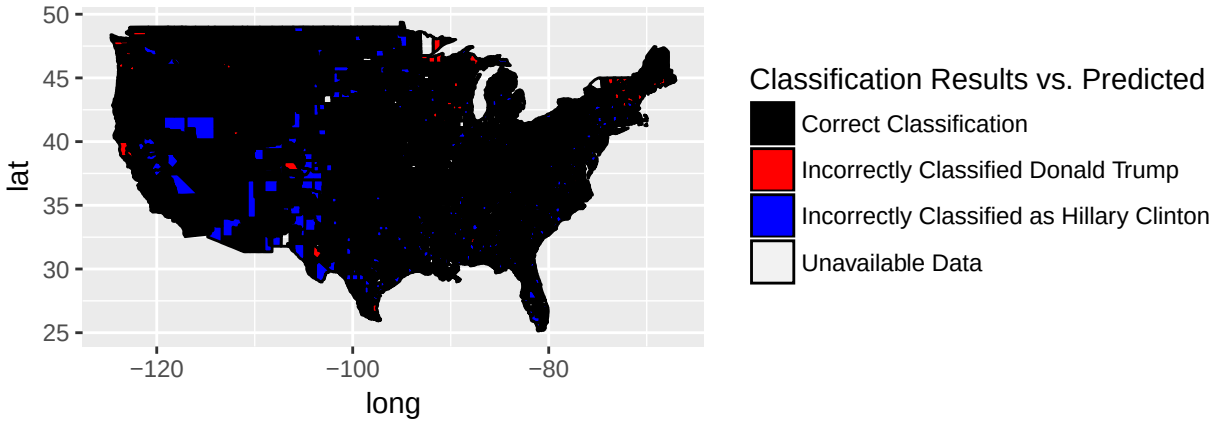
```
#building test set using pca 13

tst.pca13=test.pca[,1:14]


logistic.pcatest.predict <- predict(logistic.pcamodel.core, tst.pca13, type = "response")
tst.pca13 <- tst.pca13 %>% mutate(y = as.factor(ifelse(y == "Donald Trump", "Donald Trump", "Hillary Cl
tst.pcacl.pred <- tst.pca13%>% mutate(predCandidate = as.factor(ifelse(logistic.pcatest.predict <= best
logistic.pcatest.error <- calc_error_rate(tst.pcacl.pred$y, tst.pcacl.pred$predCandidate)
logistic.pcatest.error
```

```
## [1] 0.5374593
```

```
logistic.pcatrain.error
```

```
## [1] 0.1706026
```

```
# Saves predictions from Logistic modle for every county
predicted_candidates.logistic <- predict(logistic.model.core, mapping_election[4:30], type = "response"

# Changes probability vector to classification vector based off of threshold
# determined above
predicted_candidates.logistic <- as.factor(ifelse(predicted_candidates.logistic <= best, "Donald Trump"

mapping_election.log <- mapping_election
# Changes factor levels for future code to work
mapping_election.log$candidate <- as.factor(ifelse(mapping_election.log$candidate == "Donald Trump", "D

# Attaches predicted candidates onto our map data
prediction_mapper.logistic <- bind_cols(predicted = predicted_candidates.logistic,z = mapping_election.

# Creates a vector of missclassified counties
misclass.logistic <- mutate(prediction_mapper.logistic, misclass = as.factor(ifelse((predicted == candi


prediction_mappest.logistic <-left_join(prediction_map,misclass.logistic, by = c("region", "subregion"))

ggplot(data = prediction_mappest.logistic) +
  geom_polygon(aes(x = long, y = lat, fill = misclass, group = group), color = "black") +
  coord_fixed(1.3) +
  scale_fill_manual(name = "Classification Results vs. Predicted", labels = c("Correct Classification",
```



From our logistic models, it is clear that we should not be using principle components to perform non-linear

regression. In comparison to our other models, it is not as good as our decision tree.