Analysis will start with uploading and cleaning of the data.

#this upload is based off of a pc's directory, be sure to change it to your specific one.  
  
campaigns <- read.csv('C:/Users/nicky/Desktop/Syracuse Grad School/IST 707/Final Project/CandidateSummaryAction.csv')

#data set needs to be cleaned of data that is not needed:  
#str(campaigns)

Unneeded columns were dropped below

campaigns <- subset(campaigns, select = -c(can\_off\_dis, tot\_rec, tot\_dis, cas\_on\_han\_beg\_of\_per,  
 cas\_on\_han\_clo\_of\_per, deb\_owe\_by\_com, deb\_owe\_to\_com,  
 cov\_sta\_dat, cov\_end\_dat, can\_str1, can\_str2, can\_cit,  
 can\_zip, off\_to\_ope\_exp, off\_to\_fun, off\_to\_leg\_acc,  
 exe\_leg\_acc\_dis, fun\_dis, can\_loa\_rep, oth\_loa\_rep,  
 tot\_loa\_rep, ind\_ref, par\_com\_ref, oth\_com\_ref,  
 tot\_con\_ref, oth\_dis, tot\_dis))

#str(campaigns)

Next, to check for null values:

colSums(is.na(campaigns))

## can\_id can\_nam can\_off can\_off\_sta   
## 0 0 0 0   
## can\_par\_aff can\_inc\_cha\_ope\_sea can\_sta ind\_ite\_con   
## 0 0 0 0   
## ind\_uni\_con ind\_con par\_com\_con oth\_com\_con   
## 0 0 0 0   
## can\_con tot\_con tra\_fro\_oth\_aut\_com can\_loa   
## 0 0 0 0   
## oth\_loa tot\_loa oth\_rec ope\_exp   
## 0 0 0 0   
## tra\_to\_oth\_aut\_com net\_con net\_ope\_exp winner   
## 0 0 0 0   
## votes   
## 1435

Votes data is unreliable and will have to be removed as a column. Author of data set said the following: ’the votes column was not part of the metadata. I had to scrape that info manually from CNN’s election results page here:

<https://www.cnn.com/election/2016/results/house>

My apologies for any errors on my part, there is definitely a chance for human error. In which districts have you found discrepancies?’

campaigns <- subset(campaigns, select = -c(votes))

#saving data into a new csv  
  
write.csv(campaigns,'C:/Users/nicky/Desktop/Syracuse Grad School/IST 707/Final Project/campaigns\_new.csv')

#id isn't needed neither  
campaigns <- subset(campaigns, select = -c(can\_id))

campaigns <- subset(campaigns, select = -c(can\_nam))

Need to change data types

str(campaigns)

## 'data.frame': 1814 obs. of 22 variables:  
## $ can\_off : chr "H" "H" "H" "H" ...  
## $ can\_off\_sta : chr "GA" "PA" "FL" "MT" ...  
## $ can\_par\_aff : chr "REP" "DEM" "REP" "REP" ...  
## $ can\_inc\_cha\_ope\_sea: chr "INCUMBENT" "CHALLENGER" "OPEN" "INCUMBENT" ...  
## $ can\_sta : chr "GA" "PA" "FL" "MT" ...  
## $ ind\_ite\_con : chr "$554,305.00 " "$1,042,280.38 " "$529,030.38 " "$2,479,616.45 " ...  
## $ ind\_uni\_con : chr "$46,969.50 " "$72,430.64 " "$13,075.00 " "$1,837,715.13 " ...  
## $ ind\_con : chr "$601,274.50 " "$1,114,711.02 " "$542,105.38 " "$4,317,331.58 " ...  
## $ par\_com\_con : chr "" "" "" "$3,545.32 " ...  
## $ oth\_com\_con : chr "$473,675.00 " "$302,834.20 " "$106,050.00 " "$660,038.51 " ...  
## $ can\_con : chr "" "" "$2,700.00 " "" ...  
## $ tot\_con : chr "$1,074,949.50 " "$1,417,545.22 " "$650,855.38 " "$4,980,915.41 " ...  
## $ tra\_fro\_oth\_aut\_com: chr "$17,710.49 " "" "" "$136,894.00 " ...  
## $ can\_loa : chr "" "" "$60,000.00 " "" ...  
## $ oth\_loa : chr "" "" "" "" ...  
## $ tot\_loa : chr "" "" "$60,000.00 " "" ...  
## $ oth\_rec : chr "" "" "" "$55,910.19 " ...  
## $ ope\_exp : chr "$908,518.98 " "$1,300,557.53 " "$656,642.76 " "$5,073,110.33 " ...  
## $ tra\_to\_oth\_aut\_com : chr "" "" "" "" ...  
## $ net\_con : chr "$1,074,949.50 " "$1,406,719.06 " "$650,855.38 " "$4,938,943.74 " ...  
## $ net\_ope\_exp : chr "$907,156.21 " "$1,298,831.83 " "$656,210.29 " "$5,055,942.15 " ...  
## $ winner : chr "Y" "Y" "Y" "Y" ...

before turning everything numeric for money attributes, need to get rid of dollar signs and commas.

campaigns$tot\_con <- as.numeric(gsub(",", "", gsub("\\$", "", campaigns$tot\_con)))  
campaigns$ind\_ite\_con <- as.numeric(gsub(",", "", gsub("\\$", "", campaigns$ind\_ite\_con)))  
campaigns$ind\_uni\_con <- as.numeric(gsub(",", "", gsub("\\$", "", campaigns$ind\_uni\_con)))

## Warning: NAs introduced by coercion

campaigns$ind\_con <- as.numeric(gsub(",", "", gsub("\\$", "", campaigns$ind\_con)))  
campaigns$par\_com\_con <- as.numeric(gsub(",", "", gsub("\\$", "", campaigns$par\_com\_con)))  
campaigns$oth\_com\_con <- as.numeric(gsub(",", "", gsub("\\$", "", campaigns$oth\_com\_con)))  
campaigns$can\_con <- as.numeric(gsub(",", "", gsub("\\$", "", campaigns$can\_con)))  
campaigns$tra\_fro\_oth\_aut\_com <- as.numeric(gsub(",", "", gsub("\\$", "", campaigns$tra\_fro\_oth\_aut\_com)))  
campaigns$can\_loa <- as.numeric(gsub(",", "", gsub("\\$", "", campaigns$can\_loa)))  
campaigns$oth\_loa <- as.numeric(gsub(",", "", gsub("\\$", "", campaigns$oth\_loa)))  
campaigns$oth\_rec <- as.numeric(gsub(",", "", gsub("\\$", "", campaigns$oth\_rec)))

## Warning: NAs introduced by coercion

campaigns$ope\_exp <- as.numeric(gsub(",", "", gsub("\\$", "", campaigns$ope\_exp)))

## Warning: NAs introduced by coercion

campaigns$tra\_to\_oth\_aut\_com <- as.numeric(gsub(",", "", gsub("\\$", "", campaigns$tra\_to\_oth\_aut\_com)))  
campaigns$net\_con <- as.numeric(gsub(",", "", gsub("\\$", "", campaigns$net\_con)))

## Warning: NAs introduced by coercion

campaigns$net\_ope\_exp <- as.numeric(gsub(",", "", gsub("\\$", "", campaigns$net\_ope\_exp)))  
campaigns$tot\_loa <- as.numeric(gsub(",", "", gsub("\\$", "", campaigns$tot\_loa)))

campaigns$can\_off <- as.factor(campaigns$can\_off)  
campaigns$can\_off\_sta <- as.factor(campaigns$can\_off\_sta)  
campaigns$can\_par\_aff <- as.factor(campaigns$can\_par\_aff)  
campaigns$can\_inc\_cha\_ope\_sea <- as.factor(campaigns$can\_inc\_cha\_ope\_sea)  
campaigns$can\_sta <- as.factor(campaigns$can\_sta)  
  
campaigns$winner <- as.factor(campaigns$winner)

NA’s have been now introduced. So this should be fixed.

colSums(is.na(campaigns))

## can\_off can\_off\_sta can\_par\_aff can\_inc\_cha\_ope\_sea   
## 0 0 0 0   
## can\_sta ind\_ite\_con ind\_uni\_con ind\_con   
## 0 244 277 198   
## par\_com\_con oth\_com\_con can\_con tot\_con   
## 1432 803 1139 119   
## tra\_fro\_oth\_aut\_com can\_loa oth\_loa tot\_loa   
## 1553 1174 1747 1161   
## oth\_rec ope\_exp tra\_to\_oth\_aut\_com net\_con   
## 1284 98 1661 173   
## net\_ope\_exp winner   
## 149 0

All the na’s are from null values in the money columns. Turning all of them to zero makes sense.

campaigns[is.na(campaigns)] <- 0

str(campaigns)

## 'data.frame': 1814 obs. of 22 variables:  
## $ can\_off : Factor w/ 3 levels "H","P","S": 1 1 1 1 1 1 1 1 1 1 ...  
## $ can\_off\_sta : Factor w/ 57 levels "AK","AL","AR",..: 12 42 11 30 6 31 55 22 41 22 ...  
## $ can\_par\_aff : Factor w/ 25 levels "AMP","CON","CST",..: 21 4 21 21 4 4 4 4 21 4 ...  
## $ can\_inc\_cha\_ope\_sea: Factor w/ 4 levels "","CHALLENGER",..: 3 2 4 3 3 3 3 3 3 3 ...  
## $ can\_sta : Factor w/ 58 levels "","AK","AL","AR",..: 13 43 12 31 7 32 55 23 42 23 ...  
## $ ind\_ite\_con : num 554305 1042280 529030 2479616 746234 ...  
## $ ind\_uni\_con : num 46970 72431 13075 1837715 150890 ...  
## $ ind\_con : num 601275 1114711 542105 4317332 897124 ...  
## $ par\_com\_con : num 0 0 0 3545 0 ...  
## $ oth\_com\_con : num 473675 302834 106050 660039 308740 ...  
## $ can\_con : num 0 0 2700 0 0 0 0 0 0 0 ...  
## $ tot\_con : num 1074950 1417545 650855 4980915 1205864 ...  
## $ tra\_fro\_oth\_aut\_com: num 17710 0 0 136894 0 ...  
## $ can\_loa : num 0 0 60000 0 0 0 0 0 0 0 ...  
## $ oth\_loa : num 0 0 0 0 0 0 0 0 0 0 ...  
## $ tot\_loa : num 0 0 60000 0 0 0 0 0 0 0 ...  
## $ oth\_rec : num 0 0 0 55910 0 ...  
## $ ope\_exp : num 908519 1300558 656643 5073110 953437 ...  
## $ tra\_to\_oth\_aut\_com : num 0 0 0 0 0 ...  
## $ net\_con : num 1074950 1406719 650855 4938944 1197677 ...  
## $ net\_ope\_exp : num 907156 1298832 656210 5055942 949489 ...  
## $ winner : Factor w/ 2 levels "","Y": 2 2 2 2 2 2 2 2 2 2 ...

sum(campaigns$tot\_con)

## [1] 2364931317

str(campaigns)

## 'data.frame': 1814 obs. of 22 variables:  
## $ can\_off : Factor w/ 3 levels "H","P","S": 1 1 1 1 1 1 1 1 1 1 ...  
## $ can\_off\_sta : Factor w/ 57 levels "AK","AL","AR",..: 12 42 11 30 6 31 55 22 41 22 ...  
## $ can\_par\_aff : Factor w/ 25 levels "AMP","CON","CST",..: 21 4 21 21 4 4 4 4 21 4 ...  
## $ can\_inc\_cha\_ope\_sea: Factor w/ 4 levels "","CHALLENGER",..: 3 2 4 3 3 3 3 3 3 3 ...  
## $ can\_sta : Factor w/ 58 levels "","AK","AL","AR",..: 13 43 12 31 7 32 55 23 42 23 ...  
## $ ind\_ite\_con : num 554305 1042280 529030 2479616 746234 ...  
## $ ind\_uni\_con : num 46970 72431 13075 1837715 150890 ...  
## $ ind\_con : num 601275 1114711 542105 4317332 897124 ...  
## $ par\_com\_con : num 0 0 0 3545 0 ...  
## $ oth\_com\_con : num 473675 302834 106050 660039 308740 ...  
## $ can\_con : num 0 0 2700 0 0 0 0 0 0 0 ...  
## $ tot\_con : num 1074950 1417545 650855 4980915 1205864 ...  
## $ tra\_fro\_oth\_aut\_com: num 17710 0 0 136894 0 ...  
## $ can\_loa : num 0 0 60000 0 0 0 0 0 0 0 ...  
## $ oth\_loa : num 0 0 0 0 0 0 0 0 0 0 ...  
## $ tot\_loa : num 0 0 60000 0 0 0 0 0 0 0 ...  
## $ oth\_rec : num 0 0 0 55910 0 ...  
## $ ope\_exp : num 908519 1300558 656643 5073110 953437 ...  
## $ tra\_to\_oth\_aut\_com : num 0 0 0 0 0 ...  
## $ net\_con : num 1074950 1406719 650855 4938944 1197677 ...  
## $ net\_ope\_exp : num 907156 1298832 656210 5055942 949489 ...  
## $ winner : Factor w/ 2 levels "","Y": 2 2 2 2 2 2 2 2 2 2 ...

Candidate state is not needed:

campaigns <- subset(campaigns, select = -c(can\_sta))

str(campaigns)

## 'data.frame': 1814 obs. of 21 variables:  
## $ can\_off : Factor w/ 3 levels "H","P","S": 1 1 1 1 1 1 1 1 1 1 ...  
## $ can\_off\_sta : Factor w/ 57 levels "AK","AL","AR",..: 12 42 11 30 6 31 55 22 41 22 ...  
## $ can\_par\_aff : Factor w/ 25 levels "AMP","CON","CST",..: 21 4 21 21 4 4 4 4 21 4 ...  
## $ can\_inc\_cha\_ope\_sea: Factor w/ 4 levels "","CHALLENGER",..: 3 2 4 3 3 3 3 3 3 3 ...  
## $ ind\_ite\_con : num 554305 1042280 529030 2479616 746234 ...  
## $ ind\_uni\_con : num 46970 72431 13075 1837715 150890 ...  
## $ ind\_con : num 601275 1114711 542105 4317332 897124 ...  
## $ par\_com\_con : num 0 0 0 3545 0 ...  
## $ oth\_com\_con : num 473675 302834 106050 660039 308740 ...  
## $ can\_con : num 0 0 2700 0 0 0 0 0 0 0 ...  
## $ tot\_con : num 1074950 1417545 650855 4980915 1205864 ...  
## $ tra\_fro\_oth\_aut\_com: num 17710 0 0 136894 0 ...  
## $ can\_loa : num 0 0 60000 0 0 0 0 0 0 0 ...  
## $ oth\_loa : num 0 0 0 0 0 0 0 0 0 0 ...  
## $ tot\_loa : num 0 0 60000 0 0 0 0 0 0 0 ...  
## $ oth\_rec : num 0 0 0 55910 0 ...  
## $ ope\_exp : num 908519 1300558 656643 5073110 953437 ...  
## $ tra\_to\_oth\_aut\_com : num 0 0 0 0 0 ...  
## $ net\_con : num 1074950 1406719 650855 4938944 1197677 ...  
## $ net\_ope\_exp : num 907156 1298832 656210 5055942 949489 ...  
## $ winner : Factor w/ 2 levels "","Y": 2 2 2 2 2 2 2 2 2 2 ...

levels(campaigns$winner)[levels(campaigns$winner) == ""] <- "N"  
summary(campaigns$winner)

## N Y   
## 1343 471

levels(campaigns$can\_inc\_cha\_ope\_sea)[levels(campaigns$can\_inc\_cha\_ope\_sea) == ""] <- "UNKNOWN"  
summary(campaigns$can\_inc\_cha\_ope\_sea)

## UNKNOWN CHALLENGER INCUMBENT OPEN   
## 2 850 425 537

Printing final cleaned csv

write.csv(campaigns,'C:/Users/nicky/Desktop/Syracuse Grad School/IST 707/Final Project/campaigns\_cleaned.csv')

summary(campaigns$can\_par\_aff)

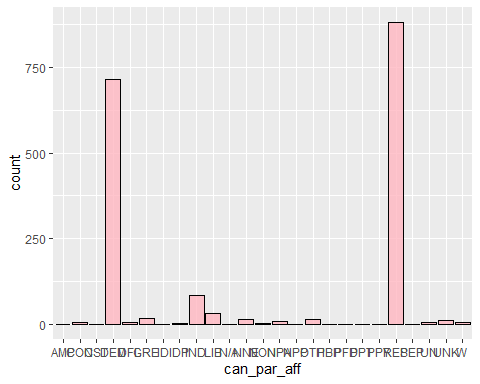
## AMP CON CST DEM DFL GRE ID IDP IND LIB N/A NNE NON NPA NPP OTH PBP PFD PPT PPY   
## 1 4 1 714 4 16 1 3 85 33 1 15 2 9 1 15 1 1 1 1   
## REP SEP UN UNK W   
## 882 1 6 12 4

library(ggplot2)

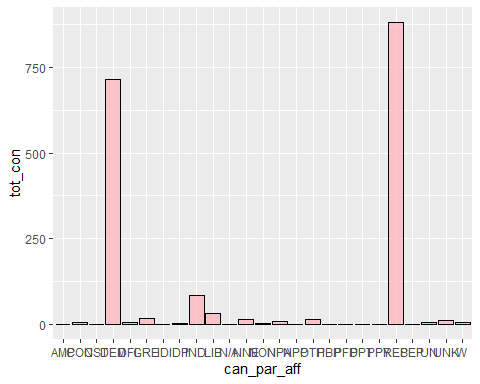
## Warning: package 'ggplot2' was built under R version 4.2.3

ggplot(campaigns, aes(x=can\_par\_aff)) + geom\_bar(aes(y=..count..),fill = 'lightpink',color = 'black',alpha = .8)

## Warning: The dot-dot notation (`..count..`) was deprecated in ggplot2 3.4.0.  
## ℹ Please use `after\_stat(count)` instead.  
## This warning is displayed once every 8 hours.  
## Call `lifecycle::last\_lifecycle\_warnings()` to see where this warning was  
## generated.



ggplot(campaigns, aes(x=can\_par\_aff,y=tot\_con)) + geom\_bar(aes(y=..count..),fill = 'lightpink',color = 'black',alpha = .8)



Income sources per party:

library(tidyverse)

## Warning: package 'tidyverse' was built under R version 4.2.3

## Warning: package 'tibble' was built under R version 4.2.3

## Warning: package 'tidyr' was built under R version 4.2.3

## Warning: package 'readr' was built under R version 4.2.3

## Warning: package 'purrr' was built under R version 4.2.3

## Warning: package 'dplyr' was built under R version 4.2.3

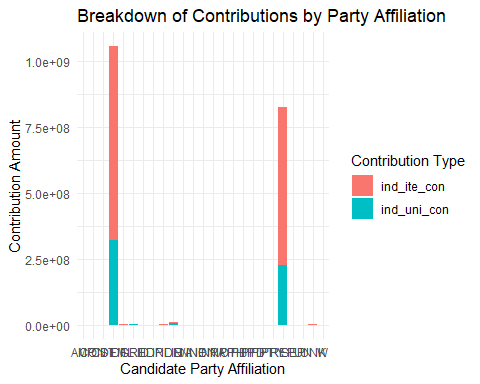
## Warning: package 'stringr' was built under R version 4.2.3

## Warning: package 'forcats' was built under R version 4.2.3

## Warning: package 'lubridate' was built under R version 4.2.3

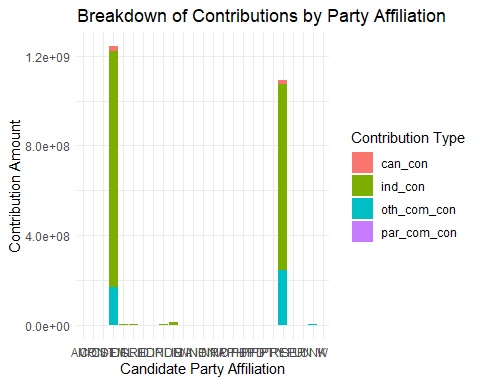
## ── Attaching core tidyverse packages ──────────────────────── tidyverse 2.0.0 ──  
## ✔ dplyr 1.1.2 ✔ readr 2.1.4  
## ✔ forcats 1.0.0 ✔ stringr 1.5.0  
## ✔ lubridate 1.9.2 ✔ tibble 3.2.1  
## ✔ purrr 1.0.1 ✔ tidyr 1.3.0  
## ── Conflicts ────────────────────────────────────────── tidyverse\_conflicts() ──  
## ✖ dplyr::filter() masks stats::filter()  
## ✖ dplyr::lag() masks stats::lag()  
## ℹ Use the ]8;;http://conflicted.r-lib.org/conflicted package]8;; to force all conflicts to become errors

df\_long2 <- campaigns %>%  
 pivot\_longer(cols = c(ind\_ite\_con, ind\_uni\_con),  
 names\_to = "contribution\_type",  
 values\_to = "amount")  
  
  
ggplot(df\_long2, aes(fill=contribution\_type, y=amount, x=can\_par\_aff)) +   
 geom\_bar(position="stack", stat="identity") +  
 labs(x = "Candidate Party Affiliation",   
 y = "Contribution Amount",   
 fill = "Contribution Type",   
 title = "Breakdown of Contributions by Party Affiliation") +  
 theme\_minimal()



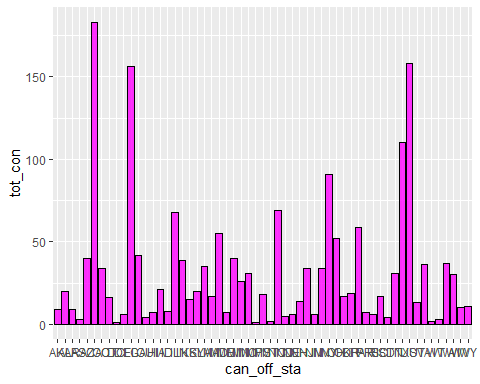
Digging deeper, individual contribution break down.

df\_long <- campaigns %>%  
 pivot\_longer(cols = c(ind\_con, par\_com\_con, oth\_com\_con, can\_con),  
 names\_to = "contribution\_type",  
 values\_to = "amount")  
  
  
ggplot(df\_long, aes(fill=contribution\_type, y=amount, x=can\_par\_aff)) +   
 geom\_bar(position="stack", stat="identity") +  
 labs(x = "Candidate Party Affiliation",   
 y = "Contribution Amount",   
 fill = "Contribution Type",   
 title = "Breakdown of Contributions by Party Affiliation") +  
 theme\_minimal()



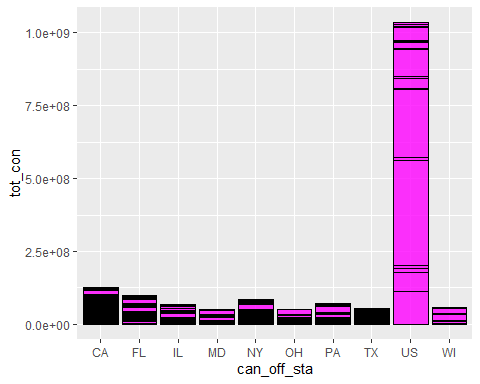
Most Expensive State Race

ggplot(campaigns, aes(x=can\_off\_sta,y=tot\_con)) + geom\_bar(aes(y=..count..),fill = 'magenta',color = 'black',alpha = .8)



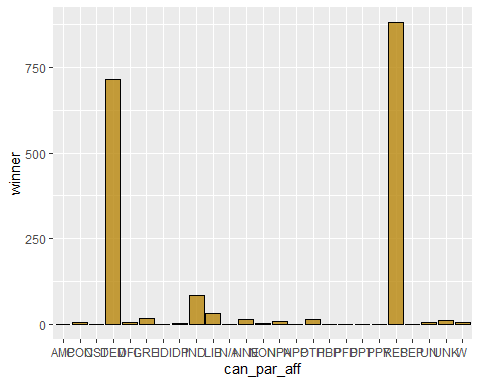
#this is too crowded

# Aggregating the data  
state\_totals <- campaigns %>%  
 group\_by(can\_off\_sta) %>%  
 summarise(tot\_con\_state = sum(tot\_con, na.rm = TRUE)) %>%  
 arrange(desc(tot\_con\_state))  
  
# Selecting the top N states  
top\_states <- state\_totals %>%  
 top\_n(10, wt = tot\_con\_state)  
  
# Filtering the original data for only these states  
campaigns\_filtered <- campaigns %>%  
 filter(can\_off\_sta %in% top\_states$can\_off\_sta)  
  
# Create the plot  
ggplot(campaigns\_filtered, aes(x=can\_off\_sta, y=tot\_con)) +   
 geom\_bar(stat="identity", fill = 'magenta', color = 'black', alpha = .8)



Party with most wins

ggplot(campaigns, aes(x=can\_par\_aff,y=winner)) + geom\_bar(aes(y=..count..),fill = 'darkgoldenrod',color = 'black',alpha = .8)



#creating a new data set with all cleaned data   
campaigns\_cl <- read.csv('C:/Users/nicky/Desktop/Syracuse Grad School/IST 707/Final Project/campaigns\_cleaned.csv')  
str(campaigns\_cl)

## 'data.frame': 1814 obs. of 22 variables:  
## $ X : int 1 2 3 4 5 6 7 8 9 10 ...  
## $ can\_off : chr "H" "H" "H" "H" ...  
## $ can\_off\_sta : chr "GA" "PA" "FL" "MT" ...  
## $ can\_par\_aff : chr "REP" "DEM" "REP" "REP" ...  
## $ can\_inc\_cha\_ope\_sea: chr "INCUMBENT" "CHALLENGER" "OPEN" "INCUMBENT" ...  
## $ ind\_ite\_con : num 554305 1042280 529030 2479616 746234 ...  
## $ ind\_uni\_con : num 46970 72431 13075 1837715 150890 ...  
## $ ind\_con : num 601275 1114711 542105 4317332 897124 ...  
## $ par\_com\_con : num 0 0 0 3545 0 ...  
## $ oth\_com\_con : num 473675 302834 106050 660039 308740 ...  
## $ can\_con : num 0 0 2700 0 0 0 0 0 0 0 ...  
## $ tot\_con : num 1074950 1417545 650855 4980915 1205864 ...  
## $ tra\_fro\_oth\_aut\_com: num 17710 0 0 136894 0 ...  
## $ can\_loa : num 0 0 60000 0 0 0 0 0 0 0 ...  
## $ oth\_loa : num 0 0 0 0 0 0 0 0 0 0 ...  
## $ tot\_loa : num 0 0 60000 0 0 0 0 0 0 0 ...  
## $ oth\_rec : num 0 0 0 55910 0 ...  
## $ ope\_exp : num 908519 1300558 656643 5073110 953437 ...  
## $ tra\_to\_oth\_aut\_com : num 0 0 0 0 0 ...  
## $ net\_con : num 1074950 1406719 650855 4938944 1197677 ...  
## $ net\_ope\_exp : num 907156 1298832 656210 5055942 949489 ...  
## $ winner : chr "Y" "Y" "Y" "Y" ...

In order for association rule mining to work, all attributes that will be used need to be turned into characters.Since not all numerical columns need to be chosen since they are all additions of each other, all that will be needed is the total columns for the different expenses.

arm\_df <- subset(campaigns\_cl, select = -c(X,ind\_con,tot\_con,tot\_loa))

Next, summary will be used so that the ranges can be determined for association rule mining. costs will be broken into 5 categories - none, low, medium-low, medium, medium-high, high, huge.

summary(arm\_df)

## can\_off can\_off\_sta can\_par\_aff can\_inc\_cha\_ope\_sea  
## Length:1814 Length:1814 Length:1814 Length:1814   
## Class :character Class :character Class :character Class :character   
## Mode :character Mode :character Mode :character Mode :character   
##   
##   
##   
## ind\_ite\_con ind\_uni\_con par\_com\_con oth\_com\_con   
## Min. : 0 Min. : 0 Min. : 0 Min. : 0   
## 1st Qu.: 3748 1st Qu.: 863 1st Qu.: 0 1st Qu.: 0   
## Median : 56368 Median : 9096 Median : 0 Median : 1312   
## Mean : 741736 Mean : 309212 Mean : 1508 Mean : 229046   
## 3rd Qu.: 420491 3rd Qu.: 41342 3rd Qu.: 0 3rd Qu.: 280571   
## Max. :266757494 Max. :134684157 Max. :449200 Max. :3954545   
## can\_con tra\_fro\_oth\_aut\_com can\_loa oth\_loa   
## Min. : 0 Min. : 0 Min. : 0 Min. : 0   
## 1st Qu.: 0 1st Qu.: 0 1st Qu.: 0 1st Qu.: 0   
## Median : 0 Median : 0 Median : 0 Median : 0   
## Mean : 22068 Mean : 163695 Mean : 84750 Mean : 3047   
## 3rd Qu.: 1141 3rd Qu.: 0 3rd Qu.: 7836 3rd Qu.: 0   
## Max. :13414225 Max. :141290000 Max. :47508505 Max. :2400000   
## oth\_rec ope\_exp tra\_to\_oth\_aut\_com net\_con   
## Min. : 0.0 Min. : 0 Min. : 0 Min. :0.000e+00   
## 1st Qu.: 0.0 1st Qu.: 12327 1st Qu.: 0 1st Qu.:8.176e+03   
## Median : 0.0 Median : 101493 Median : 0 Median :8.968e+04   
## Mean : 5005.0 Mean : 1339195 Mean : 6959 Mean :5.156e+06   
## 3rd Qu.: 12.7 3rd Qu.: 701483 3rd Qu.: 0 3rd Qu.:9.210e+05   
## Max. :1427900.7 Max. :445661956 Max. :2852535 Max. :2.526e+09   
## net\_ope\_exp winner   
## Min. :0.000e+00 Length:1814   
## 1st Qu.:1.140e+04 Class :character   
## Median :1.009e+05 Mode :character   
## Mean :5.150e+06   
## 3rd Qu.:7.064e+05   
## Max. :2.467e+09

making characters for each attribute:

arm\_df$ind\_ite\_con <- cut(arm\_df$ind\_ite\_con, breaks = c(0,1,10000,56000,420000,1000000,50000000,100000000), labels=c('None','low','medium-low','medium','medium-high','high','huge'), right=FALSE)

Now for the rest of the attributes

arm\_df$ind\_uni\_con <- cut(arm\_df$ind\_uni\_con, breaks = c(0,1,10000,56000,420000,1000000,50000000,100000000), labels=c('None','low','medium-low','medium','medium-high','high','huge'), right=FALSE)  
  
arm\_df$par\_com\_con <- cut(arm\_df$par\_com\_con, breaks = c(0,1,10000,56000,420000,1000000,50000000,100000000), labels=c('None','low','medium-low','medium','medium-high','high','huge'), right=FALSE)  
  
arm\_df$oth\_com\_con <- cut(arm\_df$oth\_com\_con, breaks = c(0,1,10000,56000,420000,1000000,50000000,100000000), labels=c('None','low','medium-low','medium','medium-high','high','huge'), right=FALSE)  
  
arm\_df$can\_con <- cut(arm\_df$can\_con, breaks = c(0,1,10000,56000,420000,1000000,50000000,100000000), labels=c('None','low','medium-low','medium','medium-high','high','huge'), right=FALSE)  
  
arm\_df$tra\_fro\_oth\_aut\_com <- cut(arm\_df$tra\_fro\_oth\_aut\_com, breaks = c(0,1,10000,56000,420000,1000000,50000000,100000000), labels=c('None','low','medium-low','medium','medium-high','high','huge'), right=FALSE)  
  
arm\_df$can\_loa <- cut(arm\_df$can\_loa, breaks = c(0,1,10000,56000,420000,1000000,50000000,100000000), labels=c('None','low','medium-low','medium','medium-high','high','huge'), right=FALSE)  
  
arm\_df$oth\_loa <- cut(arm\_df$oth\_loa, breaks = c(0,1,10000,56000,420000,1000000,50000000,100000000), labels=c('None','low','medium-low','medium','medium-high','high','huge'), right=FALSE)  
  
arm\_df$oth\_rec <- cut(arm\_df$oth\_rec, breaks = c(0,1,10000,56000,420000,1000000,50000000,100000000), labels=c('None','low','medium-low','medium','medium-high','high','huge'), right=FALSE)  
  
arm\_df$ope\_exp <- cut(arm\_df$ope\_exp, breaks = c(0,1,10000,56000,420000,1000000,50000000,100000000), labels=c('None','low','medium-low','medium','medium-high','high','huge'), right=FALSE)  
  
arm\_df$tra\_to\_oth\_aut\_com <- cut(arm\_df$tra\_to\_oth\_aut\_com, breaks = c(0,1,10000,56000,420000,1000000,50000000,100000000), labels=c('None','low','medium-low','medium','medium-high','high','huge'), right=FALSE)  
  
arm\_df$net\_con <- cut(arm\_df$net\_con, breaks = c(0,1,10000,56000,420000,1000000,50000000,100000000), labels=c('None','low','medium-low','medium','medium-high','high','huge'), right=FALSE)  
  
arm\_df$net\_ope\_exp <- cut(arm\_df$net\_ope\_exp, breaks = c(0,1,10000,56000,420000,1000000,50000000,100000000), labels=c('None','low','medium-low','medium','medium-high','high','huge'), right=FALSE)

str(arm\_df)

## 'data.frame': 1814 obs. of 18 variables:  
## $ can\_off : chr "H" "H" "H" "H" ...  
## $ can\_off\_sta : chr "GA" "PA" "FL" "MT" ...  
## $ can\_par\_aff : chr "REP" "DEM" "REP" "REP" ...  
## $ can\_inc\_cha\_ope\_sea: chr "INCUMBENT" "CHALLENGER" "OPEN" "INCUMBENT" ...  
## $ ind\_ite\_con : Factor w/ 7 levels "None","low","medium-low",..: 5 6 5 6 5 4 4 5 5 6 ...  
## $ ind\_uni\_con : Factor w/ 7 levels "None","low","medium-low",..: 3 4 3 6 4 4 4 4 4 3 ...  
## $ par\_com\_con : Factor w/ 7 levels "None","low","medium-low",..: 1 1 1 2 1 2 2 1 1 1 ...  
## $ oth\_com\_con : Factor w/ 7 levels "None","low","medium-low",..: 5 4 4 5 4 4 5 4 6 5 ...  
## $ can\_con : Factor w/ 7 levels "None","low","medium-low",..: 1 1 2 1 1 1 1 1 1 1 ...  
## $ tra\_fro\_oth\_aut\_com: Factor w/ 7 levels "None","low","medium-low",..: 3 1 1 4 1 1 1 1 4 4 ...  
## $ can\_loa : Factor w/ 7 levels "None","low","medium-low",..: 1 1 4 1 1 1 1 1 1 1 ...  
## $ oth\_loa : Factor w/ 7 levels "None","low","medium-low",..: 1 1 1 1 1 1 1 1 1 1 ...  
## $ oth\_rec : Factor w/ 7 levels "None","low","medium-low",..: 1 1 1 3 1 1 1 1 2 3 ...  
## $ ope\_exp : Factor w/ 7 levels "None","low","medium-low",..: 5 6 5 6 5 5 5 5 6 6 ...  
## $ tra\_to\_oth\_aut\_com : Factor w/ 7 levels "None","low","medium-low",..: 1 1 1 1 1 2 1 1 1 1 ...  
## $ net\_con : Factor w/ 7 levels "None","low","medium-low",..: 6 6 5 6 6 5 5 6 6 6 ...  
## $ net\_ope\_exp : Factor w/ 7 levels "None","low","medium-low",..: 5 6 5 6 5 5 5 5 6 6 ...  
## $ winner : chr "Y" "Y" "Y" "Y" ...

Turning remaining attributes into factors as well to finalize the set for ARM

arm\_df$can\_off <- as.factor(arm\_df$can\_off)  
arm\_df$can\_off\_sta <- as.factor(arm\_df$can\_off\_sta)  
arm\_df$can\_par\_aff <- as.factor(arm\_df$can\_par\_aff)  
arm\_df$can\_inc\_cha\_ope\_sea <- as.factor(arm\_df$can\_inc\_cha\_ope\_sea)  
arm\_df$winner <- as.factor(arm\_df$winner)

str(arm\_df)

## 'data.frame': 1814 obs. of 18 variables:  
## $ can\_off : Factor w/ 3 levels "H","P","S": 1 1 1 1 1 1 1 1 1 1 ...  
## $ can\_off\_sta : Factor w/ 57 levels "AK","AL","AR",..: 12 42 11 30 6 31 55 22 41 22 ...  
## $ can\_par\_aff : Factor w/ 25 levels "AMP","CON","CST",..: 21 4 21 21 4 4 4 4 21 4 ...  
## $ can\_inc\_cha\_ope\_sea: Factor w/ 4 levels "CHALLENGER","INCUMBENT",..: 2 1 3 2 2 2 2 2 2 2 ...  
## $ ind\_ite\_con : Factor w/ 7 levels "None","low","medium-low",..: 5 6 5 6 5 4 4 5 5 6 ...  
## $ ind\_uni\_con : Factor w/ 7 levels "None","low","medium-low",..: 3 4 3 6 4 4 4 4 4 3 ...  
## $ par\_com\_con : Factor w/ 7 levels "None","low","medium-low",..: 1 1 1 2 1 2 2 1 1 1 ...  
## $ oth\_com\_con : Factor w/ 7 levels "None","low","medium-low",..: 5 4 4 5 4 4 5 4 6 5 ...  
## $ can\_con : Factor w/ 7 levels "None","low","medium-low",..: 1 1 2 1 1 1 1 1 1 1 ...  
## $ tra\_fro\_oth\_aut\_com: Factor w/ 7 levels "None","low","medium-low",..: 3 1 1 4 1 1 1 1 4 4 ...  
## $ can\_loa : Factor w/ 7 levels "None","low","medium-low",..: 1 1 4 1 1 1 1 1 1 1 ...  
## $ oth\_loa : Factor w/ 7 levels "None","low","medium-low",..: 1 1 1 1 1 1 1 1 1 1 ...  
## $ oth\_rec : Factor w/ 7 levels "None","low","medium-low",..: 1 1 1 3 1 1 1 1 2 3 ...  
## $ ope\_exp : Factor w/ 7 levels "None","low","medium-low",..: 5 6 5 6 5 5 5 5 6 6 ...  
## $ tra\_to\_oth\_aut\_com : Factor w/ 7 levels "None","low","medium-low",..: 1 1 1 1 1 2 1 1 1 1 ...  
## $ net\_con : Factor w/ 7 levels "None","low","medium-low",..: 6 6 5 6 6 5 5 6 6 6 ...  
## $ net\_ope\_exp : Factor w/ 7 levels "None","low","medium-low",..: 5 6 5 6 5 5 5 5 6 6 ...  
## $ winner : Factor w/ 2 levels "N","Y": 2 2 2 2 2 2 2 2 2 2 ...

#needed libraries  
#install.packages("arules")  
library(arules)

## Warning: package 'arules' was built under R version 4.2.3

## Loading required package: Matrix

## Warning: package 'Matrix' was built under R version 4.2.2

##   
## Attaching package: 'Matrix'

## The following objects are masked from 'package:tidyr':  
##   
## expand, pack, unpack

##   
## Attaching package: 'arules'

## The following object is masked from 'package:dplyr':  
##   
## recode

## The following objects are masked from 'package:base':  
##   
## abbreviate, write

next, converting data frame to a transactions object.

trans <- as(arm\_df, "transactions")

Mining the association rules with the apriori function. Using a minimum support of 0.1 and a minimum confidence of 0.8:

rules <- apriori(trans, parameter = list(supp = 0.1, conf = 0.8))

## Apriori  
##   
## Parameter specification:  
## confidence minval smax arem aval originalSupport maxtime support minlen  
## 0.8 0.1 1 none FALSE TRUE 5 0.1 1  
## maxlen target ext  
## 10 rules TRUE  
##   
## Algorithmic control:  
## filter tree heap memopt load sort verbose  
## 0.1 TRUE TRUE FALSE TRUE 2 TRUE  
##   
## Absolute minimum support count: 181   
##   
## set item appearances ...[0 item(s)] done [0.00s].  
## set transactions ...[174 item(s), 1814 transaction(s)] done [0.00s].  
## sorting and recoding items ... [50 item(s)] done [0.00s].  
## creating transaction tree ... done [0.00s].  
## checking subsets of size 1 2 3 4 5 6 7 8 9 10

## Warning in apriori(trans, parameter = list(supp = 0.1, conf = 0.8)): Mining  
## stopped (maxlen reached). Only patterns up to a length of 10 returned!

## done [0.03s].  
## writing ... [40591 rule(s)] done [0.00s].  
## creating S4 object ... done [0.01s].

inspecting rules by lift and confidence:

inspect(sort(rules, by="lift")[1:10])

## lhs rhs support confidence coverage lift count  
## [1] {par\_com\_con=None,   
## ope\_exp=low,   
## net\_con=low} => {net\_ope\_exp=low} 0.1085998 1 0.1085998 6.693727 197  
## [2] {par\_com\_con=None,   
## oth\_com\_con=None,   
## ope\_exp=low,   
## net\_con=low} => {net\_ope\_exp=low} 0.1047409 1 0.1047409 6.693727 190  
## [3] {par\_com\_con=None,   
## oth\_rec=None,   
## ope\_exp=low,   
## net\_con=low} => {net\_ope\_exp=low} 0.1014333 1 0.1014333 6.693727 184  
## [4] {par\_com\_con=None,   
## ope\_exp=low,   
## net\_con=low,   
## winner=N} => {net\_ope\_exp=low} 0.1085998 1 0.1085998 6.693727 197  
## [5] {par\_com\_con=None,   
## tra\_fro\_oth\_aut\_com=None,   
## ope\_exp=low,   
## net\_con=low} => {net\_ope\_exp=low} 0.1047409 1 0.1047409 6.693727 190  
## [6] {par\_com\_con=None,   
## ope\_exp=low,   
## tra\_to\_oth\_aut\_com=None,   
## net\_con=low} => {net\_ope\_exp=low} 0.1047409 1 0.1047409 6.693727 190  
## [7] {par\_com\_con=None,   
## oth\_loa=None,   
## ope\_exp=low,   
## net\_con=low} => {net\_ope\_exp=low} 0.1063947 1 0.1063947 6.693727 193  
## [8] {par\_com\_con=None,   
## oth\_com\_con=None,   
## ope\_exp=low,   
## net\_con=low,   
## winner=N} => {net\_ope\_exp=low} 0.1047409 1 0.1047409 6.693727 190  
## [9] {par\_com\_con=None,   
## oth\_com\_con=None,   
## tra\_fro\_oth\_aut\_com=None,   
## ope\_exp=low,   
## net\_con=low} => {net\_ope\_exp=low} 0.1014333 1 0.1014333 6.693727 184  
## [10] {par\_com\_con=None,   
## oth\_com\_con=None,   
## ope\_exp=low,   
## tra\_to\_oth\_aut\_com=None,   
## net\_con=low} => {net\_ope\_exp=low} 0.1014333 1 0.1014333 6.693727 184

inspect(sort(rules, by="confidence")[1:10])

## lhs rhs support confidence coverage   
## [1] {can\_loa=low} => {winner=N} 0.1174201 1 0.1174201  
## [2] {ind\_ite\_con=None} => {winner=N} 0.1345094 1 0.1345094  
## [3] {net\_ope\_exp=low} => {winner=N} 0.1493936 1 0.1493936  
## [4] {ope\_exp=low} => {winner=N} 0.1670342 1 0.1670342  
## [5] {net\_con=low} => {winner=N} 0.1725469 1 0.1725469  
## [6] {net\_con=medium-low} => {winner=N} 0.1846748 1 0.1846748  
## [7] {net\_ope\_exp=medium-low} => {winner=N} 0.1990077 1 0.1990077  
## [8] {ind\_ite\_con=low} => {winner=N} 0.2039691 1 0.2039691  
## [9] {ope\_exp=medium-low} => {winner=N} 0.2105843 1 0.2105843  
## [10] {can\_loa=low, oth\_rec=None} => {winner=N} 0.1019846 1 0.1019846  
## lift count  
## [1] 1.350707 213   
## [2] 1.350707 244   
## [3] 1.350707 271   
## [4] 1.350707 303   
## [5] 1.350707 313   
## [6] 1.350707 335   
## [7] 1.350707 361   
## [8] 1.350707 370   
## [9] 1.350707 382   
## [10] 1.350707 185

For all the lift rules, lift is only at 1 which shows that these rules actually are independent from one another. Ex) x occurs as often as y. So these rules will not tell us anything interesting.

Confident is another story and actually reveals some information. As a reminder: The confidence of an association rules (X => Y) measures how often Y is found in transactions that contain X.

The top 3 rules show the following:

1. {can\_loa=low} => {winner=N}

This rule suggests that if the loan amount taken by the candidate is low (can\_loa=low), then it’s likely the candidate does not win the election (winner=N). The confidence of this rule is 0.1174201, which means this rule is correct about 11.7% of the time. The Lift of this rule is 1.350707, suggesting that the rule is 1.35 times more likely to be correct than a random assignment.

1. {ind\_ite\_con=None} => {winner=N}

This rule suggests that if there are no itemized individual contributions (ind\_ite\_con=None), then it’s likely the candidate does not win the election (winner=N). The confidence of this rule is 0.1345094, which means this rule is correct about 13.4% of the time. The Lift of this rule is 1.350707, suggesting that the rule is 1.35 times more likely to be correct than a random assignment.

1. {net\_ope\_exp=low} => {winner=N}

This rule suggests that if the net operational expenditure is low (net\_ope\_exp=low), then it’s likely the candidate does not win the election (winner=N). The confidence of this rule is 0.1493936, which means this rule is correct about 14.9% of the time. The Lift of this rule is 1.350707, suggesting that the rule is 1.35 times more likely to be correct than a random assignment.

These rules suggest a pattern in which a low spending candidate or low contribution receiving candidate often does not lead to not winning the election. The higher the confidence, the more frequently the rule has been found to be true. The Lift values greater than 1 suggest these rules are significant, as they are more likely than chance. #################### The next step will be using actual predictive algorithm on the data. We will use the older data set since Naive Bayes can handle both numerical and discrete variables.

library(e1071)

## Warning: package 'e1071' was built under R version 4.2.3

Need to take out some unneeded variables:

str(campaigns\_cl)

## 'data.frame': 1814 obs. of 22 variables:  
## $ X : int 1 2 3 4 5 6 7 8 9 10 ...  
## $ can\_off : chr "H" "H" "H" "H" ...  
## $ can\_off\_sta : chr "GA" "PA" "FL" "MT" ...  
## $ can\_par\_aff : chr "REP" "DEM" "REP" "REP" ...  
## $ can\_inc\_cha\_ope\_sea: chr "INCUMBENT" "CHALLENGER" "OPEN" "INCUMBENT" ...  
## $ ind\_ite\_con : num 554305 1042280 529030 2479616 746234 ...  
## $ ind\_uni\_con : num 46970 72431 13075 1837715 150890 ...  
## $ ind\_con : num 601275 1114711 542105 4317332 897124 ...  
## $ par\_com\_con : num 0 0 0 3545 0 ...  
## $ oth\_com\_con : num 473675 302834 106050 660039 308740 ...  
## $ can\_con : num 0 0 2700 0 0 0 0 0 0 0 ...  
## $ tot\_con : num 1074950 1417545 650855 4980915 1205864 ...  
## $ tra\_fro\_oth\_aut\_com: num 17710 0 0 136894 0 ...  
## $ can\_loa : num 0 0 60000 0 0 0 0 0 0 0 ...  
## $ oth\_loa : num 0 0 0 0 0 0 0 0 0 0 ...  
## $ tot\_loa : num 0 0 60000 0 0 0 0 0 0 0 ...  
## $ oth\_rec : num 0 0 0 55910 0 ...  
## $ ope\_exp : num 908519 1300558 656643 5073110 953437 ...  
## $ tra\_to\_oth\_aut\_com : num 0 0 0 0 0 ...  
## $ net\_con : num 1074950 1406719 650855 4938944 1197677 ...  
## $ net\_ope\_exp : num 907156 1298832 656210 5055942 949489 ...  
## $ winner : chr "Y" "Y" "Y" "Y" ...

nb\_df <- subset(campaigns\_cl, select = -c(X))  
  
nb\_df$can\_off <- as.factor(nb\_df$can\_off)  
nb\_df$can\_off\_sta <- as.factor(nb\_df$can\_off\_sta)  
nb\_df$can\_par\_aff <- as.factor(nb\_df$can\_par\_aff)  
nb\_df$can\_inc\_cha\_ope\_sea <- as.factor(nb\_df$can\_inc\_cha\_ope\_sea)  
nb\_df$winner <- as.factor(nb\_df$winner)

str(nb\_df)

## 'data.frame': 1814 obs. of 21 variables:  
## $ can\_off : Factor w/ 3 levels "H","P","S": 1 1 1 1 1 1 1 1 1 1 ...  
## $ can\_off\_sta : Factor w/ 57 levels "AK","AL","AR",..: 12 42 11 30 6 31 55 22 41 22 ...  
## $ can\_par\_aff : Factor w/ 25 levels "AMP","CON","CST",..: 21 4 21 21 4 4 4 4 21 4 ...  
## $ can\_inc\_cha\_ope\_sea: Factor w/ 4 levels "CHALLENGER","INCUMBENT",..: 2 1 3 2 2 2 2 2 2 2 ...  
## $ ind\_ite\_con : num 554305 1042280 529030 2479616 746234 ...  
## $ ind\_uni\_con : num 46970 72431 13075 1837715 150890 ...  
## $ ind\_con : num 601275 1114711 542105 4317332 897124 ...  
## $ par\_com\_con : num 0 0 0 3545 0 ...  
## $ oth\_com\_con : num 473675 302834 106050 660039 308740 ...  
## $ can\_con : num 0 0 2700 0 0 0 0 0 0 0 ...  
## $ tot\_con : num 1074950 1417545 650855 4980915 1205864 ...  
## $ tra\_fro\_oth\_aut\_com: num 17710 0 0 136894 0 ...  
## $ can\_loa : num 0 0 60000 0 0 0 0 0 0 0 ...  
## $ oth\_loa : num 0 0 0 0 0 0 0 0 0 0 ...  
## $ tot\_loa : num 0 0 60000 0 0 0 0 0 0 0 ...  
## $ oth\_rec : num 0 0 0 55910 0 ...  
## $ ope\_exp : num 908519 1300558 656643 5073110 953437 ...  
## $ tra\_to\_oth\_aut\_com : num 0 0 0 0 0 ...  
## $ net\_con : num 1074950 1406719 650855 4938944 1197677 ...  
## $ net\_ope\_exp : num 907156 1298832 656210 5055942 949489 ...  
## $ winner : Factor w/ 2 levels "N","Y": 2 2 2 2 2 2 2 2 2 2 ...

# Spliting data into train and test datasets  
  
set.seed(123)  
train\_indices\_nb <- sample(1:nrow(nb\_df), nrow(nb\_df)\*0.7)  
train\_df\_nb <- nb\_df[train\_indices\_nb, ]  
test\_df\_nb <- nb\_df[-train\_indices\_nb, ]

Training the naive bayes model

naive\_model <- naiveBayes(winner ~ ., data = train\_df\_nb)

predictions\_nb <- predict(naive\_model, test\_df\_nb)

library(caret)

## Warning: package 'caret' was built under R version 4.2.3

## Loading required package: lattice

## Warning: package 'lattice' was built under R version 4.2.3

##   
## Attaching package: 'caret'

## The following object is masked from 'package:purrr':  
##   
## lift

confusionMatrix(predictions\_nb, test\_df\_nb$winner)

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction N Y  
## N 64 1  
## Y 330 150  
##   
## Accuracy : 0.3927   
## 95% CI : (0.3514, 0.4351)  
## No Information Rate : 0.7229   
## P-Value [Acc > NIR] : 1   
##   
## Kappa : 0.0932   
##   
## Mcnemar's Test P-Value : <2e-16   
##   
## Sensitivity : 0.1624   
## Specificity : 0.9934   
## Pos Pred Value : 0.9846   
## Neg Pred Value : 0.3125   
## Prevalence : 0.7229   
## Detection Rate : 0.1174   
## Detection Prevalence : 0.1193   
## Balanced Accuracy : 0.5779   
##   
## 'Positive' Class : N   
##

The confusion matrix that the naive bayes algorithm output and the statistics provide an overview of the performance of the model.

Meaning of the confusion matrix. True Negative (N, N): The model correctly predicted 64 times that the candidate would not win. False Negative (Y, N): The model incorrectly predicted 1 time that a winning candidate would not win. False Positive (N, Y): The model incorrectly predicted 330 times that a losing candidate would win. True Positive (Y, Y): The model correctly predicted 150 times that the candidate would win.

Accuracy is the proportion of true results. Here, the accuracy is 0.3927, or about 39.27%. This means that the model correctly predicts the outcome about 39.27% of the time.

The Kappa statistic is a measure of how much better the predictions of the model are compared to predictions made by chance. Here, the kappa is 0.0932, which is quite low (a perfect model has a kappa of 1), indicating that the model is not much better than a random model.

Sensitivity is the proportion of actual positives that are correctly identified. In this case, the sensitivity is 0.1624, meaning that the model identifies about 16.24% of the winning candidates correctly.

Specificity measures the proportion of actual negatives that are correctly identified. Here, the specificity is 0.9934, suggesting the model is very good at identifying candidates that won’t win.

The Positive Predictive Value is the proportion of positive cases that were correctly identified by the model. In this case, it’s 0.9846 or 98.46% of candidates predicted not to win, did indeed not win. However, the Negative Predictive Value is low (0.3125 or 31.25%), meaning that of all the candidates that were predicted to win, only 31.25% actually won.

The No Information Rate is a baseline comparison, which is the accuracy that can be achieved without a model by always guessing the most frequent category. In this case, guessing the candidate does not win all the time would be right 72.29% of the time.

The p-value associated with the accuracy being greater than the no information rate is 1, which indicates no evidence of the model doing better than the no information rate.

Overall, this model seems to be better at predicting losers than predicting winners (low sensitivity and negative predictive value). The low accuracy, kappa, and high p-value suggest that there might be a need to improve this model, possibly by revising the feature set, using a different model, or adjusting the class imbalance.

The next is to use the kNN algorithm. The first step is to actually find out the optimal k to assign. To do this, we will use k-fold process:

#libraries  
library(class)

## Warning: package 'class' was built under R version 4.2.3

library(caret)

knn\_df <- subset(campaigns\_cl, select = -c(X))

training control

train\_control\_knn <- trainControl(method="cv", number=10)

Scaling the numeric data

knn\_df[,sapply(knn\_df, is.numeric)] <- scale(knn\_df[,sapply(knn\_df, is.numeric)])

str(knn\_df)

## 'data.frame': 1814 obs. of 21 variables:  
## $ can\_off : chr "H" "H" "H" "H" ...  
## $ can\_off\_sta : chr "GA" "PA" "FL" "MT" ...  
## $ can\_par\_aff : chr "REP" "DEM" "REP" "REP" ...  
## $ can\_inc\_cha\_ope\_sea: chr "INCUMBENT" "CHALLENGER" "OPEN" "INCUMBENT" ...  
## $ ind\_ite\_con : num -0.02657 0.042606 -0.030153 0.246364 0.000638 ...  
## $ ind\_uni\_con : num -0.0605 -0.0546 -0.0683 0.3527 -0.0365 ...  
## $ ind\_con : num -0.04156 0.00592 -0.04703 0.30208 -0.0142 ...  
## $ par\_com\_con : num -0.125 -0.125 -0.125 0.169 -0.125 ...  
## $ oth\_com\_con : num 0.515 0.155 -0.259 0.908 0.168 ...  
## $ can\_con : num -0.0567 -0.0567 -0.0498 -0.0567 -0.0567 ...  
## $ tot\_con : num -0.02087 0.01039 -0.05957 0.33551 -0.00893 ...  
## $ tra\_fro\_oth\_aut\_com: num -0.03748 -0.04202 -0.04202 -0.00688 -0.04202 ...  
## $ can\_loa : num -0.0724 -0.0724 -0.0211 -0.0724 -0.0724 ...  
## $ oth\_loa : num -0.0498 -0.0498 -0.0498 -0.0498 -0.0498 ...  
## $ tot\_loa : num -0.0746 -0.0746 -0.0236 -0.0746 -0.0746 ...  
## $ oth\_rec : num -0.0925 -0.0925 -0.0925 0.9413 -0.0925 ...  
## $ ope\_exp : num -0.03214 -0.00288 -0.05094 0.27866 -0.02879 ...  
## $ tra\_to\_oth\_aut\_com : num -0.0833 -0.0833 -0.0833 -0.0833 -0.0833 ...  
## $ net\_con : num -0.04926 -0.04525 -0.05438 -0.00262 -0.04778 ...  
## $ net\_ope\_exp : num -0.05224 -0.04741 -0.05533 -0.00116 -0.05171 ...  
## $ winner : chr "Y" "Y" "Y" "Y" ...

removing winner

knn\_df\_class <- knn\_df$winner  
knn\_df$winner <- NULL

Cannot perform knn without removing remaining categorical variables. Need to use a technique called hot encoding to change them into numerical ones.

df\_encoded <- cbind(knn\_df[, !(names(knn\_df) %in% "can\_off")], model.matrix(~can\_off - 1, knn\_df))

str(df\_encoded)

## 'data.frame': 1814 obs. of 22 variables:  
## $ can\_off\_sta : chr "GA" "PA" "FL" "MT" ...  
## $ can\_par\_aff : chr "REP" "DEM" "REP" "REP" ...  
## $ can\_inc\_cha\_ope\_sea: chr "INCUMBENT" "CHALLENGER" "OPEN" "INCUMBENT" ...  
## $ ind\_ite\_con : num -0.02657 0.042606 -0.030153 0.246364 0.000638 ...  
## $ ind\_uni\_con : num -0.0605 -0.0546 -0.0683 0.3527 -0.0365 ...  
## $ ind\_con : num -0.04156 0.00592 -0.04703 0.30208 -0.0142 ...  
## $ par\_com\_con : num -0.125 -0.125 -0.125 0.169 -0.125 ...  
## $ oth\_com\_con : num 0.515 0.155 -0.259 0.908 0.168 ...  
## $ can\_con : num -0.0567 -0.0567 -0.0498 -0.0567 -0.0567 ...  
## $ tot\_con : num -0.02087 0.01039 -0.05957 0.33551 -0.00893 ...  
## $ tra\_fro\_oth\_aut\_com: num -0.03748 -0.04202 -0.04202 -0.00688 -0.04202 ...  
## $ can\_loa : num -0.0724 -0.0724 -0.0211 -0.0724 -0.0724 ...  
## $ oth\_loa : num -0.0498 -0.0498 -0.0498 -0.0498 -0.0498 ...  
## $ tot\_loa : num -0.0746 -0.0746 -0.0236 -0.0746 -0.0746 ...  
## $ oth\_rec : num -0.0925 -0.0925 -0.0925 0.9413 -0.0925 ...  
## $ ope\_exp : num -0.03214 -0.00288 -0.05094 0.27866 -0.02879 ...  
## $ tra\_to\_oth\_aut\_com : num -0.0833 -0.0833 -0.0833 -0.0833 -0.0833 ...  
## $ net\_con : num -0.04926 -0.04525 -0.05438 -0.00262 -0.04778 ...  
## $ net\_ope\_exp : num -0.05224 -0.04741 -0.05533 -0.00116 -0.05171 ...  
## $ can\_offH : num 1 1 1 1 1 1 1 1 1 1 ...  
## $ can\_offP : num 0 0 0 0 0 0 0 0 0 0 ...  
## $ can\_offS : num 0 0 0 0 0 0 0 0 0 0 ...

This adds a lot of dimensional but does the job to change variables into numerical ones. So the same will be done for the rest: <https://stackoverflow.com/questions/48649443/how-to-one-hot-encode-several-categorical-variables-in-r>

df\_encoded <- cbind(df\_encoded[, !(names(df\_encoded) %in% "can\_off\_sta")], model.matrix(~can\_off\_sta - 1, df\_encoded))  
df\_encoded <- cbind(df\_encoded[, !(names(df\_encoded) %in% "can\_par\_aff")], model.matrix(~can\_par\_aff - 1, df\_encoded))  
df\_encoded <- cbind(df\_encoded[, !(names(df\_encoded) %in% "can\_inc\_cha\_ope\_sea")], model.matrix(~can\_inc\_cha\_ope\_sea - 1, df\_encoded))

str(df\_encoded)

## 'data.frame': 1814 obs. of 105 variables:  
## $ ind\_ite\_con : num -0.02657 0.042606 -0.030153 0.246364 0.000638 ...  
## $ ind\_uni\_con : num -0.0605 -0.0546 -0.0683 0.3527 -0.0365 ...  
## $ ind\_con : num -0.04156 0.00592 -0.04703 0.30208 -0.0142 ...  
## $ par\_com\_con : num -0.125 -0.125 -0.125 0.169 -0.125 ...  
## $ oth\_com\_con : num 0.515 0.155 -0.259 0.908 0.168 ...  
## $ can\_con : num -0.0567 -0.0567 -0.0498 -0.0567 -0.0567 ...  
## $ tot\_con : num -0.02087 0.01039 -0.05957 0.33551 -0.00893 ...  
## $ tra\_fro\_oth\_aut\_com : num -0.03748 -0.04202 -0.04202 -0.00688 -0.04202 ...  
## $ can\_loa : num -0.0724 -0.0724 -0.0211 -0.0724 -0.0724 ...  
## $ oth\_loa : num -0.0498 -0.0498 -0.0498 -0.0498 -0.0498 ...  
## $ tot\_loa : num -0.0746 -0.0746 -0.0236 -0.0746 -0.0746 ...  
## $ oth\_rec : num -0.0925 -0.0925 -0.0925 0.9413 -0.0925 ...  
## $ ope\_exp : num -0.03214 -0.00288 -0.05094 0.27866 -0.02879 ...  
## $ tra\_to\_oth\_aut\_com : num -0.0833 -0.0833 -0.0833 -0.0833 -0.0833 ...  
## $ net\_con : num -0.04926 -0.04525 -0.05438 -0.00262 -0.04778 ...  
## $ net\_ope\_exp : num -0.05224 -0.04741 -0.05533 -0.00116 -0.05171 ...  
## $ can\_offH : num 1 1 1 1 1 1 1 1 1 1 ...  
## $ can\_offP : num 0 0 0 0 0 0 0 0 0 0 ...  
## $ can\_offS : num 0 0 0 0 0 0 0 0 0 0 ...  
## $ can\_off\_staAK : num 0 0 0 0 0 0 0 0 0 0 ...  
## $ can\_off\_staAL : num 0 0 0 0 0 0 0 0 0 0 ...  
## $ can\_off\_staAR : num 0 0 0 0 0 0 0 0 0 0 ...  
## $ can\_off\_staAS : num 0 0 0 0 0 0 0 0 0 0 ...  
## $ can\_off\_staAZ : num 0 0 0 0 0 0 0 0 0 0 ...  
## $ can\_off\_staCA : num 0 0 0 0 1 0 0 0 0 0 ...  
## $ can\_off\_staCO : num 0 0 0 0 0 0 0 0 0 0 ...  
## $ can\_off\_staCT : num 0 0 0 0 0 0 0 0 0 0 ...  
## $ can\_off\_staDC : num 0 0 0 0 0 0 0 0 0 0 ...  
## $ can\_off\_staDE : num 0 0 0 0 0 0 0 0 0 0 ...  
## $ can\_off\_staFL : num 0 0 1 0 0 0 0 0 0 0 ...  
## $ can\_off\_staGA : num 1 0 0 0 0 0 0 0 0 0 ...  
## $ can\_off\_staGU : num 0 0 0 0 0 0 0 0 0 0 ...  
## $ can\_off\_staHI : num 0 0 0 0 0 0 0 0 0 0 ...  
## $ can\_off\_staIA : num 0 0 0 0 0 0 0 0 0 0 ...  
## $ can\_off\_staID : num 0 0 0 0 0 0 0 0 0 0 ...  
## $ can\_off\_staIL : num 0 0 0 0 0 0 0 0 0 0 ...  
## $ can\_off\_staIN : num 0 0 0 0 0 0 0 0 0 0 ...  
## $ can\_off\_staKS : num 0 0 0 0 0 0 0 0 0 0 ...  
## $ can\_off\_staKY : num 0 0 0 0 0 0 0 0 0 0 ...  
## $ can\_off\_staLA : num 0 0 0 0 0 0 0 0 0 0 ...  
## $ can\_off\_staMA : num 0 0 0 0 0 0 0 1 0 1 ...  
## $ can\_off\_staMD : num 0 0 0 0 0 0 0 0 0 0 ...  
## $ can\_off\_staME : num 0 0 0 0 0 0 0 0 0 0 ...  
## $ can\_off\_staMI : num 0 0 0 0 0 0 0 0 0 0 ...  
## $ can\_off\_staMN : num 0 0 0 0 0 0 0 0 0 0 ...  
## $ can\_off\_staMO : num 0 0 0 0 0 0 0 0 0 0 ...  
## $ can\_off\_staMP : num 0 0 0 0 0 0 0 0 0 0 ...  
## $ can\_off\_staMS : num 0 0 0 0 0 0 0 0 0 0 ...  
## $ can\_off\_staMT : num 0 0 0 1 0 0 0 0 0 0 ...  
## $ can\_off\_staNC : num 0 0 0 0 0 1 0 0 0 0 ...  
## $ can\_off\_staND : num 0 0 0 0 0 0 0 0 0 0 ...  
## $ can\_off\_staNE : num 0 0 0 0 0 0 0 0 0 0 ...  
## $ can\_off\_staNH : num 0 0 0 0 0 0 0 0 0 0 ...  
## $ can\_off\_staNJ : num 0 0 0 0 0 0 0 0 0 0 ...  
## $ can\_off\_staNM : num 0 0 0 0 0 0 0 0 0 0 ...  
## $ can\_off\_staNV : num 0 0 0 0 0 0 0 0 0 0 ...  
## $ can\_off\_staNY : num 0 0 0 0 0 0 0 0 0 0 ...  
## $ can\_off\_staOH : num 0 0 0 0 0 0 0 0 0 0 ...  
## $ can\_off\_staOK : num 0 0 0 0 0 0 0 0 0 0 ...  
## $ can\_off\_staOR : num 0 0 0 0 0 0 0 0 1 0 ...  
## $ can\_off\_staPA : num 0 1 0 0 0 0 0 0 0 0 ...  
## $ can\_off\_staPR : num 0 0 0 0 0 0 0 0 0 0 ...  
## $ can\_off\_staRI : num 0 0 0 0 0 0 0 0 0 0 ...  
## $ can\_off\_staSC : num 0 0 0 0 0 0 0 0 0 0 ...  
## $ can\_off\_staSD : num 0 0 0 0 0 0 0 0 0 0 ...  
## $ can\_off\_staTN : num 0 0 0 0 0 0 0 0 0 0 ...  
## $ can\_off\_staTX : num 0 0 0 0 0 0 0 0 0 0 ...  
## $ can\_off\_staUS : num 0 0 0 0 0 0 0 0 0 0 ...  
## $ can\_off\_staUT : num 0 0 0 0 0 0 0 0 0 0 ...  
## $ can\_off\_staVA : num 0 0 0 0 0 0 0 0 0 0 ...  
## $ can\_off\_staVI : num 0 0 0 0 0 0 0 0 0 0 ...  
## $ can\_off\_staVT : num 0 0 0 0 0 0 0 0 0 0 ...  
## $ can\_off\_staWA : num 0 0 0 0 0 0 0 0 0 0 ...  
## $ can\_off\_staWI : num 0 0 0 0 0 0 1 0 0 0 ...  
## $ can\_off\_staWV : num 0 0 0 0 0 0 0 0 0 0 ...  
## $ can\_off\_staWY : num 0 0 0 0 0 0 0 0 0 0 ...  
## $ can\_par\_affAMP : num 0 0 0 0 0 0 0 0 0 0 ...  
## $ can\_par\_affCON : num 0 0 0 0 0 0 0 0 0 0 ...  
## $ can\_par\_affCST : num 0 0 0 0 0 0 0 0 0 0 ...  
## $ can\_par\_affDEM : num 0 1 0 0 1 1 1 1 0 1 ...  
## $ can\_par\_affDFL : num 0 0 0 0 0 0 0 0 0 0 ...  
## $ can\_par\_affGRE : num 0 0 0 0 0 0 0 0 0 0 ...  
## $ can\_par\_affID : num 0 0 0 0 0 0 0 0 0 0 ...  
## $ can\_par\_affIDP : num 0 0 0 0 0 0 0 0 0 0 ...  
## $ can\_par\_affIND : num 0 0 0 0 0 0 0 0 0 0 ...  
## $ can\_par\_affLIB : num 0 0 0 0 0 0 0 0 0 0 ...  
## $ can\_par\_affN/A : num 0 0 0 0 0 0 0 0 0 0 ...  
## $ can\_par\_affNNE : num 0 0 0 0 0 0 0 0 0 0 ...  
## $ can\_par\_affNON : num 0 0 0 0 0 0 0 0 0 0 ...  
## $ can\_par\_affNPA : num 0 0 0 0 0 0 0 0 0 0 ...  
## $ can\_par\_affNPP : num 0 0 0 0 0 0 0 0 0 0 ...  
## $ can\_par\_affOTH : num 0 0 0 0 0 0 0 0 0 0 ...  
## $ can\_par\_affPBP : num 0 0 0 0 0 0 0 0 0 0 ...  
## $ can\_par\_affPFD : num 0 0 0 0 0 0 0 0 0 0 ...  
## $ can\_par\_affPPT : num 0 0 0 0 0 0 0 0 0 0 ...  
## $ can\_par\_affPPY : num 0 0 0 0 0 0 0 0 0 0 ...  
## $ can\_par\_affREP : num 1 0 1 1 0 0 0 0 1 0 ...  
## $ can\_par\_affSEP : num 0 0 0 0 0 0 0 0 0 0 ...  
## $ can\_par\_affUN : num 0 0 0 0 0 0 0 0 0 0 ...  
## [list output truncated]

Performing kNN to find optimal K

knn\_fit <- train(df\_encoded, knn\_df\_class, method="knn", tuneLength=10, trControl=train\_control\_knn)

print(knn\_fit$bestTune)

## k  
## 9 21

Now that we have the optimal K, we can also use the Knn model:

knn\_df2 <- subset(campaigns\_cl, select = -c(X))  
  
# Scaling the data again  
knn\_df2[,sapply(knn\_df2, is.numeric)] <- scale(knn\_df2[,sapply(knn\_df2, is.numeric)])  
  
#turning categorical variables into numeric  
knn\_df2 <- cbind(knn\_df2[, !(names(knn\_df2) %in% "can\_off")], model.matrix(~can\_off - 1, knn\_df2))  
knn\_df2 <- cbind(knn\_df2[, !(names(knn\_df2) %in% "can\_off\_sta")], model.matrix(~can\_off\_sta - 1, knn\_df2))  
knn\_df2 <- cbind(knn\_df2[, !(names(knn\_df2) %in% "can\_par\_aff")], model.matrix(~can\_par\_aff - 1, knn\_df2))  
knn\_df2 <- cbind(knn\_df2[, !(names(knn\_df2) %in% "can\_inc\_cha\_ope\_sea")], model.matrix(~can\_inc\_cha\_ope\_sea - 1, knn\_df2))  
  
# Spliting the data into a training set and a test set  
set.seed(123)  
data\_index <- sample(1:nrow(knn\_df2), nrow(knn\_df2)\*0.7)  
train\_set <- knn\_df2[data\_index,]  
test\_set <- knn\_df2[-data\_index,]  
  
# The 'class' column is our target variable  
train\_labels <- train\_set$winner  
test\_labels <- test\_set$winner  
  
# Removing the 'class' column from the training and test set  
train\_set$winner <- NULL  
test\_set$winner <- NULL  
  
# Performing kNN  
knn\_pred <- knn(train = train\_set, test = test\_set, cl = train\_labels, k=11) # using 11 because that was the optimal.  
  
# Checking the accuracy of the model  
mean(test\_labels == knn\_pred)

## [1] 0.946789

0.946789, is the accuracy of the knn model. Mean Accuracy computes the number of correct predictions divided by the total number of predictions.The accuracy of 0.946789 means that the model correctly predicted the outcome approximately 94.68%. This is generally a good result, but not if the cost of a false negative is very high. While in other applications, we might not mind a high false negative amount, this might be the case if there are costs on the line.

SVM Model was next on the list and also only takes numeric attributes so that same scaled and numeric data set from above.

library(e1071)  
  
svm\_df <- subset(campaigns\_cl, select = -c(X))  
  
  
# Scaling the data again  
svm\_df[,sapply(svm\_df, is.numeric)] <- scale(svm\_df[,sapply(svm\_df, is.numeric)])  
  
#turning categorical variables into numeric  
svm\_df <- cbind(svm\_df[, !(names(svm\_df) %in% "can\_off")], model.matrix(~can\_off - 1, svm\_df))  
svm\_df <- cbind(svm\_df[, !(names(svm\_df) %in% "can\_off\_sta")], model.matrix(~can\_off\_sta - 1, svm\_df))  
svm\_df <- cbind(svm\_df[, !(names(svm\_df) %in% "can\_par\_aff")], model.matrix(~can\_par\_aff - 1, svm\_df))  
svm\_df <- cbind(svm\_df[, !(names(svm\_df) %in% "can\_inc\_cha\_ope\_sea")], model.matrix(~can\_inc\_cha\_ope\_sea - 1, svm\_df))  
svm\_df$winner <- ifelse(svm\_df$winner == "Y", 1, 0)  
  
set.seed(123)   
split\_index <- sample(2, nrow(svm\_df), replace=TRUE, prob=c(0.7,0.3))  
train\_data <- svm\_df[split\_index==1,]  
test\_data <- svm\_df[split\_index==2,]  
  
svm\_model <- svm(winner ~ ., data = train\_data, kernel = "radial", scale = TRUE)

## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):  
## Variable(s) 'can\_off\_staAS' and 'can\_off\_staMT' and 'can\_off\_staVI' and  
## 'X.can\_par\_affN.A.' and 'can\_par\_affNPP' and 'can\_par\_affPBP' and  
## 'can\_par\_affPPY' constant. Cannot scale data.

predictions <- predict(svm\_model, newdata = test\_data)  
  
  
confusion\_matrix <- table(pred = predictions, true = test\_data$winner)  
accuracy <- sum(diag(confusion\_matrix)) / sum(confusion\_matrix)  
print(paste('Accuracy:', accuracy))

## [1] "Accuracy: 0.00183823529411765"

Such a low accuracy rate was expected because of the huge disparity between losers and winners.

winner\_counts <- table(svm\_df$winner)  
  
print(winner\_counts)

##   
## 0 1   
## 1343 471

471/1343

## [1] 0.3507074

As we can see above, only 35% of the data are winners which could be why the model was performing so badly.

Next is decision trees and some columns will have to removed because decision trees are not good with data they havent seen. Some attributes such as party or state do not matter as much. This data set will have to be cleared of all the outlier parties that are not republican, democrat or independent.

#str(campaigns\_cl)

library(rpart)

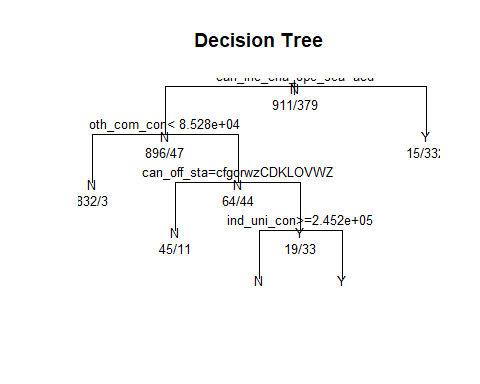
## Warning: package 'rpart' was built under R version 4.2.3

dt\_df <- subset(campaigns\_cl, select = -c(X))  
dt\_df<- dt\_df[dt\_df$can\_par\_aff %in% c("REP", "DEM", "IDP"), ]  
  
set.seed(123)   
  
split\_index <- sample(2, nrow(dt\_df), replace=TRUE, prob=c(0.8,0.2))  
train\_data <- dt\_df[split\_index==1,]  
test\_data <- dt\_df[split\_index==2,]  
  
# Training  
tree\_model <- rpart(winner ~ ., data = train\_data, method = "class")  
  
# Printing the decision tree model  
print(tree\_model)

## n= 1290   
##   
## node), split, n, loss, yval, (yprob)  
## \* denotes terminal node  
##   
## 1) root 1290 379 N (0.706201550 0.293798450)   
## 2) can\_inc\_cha\_ope\_sea=CHALLENGER,OPEN,UNKNOWN 943 47 N (0.950159067 0.049840933)   
## 4) oth\_com\_con< 85283.5 835 3 N (0.996407186 0.003592814) \*  
## 5) oth\_com\_con>=85283.5 108 44 N (0.592592593 0.407407407)   
## 10) can\_off\_sta=AR,CA,CO,IA,IN,MD,MN,MT,NC,NY,OH,PA,US,UT,WV 56 11 N (0.803571429 0.196428571) \*  
## 11) can\_off\_sta=AZ,DE,FL,GA,IL,KS,KY,LA,MI,NE,NH,NJ,NV,TN,TX,VA,WA,WI,WY 52 19 Y (0.365384615 0.634615385)   
## 22) ind\_uni\_con>=245244.4 13 4 N (0.692307692 0.307692308) \*  
## 23) ind\_uni\_con< 245244.4 39 10 Y (0.256410256 0.743589744) \*  
## 3) can\_inc\_cha\_ope\_sea=INCUMBENT 347 15 Y (0.043227666 0.956772334) \*

# Plotting the decision tree  
plot(tree\_model, uniform=TRUE, main="Decision Tree")  
text(tree\_model, use.n=TRUE, all=TRUE, cex=.8)

## Warning in labels.rpart(x, minlength = minlength): more than 52 levels in a  
## predicting factor, truncated for printout



# Making predictions on testing data  
predictions <- predict(tree\_model, newdata = test\_data, type = "class")  
  
# accuracy  
confusion\_matrix <- table(pred = predictions, true = test\_data$winner)  
accuracy <- sum(diag(confusion\_matrix)) / sum(confusion\_matrix)  
print(paste('Accuracy:', accuracy))

## [1] "Accuracy: 0.919093851132686"

The root of the tree (node 1) includes all 1290 observations. The base prediction (yval) is ‘N’, because the majority (about 70.6%) of the candidates did not win.

The first split (node 2) is based on the variable ‘can\_inc\_cha\_ope\_sea’. If a candidate is an ‘INCUMBENT’, the model predicts a win (node 3) with a very high probability (95.7%).

If a candidate is a ‘CHALLENGER’, ‘OPEN’, or ‘UNKNOWN’, the model makes further splits based on the ‘oth\_com\_con’ and ‘can\_off\_sta’ variables (nodes 4 and 5).

For ‘oth\_com\_con’ < 85283.5 (node 4), the model predicts ‘N’ (not win) with a very high probability (99.6%).

For ‘oth\_com\_con’ >= 85283.5 (node 5), the model makes further decisions based on the ‘can\_off\_sta’ variable. If ‘can\_off\_sta’ belongs to ‘AR,CA,CO,IA,IN,MD,MN,MT,NC,NY,OH,PA,US,UT,WV’, the model predicts ‘N’ (node 10), and if ‘can\_off\_sta’ belongs to ‘AZ,DE,FL,GA,IL,KS,KY,LA,MI,NE,NH,NJ,NV,TN,TX,VA,WA,WI,WY’, the model makes further decisions based on the ‘ind\_uni\_con’ variable (node 11).

If ‘ind\_uni\_con’ >= 245244.4 (node 22), the model predicts ‘N’, and if ‘ind\_uni\_con’ < 245244.4 (node 23), the model predicts ‘Y’ (win) with a high probability (74.4%).

The accuracy of this model on the test set is about 91.9%, which means the model correctly predicted the winner for about 91.9% of the cases in the test set.

Finally, random forest:

library(randomForest)

## Warning: package 'randomForest' was built under R version 4.2.3

## randomForest 4.7-1.1

## Type rfNews() to see new features/changes/bug fixes.

##   
## Attaching package: 'randomForest'

## The following object is masked from 'package:dplyr':  
##   
## combine

## The following object is masked from 'package:ggplot2':  
##   
## margin

rf\_df <- subset(campaigns\_cl, select = -c(X))  
rf\_df<- rf\_df[rf\_df$can\_par\_aff %in% c("REP", "DEM", "IDP"), ]  
rf\_df$winner <- as.factor(rf\_df$winner)  
  
set.seed(123)   
  
split\_index <- sample(2, nrow(rf\_df), replace=TRUE, prob=c(0.8,0.2))  
train\_data <- rf\_df[split\_index==1,]  
test\_data <- rf\_df[split\_index==2,]  
  
# Training  
rf\_model <- randomForest(winner ~ ., data = train\_data, ntree = 500, mtry = 3, importance = TRUE)  
  
# Printing the decision tree model  
print(rf\_model)

##   
## Call:  
## randomForest(formula = winner ~ ., data = train\_data, ntree = 500, mtry = 3, importance = TRUE)   
## Type of random forest: classification  
## Number of trees: 500  
## No. of variables tried at each split: 3  
##   
## OOB estimate of error rate: 4.73%  
## Confusion matrix:  
## N Y class.error  
## N 882 29 0.03183315  
## Y 32 347 0.08443272

# Making predictions on testing data  
predictions <- predict(rf\_model, newdata = test\_data)  
  
# accuracy  
confusion\_matrix <- table(pred = predictions, true = test\_data$winner)  
accuracy <- sum(diag(confusion\_matrix)) / sum(confusion\_matrix)  
print(paste('Accuracy:', accuracy))

## [1] "Accuracy: 0.964401294498382"

importance(rf\_model)

## N Y MeanDecreaseAccuracy MeanDecreaseGini  
## can\_off 3.6687749 9.135406 9.6956222 4.028365  
## can\_off\_sta 3.5091439 -1.090701 1.8474877 5.576789  
## can\_par\_aff 3.0052598 1.674497 3.1465468 1.635088  
## can\_inc\_cha\_ope\_sea 6.7395409 22.421383 20.4818511 39.904888  
## ind\_ite\_con 8.1929490 7.461729 11.5065683 29.779888  
## ind\_uni\_con 17.6943208 -5.647981 12.5130932 21.975400  
## ind\_con 11.0618658 5.430790 13.0639809 33.878625  
## par\_com\_con 8.5858301 -5.151288 3.6509799 6.049416  
## oth\_com\_con 19.9173028 32.746465 30.8609396 117.448295  
## can\_con 1.2823391 8.375000 8.2206112 7.135696  
## tot\_con 11.8741749 13.028338 16.8340573 58.879761  
## tra\_fro\_oth\_aut\_com 2.0986875 11.337795 10.5593462 8.297627  
## can\_loa -0.1593137 9.448784 9.2710152 10.103119  
## oth\_loa -4.2197569 2.119535 -0.7763937 0.596057  
## tot\_loa 1.9184383 9.677746 10.3207858 11.576751  
## oth\_rec 1.4804970 9.607325 8.5499606 10.333671  
## ope\_exp 9.9839737 6.299865 11.7793831 46.783178  
## tra\_to\_oth\_aut\_com 5.9596187 -2.451487 4.7022175 2.845605  
## net\_con 11.6297089 16.455821 18.7150979 69.506118  
## net\_ope\_exp 10.6030400 6.525396 11.7202757 47.062832

The output shows a lot of details.

Type of random forest: classification - means we used a classification model. The target variable, winner, is categorical.

Number of trees: 500. This is the number of trees that were used to vote for the class in classification or averaged in regression.

No. of variables tried at each split: 3 - At each split in each decision tree, 3 variables were randomly selected as candidates for splitting.

OOB estimate of error rate: 4.73% - This is the Out-of-bag error estimate, which is a method of measuring the prediction error of random forests. This error rate suggests that the model is quite good, as the error rate is very low.

The confusion matrix provides a summary of the model’s performance. It’s a 2x2 matrix: “Y” and “N”. The model has classified 882 instances correctly as ‘N’ and 29 incorrectly as ‘Y’. Similarly, it has classified 347 instances correctly as ‘Y’ and 32 incorrectly as ‘N’. The class error is calculated as the number of wrong predictions divided by the total predictions for each class.

Accuracy: 0.964401294498382 - This is the overall accuracy of the model on the test data. It is very high, which indicates that the model did a great job in classifying the winner.

str(rf\_df)

## 'data.frame': 1599 obs. of 21 variables:  
## $ can\_off : chr "H" "H" "H" "H" ...  
## $ can\_off\_sta : chr "GA" "PA" "FL" "MT" ...  
## $ can\_par\_aff : chr "REP" "DEM" "REP" "REP" ...  
## $ can\_inc\_cha\_ope\_sea: chr "INCUMBENT" "CHALLENGER" "OPEN" "INCUMBENT" ...  
## $ ind\_ite\_con : num 554305 1042280 529030 2479616 746234 ...  
## $ ind\_uni\_con : num 46970 72431 13075 1837715 150890 ...  
## $ ind\_con : num 601275 1114711 542105 4317332 897124 ...  
## $ par\_com\_con : num 0 0 0 3545 0 ...  
## $ oth\_com\_con : num 473675 302834 106050 660039 308740 ...  
## $ can\_con : num 0 0 2700 0 0 0 0 0 0 0 ...  
## $ tot\_con : num 1074950 1417545 650855 4980915 1205864 ...  
## $ tra\_fro\_oth\_aut\_com: num 17710 0 0 136894 0 ...  
## $ can\_loa : num 0 0 60000 0 0 0 0 0 0 0 ...  
## $ oth\_loa : num 0 0 0 0 0 0 0 0 0 0 ...  
## $ tot\_loa : num 0 0 60000 0 0 0 0 0 0 0 ...  
## $ oth\_rec : num 0 0 0 55910 0 ...  
## $ ope\_exp : num 908519 1300558 656643 5073110 953437 ...  
## $ tra\_to\_oth\_aut\_com : num 0 0 0 0 0 ...  
## $ net\_con : num 1074950 1406719 650855 4938944 1197677 ...  
## $ net\_ope\_exp : num 907156 1298832 656210 5055942 949489 ...  
## $ winner : Factor w/ 2 levels "N","Y": 2 2 2 2 2 2 2 2 2 2 ...