Introduction

The goal of this project is to design a machine learning model that determines consumer debt relief from applicant complaints filed with the Consumer Financial Protection Bureau (CFPB) for the continental US. The data used for this analysis is from CFPB, the Urban Institute and the United States Census Bureau. The CFPB complaint dataset was combined with county debt dataset from the Urban Institute, as well as county level demographic data from the 2020 US Census. These datasets were merged to account for county level differences in demographics, incomes, and debt owed as important features for predicting relief status for individuals filing debt relief complaints in these counties. The hypothesis being that geospatial social-economic factors play a role in determining an individual's debt relief.

These datasets were cleaned before combining and implementing models. The cleaning process involved consolidation of factor variables related to the type of debt complaints, the responses and nature of communicating debt notifications, company responses to applicant complaints, and consumer consent. Other numerical information relating to county level debt, income, and demographics were adjusted to maintain comparability, and missing data was imputed with various techniques. Demographic county totals from the US Census were tabulated manually and scaled as county proportions, to compare across the continental US counties. County level debt involved tree imputation of missing data along with a Principal Component Analysis to reduce the various debt types into their correlations.

Data Cleaning

Many inputs in the "Sub Product" variable reference the same overarching category of debt. This is likely due to variability in filing this field from the consumer, and any possible survey changes by year. The consolidated category **"Misc Debt"** covers "Other Debt", "Other (i.e., phone, health club, etc.)", and "Telecommunications debt" because they are similar to each other and uncorrelated with each other. The consolidated category **"Federal Student Loans"** covers "Federal Student Loan Debt", and "Federal Student Loan". Conversely, "Private Student Loan Debt", and "Non Federal Student" were lumped as **"Non-Federal Student Loan"** for private student loan debts. Other categories like "Auto debt", "Credit Card Debt", "Rental Debt", "Home Debt", and "Payday Debt" were consolidated based on the various direct referencing of said categories.

The levels for "Issues" field references high level classifications based on results. The category **"Collect Debt Not Owed"** encompasses "Attempts to collect debt not owed", and "Cont'd attempts to collect debt not owed" which both result in uncollected debts. The responses: "Took or threatened to take negative or legal action", and "Taking/threatening an illegal action" were grouped by the common trait and level name **"Aggressive Response"**. The levels "Threatened to contact someone or share information improperly", "Communication tactics", "False statements or representation", "Disclosure verification of

debt", "Improper contact or sharing of info" were grouped as **"Improper communication tactic"**. The levels regarding written and electronic notification about debt were grouped together as **"Debt Notifications"**. This classification loses information for respondents' internet availability, to prioritize overall receipt of debt notification.

The level **"No Comment"** consolidates "Company has responded to the consumer and the CFPB and chooses not to provide a public response", and "Company chooses not to provide a public response". Both responses result in no comments, which becomes the grouping. The level **"Improve Service"** consolidates "Company believes complaint represents an opportunity for improvement to better serve consumers", and "Company believes the complaint provided an opportunity to answer consumer's questions". This characteristic combines sentences talking about improvement in their services due to consumer complaints. The level **"Misunderstanding"** consolidates "Company believes the complaint is the result of a misunderstanding", and "Company believes complaint is the result of an isolated error" combines sentences pertaining to misunderstanding of a consumer.

The variable: **"Consumer consent provided"** has levels grouped as "Consent Provided", and lumps "No Consent Provided" from the original levels of "None", "Consent not provided", "other, consent withdrawn", and "N/A". Essentially this creates the variable as a simple two class characteristic for consent provided and consent not provided.

Zip codes had errors in the debt data which necessitated cleaning. Zip codes with special characters, and length less than five digits were identified. Zip codes with less than five digits were manipulated with a leading zero and then verified for correctness in a list of the unique 50 United States zip codes. Incorrect unique zip codes were under 70 observations, which was a low total number of our training dataset with around 47,000 observations. These incorrect zips were mode imputed based on their state to correct this issue. A t-test was performed between correct and incorrect zip codes and the null hypothesis was not rejected. This implies that the subset of incorrect zips are not significantly different from the correct zips. Despite this, mode imputation of incorrect zips by their corresponding state maintained our number of observations. Modal imputation reassigns an incorrect zip to the most frequently occurring zip code for that state.

The data was also subsetted for only the fifty United States. County level census data was collected based on ages below 24 years and above 65 years to represent the key working age population. Hispanic ethnicity, gender totals, and race variables were tabulated per county fip from the census data. Then these totals were created to proportions for each unique county so county information could be comparable in our data. Binaries for older Americans and servicemen were established based on the individual consumer debt filing.

A principal component analysis was performed on the county level debt data from the Urban Institute. There were 25 variables involved in this reduction, and the variables explained the type of debt such as medical, auto, student loans etc, and separated by non communities of color, and communities of color for each county. The correlation plot (in

appendix) shows that the variables are highly correlated, whether positive or negative. This makes this subset of variables a strong candidate for PCA, since we can likely reduce the data into components that retain the debt information while reducing the complexity of the dataset. Based on the scree plot, we chose 4 principal components, which explain over 93% of the variance. Considering these components attempt to capture the euclidean distances of interacted race variables on debt we are conceptualizing our four components as the correlations for the four largest debt holdings with the binary racial categories of non people of color and people of color. The data for people of color had many missing entries, which may be due to systemic access issues during the surveying process. Missing entries were imputed with a tree method in the PCA package, and a map of imputed cells was created for data transparency.

The variables: "Sub.product", "Issue", "County Age Proportion >64", “County Proportion Female", "Average County Household Income for Communities of Color", and “County Share of People of Color" were incorporated in a mixed clustering analysis determining medical clusters. The variables "Sub.product", "Issue" were categorical level variables, and the rest of the variables were numeric proportions of each county. The selection of these variables attempts to isolate gender and racial groups that face institutional, systemic social challenges. The clusters showcase that credit card debt and medical debt were the most pervasive among each cluster. However, the fit of the medical clusters had a relatively high mean squared error of ~210,000.

Models

Following the addition of the principal components and medical debt clusters onto the cleaned dataset, we checked again for missing values and imputed the mean/mode of the variable. Models under consideration included: linear probability and logit (both with lasso regularization), regression tree, random forest, and gradient boosting. Despite its lack of interpretability when compared with models such as logit, we chose XGBoost because it had the highest cross-validated F1 score of the group at 0.30. As a group we explored the winsorized approach with and without oversampling to address outliers, but our results did not make a meaningful difference in predictive ability[1]. However, oversampling improved the predictive power on the training datasets, but did not improve on the validation set. To address outliers and class imbalances (only about 12% of the variable relief was ever equal to 1 in our sample), we ran all models with class weights for the minority class of 5 times.

All of the models were generally poor fits with the data even after principal components analysis and clustering, including XGBoost. However, we chose XGBoost in part because it had the best fit of the group and performed best in predictive accuracy (based on F1 Scores) of the group. So, while we may not trust XGBoost alone here as a standalone model given its poor predictive power and dubious fit, it was the best performing model of the set that we ran on this data.

[1] Winsorization consolidated the 5% tails of the distribution of each variable into the 95% confidence interval.

Chosen Model

For our XGBoost model, we performed a grid search for the hyperparameters to optimize the fit. Following the grid search for each parameter, their optimized values were: eta = 0.05, colsample = 1/3, subsample = 1, max depth = 2, minimum child weight of 400, and gamma = 1. The relatively low value of eta guards against overfitting, but led to slower convergence. The relatively low max depth of 2 means that this model is growing relatively shorter trees, which grow faster and are less likely to keep unimportant variables, a real concern here with high existing multicollinearity in the dataset. The minimum child weight of 400 means that at least 400 observations are required in each terminal node, again leading to smaller trees.

Examining the results of the XGBoost model, we find that Sub product credit card debt, timely response, frequent and repeated calls, and miscellaneous debt all appear to be important variables based on their gain, indicating that they were the most important in increasing the accuracy of the model. The order of importance largely remains the same if we examine cover and frequency, meaning that their inclusion in the model both covered more data points than other features, and appeared most frequently in the construction of trees. Among all the important variables, sub-product credit card debt represented the highest frequency of appearance in all the trees across all boosting rounds. In other words, it was used more frequently relative to other variables. Similarly, it also had the highest Gain and Coverage, indicating the importance of the variable.

While we cannot directly determine the direction and magnitude of credit card debt's effect on the probability of debt relief, we can get some indication from more interpretable models, for instance, logistic regression. In the logit regression, the coefficient on credit card debt was 7.48, indicating a strong increase in the likelihood of relief if the complaint was about credit card debt, holding all else constant. Since the data are identical between logit and gradient boosting, we would expect that relationship to hold in some capacity in the table above, indicating that credit card debt is much more likely to be discharged relative to other kinds of debt.

Ethical Considerations

The stated models are primarily tasked with minimizing error (or maximizing F1 score) without consideration of the social or moral implications of the decisions they make to achieve their objective. Debt relief determination has a major social impact, which we must acknowledge, especially where the model fails to meet ethical standards in addition to modeling assumptions. The outcome of debt relief has an impact for each individual complaint applicant which can range from small to large impacts. Thus, it is crucial for our model to determine debt relief outcomes for cases when relief is warranted. In a machine learning framework, true positives (accuracy) is an important classification evaluation of the model, but analyzing false negatives and false positives is deemed a higher priority for model evaluation. We want our model to minimize false negatives (which is denial of relief

when it is warranted) to reduce the negative socio-economic impacts for applicants as they file debt complaints especially for counties where there are higher proportions of marginalized demographic groups.

We are skeptical that our model addresses pareto outcomes for counties with higher proportions of marginalized groups. Our final model was analyzed to examine different racial, gender, or other county demographic variables that were impactful in predicting debt relief.

None of the racial categories ended up being among the five most important variables in our final model, but there are other methods of determining how race fits into the output. While gradient boosting models are somewhat challenging to interpret, we can still find changes in predicted probability of relief by evaluating all variables at their mean (or mode for categorical variables). Holding all other variables constant at their mean/mode, increasing the county ratio of people of color from 50% to 75% is associated with a 5% decline in the probability of debt relief, suggesting that race has an impact even if no race variable was top five in importance. In addition, the principal components made up of county debt collection variables had a significant number of missing values for Black-majority counties, suggesting some level of bias in data collection. Since these values were imputed with the mean, our components (and the resulting models) are inherently biased toward the center of those distributions, which is potentially a major issue if those debt collection statistics are missing for a reason related to county race characteristics. Age binaries for applicants over 64 years were dropped from the model to compare the effects of this binary, but no real impact was observed.

Conclusion

The CFPB complaint dataset was combined with the Urban Institute's county level debt dataset, and with county level demographic data from the U.S. census. This merged dataset provides the county characteristics for each, anonymized complaint applicant. The goal of this project is to analyze how the unique county-level socio-economic composition affects an individual's debt relief outcome. After cleaning, and manipulating our datasets an XGBoost model was selected as the best model. This modeling choice was selected after exploring more interpretable models such as logit and tree models against using 10-fold cross-validation. The other models either failed to produce meaningful results or did not meet modeling assumptions, or a combination of both.

The XGBoost model was then validated on another compliant dataset from CFPB after cleaning and merging the three datasets. On the validation dataset, the XGBoost model had an accuracy of 74% and an F1 Score of 0.29. While determining correct relief responses is important, in the case of debt relief, false negatives and false positives are important as well to ensure the customer recuperates their losses.

Appendix

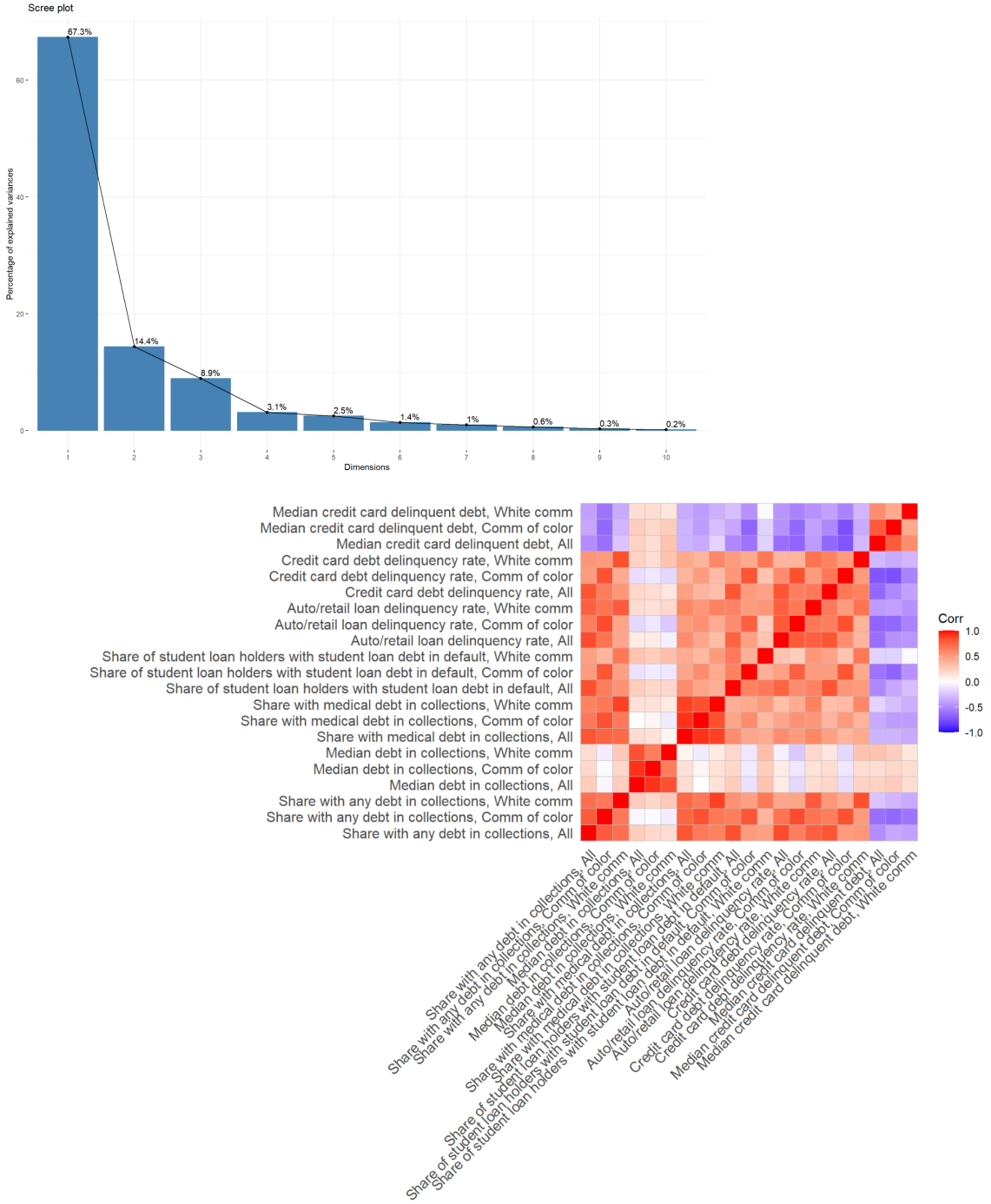


Figure 1 showcases the scree plot (top) and correlation matrix (bottom) of the county debt variables included in the PCA reduction. Four components were selected due to explaining 93.7% of the variance.

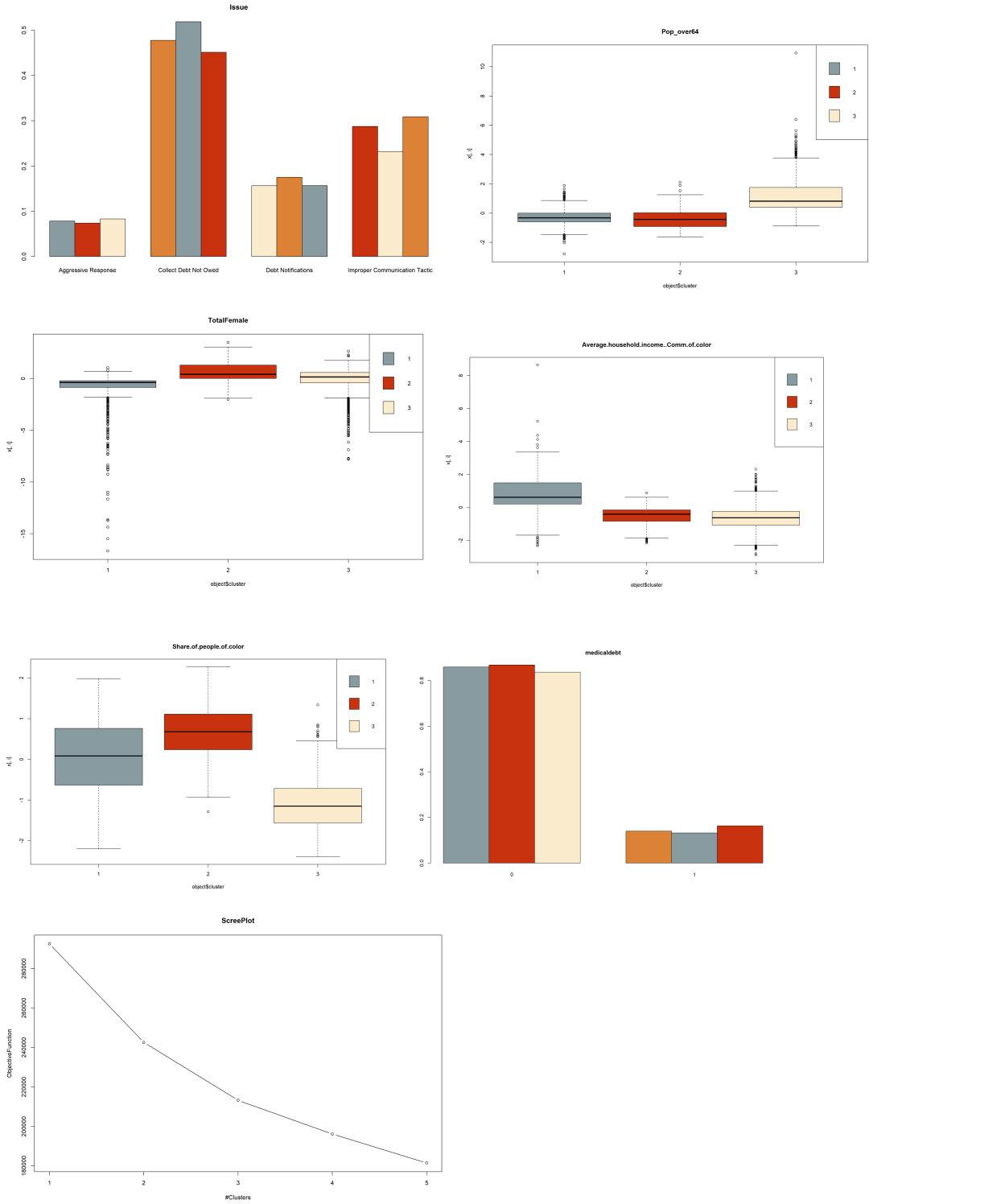


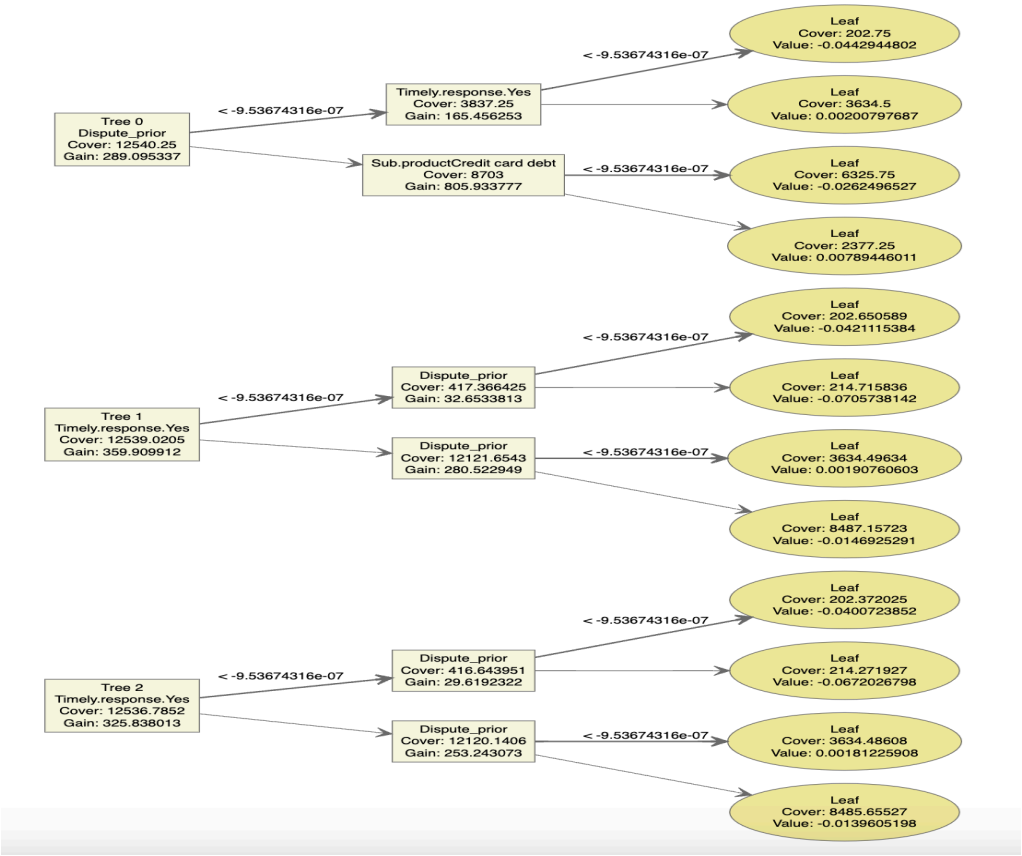
Figure 2 showcases medical cluster informational plots of the categorical variables along with a scree plot of cluster MSE.

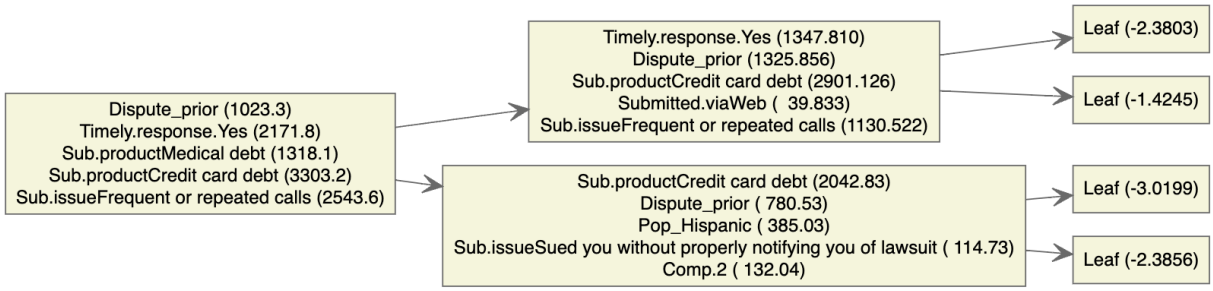
Table 1 which compares the F1 Scores of all models on the training datasets.

Cross-validated F1 Scores	
Model	**F1 Score**
Linear Probability (Lasso)	0.278
Logit (Lasso)	0.292
Regression Tree	0.279
Random Forest	0.280
Gradient Boosting (XGBoost)	0.303

Gradient Boosting/ XGBoost

XGBoost training outputs are under:

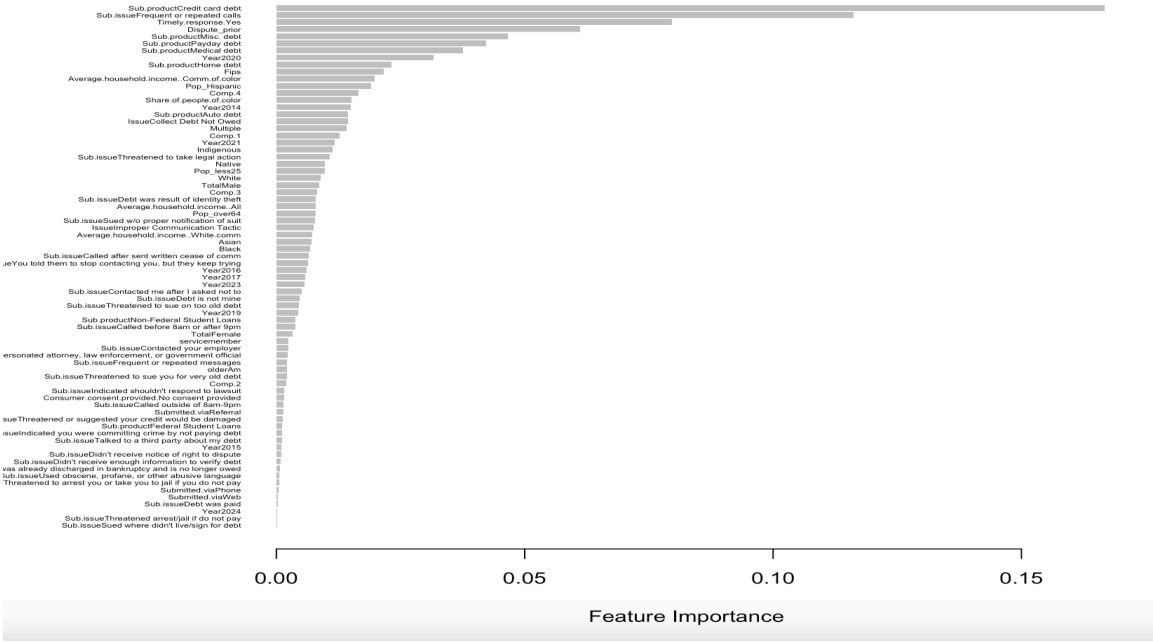




Importance frame:

Top 8 variables from Importance Matrix

Feature	Gain	Cover	Frequency
Sub.Product Credit Card Debt	0.1668	0.0467	0.0461
Sub.issue Frequent or repeated calls	0.1162	0.0413	0.0351
Timely Response. Yes	0.0796	0.0418	0.0337
Dispute_prior	0.0612	0.0382	0.0303
Sub.product Misc. debt	0.0466	0.0189	0.0186
Sub.product Payday debt	0.0422	0.0308	0.0227
Sub.product Medical debt	0.0376	0.0154	0.0131
Year2020	0.0316	0.032	0.0234



Training confusion matrix output:

conf_matrix_training

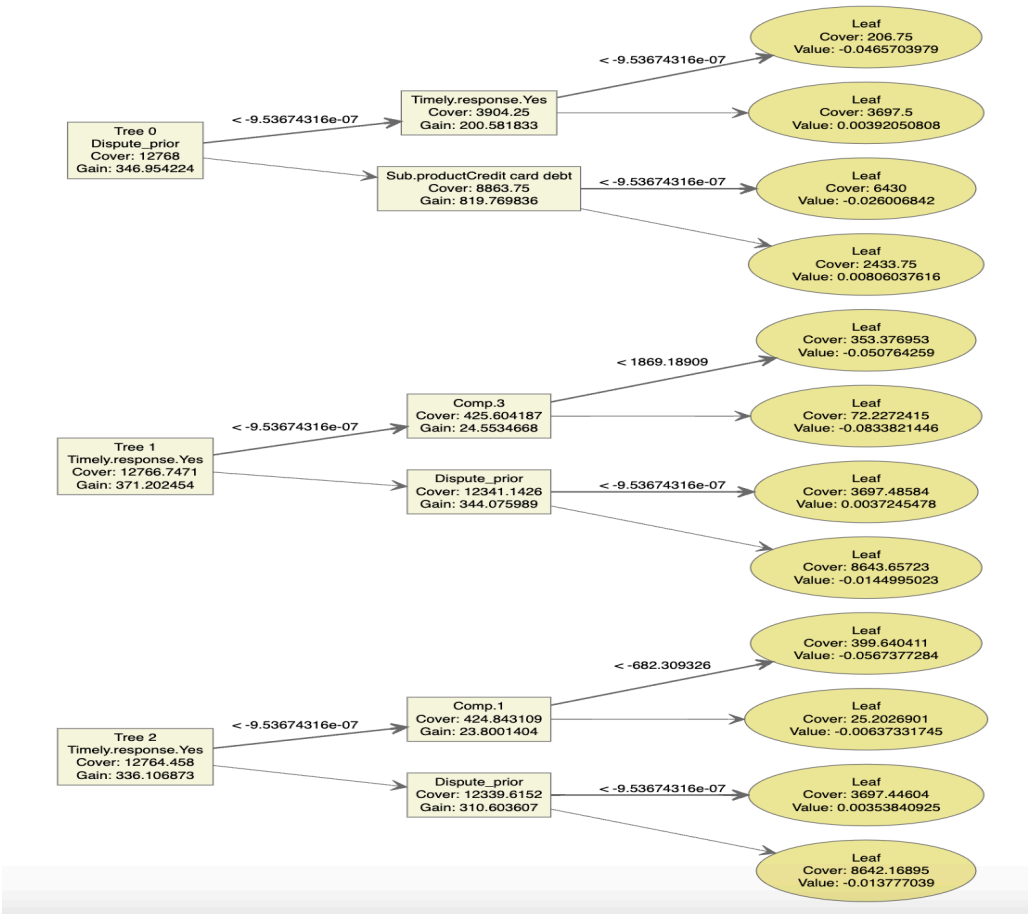
y_pred_class	ytest 0	1
0	9616	1129
1	2764	811

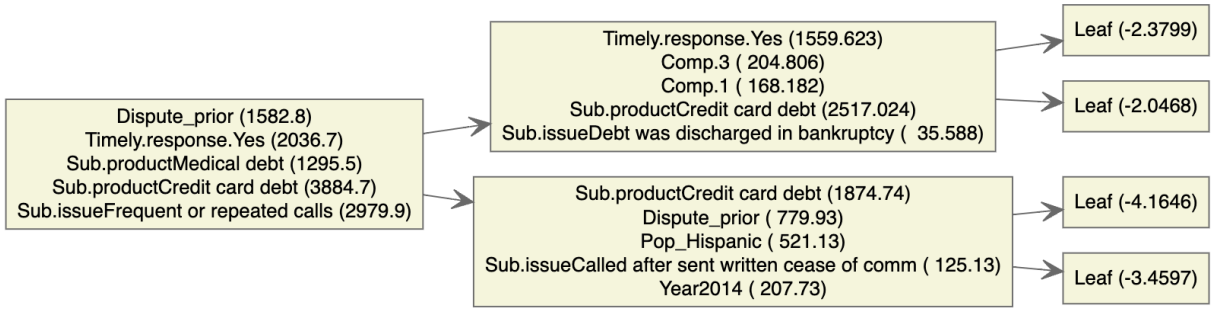
> accuracy_training<- sum(diag(conf_matrix_training)) / sum(conf_matrix_training)

accuracy_training
0.7281425

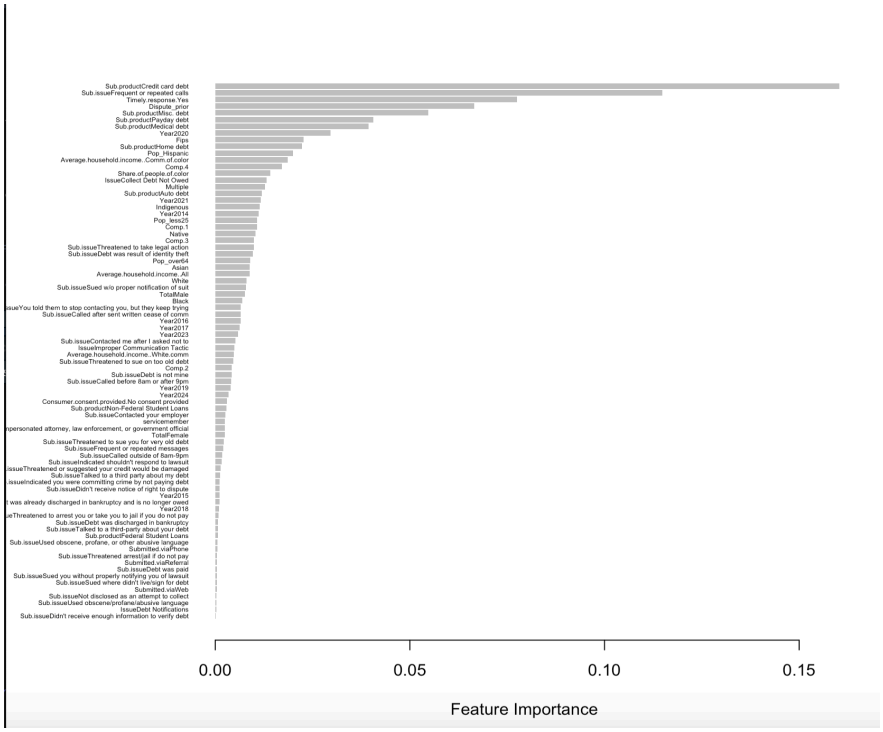
Logit, lasso, the trees, and random forests, did not meet assumptions.

Below is the XGBOOST output without older american variable:





Importance frame without Older American:



Validation XGBOOST outputs:

conf_matrix_test

	ytrain2	
y_pred_class1	0	1
0	29632	3577
1	8230	2389

> accuracy_test<- sum(diag(conf_matrix_test)) / sum(conf_matrix_test)
> accuracy_test

Accuracy = 0.730606

Scripts:

```
setwd("C:/Users/18045/Documents/R/ML/ML Project") #Sean's WD
#setwd("") #Nirav's WD
#setwd("") #Elise's WD
#setwd("") #Isaac's WD
#setwd("") #Truman's WD
#install.packages("clustMixType", "wesanderson")
library(readr)
library(data.table)
library(readxl)
library(tidyverse)
library(formattable)
```

```
library(dplyr)
library(plyr)
library(tidyr)
library(ggplot2)
library(cowplot)
library(scales)
library(corrplot)
#install.packages('clustMixType')
library(clustMixType)
#install.packages('wesanderson')
library(wesanderson)
#install.packages("zipcodeR")
library(zipcodeR)
#install.packages('stringr')
library(stringr)
library(glmnet)
library(caret)
library(caTools)
library(xgboost)
library(Matrix)

#Loading The Data----

group3 <- read_csv("group3.csv", col_types = cols(...1 = col_skip()))

fips_data <- read_csv("zip_fips.csv", col_types = cols(...1 = col_skip(),
                                                       STCOUNTYFP = col_number()))

countydebt <- read_excel("dia_lbls_all_overall_county_2022_02_14Sep2023.xlsx")

census <- read_csv("cc-est2022-all.csv")

#Isolating the Dependent Variable-----

group3$relief<- ifelse(group3$Company.response.to.consumer %in% c("Closed with monetary relief",
                                                       "Closed with relief",
                                                       "Closed with non-monetary relief"),
                       1,0)

#Cleaning-----

#drop non states
group3$drop <- as.numeric(group3$State %in% c("NONE", "None", "DC", "AA", "AS", "AP", "AE", "FM","GU", "MH", "MP", "PR",
"VI", "UNITED STATES MINOR OUTLYING ISLANDS"))
group3 <- subset(group3, !(State %in% c("AE", 'AP'))) #dropping military bases

fips_data <- fips_data[!fips_data$STATE %in% c("NONE", "None", "DC", "AA", "AS", 'AP', "AE", "FM","GU", "MH", "MP", "PR", "VI",
"UNITED STATES MINOR OUTLYING ISLANDS"),]

ctfips <-  (ifelse((fips_data$STATE == 'CT' & str_detect(fips_data$STCOUNTYFP, "9")),
            as.numeric(paste0("01", fips_data$STCOUNTYFP)), fips_data$STCOUNTYFP))

fips_data$STCOUNTYFP <- ctfips

countydebt$`County FIPS` <- as.numeric(countydebt$`County FIPS`)

#Named Factors ----
group3 <- group3 %>%
  mutate(Company.response.to.consumer = case_when(Company.response.to.consumer == "In progress" ~ 0,
                                       Company.response.to.consumer == "Closed with explanation" ~ 1,
                                       Company.response.to.consumer == "Untimely response" ~ 2,
```

```
                    Company.response.to.consumer == "Closed with non-monetary relief" ~ 3,
                    Company.response.to.consumer == "Closed with monetary relief" ~ 4,
                    Company.response.to.consumer == "Closed" ~ 5),
        Sub.product = case_when(Sub.product == "Other debt" ~ "Misc. debt",
                Sub.product == 'Other (i.e. phone, health club, etc.)' ~ "Misc. debt",
                Sub.product == "Telecommunications debt" ~ "Misc. debt",
                Sub.product == 'I do not know' ~ "Misc. debt",
                Sub.product == "Federal student loan debt" ~ "Federal Student Loans",
                Sub.product == "Federal student loan"  ~ "Federal Student Loans",
                Sub.product == "Private student loan debt" ~ "Non-Federal Student Loans",
                Sub.product == "Non-federal student loan"  ~ "Non-Federal Student Loans",
                Sub.product == "Auto" ~ "Auto debt",
                Sub.product == "Auto debt" ~ "Auto debt",
                Sub.product == "Credit card debt" ~ "Credit card debt",
                Sub.product == "Credit card" ~ "Credit card debt",
                Sub.product == 'Mortgage debt'~ "Home debt", #fixed a typo in mortgage (by Nirav).
                Sub.product == "Mortgage" ~ "Home debt",
                Sub.product == 'Medical' ~ "Medical debt",
                Sub.product == "Medical debt" ~ "Medical debt",
                Sub.product == "Rental debt" ~ "Rental debt",
                Sub.product == "Payday loan debt" ~ "Payday debt",
                Sub.product == "Payday loan" ~ "Payday debt"),
        Issue = case_when(Issue == "Attempts to collect debt not owed" ~ "Collect Debt Not Owed",
                Issue == 'Cont\'d attempts collect debt not owed' ~ "Collect Debt Not Owed",
                Issue == "Took or threatened to take negative or legal action" ~ "Aggressive Response",
                Issue == 'Taking/threatening an illegal action' ~ "Aggressive Response",
                Issue == "Threatened to contact someone or share information improperly" ~ "Improper Communication Tactic",
                Issue == "Communication tactics" ~ "Improper Communication Tactic",
                Issue == "False statements or representation" ~ "Improper Communication Tactic",
                Issue == "Disclosure verification of debt" ~ "Improper Communication Tactic",
                Issue == "Improper contact or sharing of info" ~ "Improper Communication Tactic",
                Issue == "Written notification about debt" ~ "Debt Notifications",
                Issue == "Electronic communications"  ~ "Debt Notifications"),
        Company.public.response = case_when(Company.public.response == "Company has responded to the consumer and the
CFPB and chooses not to provide a public response" ~ "No Comment",
                    Company.public.response == "Company chooses not to provide a public response" ~ "No Comment",
                    Company.public.response == "Company believes complaint represents an opportunity for
improvement to better serve consumers" ~ "Improve Service",
                    Company.public.response == "Company believes the complaint provided an opportunity to answer
consumer's questions" ~ "Improve Service",
                    Company.public.response == "Company believes the complaint is the result of a misunderstanding" ~
"Misunderstanding",
                    Company.public.response == "Company believes complaint is the result of an isolated error" ~
"Misunderstanding"),
        Consumer.consent.provided. = case_when(Consumer.consent.provided. == "Consent provided" ~ "Consent provided",
                    Consumer.consent.provided. == "None"  ~ "No consent provided",
                    Consumer.consent.provided. == "Consent not provided"  ~ "No consent provided",
                    Consumer.consent.provided. == "Other"  ~ "No consent provided",
                    Consumer.consent.provided. == "Consent withdrawn"  ~ "No consent provided",
                    Consumer.consent.provided. == "N/A" ~ "No consent provided"))

#Dummy for after date (form changes NA imputation)
group3$Date.received <- as.Date(group3$Date.received, format = "%m/%d/%y")
group3$Dispute_prior <- ifelse(group3$Date.received  > as.Date('04/24/17', format= "%m/%d/%y"), 1,0)

#older Americans and servicefolk
group3$servicemember <- ifelse(str_detect(group3$Tags, "Servicemember"), 1, 0)
group3$olderAm <- ifelse(str_detect(group3$Tags, "Older American"), 1, 0)

#####Zip Code Cleaning#### -----
USA_zippop <- zip_code_db
```

```
#Clean Zip Reference data first ----

fips_data$ZIP <- as.character(fips_data$ZIP)
group3$ZIP.code <- as.character(group3$ZIP.code)

#check zips including numbers
table(str_detect( as.character(fips_data$ZIP), "[0-9]+$")) #looks like they're all numbers

#check zips less than 5 characters
table(nchar(fips_data$ZIP) < 5) #2900 entries less than 5 digits - leading zeros?
invalidzip <- fips_data$ZIP[nchar(fips_data$ZIP) < 5]

table(nchar(fips_data$ZIP) > 5) #No entries greater than 5 digits

#Check leading zeros
table(grepl("^0", group3$ZIP.code)) #looks like they already have leading zeros

#LEADING Zero Code replacement
#Place a leading zero for the problem zips
zip <- as.numeric(fips_data$ZIP)
for(i in 1:length(zip)){
  if(nchar(as.numeric(zip[i])) < 5){
            zip[i] <- paste0("0", zip[i])
  }
}

fips_data$ZIP <- (zip)
table(nchar(fips_data$ZIP) < 5 ) #incorrect zip identification

error_zip <- fips_data[nchar(fips_data$ZIP) < 5, ]

#manual fix
fips_data$ZIP[fips_data$ZIP == error_zip$ZIP] <- '00501'

#Group3 Clean Zips -----

#check zips including numbers
table(str_detect( as.character(group3$ZIP.code), "[0-9]+$")) #looks like they're all numbers

#check zips less than 5 characters
table(nchar(group3$ZIP.code) < 5) #No entries less than 5 digits
table(nchar(group3$ZIP.code) > 5) #No entries greater than 5 digits

#Check leading zeros
table(grepl("^0", group3$ZIP.code)) #looks like they already have leading zeros

unique_zips <- unique(group3$ZIP.code)

zip_binary_map <- unique_zips %in% fips_data$ZIP

table(zip_binary_map)

error_zips <- unique_zips[!zip_binary_map] #erroneous zips

#T.test of error zips -----
t.test(table(zip_binary_map), alternative = 'two.sided') #fail to reject the null - the error zips are not significant at 5% level

for (error_state in unique(group3$State[group3$ZIP.code %in% error_zips])) {
  mode_zip <- names(sort(table(group3$ZIP.code[group3$State == error_state]), decreasing = TRUE)[1])
  group3$ZIP.code[group3$ZIP.code %in% error_zips & group3$State == error_state] <- mode_zip
```

```
}

#LEADING Zero Code replacement for the problem zips
#zip <- as.character(group3$ZIP.code[zip_binary_map == FALSE])
#for(i in 1:length(zip)){
#  if(nchar(as.numeric(zip[i])) < 5){
#             zip[i] <- paste0("0", zip[i])
#  }
#}

#insert zips in data

#Percusive Maintainence ---
# run the loop until there are no issues

error_zips <- unique_zips[!zip_binary_map] #82 erroneous zips
table(zip_binary_map)

for (error_state in unique(group3$State[group3$ZIP.code %in% error_zips])) {
  mode_zip <- names(sort(table(group3$ZIP.code[group3$State == error_state]), decreasing = TRUE)[1])
  group3$ZIP.code[group3$ZIP.code %in% error_zips & group3$State == error_state] <- mode_zip
}

#Check if this worked
unique_zips <- unique(group3$ZIP.code)
zip_binary_map <- unique_zips %in% fips_data$ZIP
table(zip_binary_map) #looks like there are no issues anymore

rm(i, zip, zip_binary_map, unique_zips) #remove these variables when done.

#FIPS Fix ------

state_data <- data.frame(
  Postal_Abbr = c(
             "AL", "AK", "AZ", "AR", "CA", "CO", "CT", "DE", "FL", "GA", "HI", "ID", "IL", "IN", "IA", "KS", "KY", "LA",
             "ME", "MD", "MA", "MI", "MN", "MS", "MO", "MT", "NE", "NV", "NH", "NJ", "NM", "NY", "NC", "ND", "OH", "OK", "OR",
             "PA", "RI", "SC", "SD", "TN", "TX", "UT", "VT", "VA", "WA", "WV", "WI", "WY"
  ),
  FIPS_Code = c(
             "0", "0", "0", "0", "0", "0", "0", "10", "12", "13", "15", "16", "17", "18", "19", "20", "21", "22", "23",
             "24", "25", "26", "27", "28", "29", "30", "31", "32", "33", "34", "35", "36", "37", "38", "39", "40", "41", "42",
             "44", "45", "46", "47", "48", "49", "50", "51", "53", "54", "55", "56"
  )
)

countydebt$`County FIPS` <- as.character(paste0("0", countydebt$`County FIPS`))

fips_data$STCOUNTYFP <- as.character(paste0("0", fips_data$STCOUNTYFP))

#Q3 ----

#merge
merg_zips <- merge(group3, fips_data, by.x =  'ZIP.code', by.y = 'ZIP')

#Q4 -----

merg_fips <- merge(merg_zips, countydebt,  by.x = 'STCOUNTYFP', by.y = 'County FIPS')

#Q5----

census_fips <- (paste(census$STATE, census$COUNTY, sep = ""))
```

```
census_fips <- ifelse(!(census_fips %in% fips_data$STCOUNTYFP), paste0('0', census_fips), census_fips)
table(census_fips %in% fips_data$STCOUNTYFP)

census <- cbind(census, census_fips)

county_totals <- subset(census, (AGEGRP == 0 & YEAR == 1))

#Age dummies

unique_fips <- unique(census$census_fips)
results_list <- list()
results_list2 <- list()
fip <- 0

for (fip in unique_fips) {
  result <- census %>%
            filter(census_fips == fip, YEAR == 1, AGEGRP >= 1 & AGEGRP <= 5) %>%
            summarise(TOTAL_POP = sum(TOT_POP))

  results_list[[fip]] <- result

  result2 <- census %>%
            filter(census_fips == fip, YEAR == 1, AGEGRP >= 14 & AGEGRP <= 18) %>%
            summarise(TOTAL_POP = sum(TOT_POP))

  results_list2[[fip]] <- result2
}

combined_results <- bind_rows(results_list, .id = "Pop_less25")
combined_results2 <- bind_rows(results_list2, .id = "Pop_over64")

#Ethnicity

colnames(county_totals)

county_demos <- cbind(bind_rows(results_list, .id = "Pop_less25"), bind_rows(results_list2, .id = "Pop_over64"))

county_demos$Pop_Hispanic <- rowSums(county_totals[, c('H_FEMALE', 'H_MALE')])

colnames(county_demos) <- c("Fips", "Pop_less25", "Fips", "Pop_over64", "Pop_Hispanic")
county_demos <- county_demos[,-3]

#include male and female county totals
county_demos$TotalMale <- county_totals$TOT_MALE
county_demos$TotalFemale <- county_totals$TOT_FEMALE

#Race counts per county

races <- county_totals[,c("WA_MALE", "WA_FEMALE", "BA_MALE", "BA_FEMALE",
  "IA_MALE", "IA_FEMALE", "AA_MALE", "AA_FEMALE", "NA_MALE", "NA_FEMALE", "TOM_MALE", 'TOM_FEMALE')]

tot_white <- rowSums(races[,1:2])
tot_black <- rowSums(races[,3:4])
tot_indigenous <- rowSums(races[,5:6])
tot_asian <- rowSums(races[,7:8])
tot_native <- rowSums(races[,9:10])
tot_two <- rowSums(races[,11:12])

total_population <- data.frame(White = tot_white,
                               Black = tot_black,
```

```
                    Indigenous = tot_indigenous,
                    Asian = tot_asian,
                    Native = tot_native,
                    Multiple = tot_two,
                    County_pop = county_totals$TOT_POP,
                    Fips = county_totals$census_fips)

total_population$County_pop - rowSums(total_population[,1:6]) #residual census check

county_demos <- left_join(county_demos, total_population, by = "Fips")

result <- apply(county_demos[, 2:12], 2, function(x) {
  x / total_population$County_pop
})

county_demos[,2:12] <- result
county_demos <- county_demos[,-13]

#Q6-----

#matching fips to zips

merg_county_demos <- merge(county_demos, fips_data, by.x = 'Fips', by.y = "STCOUNTYFP")

clean_group3 <- left_join(group3, merg_county_demos, by = c('ZIP.code' = 'ZIP'))

hopefully_all <- left_join(clean_group3, countydebt, by = c('Fips' = 'County FIPS'))

#making csvs to save time in cleanup
write.csv(county_demos, "county_demos.csv")
write.csv(merg_fips, 'merg_fips.csv')
write.csv(clean_group3, 'clean_group3.csv')
write.csv(hopefully_all, 'hopefully.csv')

#Q7------

colnames(hopefully_all)
temp <-  hopefully_all[, c(40:60)]
temp <- sapply(temp, as.numeric)

imputation_map <- is.na(temp) #map of our NA values before we imputate

temp_imputed <- apply(temp, 2, function(x) {
  x[is.na(x)] <- mean(x, na.rm = TRUE)
  return(x)
})

#### PCA
library(ggcorrplot)
library("FactoMineR")
library(factoextra)
# PCA is based on correlations, not distance.
## So we need to store the correlation matrix.

corr_matrix <- cor(temp_imputed)
corr_matrix <- scale(corr_matrix)
ggcorrplot(corr_matrix)

## do PCA
debt.pca <- princomp(corr_matrix)
summary(debt.pca)
```

```
fviz_eig(debt.pca, addlabels = TRUE)

comps <- debt.pca$scores[,1:4]

# loadings for first 5 components
debt.pca$loadings[, 1:4]

#scree plot
variance_explained <- debt.pca$sd^2 / sum(debt.pca$sd^2)*100

variance_explained[1:7]

qplot(c(1:21), variance_explained) +
  geom_line() +
  xlab("Principal Component") +
  ylab("Variance Explained") +
  ggtitle("Scree Plot") +
  ylim(0, 100)

#get 4 components
feature_vector <- (debt.pca$loadings)
feature_vector_transposed <- t(feature_vector)

original_data_transposed <- t(temp_imputed)

dim(original_data_transposed)
dim(feature_vector)

data_reoriented <- feature_vector_transposed %*% original_data_transposed

data_reoriented_df <- as.data.frame(data_reoriented)
data_reoriented_df <- t(data_reoriented_df)

comps <- data_reoriented_df[,c(1:4)]

comps <- as.data.frame(comps)

#now merge to the dataset
temp2 <- cbind(hopefully_all[,-c(37:60)], comps)

colnames(temp2)

write.csv(temp2, 'straightouttacompton.csv')

######## PRIOR CODE GENERATES CSVs. LOAD CSV FROM HERE ONWARD ########--------

NWA <- read.csv('straightouttacompton.csv')
NWA <- NWA[,-1]

colnames(NWA)

#Q8 ----

### Clustering code
# make a separate dataset, and then make sure each variable is the right class.
set.seed(23748234)

ca <- NWA[, c("Sub.product",
              "Issue",
              "Pop_over64",
              "TotalFemale",
```

```
            "Average.household.income..Comm.of.color",
            "Share.of.people.of.color")      ]

ca[,1] <- as.factor(ca[,1] )
ca[,2] <- as.factor(ca[,2] )
ca[,3] <- scale(as.numeric(ca[,3] ))
ca[,4] <- scale(as.numeric(ca[,4] ))
ca[,5] <- scale(as.numeric(ca[,5] ))
ca[,6] <- scale(as.numeric(ca[,6] ))

colnames(ca) <- c("Sub.product",
            "Issue",
            "Pop_over64",
            "TotalFemale",
            "Average.household.income..Comm.of.color",
            "Share.of.people.of.color")

ca$medicaldebt <- as.factor(ifelse(ca$Sub.product == "Medical debt", 1,0))

impute_map_medical <- is.na(ca) #map of imputation

# 2 cluster
kpres2 <- kproto(x=ca[,c(2:7)], k=2, na.rm = 'imp.internal')

# 3 cluster
kpres3 <- kproto(x=ca[,c(2:7)], k=3,  na.rm = 'imp.internal')

# Plots
clprofiles(kpres3, ca[,c(2:7)], col = wes_palette("Royal1", 4, type = "continuous")) #c('blue', 'red', 'green'))

# scree plot

n.scree<-ncol(ca)-2
Es<- numeric(n.scree)
for(i in 1:n.scree){
  kpres <- kproto(ca,k=i,nstart=5, verbose = FALSE)
  Es[i] <- kpres$tot.withinss
}

plot(1:n.scree, Es[1:5], type="b", ylab="ObjectiveFunction",
            xlab="#Clusters", main="ScreePlot") #figure2

NWA$MedicalDebtClusters <- kpres3$cluster

colnames(NWA)

write.csv(NWA, 'q9.csv')

#Q9 -----

q9 <- read.csv('q9.csv')
colnames(q9)

library(lubridate)

q9$Year <- as.factor(year(q9$Date.received))

q9 <- q9 %>%
  select(Sub.product, Issue, Sub.issue, Consumer.consent.provided., Submitted.via, Timely.response. , relief, drop,
            Dispute_prior, servicemember, olderAm, Fips,
            Pop_less25, Pop_over64, Pop_Hispanic, White, Black, Asian, Native, Indigenous, Multiple, TotalMale, TotalFemale,
```

```
  Share.of.people.of.color, Average.household.income..All, Average.household.income..Comm.of.color,
Average.household.income..White.comm,
 Comp.1, Comp.2,Comp.3, Comp.4, MedicalDebtClusters,  Year)

#Transformations -------

#make factors
q9_s <- data.frame(
  lapply(q9, function(x) {
            if(is.character(x)) factor(x) else x
  })
)

'''
logged_vs <- log(q9_s[,c(13:25)])
lognames <- colnames(logged_vs)
lognames <- paste("log", lognames)
colnames(logged_vs) <- lognames

squared_vs <- q9_s[,c(13:25)]^2
sqnames <- colnames(squared_vs)
sqnames <- paste("sq", sqnames)
colnames(squared_vs) <- sqnames

standardized_vs <- scale(q9_s[,c(13:25)])
stan_names <- colnames(standardized_vs)
stan_names <- paste("stan", stan_names)
colnames(standardized_vs) <- stan_names

q9_s <- cbind(q9_s, logged_vs, squared_vs, standardized_vs)
'''

q9_s <- q9_s %>%
  filter(complete.cases(.))

write.csv(q9_s, "q9_s.csv")

##### train and test ------
library(caret)
train_control <- trainControl(method = "cv",
                              number = 10)

library(caTools)
set.seed(275142345)
split <- sample.split(q9_s, SplitRatio = 0.7)
train <- subset(q9_s, split == "TRUE")
test <- subset(q9_s, split == "FALSE")

#Q10-----

#Linear Probability (Lasso):----
set.seed(03110)

X <- model.matrix(relief ~ ., data = q9_s)[, -1]
Y <- q9_s$relief
# Define class weights
class_weights <- ifelse(Y == 1, 5, 1)

cv_fit <- cv.glmnet(X, Y, alpha = 1, family = "gaussian", nfolds = 10, weights = class_weights)

optimal_lambda <- cv_fit$lambda.min
```

```
predicted_probs <- predict(cv_fit, newx = X, s = "lambda.min", type = "response")
predicted_classes <- ifelse(predicted_probs > 0.5, 1, 0)

classification_error <- mean(predicted_classes != Y)
print(paste("Classification Error:", classification_error))

residuals <- as.numeric(Y) - predicted_probs
qqnorm(residuals)
qqline(residuals)

library(caret)

conf_matrix <- confusionMatrix(factor(predicted_classes), factor(Y))

print(conf_matrix)
#F1 0.278
#Note: We found that packages such as MLmetrics, which calculate F1 scores internally, were massively overinflating our F1 scores, so we calculated them all by hand using the confusion matricies.

#Logit (Lasso):----
set.seed(03110)

# Define class weights

X <- model.matrix(relief ~ ., data = q9_s)[, -1]
Y <- q9_s$relief
class_weights <- ifelse(Y == 1, 5, 1)

cv_fit <- cv.glmnet(X, Y, alpha = 1, family = "binomial", link = "logit", nfolds = 10, weights = class_weights)

optimal_lambda <- cv_fit$lambda.min

predicted_probs <- predict(cv_fit, newx = X, s = "lambda.min", type = "response")
predicted_classes <- predict(cv_fit, newx = X, s = "lambda.min", type = "class")

classification_error <- mean(predicted_classes != Y)
print(paste("Classification Error:", classification_error))

residuals <- as.numeric(Y) - predicted_probs
qqnorm(residuals)
qqline(residuals)


library(caret)

predicted_classes <- ifelse(predicted_probs > 0.5, 1, 0)

conf_matrix <- confusionMatrix(factor(predicted_classes), factor(Y))

print(conf_matrix)

#F1 0.292

#Regression tree ------
library(rpart)
library(rpart.plot)
library(MLmetrics)

set.seed(082020)
minsplit <- 20
```

```
# Define class weights
class_weights <- ifelse(q9_s$relief == 1, 5, 1)

rpart.control_params <- rpart.control(minsplit = minsplit, minbucket = round(minsplit/3), cp = 0.01,
                                      maxcompete = 4, maxsurrogate = 5, usesurrogate = 2, xval = 10,
                                      surrogatestyle = 0, maxdepth = 30)


relief.tree <- rpart(relief ~ ., data = q9_s, control = rpart.control_params, method = "class", weights = class_weights)


summary(relief.tree)


rpart.plot(relief.tree)


relief.tree$variable.importance


predicted_labels <- predict(relief.tree, type = "class")

misclassification_rate <- mean(predicted_labels != q9_s$relief)

print(misclassification_rate)

library(caret)

#predicted_classes <- ifelse(predicted_probs > 0.5, 1, 0)

conf_matrix <- confusionMatrix(factor(predicted_labels), factor(Y))

print(conf_matrix)

#F1 0.279

# Random forest----
library(caret)
library(randomForest)

X <- model.matrix(relief ~ ., data = q9_s)[, -1]
Y <- q9_s$relief
class_weights <- ifelse(Y == 1, 5, 1)

set.seed(275142)

ctrl <- trainControl(classProbs = TRUE, method = "cv", number = 10)

RF <- train(relief ~ ., data = q9_s, method = "rf",
            trControl = ctrl, importance = TRUE, ntree = 500,
            class.weights = class_weights, verbose = TRUE)

# Importance frame
importance_frame <- varImp(RF)
print(importance_frame)

# Plot variable depth
plot(varImpPlot(RF))

# Conditional min depth
```

```
# Picking the 5 most important variables
vars <- rownames(importance_frame)[1:5]
interactions_frame <- minDepthInteraction(RF, predictors = vars)
print(interactions_frame)
plot(interactions_frame)


misclassification_rate <- mean(predicted_labels != q9_s$relief)

print(misclassification_rate)

library(caret)

predicted_labels <- ifelse(predicted_probs > 0.5, 1, 0)

conf_matrix <- confusionMatrix(factor(predicted_labels), factor(Y))

print(conf_matrix)

#F1: 0.28

################################################## XG Boost
############################################################
#Loading the training dataset:
q9_s<-read.csv("q9_s.csv")
view(q9_s)


#Subsetting the oversampled data:
q9_s<- q9_s %>%
  select(Sub.product, Issue, Sub.issue, Consumer.consent.provided., Submitted.via, Timely.response. , relief, drop,
            Dispute_prior, servicemember, olderAm,
            Fips, Pop_less25, Pop_over64, Pop_Hispanic, White, Black, Asian, Indigenous, Native, Multiple, TotalMale, TotalFemale,
            Share.of.people.of.color, Average.household.income..All, Average.household.income..Comm.of.color,
Average.household.income..White.comm,
            Comp.1, Comp.2,Comp.3, Comp.4, MedicalDebtClusters,Year)

view(q9_s)


q9_s<- data.frame(
  lapply(q9_s, function(x) {
            if(is.character(x)) factor(x) else x
  })
)
'''
##########Winsorize code############
#### diminish the effect of outliers in the data set- Using the winsorization function
#Winsorize with custom thresholds (10th and 90th percentile)

install.packages(DescTools)
library (DescTools)

winsorize <- function(x, lower = quantile(x, 0.1), upper = quantile(x, 0.90)) {
  x[x < lower] <- lower
  x[x > upper] <- upper
  return(x)
}

q9_s$Pop_less25<- winsorize(q9_s$Pop_less25)
q9_s$Pop_over64<- winsorize(q9_s$Pop_over64)
```

```
q9_s$Pop_Hispanic<- winsorize(q9_s$Pop_Hispanic)
q9_s$White<- winsorize(q9_s$White)
q9_s$Black<- winsorize(q9_s$Black)
q9_s$Asian<- winsorize(q9_s$Asian)
q9_s$Native<- winsorize(q9_s$Native)
q9_s$Indigenous<- winsorize(q9_s$Indigenous)
q9_s$TotalMale<- winsorize(q9_s$TotalMale)
q9_s$TotalFemale<- winsorize(q9_s$TotalFemale)
q9_s$Share.of.people.of.color<- winsorize(q9_s$Share.of.people.of.color)
q9_s$Average.household.income..All<- winsorize(q9_s$Average.household.income..All)
q9_s$Average.household.income..Comm.of.color<- winsorize(q9_s$Average.household.income..Comm.of.color)
q9_S$Average.household.income..White.comm<- winsorize(q9_s$Average.household.income..White.comm)
'''
############################## XG Boost ########################################
#Loading the training dataset:
q9_s<-read.csv("q9_s.csv")
view(q9_s)

#Subsetting the training data:
q9_s<- q9_s %>%
  select(Sub.product, Issue, Sub.issue, Consumer.consent.provided., Submitted.via, Timely.response. , relief, drop,
         Dispute_prior, servicemember, olderAm,Fips, Pop_less25, Pop_over64, Pop_Hispanic, White, Black, Asian, Indigenous,
Native, Multiple, TotalMale, TotalFemale,
         Share.of.people.of.color, Average.household.income..All, Average.household.income..Comm.of.color,
Average.household.income..White.comm,
         Comp.1, Comp.2,Comp.3, Comp.4, MedicalDebtClusters,Year)

view(q9_s)


q9_s<- data.frame(
  lapply(q9_s, function(x) {
         if(is.character(x)) factor(x) else x
  })
)

#Converting year to factor:
q9_s <- q9_s %>%
  mutate(Year = factor(Year))




############################## XG Boost ########################################
set.seed(27514)
split.imp <- sample.split(q9_s, SplitRatio = 0.7)
train1 <-  subset(q9_s, split.imp == "TRUE")
test1 <- subset(q9_s, split.imp == "FALSE")

# make them numeric matrix
xtrain<-sparse.model.matrix(relief ~. -1, data = train1)
ytrain <- as.array(train1$relief)
xtest <- sparse.model.matrix(relief ~ .-1, data = test1)
ytest <- as.array(test1$relief)

### run train model
xgb1 <- xgboost(data = xtrain, label = ytrain,
         nrounds = 500)

# run test model
y_pred <- predict(xgb1, xtest)
```

```
y_pred
# get MSE
test.MSE<-mean((ytest - y_pred)^2)
test.MSE
# get residual (if continuous outcome)
r<-ytest - y_pred
qqnorm(r)

### plot
xgb.plot.multi.trees(xgb1)

#Various different hyperparameters:
#eta:
eta=c(0.05, 0.1, 0.2,0.5,1)

# = colsample_bylevel candidates = #
cs=c(1/3,2/3,1)

# = max_depth candidates = #
md=c(2,4,6,10)

# = sub_sample candidates = #
ss=c(0.25,0.5,0.75,1)

# = standard model is the second value of each vector above = #
standard=c(2,2,3,2)

# = min_child_weights candidates = #
mcw=c(1,10,100,400)

# = gamma candidates = #
gamma=c(0.1,1,10,100)

#Running different hyperparameters:
#a) eta:
set.seed(13856)
conv_eta = matrix(NA,500,length(eta))
pred_eta = matrix(NA,nrow(test1), length(eta))
colnames(conv_eta) = colnames(pred_eta) = eta

for(i in 1:length(eta)){
  params=list(eta = eta[i], colsample_bylevel=cs[standard[2]],
              subsample = ss[standard[4]], max_depth = md[standard[3]],
              min_child_weigth = 1)
  xgb=xgboost(xtrain, label = ytrain, nrounds = 500, params = params)
  conv_eta[,i] = xgb$evaluation_log$train_rmse
  pred_eta[,i] = predict(xgb, xtest)
}


conv_eta = data.frame(iter=1:500, conv_eta)
conv_eta = melt(conv_eta, id.vars = "iter")
ggplot(data = conv_eta) + geom_line(aes(x = iter, y = value, color = variable))
(RMSE_eta = sqrt(colMeans((as.numeric(ytest)-pred_eta)^2)))


#b) Colsample_bylevel:
set.seed(1284654)
conv_cs = matrix(NA,500,length(cs))
pred_cs = matrix(NA,nrow(test1), length(cs))
colnames(conv_cs) = colnames(pred_cs) = cs
```

```
for(i in 1:length(cs)){
  params = list(eta = eta[standard[1]], colsample_bylevel = cs[i],
                subsample = ss[standard[4]], max_depth = md[standard[3]],
                min_child_weigth = 1)
  xgb=xgboost(xtrain, label = ytrain,nrounds = 500, params = params)
  conv_cs[,i] = xgb$evaluation_log$train_rmse
  pred_cs[,i] = predict(xgb, xtest)
}

conv_cs = data.frame(iter=1:500, conv_cs)
conv_cs = melt(conv_cs, id.vars = "iter")
ggplot(data = conv_cs) + geom_line(aes(x = iter, y = value, color = variable))
(RMSE_cs = sqrt(colMeans((as.numeric(ytest)-pred_cs)^2)))


#c) Max Depth:
set.seed(1284654)
conv_md=matrix(NA,500,length(md))
pred_md=matrix(NA,nrow(test1),length(md))
colnames(conv_md)=colnames(pred_md)=md

for(i in 1:length(md)){
  params=list(eta=eta[standard[1]],colsample_bylevel=cs[standard[2]],
                subsample=ss[standard[4]],max_depth=md[i],
                min_child_weigth=1)
  xgb=xgboost(xtrain, label = ytrain,nrounds = 500,params=params)
  conv_md[,i] = xgb$evaluation_log$train_rmse
  pred_md[,i] = predict(xgb, xtest)
}

conv_md=data.frame(iter=1:500,conv_md)
conv_md=melt(conv_md,id.vars = "iter")
ggplot(data=conv_md)+geom_line(aes(x=iter,y=value,color=variable))
(RMSE_md=sqrt(colMeans((as.numeric(ytest)-pred_md)^2)))


#d) Sub Sample:
set.seed(1)
conv_ss=matrix(NA,500,length(ss))
pred_ss=matrix(NA,nrow(test1),length(ss))
colnames(conv_ss)=colnames(pred_ss)=ss

for(i in 1:length(ss)){
  params=list(eta=eta[standard[1]],colsample_bylevel=cs[standard[2]],
                subsample=ss[i],max_depth=md[standard[3]],
                min_child_weigth=1)
  xgb=xgboost(xtrain, label = ytrain,nrounds = 500,params=params)
  conv_ss[,i] = xgb$evaluation_log$train_rmse
  pred_ss[,i] = predict(xgb, xtest)
}

conv_ss=data.frame(iter=1:500,conv_ss)
conv_ss=melt(conv_ss,id.vars = "iter")
ggplot(data=conv_ss)+geom_line(aes(x=iter,y=value,color=variable))
(RMSE_ss=sqrt(colMeans((as.numeric(ytest)-pred_ss)^2)))


#e) min_child weight:
set.seed(12754)
conv_mcw = matrix(NA,500,length(mcw))
```

```
pred_mcw = matrix(NA,nrow(test1), length(mcw))
colnames(conv_mcw) = colnames(pred_mcw) = mcw

for(i in 1:length(mcw)){
  params = list(eta = 0.1, colsample_bylevel=2/3,
                subsample = 1, max_depth = 6,
                min_child_weight = mcw[i], gamma = 0)
  xgb = xgboost(xtrain, label = ytrain, nrounds = 500, params = params)
  conv_mcw[,i] = xgb$evaluation_log$train_rmse
  pred_mcw[,i] = predict(xgb, xtest)
}

conv_mcw = data.frame(iter=1:500, conv_mcw)
conv_mcw = melt(conv_mcw, id.vars = "iter")
ggplot(data = conv_mcw) + geom_line(aes(x = iter, y = value, color = variable))
(RMSE_mcw = sqrt(colMeans((as.numeric(ytest)-pred_mcw)^2)))

#f) Gamma:
set.seed(12897564)
conv_gamma = matrix(NA,500,length(gamma))
pred_gamma = matrix(NA,nrow(test1), length(gamma))
colnames(conv_gamma) = colnames(pred_gamma) = gamma

for(i in 1:length(gamma)){
  params = list(eta = 0.1, colsample_bylevel=2/3,
                subsample = 1, max_depth = 6, min_child_weight = 1,
                gamma = gamma[i])
  xgb = xgboost(xtrain, label = ytrain, nrounds = 500, params = params)
  conv_gamma[,i] = xgb$evaluation_log$train_rmse
  pred_gamma[,i] = predict(xgb, xtest)
}

conv_gamma = data.frame(iter=1:500, conv_gamma)
conv_gamma = melt(conv_gamma, id.vars = "iter")
ggplot(data = conv_gamma) + geom_line(aes(x = iter, y = value, color = variable))
(RMSE_gamma = sqrt(colMeans((as.numeric(ytest)-pred_gamma)^2)))


#Setting the values:
xgb1 <- xgboost(data = xtrain, label = ytrain,
                nrounds = 500)

#set these values manually by looking at RMSE values:
params = list(eta = .05, colsample_bylevel = 1/3,
                subsample = 1, max_depth = 2,
                min_child_weigth = 400, gamma = 1)
params


xgb.train <- xgboost(data = xtrain, label = ytrain,
                           params = params,weight = ifelse(ytrain == 1, 5, 1),
                           nrounds = 500, objective = "binary:logistic")


xgb.test <- xgboost(data = xtest, label = ytest,
                params = params,
                nrounds = 500, objective = "binary:logistic")

# run test model
y_pred <- predict(xgb.train, xtest)
y_pred
```

```
# get MSE
test.MSE2<-mean((ytest - y_pred)^2)
test.MSE2
# get residual (if continuous outcome)
r2<-ytest - y_pred
plot(r2, ylab = "residuals", main = "XGB residuals")
qqnorm(r2)


#Plot:
#get the first three trees
xgb.plot.tree(model = xgb.train, trees = 0:2)
xgb.plot.multi.trees(xgb.train)
importance_matrix <- xgb.importance(model = xgb.train)
importance_matrix
xgb.plot.importance(importance_matrix, xlab = "Feature Importance")


#First Converting predicted probabilities to class labels
y_pred_class<- ifelse(y_pred > 0.48, 1, 0)

# Creating the confusion matrix manually:
conf_matrix_training<- table(y_pred_class, ytest)
conf_matrix_training

#Calculating the accuracy of the training model:
accuracy_training<- sum(diag(conf_matrix_training)) / sum(conf_matrix_training)
accuracy_training


#################################### XG Boost without olderAM variable
################################################

#Dropping the 'older american variable, i.e., olderAM' and re-running the XG BOOST:
attach(q9_s)

#Dropping the olderAM variable:
q9_s1<-q9_s%>%
  select(-olderAm)
view(q9_s1)


#################################### XG Boost 1 ##########################################################
#convert all character columns to factor
q9_s1<- data.frame(
  lapply(q9_s1, function(x) {
            if(is.character(x)) factor(x) else x
  })
)


#Converting year to factor:
q9_s <- q9_s %>%
  mutate(Year = factor(Year))


set.seed(27514)
split.imp1 <- sample.split(q9_s1, SplitRatio = 0.7)
train2 <-  subset(q9_s1, split.imp == "TRUE")
test2 <- subset(q9_s1, split.imp == "FALSE")
```

```
# make them numeric matrix
xtrain1<-sparse.model.matrix(relief ~. -1, data = train2)
ytrain1<- as.array(train2$relief)
xtest1<- sparse.model.matrix(relief ~ .-1, data = test2)
ytest1<- as.array(test2$relief)

### running the default train model
xgb2<- xgboost(data = xtrain1, label = ytrain1,
               nrounds = 500)

# run test model
y_pred1<- predict(xgb2, xtest1)
y_pred1

# get MSE
test.MSE1<-mean((ytest1 - y_pred1)^2)
test.MSE1

# get residual (if continuous outcome)
r1<-ytest1 - y_pred1
qqnorm(r1)

### plot
xgb.plot.multi.trees(xgb2)

#Setting hyperparameters:
#eta:
eta=c(0.05, 0.1, 0.2,0.5,1)

# = colsample_bylevel candidates = #
cs=c(1/3,2/3,1)

# = max_depth candidates = #
md=c(2,4,6,10)

# = sub_sample candidates = #
ss=c(0.25,0.5,0.75,1)

# = standard model is the second value of each vector above = #
standard=c(2,2,3,2)

# = min_child_weights candidates = #
mcw=c(1,10,100,400)

# = gamma candidates = #
gamma=c(0.1,1,10,100)

#a) ETA search:
set.seed(13856)
conv_eta1 = matrix(NA,500,length(eta))
pred_eta1 = matrix(NA,nrow(test2), length(eta))
colnames(conv_eta1) = colnames(pred_eta1) = eta

for(i in 1:length(eta)){
  params=list(eta = eta[i], colsample_bylevel=cs[standard[2]],
              subsample = ss[standard[4]], max_depth = md[standard[3]],
              min_child_weigth = 1)
  xgb=xgboost(xtrain1, label = ytrain1, nrounds = 500, params = params)
  conv_eta1[,i] = xgb$evaluation_log$train_rmse
  pred_eta1[,i] = predict(xgb, xtest1)
}
```

```
conv_eta1 = data.frame(iter=1:500, conv_eta1)
conv_eta1 = melt(conv_eta1, id.vars = "iter")
ggplot(data = conv_eta1) + geom_line(aes(x = iter, y = value, color = variable))
(RMSE_eta1 = sqrt(colMeans((as.numeric(ytest1)-pred_eta1)^2)))

#b) Colsample_bylevel:
set.seed(1284654)
conv_cs1 = matrix(NA,500,length(cs))
pred_cs1 = matrix(NA,nrow(test2), length(cs))
colnames(conv_cs1) = colnames(pred_cs1) = cs

for(i in 1:length(cs)){
  params = list(eta = eta[standard[1]], colsample_bylevel = cs[i],
                subsample = ss[standard[4]], max_depth = md[standard[3]],
                min_child_weigth = 1)
  xgb=xgboost(xtrain1, label = ytrain1,nrounds = 500, params = params)
  conv_cs1[,i] = xgb$evaluation_log$train_rmse
  pred_cs1[,i] = predict(xgb, xtest1)
}

conv_cs1 = data.frame(iter=1:500, conv_cs1)
conv_cs1 = melt(conv_cs1, id.vars = "iter")
ggplot(data = conv_cs1) + geom_line(aes(x = iter, y = value, color = variable))
(RMSE_cs1 = sqrt(colMeans((as.numeric(ytest1)-pred_cs1)^2)))

#c) Max Depth:
set.seed(1284654)
conv_md1=matrix(NA,500,length(md))
pred_md1=matrix(NA,nrow(test2),length(md))
colnames(conv_md1)=colnames(pred_md1)=md

for(i in 1:length(md)){
  params=list(eta=eta[standard[1]],colsample_bylevel=cs[standard[2]],
              subsample=ss[standard[4]],max_depth=md[i],
              min_child_weigth=1)
  xgb=xgboost(xtrain1, label = ytrain1,nrounds = 500,params=params)
  conv_md1[,i] = xgb$evaluation_log$train_rmse
  pred_md1[,i] = predict(xgb, xtest1)
}

conv_md1=data.frame(iter=1:500,conv_md1)
conv_md1=melt(conv_md1,id.vars = "iter")
ggplot(data=conv_md1)+geom_line(aes(x=iter,y=value,color=variable))
(RMSE_md1=sqrt(colMeans((as.numeric(ytest1)-pred_md1)^2)))

#d) Sub Sample:
set.seed(1)
conv_ss1=matrix(NA,500,length(ss))
pred_ss1=matrix(NA,nrow(test2),length(ss))
colnames(conv_ss1)=colnames(pred_ss1)=ss

for(i in 1:length(ss)){
  params=list(eta=eta[standard[1]],colsample_bylevel=cs[standard[2]],
              subsample=ss[i],max_depth=md[standard[3]],
              min_child_weigth=1)
  xgb=xgboost(xtrain1, label = ytrain1,nrounds = 500,params=params)
  conv_ss1[,i] = xgb$evaluation_log$train_rmse
  pred_ss1[,i] = predict(xgb, xtest1)
}
```

```
conv_ss1=data.frame(iter=1:500,conv_ss1)
conv_ss1=melt(conv_ss1,id.vars = "iter")
ggplot(data=conv_ss1)+geom_line(aes(x=iter,y=value,color=variable))
(RMSE_ss1=sqrt(colMeans((as.numeric(ytest1)-pred_ss1)^2)))

#e) min_child weight:
set.seed(12754)
conv_mcw1= matrix(NA,500,length(mcw))
pred_mcw1= matrix(NA,nrow(test2), length(mcw))
colnames(conv_mcw1) = colnames(pred_mcw1) = mcw

for(i in 1:length(mcw)){
  params = list(eta = 0.1, colsample_bylevel=2/3,
                subsample = 1, max_depth = 6,
                min_child_weight = mcw[i], gamma = 0)
  xgb = xgboost(xtrain1, label = ytrain1, nrounds = 500, params = params)
  conv_mcw1[,i] = xgb$evaluation_log$train_rmse
  pred_mcw1[,i] = predict(xgb, xtest1)
}

conv_mcw1= data.frame(iter=1:500, conv_mcw1)
conv_mcw1= melt(conv_mcw1, id.vars = "iter")
ggplot(data = conv_mcw1) + geom_line(aes(x = iter, y = value, color = variable))
(RMSE_mcw1 = sqrt(colMeans((as.numeric(ytest1)-pred_mcw1)^2)))

#f) Gamma:
set.seed(12897564)
conv_gamma1= matrix(NA,500,length(gamma))
pred_gamma1= matrix(NA,nrow(test2), length(gamma))
colnames(conv_gamma1) = colnames(pred_gamma1) = gamma

for(i in 1:length(gamma)){
  params = list(eta = 0.1, colsample_bylevel=2/3,
                subsample = 1, max_depth = 6, min_child_weight = 1,
                gamma = gamma[i])
  xgb = xgboost(xtrain1, label = ytrain1, nrounds = 500, params = params)
  conv_gamma1[,i] = xgb$evaluation_log$train_rmse
  pred_gamma1[,i] = predict(xgb, xtest1)
}

conv_gamma1 = data.frame(iter=1:500, conv_gamma1)
conv_gamma1 = melt(conv_gamma1, id.vars = "iter")
ggplot(data = conv_gamma1) + geom_line(aes(x = iter, y = value, color = variable))
(RMSE_gamma1 = sqrt(colMeans((as.numeric(ytest1)-pred_gamma1)^2)))


#Setting the values:
xgb2<- xgboost(data = xtrain1, label = ytrain1,
               nrounds = 500)

#set these values manually by looking at RMSE values:
params1 = list(eta = .05, colsample_bylevel = 1/3,
               subsample = 1 , max_depth = 2,
               min_child_weigth = 400, gamma = 1)
params1

xgb.train1 <- xgboost(data = xtrain1, label = ytrain1,
                      params = params1,weight = ifelse(ytrain1 == 1, 5, 1),
                      nrounds = 500, objective = "binary:logistic")

xgb.test1 <- xgboost(data = xtest1, label = ytest1,
```

```
                    params = params1,
                    nrounds = 500, objective = "binary:logistic")

# run test model
y_pred1.1 <- predict(xgb.train1, xtest1)
y_pred1.1
# get MSE
test.MSE3<-mean((ytest1 - y_pred1.1)^2)
test.MSE3
# get residual (if continuous outcome)
r3<-ytest1 - y_pred1.1
plot(r3, ylab = "residuals", main = "XGB residuals")
qqnorm(r3)

#Feature importance plots:
#get the first three trees
xgb.plot.tree(model = xgb.train1, trees = 0:2)
xgb.plot.multi.trees(xgb.train1)
importance_matrix1 <- xgb.importance(model = xgb.train1)
importance_matrix1
xgb.plot.importance(importance_matrix1, xlab = "Feature Importance")


######################################## Neural Network ----
nn.pred = predict(nn.model, newdata = nn.test.data)

nn.prec = vector('double')
nn.recall = vector('double')

nn.threshold = seq(0.01,0.9,0.01)

best.threshold = 1
best.f1 = 0
for(i in 1:length(nn.threshold))
{
 print(i)
 cm = table(as.vector(nn.test.data[,1]),
          ifelse(nn.pred>=nn.threshold[i],1,0))
 if(dim(cm)[2] == 2)
 {
          nn.prec = c(nn.prec,cm[2,2] / (cm[2,2] + cm[1,2]))
          nn.recall = c(nn.recall,cm[2,2] / (cm[2,2] + cm[2,1]))
 }
 else{
          nn.prec = c(nn.prec,0)
          nn.recall = c(nn.recall,0)
 }
 temp.f1 = 2 * nn.prec[i] * nn.recall[i] / (nn.prec[i] + nn.recall[i])
 print(temp.f1)
 if(!is.nan(temp.f1))
 {
          if(temp.f1 > best.f1)
          {
          best.f1 = temp.f1
          best.threshold = nn.threshold[i]
          }
 }
}
plot(nn.recall,nn.prec)

cm.best = table(as.vector(nn.test.data[,1]),
```

```
        ifelse(nn.pred >= best.threshold,1,0))

cm.best.df = as.data.frame(cm.best)

setwd("C:/Users/18045/Documents/R/ML/ML Project") #Sean's WD
#setwd("") #Nirav's WD
#setwd("") #Elise's WD
#setwd("") #Isaac's WD
#setwd("") #Truman's WD
#install.packages("clustMixType", "wesanderson")
library(readr)
library(data.table)
library(readxl)
library(tidyverse)
library(formattable)
library(dplyr)
library(plyr)
library(tidyr)
library(ggplot2)
library(cowplot)
library(scales)
library(corrplot)
#install.packages('clustMixType')
library(clustMixType)
#install.packages('wesanderson')
library(wesanderson)
#install.packages("zipcodeR")
library(zipcodeR)
#install.packages('stringr')
library(stringr)
library(glmnet)
library(caret)
library(caTools)
library(xgboost)
library(Matrix)

#Loading The Data----

testgroup3 <- read_csv("test_all.csv", col_types = cols(...1 = col_skip(),
                                                        X = col_skip()))

county_demos <- read_csv("county_demos.csv", col_types = cols(...1 = col_skip()))

countydebt <- read_excel("dia_lbls_all_overall_county_2022_02_14Sep2023.xlsx")

fips_data <- read_csv("zip_fips.csv", col_types = cols(...1 = col_skip(),
                                                       STCOUNTYFP = col_number()))

#Isolating the Dependent Variable-----

testgroup3$relief<- ifelse(testgroup3$Company.response.to.consumer %in% c("Closed with monetary relief",
                                                        "Closed with relief",
                                                        "Closed with non-monetary relief"),
                           1,0)

#Cleaning-----

#drop non states
testgroup3$drop <- as.numeric(testgroup3$State %in% c("NONE", "None", "DC", "AA", "AS", "AP", "AE", "FM","GU", "MH", "MP",
"PR", "VI", "UNITED STATES MINOR OUTLYING ISLANDS"))
testgroup3 <- subset(testgroup3, !(State %in% c("AE", 'AP'))) #dropping military bases
```

```
fips_data <- fips_data[!fips_data$STATE %in% c("NONE", "None", "DC", "AA", "AS", 'AP', "AE", "FM","GU", "MH", "MP", "PR", "VI",
"UNITED STATES MINOR OUTLYING ISLANDS"),]

ctfips <-  (ifelse((fips_data$STATE == 'CT' & str_detect(fips_data$STCOUNTYFP, "9")),
            as.numeric(paste0("01", fips_data$STCOUNTYFP)), fips_data$STCOUNTYFP))

fips_data$STCOUNTYFP <- ctfips

countydebt$`County FIPS` <- as.numeric(countydebt$`County FIPS`)

#Named Factors ----
testgroup3 <- testgroup3 %>%
  mutate(Company.response.to.consumer = case_when(Company.response.to.consumer == "In progress" ~ 0,
                                    Company.response.to.consumer == "Closed with explanation" ~ 1,
                                    Company.response.to.consumer == "Untimely response" ~ 2,
                                    Company.response.to.consumer == "Closed with non-monetary relief" ~ 3,
                                    Company.response.to.consumer == "Closed with monetary relief" ~ 4,
                                    Company.response.to.consumer == "Closed" ~ 5),
            Sub.product = case_when(Sub.product == "Other debt" ~ "Misc. debt",
                      Sub.product == 'Other (i.e. phone, health club, etc.)' ~ "Misc. debt",
                      Sub.product == "Telecommunications debt" ~ "Misc. debt",
                      Sub.product == 'I do not know' ~ "Misc. debt",
                      Sub.product == "Federal student loan debt" ~ "Federal Student Loans",
                      Sub.product == "Federal student loan"   ~ "Federal Student Loans",
                      Sub.product == "Private student loan debt" ~ "Non-Federal Student Loans",
                      Sub.product == "Non-federal student loan"  ~ "Non-Federal Student Loans",
                      Sub.product == "Auto" ~ "Auto debt",
                      Sub.product == "Auto debt" ~ "Auto debt",
                      Sub.product == "Credit card debt" ~ "Credit card debt",
                      Sub.product == "Credit card" ~ "Credit card debt",
                      Sub.product == 'Mortgage debt'~ "Home debt", #fixed a typo in mortgage (by Nirav).
                      Sub.product == "Mortgage" ~ "Home debt",
                      Sub.product == 'Medical' ~ "Medical debt",
                      Sub.product == "Medical debt" ~ "Medical debt",
                      Sub.product == "Rental debt" ~ "Rental debt",
                      Sub.product == "Payday loan debt" ~ "Payday debt",
                      Sub.product == "Payday loan" ~ "Payday debt"),
            Issue = case_when(Issue == "Attempts to collect debt not owed" ~ "Collect Debt Not Owed",
                      Issue == 'Cont\'d attempts collect debt not owed' ~ "Collect Debt Not Owed",
                      Issue == "Took or threatened to take negative or legal action" ~ "Aggressive Response",
                      Issue == 'Taking/threatening an illegal action' ~ "Aggressive Response",
                      Issue == "Threatened to contact someone or share information improperly" ~ "Improper Communication Tactic",
                      Issue == "Communication tactics" ~ "Improper Communication Tactic",
                      Issue == "False statements or representation" ~ "Improper Communication Tactic",
                      Issue == "Disclosure verification of debt" ~ "Improper Communication Tactic",
                      Issue == "Improper contact or sharing of info" ~ "Improper Communication Tactic",
                      Issue == "Written notification about debt" ~ "Debt Notifications",
                      Issue == "Electronic communications"   ~ "Debt Notifications"),
            Company.public.response = case_when(Company.public.response == "Company has responded to the consumer and the
CFPB and chooses not to provide a public response" ~ "No Comment",
                                    Company.public.response == "Company chooses not to provide a public response" ~ "No Comment",
                                    Company.public.response == "Company believes complaint represents an opportunity for
improvement to better serve consumers" ~ "Improve Service",
                                    Company.public.response == "Company believes the complaint provided an opportunity to answer
consumer's questions" ~ "Improve Service",
                                    Company.public.response == "Company believes the complaint is the result of a misunderstanding" ~
"Misunderstanding",
                                    Company.public.response == "Company believes complaint is the result of an isolated error" ~
"Misunderstanding"),
            Consumer.consent.provided. = case_when(Consumer.consent.provided. == "Consent provided" ~ "Consent provided",
```

```
                                Consumer.consent.provided. == "None"  ~ "No consent provided",
                                Consumer.consent.provided. == "Consent not provided"  ~ "No consent provided",
                                Consumer.consent.provided. == "Other"  ~ "No consent provided",
                                Consumer.consent.provided. == "Consent withdrawn"   ~ "No consent provided",
                                Consumer.consent.provided. == "N/A" ~ "No consent provided"),
            Submitted.via = case_when(Submitted.via == 'Email' ~ 'Web',
                        Submitted.via == 'Phone' ~ 'Phone',
                        Submitted.via == 'Postal mail' ~ 'Postal mail',
                        Submitted.via == 'Referral' ~ 'Referral',
                        Submitted.via == 'Web' ~ 'Web',
                        Submitted.via == 'Web Referral' ~ 'Web Referral',
                        Submitted.via == 'Fax' ~ 'Fax'))

#Dummy for after date (form changes NA imputation)
testgroup3$Date.received <- as.Date(testgroup3$Date.received, format = "%m/%d/%y")
testgroup3$Dispute_prior <- ifelse(testgroup3$Date.received  > as.Date('04/24/17', format= "%m/%d/%y"), 1,0)

#older Americans and servicefolk
testgroup3$servicemember <- ifelse(str_detect(testgroup3$Tags, "Servicemember"), 1, 0)
testgroup3$olderAm <- ifelse(str_detect(testgroup3$Tags, "Older American"), 1, 0)

#####Zip Code Cleaning#### -----
USA_zippop <- zip_code_db

#Clean Zip Reference data first ----

fips_data$ZIP <- as.character(fips_data$ZIP)
testgroup3$ZIP.code <- as.character(testgroup3$ZIP.code)

#check zips including numbers
table(str_detect( as.character(fips_data$ZIP), "[0-9]+$")) #looks like they're all numbers

#check zips less than 5 characters
table(nchar(fips_data$ZIP) < 5) #2900 entries less than 5 digits - leading zeros?
invalidzip <- fips_data$ZIP[nchar(fips_data$ZIP) < 5]

table(nchar(fips_data$ZIP) > 5) #No entries greater than 5 digits

#Check leading zeros
table(grepl("^0", testgroup3$ZIP.code)) #looks like they already have leading zeros

#LEADING Zero Code replacement
#Place a leading zero for the problem zips
zip <- as.numeric(fips_data$ZIP)
for(i in 1:length(zip)){
  if(nchar(as.numeric(zip[i])) < 5){
            zip[i] <- paste0("0", zip[i])
  }
}

fips_data$ZIP <- (zip)
table(nchar(fips_data$ZIP) < 5 ) #incorrect zip identification

error_zip <- fips_data[nchar(fips_data$ZIP) < 5, ]

#manual fix
fips_data$ZIP[fips_data$ZIP == error_zip$ZIP] <- '00501'

#Group3 Clean Zips -----

#check zips including numbers
```

```
table(str_detect( as.character(testgroup3$ZIP.code), "[0-9]+$")) #looks like they're all numbers

#check zips less than 5 characters
table(nchar(testgroup3$ZIP.code) < 5) #No entries less than 5 digits
table(nchar(testgroup3$ZIP.code) > 5) #No entries greater than 5 digits

#Check leading zeros
table(grepl("^0", testgroup3$ZIP.code)) #looks like they already have leading zeros

unique_zips <- unique(testgroup3$ZIP.code)

zip_binary_map <- unique_zips %in% fips_data$ZIP

table(zip_binary_map)

error_zips <- unique_zips[!zip_binary_map] #erroneous zips

#T.test of error zips -----
t.test(table(zip_binary_map), alternative = 'two.sided') #fail to reject the null - the error zips are not significant at 5% level

for (error_state in unique(testgroup3$State[testgroup3$ZIP.code %in% error_zips])) {
  mode_zip <- names(sort(table(testgroup3$ZIP.code[testgroup3$State == error_state]), decreasing = TRUE)[1])
  testgroup3$ZIP.code[testgroup3$ZIP.code %in% error_zips & testgroup3$State == error_state] <- mode_zip
}

#LEADING Zero Code replacement for the problem zips
#zip <- as.character(group3$ZIP.code[zip_binary_map == FALSE])
#for(i in 1:length(zip)){
#  if(nchar(as.numeric(zip[i])) < 5){
#             zip[i] <- paste0("0", zip[i])
#  }
#}

#insert zips in data

#Percusive Maintainence ---
# run the loop until there are no issues

error_zips <- unique_zips[!zip_binary_map] #82 erroneous zips
table(zip_binary_map)

for (error_state in unique(testgroup3$State[testgroup3$ZIP.code %in% error_zips])) {
  mode_zip <- names(sort(table(testgroup3$ZIP.code[testgroup3$State == error_state]), decreasing = TRUE)[1])
  group3$ZIP.code[testgroup3$ZIP.code %in% error_zips & testgroup3$State == error_state] <- mode_zip
}

#Check if this worked
unique_zips <- unique(testgroup3$ZIP.code)
zip_binary_map <- unique_zips %in% fips_data$ZIP
table(zip_binary_map) #looks like there are no issues anymore

#rm(i, zip, zip_binary_map, unique_zips, unclean_zips) #remove these variables when done.

#FIPS Fix ------

state_data <- data.frame(
  Postal_Abbr = c(
             "AL", "AK", "AZ", "AR", "CA", "CO", "CT", "DE", "FL", "GA", "HI", "ID", "IL", "IN", "IA", "KS", "KY", "LA",
             "ME", "MD", "MA", "MI", "MN", "MS", "MO", "MT", "NE", "NV", "NH", "NJ", "NM", "NY", "NC", "ND", "OH", "OK", "OR",
             "PA", "RI", "SC", "SD", "TN", "TX", "UT", "VT", "VA", "WA", "WV", "WI", "WY"
  ),
```

```
  FIPS_Code = c(
            "0", "0", "0", "0", "0", "0", "0", "10", "12", "13", "15", "16", "17", "18", "19", "20", "21", "22", "23",
            "24", "25", "26", "27", "28", "29", "30", "31", "32", "33", "34", "35", "36", "37", "38", "39", "40", "41", "42",
            "44", "45", "46", "47", "48", "49", "50", "51", "53", "54", "55", "56"
  )
)

countydebt$`County FIPS` <- as.character(paste0("0", countydebt$`County FIPS`))

fips_data$STCOUNTYFP <- as.character(paste0("0", fips_data$STCOUNTYFP))

#Merges ----

merg_zips <- merge(testgroup3, fips_data, by.x =  'ZIP.code', by.y = 'ZIP')

merg_fips <- merge(merg_zips, countydebt,  by.x = 'STCOUNTYFP', by.y = 'County FIPS')

cleantest <- merge(merg_fips, county_demos,  by.x = 'STCOUNTYFP', by.y = 'Fips')

#PCA ----

colnames(cleantest)

temp <- cleantest[, c(32:52)]
temp <- sapply(temp, as.numeric)

imputation_map_pca_val <- is.na(temp) #map of our imputations

temp_imputed <- apply(temp, 2, function(x) {
  x[is.na(x)] <- mean(x, na.rm = TRUE)
  return(x)
})

#### PCA
library(ggcorrplot)
library("FactoMineR")
library(factoextra)

corr_matrix <- cor(temp_imputed)
corr_matrix <- scale(corr_matrix)
ggcorrplot(corr_matrix)

## do PCA
debt.pca <- princomp(corr_matrix)
summary(debt.pca)
fviz_eig(debt.pca, addlabels = TRUE)

comps <- debt.pca$scores[,1:4]

# loadings for first 5 components
debt.pca$loadings[, 1:4]

#scree plot
variance_explained <- debt.pca$sd^2 / sum(debt.pca$sd^2)*100

variance_explained[1:7]

qplot(c(1:21), variance_explained) +
  geom_line() +
  xlab("Principal Component") +
  ylab("Variance Explained") +
```

```
  ggtitle("Scree Plot") +
  ylim(0, 100)


#get 4 components
feature_vector <- (debt.pca$loadings)
feature_vector_transposed <- t(feature_vector)

original_data_transposed <- t(temp_imputed)

dim(original_data_transposed)
dim(feature_vector)

data_reoriented <- feature_vector_transposed %*% original_data_transposed

data_reoriented_df <- as.data.frame(data_reoriented)
data_reoriented_df <- t(data_reoriented_df)

comps <- data_reoriented_df[,c(1:4)]

comps <- as.data.frame(comps)

#now merge to the dataset
temp2 <- cbind(cleantest[, -c(29:52)], comps)

colnames(temp2)

write.csv(temp2, "NWA_test.csv")

#### Clustering code ------
# make a separate dataset, and then make sure each variable is the right class.
set.seed(23748234)

NWA_test <- read_csv("NWA_test.csv", col_types = cols(...1 = col_skip()))

ca <- NWA_test[, c("Sub.product",
                   "Issue",
                   "Pop_over64",
                   "TotalFemale",
                   "Average household income, Comm of color",
                   "Share of people of color")]

ca[,1] <- as.factor(ca[,1] )
ca[,2] <- as.factor(ca[,2] )
ca[,3] <- scale(as.numeric(ca[,3] ))
ca[,4] <- scale(as.numeric(ca[,4] ))
ca[,5] <- scale(as.numeric(ca[,5] ))
ca[,6] <- scale(as.numeric(ca[,6] ))

colnames(ca) <- c("Sub.product",
                  "Issue",
                  "Pop_over64",
                  "TotalFemale",
                  "Average.household.income..Comm.of.color",
                  "Share.of.people.of.color")

ca$medicaldebt <- as.factor(ifelse(ca$Sub.product == "Medical debt", 1,0))

impute_map_medical_val <- is.na(ca) #imputation map for medical clusters

# 2 cluster
```

```
kpres2 <- kproto(x=ca[,c(2:7)], k=2, na.rm = 'imp.internal')

# 3 cluster
kpres3 <- kproto(x=ca[,c(2:7)], k=3,  na.rm = 'imp.internal')

# Plots
clprofiles(kpres3, ca[,c(2:7)], col = wes_palette("Royal1", 4, type = "continuous"))

# scree plot

n.scree<-ncol(ca)-2
Es<- numeric(n.scree)
for(i in 1:n.scree){
  kpres <- kproto(ca,k=i,nstart=5, verbose = FALSE)
  Es[i] <- kpres$tot.withinss
}

plot(1:n.scree, Es[1:5], type="b", ylab="ObjectiveFunction",
             xlab="#Clusters", main="ScreePlot") #figure2

NWA_test$MedicalDebtClusters <- kpres3$cluster

colnames(NWA_test)

write.csv(NWA_test, 'q9_test.csv')

#Q9 -----

q9_test <- read_csv("q9_test.csv", col_types = cols(...1 = col_skip()))
colnames(q9_test)
#rename to match
q9_test$Fips <- q9_test$STCOUNTYFP

q9_test$`Share.of.people.of.color` <- q9_test$`Share of people of color`
q9_test$`Average.household.income..All` <- q9_test$`Average household income, All`
q9_test$`Average.household.income..Comm.of.color` <- q9_test$`Average household income, Comm of color`
q9_test$`Average.household.income..White.comm` <- q9_test$`Average household income, White comm`

library(lubridate)

q9_test$Year <- as.factor(year(q9_test$Date.received))

q9_test <- q9_test %>%
  select(Sub.product, Issue, Sub.issue, Consumer.consent.provided., Submitted.via, Timely.response. , relief, drop,
             Dispute_prior, servicemember, olderAm,
             Fips, Pop_less25, Pop_over64, Pop_Hispanic, White, Black, Asian, Indigenous, Native, Multiple, TotalMale, TotalFemale,
             Share.of.people.of.color, Average.household.income..All, Average.household.income..Comm.of.color,
Average.household.income..White.comm,
             Comp.1, Comp.2,Comp.3, Comp.4, MedicalDebtClusters,  Year)

table(q9_test$Year)

#Transformations -------

#make factors
q9_s_test <- data.frame(
  lapply(q9_test, function(x) {
             if(is.character(x)) factor(x) else x
  })
)
```

```
q9_s_test <- q9_s_test %>%
  filter(complete.cases(.))

write.csv(q9_s_test, "q9_s_test.csv")


################################################## Validation XG BOOST
##################################################
q9_s_test<-read.csv("q9_s_test.csv")
view(q9_s_test)
attach(q9_s_test)


#rec-coding email character:
q9_s_test<- q9_s_test %>%
  mutate( Submitted.via = case_when(Submitted.via == 'Email' ~ 'Web',
                            Submitted.via == 'Phone' ~ 'Phone',
                            Submitted.via == 'Postal mail' ~ 'Postal mail',
                            Submitted.via == 'Referral' ~ 'Referral',
                            Submitted.via == 'Web' ~ 'Web',
                            Submitted.via == 'Fax' ~ 'Fax',
                            Submitted.via == 'Web Referral' ~ 'Web Referral'))


#Dropping the index (X) variable:
q9_s_test<-q9_s_test%>%
  select(-X)

view(q9_s_test)

#Converting year to factor:
q9_s_test<- q9_s_test %>%
  mutate(Year = factor(Year))

#making them factors:
q9_s_test<- data.frame(
  lapply(q9_s_test, function(x) {
            if(is.character(x)) factor(x) else x
  })
)

set.seed(27514)

#making them numeric matrix
xtrain2<-sparse.model.matrix(relief ~. -1, data = q9_s_test)
ytrain2<- as.array(q9_s_test$relief)


### run the default test XGBOOST model;
xgb3<- xgboost(data = xtrain2, label = ytrain2,
            nrounds = 500)

#parameters coming from training data:
params = list(eta = .05, colsample_bylevel = 1/3,
            subsample = 1, max_depth = 2,
            min_child_weigth = 400, gamma = 1)
params

# run test model
y_pred_test<- predict(xgb.train, xtrain2)
y_pred_test
```

```
#First Converting predicted probabilities to class labels
y_pred_class1<- ifelse(y_pred_test > 0.5, 1, 0)


# Creating the confusion matrix manually:
conf_matrix_test<- table(y_pred_class1, ytrain2)
conf_matrix_test


#Calculating the accuracy of the training model:
accuracy_test<- sum(diag(conf_matrix_test)) / sum(conf_matrix_test)
accuracy_test
```