Final Project - Bank Dataset

Aaron Abromowitz, Stephanie Duarte, Dammy Owolabi

2024-04-20

Youtube Link: <https://youtu.be/oeHvLTXBvNQ?si=Wxctb9p1mOr-wJOV>

# EDA

The first thing that is done when you get a dataset is to peform an Exploratory Data Anslysis, to see what characteristics the data has. The variable we are trying to predict is the y column whic represents if the client has subscribed to a term deposit. The possible values are “yes” or “no”.

## Create training and test set

Going forward, we will use the training set for all analysis and model building. The test set will be used at the end to get metrics for the various models we create.

# Pull in data  
data<-read.csv('https://raw.githubusercontent.com/stedua22/6372-Project-2/main/bank-additional-full.csv',stringsAsFactors = T, sep=";")  
  
# Set levels to use for later  
data$y <- relevel(data$y, ref="yes")  
data$month <- factor(data$month, levels=c('mar','apr','may','jun','jul','aug','sep','oct','nov','dec'))  
data$day\_of\_week <- factor(data$day\_of\_week, levels=c('mon','tue','wed','thu','fri'))  
  
# Duration was removed since the dataset explanation file said that it was created after y variable was known, so shouldn't be used for prediction.  
data$duration <- c()  
  
# Create the train and test split  
train\_perc <- .8  
set.seed(1234)  
train\_indices <- sample(nrow(data), floor(train\_perc \* nrow(data)))  
train\_data <- data[train\_indices, ]  
nrow(train\_data)

## [1] 32950

test\_data <- data[-train\_indices, ]   
nrow(test\_data)

## [1] 8238

## Look at summary statistics for numeric variables

There are several numeric variables where we can look at the min/max, quartiles, median, and mean.

summary(train\_data$age)

## Min. 1st Qu. Median Mean 3rd Qu. Max.   
## 17.00 32.00 38.00 40.07 47.00 98.00

summary(train\_data$campaign)

## Min. 1st Qu. Median Mean 3rd Qu. Max.   
## 1.000 1.000 2.000 2.561 3.000 56.000

summary(train\_data$pdays)

## Min. 1st Qu. Median Mean 3rd Qu. Max.   
## 0.0 999.0 999.0 962.8 999.0 999.0

summary(train\_data$previous)

## Min. 1st Qu. Median Mean 3rd Qu. Max.   
## 0.0000 0.0000 0.0000 0.1733 0.0000 7.0000

summary(train\_data$emp.var.rate)

## Min. 1st Qu. Median Mean 3rd Qu. Max.   
## -3.40000 -1.80000 1.10000 0.07483 1.40000 1.40000

summary(train\_data$cons.price.idx)

## Min. 1st Qu. Median Mean 3rd Qu. Max.   
## 92.20 93.08 93.75 93.57 93.99 94.77

summary(train\_data$cons.conf.idx)

## Min. 1st Qu. Median Mean 3rd Qu. Max.   
## -50.80 -42.70 -41.80 -40.51 -36.40 -26.90

summary(train\_data$euribor3m)

## Min. 1st Qu. Median Mean 3rd Qu. Max.   
## 0.634 1.344 4.857 3.614 4.961 5.045

summary(train\_data$nr.employed)

## Min. 1st Qu. Median Mean 3rd Qu. Max.   
## 4964 5099 5191 5167 5228 5228

## Look at summary statistics for categorical variables

There are several categorical variables. Summary statistics don’t make as much sense for them, but you can look at the distribution of values in the different categories.

summary(train\_data$job)

## admin. blue-collar entrepreneur housemaid management   
## 8283 7426 1189 836 2355   
## retired self-employed services student technician   
## 1391 1130 3185 712 5359   
## unemployed unknown   
## 827 257

summary(summary(train\_data$job))

## Min. 1st Qu. Median Mean 3rd Qu. Max.   
## 257.0 833.8 1290.0 2745.8 3728.5 8283.0

length(summary(train\_data$job))

## [1] 12

summary(train\_data$marital)

## divorced married single unknown   
## 3699 19937 9248 66

summary(summary(train\_data$marital))

## Min. 1st Qu. Median Mean 3rd Qu. Max.   
## 66 2791 6474 8238 11920 19937

length(summary(train\_data$marital))

## [1] 4

summary(train\_data$education)

## basic.4y basic.6y basic.9y high.school   
## 3345 1859 4803 7646   
## illiterate professional.course university.degree unknown   
## 16 4172 9738 1371

summary(summary(train\_data$education))

## Min. 1st Qu. Median Mean 3rd Qu. Max.   
## 16 1737 3758 4119 5514 9738

length(summary(train\_data$education))

## [1] 8

summary(train\_data$default)

## no unknown yes   
## 26060 6888 2

summary(summary(train\_data$default))

## Min. 1st Qu. Median Mean 3rd Qu. Max.   
## 2 3445 6888 10983 16474 26060

length(summary(train\_data$default))

## [1] 3

summary(train\_data$housing)

## no unknown yes   
## 14918 794 17238

summary(summary(train\_data$housing))

## Min. 1st Qu. Median Mean 3rd Qu. Max.   
## 794 7856 14918 10983 16078 17238

length(summary(train\_data$housing))

## [1] 3

summary(train\_data$loan)

## no unknown yes   
## 27197 794 4959

summary(summary(train\_data$loan))

## Min. 1st Qu. Median Mean 3rd Qu. Max.   
## 794 2876 4959 10983 16078 27197

length(summary(train\_data$loan))

## [1] 3

summary(train\_data$contact)

## cellular telephone   
## 20989 11961

summary(summary(train\_data$contact))

## Min. 1st Qu. Median Mean 3rd Qu. Max.   
## 11961 14218 16475 16475 18732 20989

length(summary(train\_data$contact))

## [1] 2

summary(train\_data$month)

## mar apr may jun jul aug sep oct nov dec   
## 440 2098 11023 4233 5760 4931 450 572 3305 138

summary(summary(train\_data$month))

## Min. 1st Qu. Median Mean 3rd Qu. Max.   
## 138.0 480.5 2701.5 3295.0 4756.5 11023.0

length(summary(train\_data$month))

## [1] 10

summary(train\_data$day\_of\_week)

## mon tue wed thu fri   
## 6772 6513 6506 6874 6285

summary(summary(train\_data$day\_of\_week))

## Min. 1st Qu. Median Mean 3rd Qu. Max.   
## 6285 6506 6513 6590 6772 6874

length(summary(train\_data$day\_of\_week))

## [1] 5

summary(train\_data$poutcome)

## failure nonexistent success   
## 3438 28425 1087

summary(summary(train\_data$poutcome))

## Min. 1st Qu. Median Mean 3rd Qu. Max.   
## 1087 2262 3438 10983 15932 28425

length(summary(train\_data$poutcome))

## [1] 3

## Examine bank client data

The dataset includes age, job, marital, education, default, housing, and loan columns, which were identified as client data.

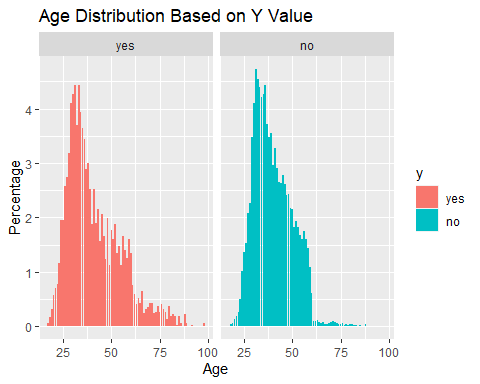
library(tidyverse)

## ── Attaching core tidyverse packages ──────────────────────── tidyverse 2.0.0 ──  
## ✔ dplyr 1.1.1 ✔ readr 2.1.4  
## ✔ forcats 1.0.0 ✔ stringr 1.5.0  
## ✔ ggplot2 3.4.1 ✔ tibble 3.2.1  
## ✔ lubridate 1.9.2 ✔ tidyr 1.3.0  
## ✔ purrr 1.0.1   
## ── 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

# Plot age  
summary <- train\_data %>%  
 group\_by(age,y) %>%  
 summarize(count=n())

## `summarise()` has grouped output by 'age'. You can override using the `.groups`  
## argument.

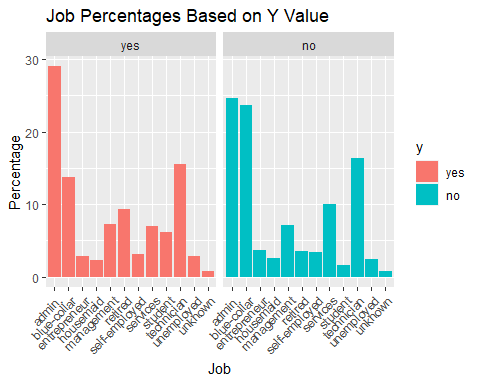
summary$perc <- 0  
summary$perc[summary$y == 'no'] <- summary$count[summary$y == 'no'] / nrow(train\_data[train\_data$y == 'no',]) \* 100  
summary$perc[summary$y == 'yes'] <- summary$count[summary$y == 'yes'] / nrow(train\_data[train\_data$y == 'yes',]) \* 100  
summary %>% ggplot(aes(x=age,y=perc,fill=y)) + geom\_bar(stat="identity") + facet\_wrap(~y) +   
 ylab('Percentage') + xlab('Age') + ggtitle('Age Distribution Based on Y Value')



# Plot job  
summary <- train\_data %>%  
 group\_by(job,y) %>%  
 summarize(count=n())

## `summarise()` has grouped output by 'job'. You can override using the `.groups`  
## argument.

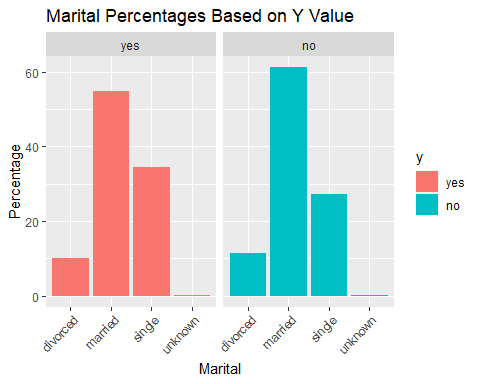
summary$perc <- 0  
summary$perc[summary$y == 'no'] <- summary$count[summary$y == 'no'] / nrow(train\_data[train\_data$y == 'no',]) \* 100  
summary$perc[summary$y == 'yes'] <- summary$count[summary$y == 'yes'] / nrow(train\_data[train\_data$y == 'yes',]) \* 100  
summary %>% ggplot(aes(x=job,y=perc,fill=y)) + geom\_bar(stat="identity") + facet\_wrap(~y) +   
 ylab('Percentage') + xlab('Job') + ggtitle('Job Percentages Based on Y Value') +   
 theme(axis.text.x = element\_text(angle = 45, hjust = 1))



# Plot marital  
summary <- train\_data %>%  
 group\_by(marital,y) %>%  
 summarize(count=n())

## `summarise()` has grouped output by 'marital'. You can override using the  
## `.groups` argument.

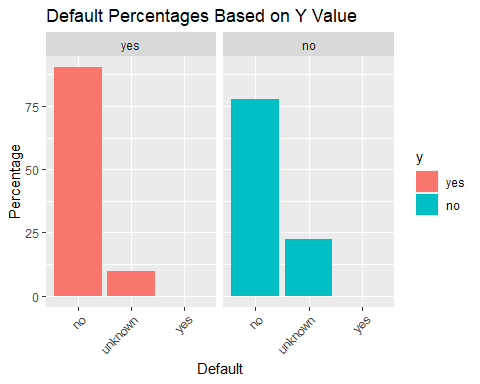
summary$perc <- 0  
summary$perc[summary$y == 'no'] <- summary$count[summary$y == 'no'] / nrow(train\_data[train\_data$y == 'no',]) \* 100  
summary$perc[summary$y == 'yes'] <- summary$count[summary$y == 'yes'] / nrow(train\_data[train\_data$y == 'yes',]) \* 100  
summary %>% ggplot(aes(x=marital,y=perc,fill=y)) + geom\_bar(stat="identity") + facet\_wrap(~y) +   
 ylab('Percentage') + xlab('Marital') + ggtitle('Marital Percentages Based on Y Value') +   
 theme(axis.text.x = element\_text(angle = 45, hjust = 1))



# Plot default  
summary <- train\_data %>%  
 group\_by(default,y) %>%  
 summarize(count=n())

## `summarise()` has grouped output by 'default'. You can override using the  
## `.groups` argument.

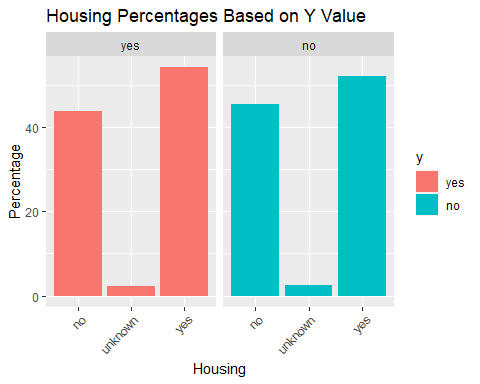
summary$perc <- 0  
summary$perc[summary$y == 'no'] <- summary$count[summary$y == 'no'] / nrow(train\_data[train\_data$y == 'no',]) \* 100  
summary$perc[summary$y == 'yes'] <- summary$count[summary$y == 'yes'] / nrow(train\_data[train\_data$y == 'yes',]) \* 100  
summary %>% ggplot(aes(x=default,y=perc,fill=y)) + geom\_bar(stat="identity") + facet\_wrap(~y) +   
 ylab('Percentage') + xlab('Default') + ggtitle('Default Percentages Based on Y Value') +   
 theme(axis.text.x = element\_text(angle = 45, hjust = 1))



# Plot housing  
summary <- train\_data %>%  
 group\_by(housing,y) %>%  
 summarize(count=n())

## `summarise()` has grouped output by 'housing'. You can override using the  
## `.groups` argument.

summary$perc <- 0  
summary$perc[summary$y == 'no'] <- summary$count[summary$y == 'no'] / nrow(train\_data[train\_data$y == 'no',]) \* 100  
summary$perc[summary$y == 'yes'] <- summary$count[summary$y == 'yes'] / nrow(train\_data[train\_data$y == 'yes',]) \* 100  
summary %>% ggplot(aes(x=housing,y=perc,fill=y)) + geom\_bar(stat="identity") + facet\_wrap(~y) +   
 ylab('Percentage') + xlab('Housing') + ggtitle('Housing Percentages Based on Y Value') +   
 theme(axis.text.x = element\_text(angle = 45, hjust = 1))

 None of these variables appear to show any strong indicator for yes or no. It looks like higher ages tend to lean more towards yes, and certain jobs (Ex: admin) lean more towards yes. When Y = no, there are more than about double the unknown values, but still far under 50%.

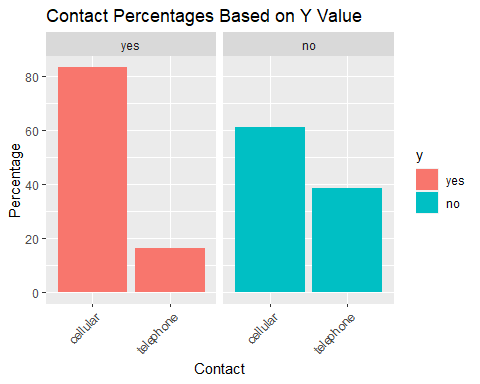
## Examine data related with the last contact of the current campaign

The dataset includes contact communication type, month of contact, and day of week of contac, for the last contact of the current campaign to sell term deposits.

# Plot contact  
summary <- train\_data %>%  
 group\_by(contact,y) %>%  
 summarize(count=n())

## `summarise()` has grouped output by 'contact'. You can override using the  
## `.groups` argument.

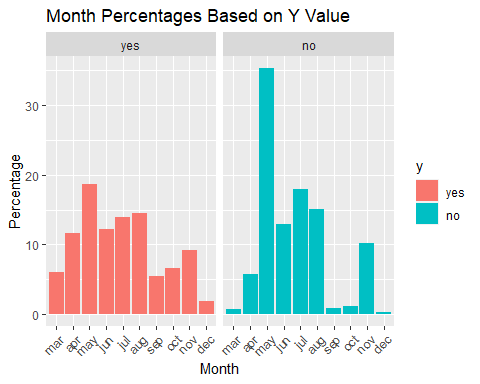
summary$perc <- 0  
summary$perc[summary$y == 'no'] <- summary$count[summary$y == 'no'] / nrow(train\_data[train\_data$y == 'no',]) \* 100  
summary$perc[summary$y == 'yes'] <- summary$count[summary$y == 'yes'] / nrow(train\_data[train\_data$y == 'yes',]) \* 100  
summary %>% ggplot(aes(x=contact,y=perc,fill=y)) + geom\_bar(stat="identity") + facet\_wrap(~y) +   
 ylab('Percentage') + xlab('Contact') + ggtitle('Contact Percentages Based on Y Value') +   
 theme(axis.text.x = element\_text(angle = 45, hjust = 1))



# Plot month  
summary <- train\_data %>%  
 group\_by(month,y) %>%  
 summarize(count=n())

## `summarise()` has grouped output by 'month'. You can override using the  
## `.groups` argument.

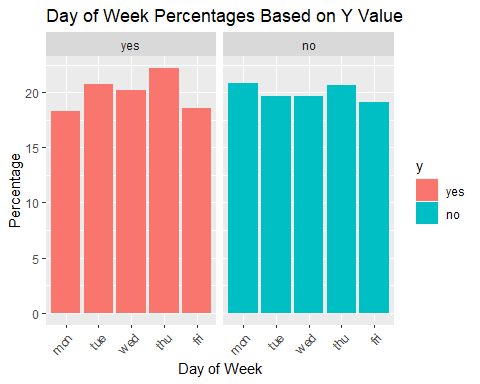
summary$perc <- 0  
summary$perc[summary$y == 'no'] <- summary$count[summary$y == 'no'] / nrow(train\_data[train\_data$y == 'no',]) \* 100  
summary$perc[summary$y == 'yes'] <- summary$count[summary$y == 'yes'] / nrow(train\_data[train\_data$y == 'yes',]) \* 100  
summary %>% ggplot(aes(x=month,y=perc,fill=y)) + geom\_bar(stat="identity") + facet\_wrap(~y) +   
 ylab('Percentage') + xlab('Month') + ggtitle('Month Percentages Based on Y Value') +   
 theme(axis.text.x = element\_text(angle = 45, hjust = 1))



# Plot day of week  
summary <- train\_data %>%  
 group\_by(day\_of\_week,y) %>%  
 summarize(count=n())

## `summarise()` has grouped output by 'day\_of\_week'. You can override using the  
## `.groups` argument.

summary$perc <- 0  
summary$perc[summary$y == 'no'] <- summary$count[summary$y == 'no'] / nrow(train\_data[train\_data$y == 'no',]) \* 100  
summary$perc[summary$y == 'yes'] <- summary$count[summary$y == 'yes'] / nrow(train\_data[train\_data$y == 'yes',]) \* 100  
summary %>% ggplot(aes(x=day\_of\_week,y=perc,fill=y)) + geom\_bar(stat="identity") + facet\_wrap(~y) +   
 ylab('Percentage') + xlab('Day of Week') + ggtitle('Day of Week Percentages Based on Y Value') +   
 theme(axis.text.x = element\_text(angle = 45, hjust = 1))

 If y is Yes, there are ~20% more likely to be contacted on your cell phone than on landline. There are certain months that also seem to have more term deposit sales in them. Day of week looks to not have much change.

## Examine other data related with the current campaign or previous campaigns

There are variables for number of contacts performed during this campaign and for this client (campaign), number of days that passed by after the client was last contacted from a previous campaign (pdays), number of contacts performed before this campaign and for this client (previous), and outcome of the previous marketing campaign (poutcome).

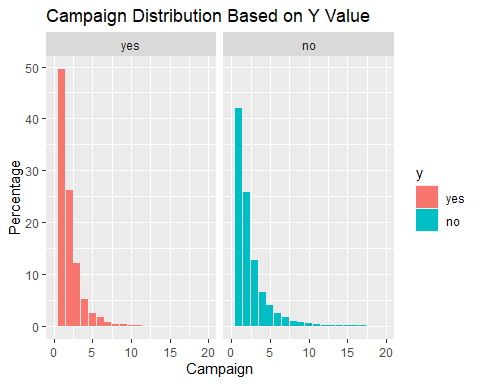
# Plot campaign  
summary <- train\_data %>%  
 group\_by(campaign,y) %>%  
 summarize(count=n())

## `summarise()` has grouped output by 'campaign'. You can override using the  
## `.groups` argument.

summary$perc <- 0  
summary$perc[summary$y == 'no'] <- summary$count[summary$y == 'no'] / nrow(train\_data[train\_data$y == 'no',]) \* 100  
summary$perc[summary$y == 'yes'] <- summary$count[summary$y == 'yes'] / nrow(train\_data[train\_data$y == 'yes',]) \* 100  
summary %>% ggplot(aes(x=campaign,y=perc,fill=y)) + geom\_bar(stat="identity") + facet\_wrap(~y) +   
 ylab('Percentage') + xlab('Campaign') + ggtitle('Campaign Distribution Based on Y Value') + xlim(c(0,20))

## Warning: Removed 23 rows containing missing values (`position\_stack()`).

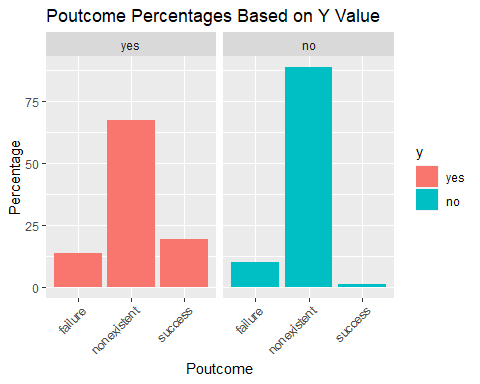
## Warning: Removed 1 rows containing missing values (`geom\_bar()`).



# Plot poutcome  
summary <- train\_data %>%  
 group\_by(poutcome,y) %>%  
 summarize(count=n())

## `summarise()` has grouped output by 'poutcome'. You can override using the  
## `.groups` argument.

summary$perc <- 0  
summary$perc[summary$y == 'no'] <- summary$count[summary$y == 'no'] / nrow(train\_data[train\_data$y == 'no',]) \* 100  
summary$perc[summary$y == 'yes'] <- summary$count[summary$y == 'yes'] / nrow(train\_data[train\_data$y == 'yes',]) \* 100  
summary %>% ggplot(aes(x=poutcome,y=perc,fill=y)) + geom\_bar(stat="identity") + facet\_wrap(~y) +   
 ylab('Percentage') + xlab('Poutcome') + ggtitle('Poutcome Percentages Based on Y Value') +   
 theme(axis.text.x = element\_text(angle = 45, hjust = 1))

 Campaign distributions seem pretty similar between Yes and No. For Poutcome, success is more common for the yes than for no.

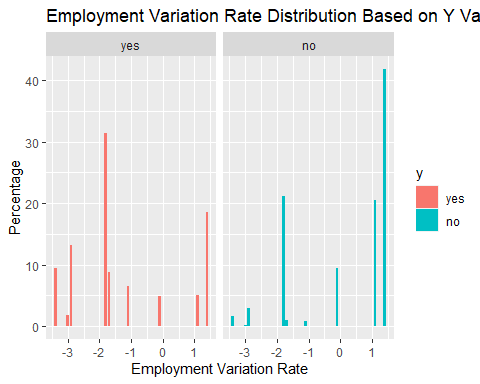
## Examine socio-economic data

There are variables for socio economic data for Employement Variation Rate, Consumer Price Index, Consumer Confidence Index, Euribor 3 month rate, and Numer of Employees.

# Plot emp.var.rate  
summary <- train\_data %>%  
 group\_by(emp.var.rate,y) %>%  
 summarize(count=n())

## `summarise()` has grouped output by 'emp.var.rate'. You can override using the  
## `.groups` argument.

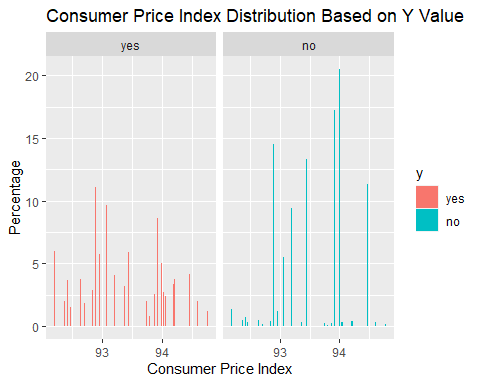
summary$perc <- 0  
summary$perc[summary$y == 'no'] <- summary$count[summary$y == 'no'] / nrow(train\_data[train\_data$y == 'no',]) \* 100  
summary$perc[summary$y == 'yes'] <- summary$count[summary$y == 'yes'] / nrow(train\_data[train\_data$y == 'yes',]) \* 100  
summary %>% ggplot(aes(x=emp.var.rate,y=perc,fill=y)) + geom\_bar(stat="identity") + facet\_wrap(~y) +   
 ylab('Percentage') + xlab('Employment Variation Rate') + ggtitle('Employment Variation Rate Distribution Based on Y Value')



# Plot cons.price.idx  
summary <- train\_data %>%  
 group\_by(cons.price.idx,y) %>%  
 summarize(count=n())

## `summarise()` has grouped output by 'cons.price.idx'. You can override using  
## the `.groups` argument.

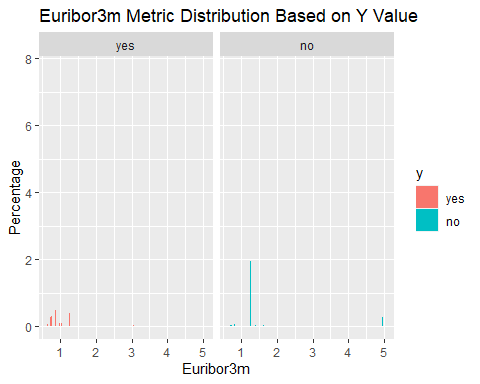
summary$perc <- 0  
summary$perc[summary$y == 'no'] <- summary$count[summary$y == 'no'] / nrow(train\_data[train\_data$y == 'no',]) \* 100  
summary$perc[summary$y == 'yes'] <- summary$count[summary$y == 'yes'] / nrow(train\_data[train\_data$y == 'yes',]) \* 100  
summary %>% ggplot(aes(x=cons.price.idx,y=perc,fill=y)) + geom\_bar(stat="identity") + facet\_wrap(~y) +   
 ylab('Percentage') + xlab('Consumer Price Index') + ggtitle('Consumer Price Index Distribution Based on Y Value')



# Plot euribor3m  
summary <- train\_data %>%  
 group\_by(euribor3m,y) %>%  
 summarize(count=n())

## `summarise()` has grouped output by 'euribor3m'. You can override using the  
## `.groups` argument.

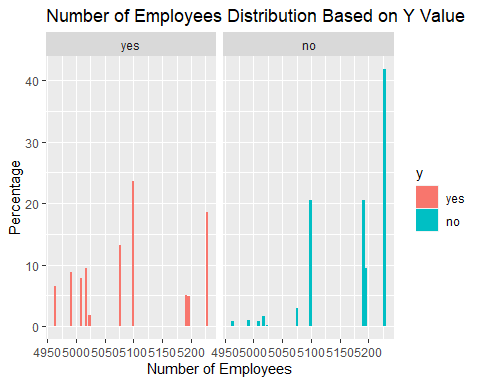
summary$perc <- 0  
summary$perc[summary$y == 'no'] <- summary$count[summary$y == 'no'] / nrow(train\_data[train\_data$y == 'no',]) \* 100  
summary$perc[summary$y == 'yes'] <- summary$count[summary$y == 'yes'] / nrow(train\_data[train\_data$y == 'yes',]) \* 100  
summary %>% ggplot(aes(x=euribor3m,y=perc,fill=y)) + geom\_bar(stat="identity") + facet\_wrap(~y) +   
 ylab('Percentage') + xlab('Euribor3m') + ggtitle('Euribor3m Metric Distribution Based on Y Value')



# Plot nr.employed  
summary <- train\_data %>%  
 group\_by(nr.employed,y) %>%  
 summarize(count=n())

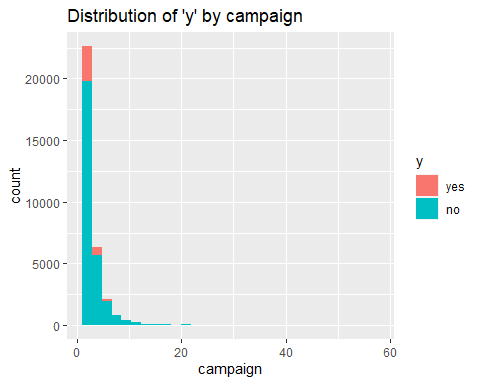
## `summarise()` has grouped output by 'nr.employed'. You can override using the  
## `.groups` argument.

summary$perc <- 0  
summary$perc[summary$y == 'no'] <- summary$count[summary$y == 'no'] / nrow(train\_data[train\_data$y == 'no',]) \* 100  
summary$perc[summary$y == 'yes'] <- summary$count[summary$y == 'yes'] / nrow(train\_data[train\_data$y == 'yes',]) \* 100  
summary %>% ggplot(aes(x=nr.employed,y=perc,fill=y)) + geom\_bar(stat="identity") + facet\_wrap(~y) +   
 ylab('Percentage') + xlab('Number of Employees') + ggtitle('Number of Employees Distribution Based on Y Value')

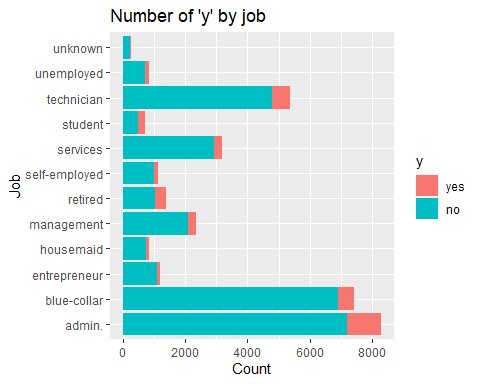


### Analyzing campaign  
  
ggplot(train\_data) +   
 geom\_histogram(mapping = aes(x=campaign, fill=y)) +   
 ggtitle("Distribution of 'y' by campaign")

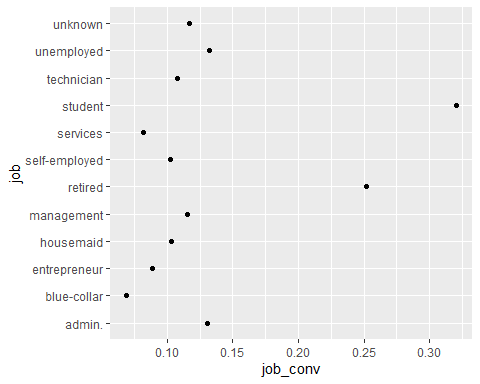
## `stat\_bin()` using `bins = 30`. Pick better value with `binwidth`.



### Analyzing JOB  
  
ggplot(train\_data) +   
 geom\_bar(mapping = aes(x=job, fill = y)) +   
 coord\_flip() + #Added coord flip here to make it more readable  
 ggtitle("Number of 'y' by job") +   
 ylab("Count") +   
 xlab("Job")

 Admin, technician and blue collar jobs are the top 3 subscribers by volume

df2 <- train\_data %>%   
 group\_by(job) %>%   
 count(y) %>%   
 mutate(job\_conv = n/sum(n)) %>%   
 filter(y == "yes")  
  
ggplot(df2, aes(x=job, y=job\_conv)) +   
 geom\_point() +   
 coord\_flip()

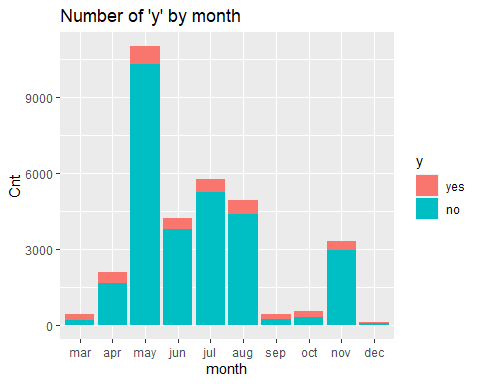


Above, I looked at the ratio of “yes” vs “no” and see that students and retired persons convert at much higher rates than those of other professions. And ‘blue collar’ has the lowest conversion rate

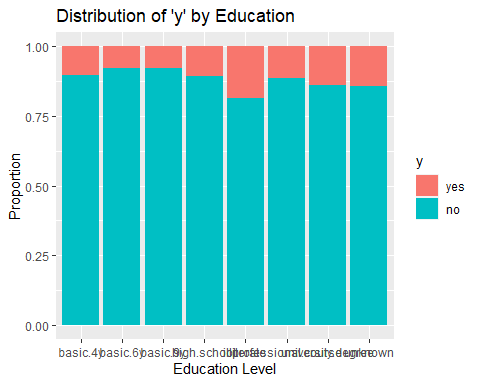
So, if they were to want to improve the cost effectiveness of their campaigns they might want to target more ‘students’ and ‘retirees’

### Analyzing By Month

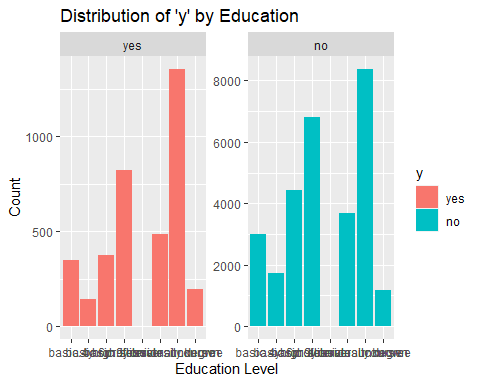
ggplot(train\_data) +   
 geom\_bar(mapping = aes(x=month, fill = y)) +   
 ggtitle("Number of 'y' by month") +  
 ylab("Cnt") + xlab("month")



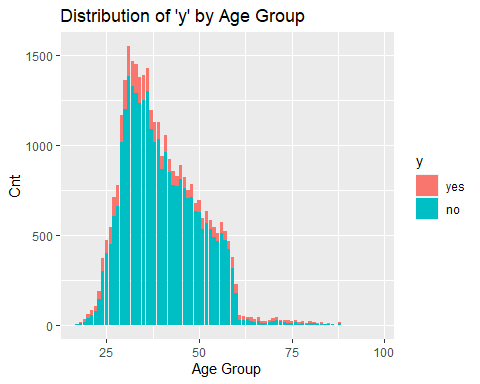
#Education  
ggplot(train\_data, aes(x = education, fill = y)) +   
 geom\_bar(position = "fill") +   
 ggtitle("Distribution of 'y' by Education") +   
 ylab("Proportion") +   
 xlab("Education Level")



ggplot(train\_data, aes(x = education, fill = y)) +   
 geom\_bar() +   
 facet\_wrap(~ y, scales = "free\_y") +   
 ggtitle("Distribution of 'y' by Education") +   
 ylab("Count") +   
 xlab("Education Level")

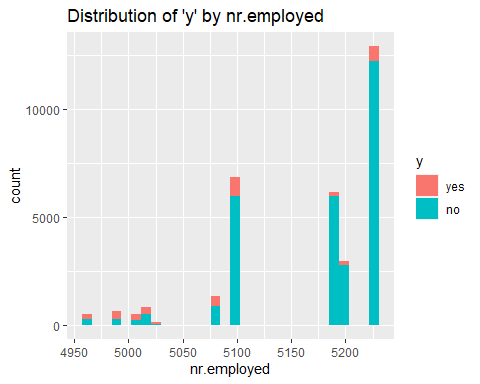


#Age  
ggplot(train\_data) +   
 geom\_bar(mapping = aes(x=age, fill = y)) +   
 ggtitle("Distribution of 'y' by Age Group") +   
 ylab("Cnt") +   
 xlab("Age Group")

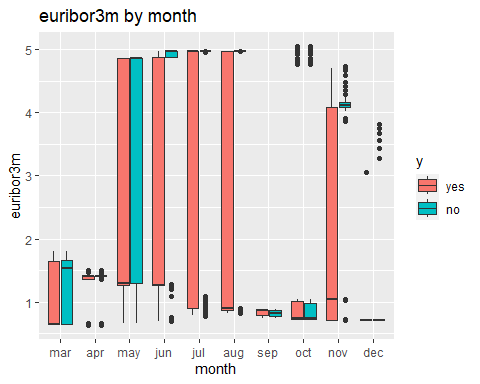


# by nr.employed  
ggplot(train\_data) + geom\_histogram(mapping = aes(x = nr.employed, fill = y)) +  
 ggtitle("Distribution of 'y' by nr.employed")

## `stat\_bin()` using `bins = 30`. Pick better value with `binwidth`.

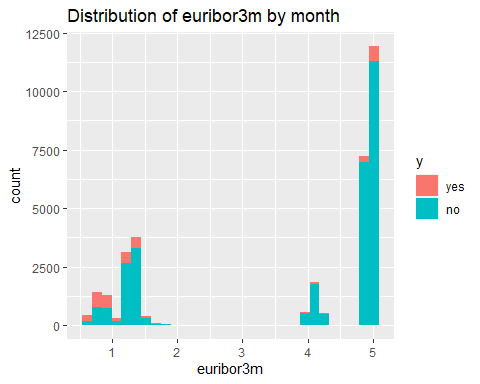


# Euribor 3 month rate  
ggplot(train\_data, aes(x = month , y = euribor3m, fill = y)) +   
 geom\_boxplot() +   
 ggtitle("euribor3m by month")



ggplot(train\_data) + geom\_histogram(mapping = aes(x = euribor3m, fill = y)) +  
 ggtitle("Distribution of euribor3m by month")

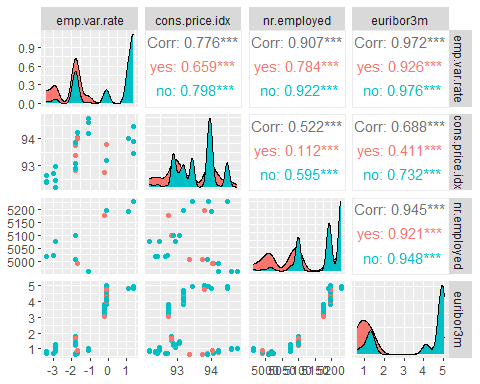
## `stat\_bin()` using `bins = 30`. Pick better value with `binwidth`.



# Correlations of socio-economic variables  
library(GGally)

## Registered S3 method overwritten by 'GGally':  
## method from   
## +.gg ggplot2

ggpairs(train\_data[,c('emp.var.rate','cons.price.idx','nr.employed','euribor3m')],aes(color = train\_data$y))

 The Consumer Price Index data seems to have more of a flat distribution for those with a term deposit. Employment Variation Rate, Euribor3m, and Number Employed all seem to have similar distributions where there is a higher percentage without term deposits for higher values of the index.

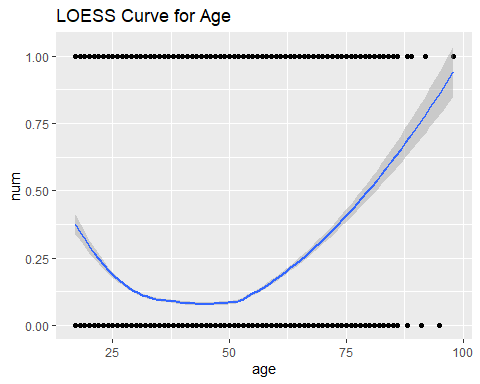
Looking at the paired correlations between out of the socio economic variables, we can see that Employment Variation Rate, Euribor3m, and Number Employed are highly correlated.

### LOESS Curves

LOESS curves can be useful to plot for numeric variables. They show if there is a general trend to higher or lower probabilities if the numeric variable increases. If they increase and then decrease, or vice-versa, this shows that the relationship between the two variables isn’t quite as simple.

# LOESS for Age  
train\_data$num <- ifelse(train\_data$y=="yes",1,0)  
train\_data %>% ggplot(aes(x=age,y=num)) +   
 geom\_point() + geom\_smooth(method="loess") +   
 ggtitle('LOESS Curve for Age')

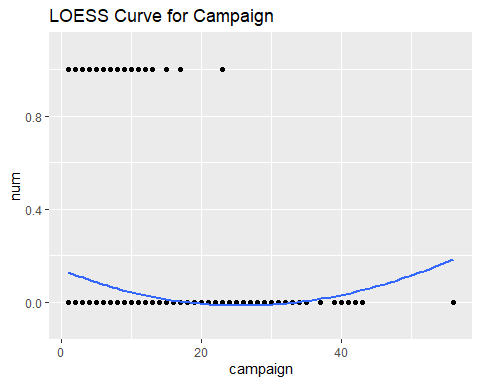
## `geom\_smooth()` using formula = 'y ~ x'



# LOESS for Campaign  
train\_data$num <- ifelse(train\_data$y=="yes",1,0)  
train\_data %>% ggplot(aes(x=campaign,y=num)) +   
 geom\_point() + geom\_smooth(method="loess", size = 1, span = 2, se = FALSE) +   
 ggtitle('LOESS Curve for Campaign') + ylim(c(-.1,1.1))

## Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.  
## ℹ Please use `linewidth` instead.  
## This warning is displayed once every 8 hours.  
## Call `lifecycle::last\_lifecycle\_warnings()` to see where this warning was  
## generated.

## `geom\_smooth()` using formula = 'y ~ x'



# LOESS for nr.employed   
train\_data %>% ggplot(aes(x=nr.employed,y=num)) +   
 geom\_point() + geom\_smooth(method="loess",span=.5) +   
 ggtitle('LOESS Curve for nr.employed') +   
 ylim(c(-.1,1.1))

## `geom\_smooth()` using formula = 'y ~ x'

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric = parametric,  
## : pseudoinverse used at 5228.1

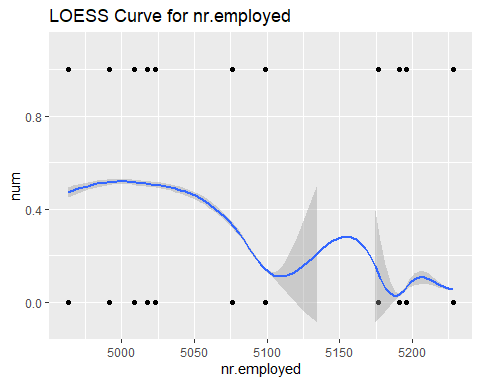
## Warning in simpleLoess(y, x, w, span, degree = degree, parametric = parametric,  
## : neighborhood radius 37.1

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric = parametric,  
## : reciprocal condition number 1.4919e-14

## Warning in predLoess(object$y, object$x, newx = if (is.null(newdata)) object$x  
## else if (is.data.frame(newdata))  
## as.matrix(model.frame(delete.response(terms(object)), : pseudoinverse used at  
## 5228.1

## Warning in predLoess(object$y, object$x, newx = if (is.null(newdata)) object$x  
## else if (is.data.frame(newdata))  
## as.matrix(model.frame(delete.response(terms(object)), : neighborhood radius  
## 37.1

## Warning in predLoess(object$y, object$x, newx = if (is.null(newdata)) object$x  
## else if (is.data.frame(newdata))  
## as.matrix(model.frame(delete.response(terms(object)), : reciprocal condition  
## number 1.4919e-14



# LOESS for emp.var.rate   
train\_data %>% ggplot(aes(x=emp.var.rate,y=num)) +   
 geom\_point() + geom\_smooth(method="loess",span=.5) +   
 ggtitle('LOESS Curve for emp.var.rate')

## `geom\_smooth()` using formula = 'y ~ x'

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric = parametric,  
## : pseudoinverse used at 1.424

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric = parametric,  
## : neighborhood radius 0.324

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric = parametric,  
## : reciprocal condition number 1.0203e-26

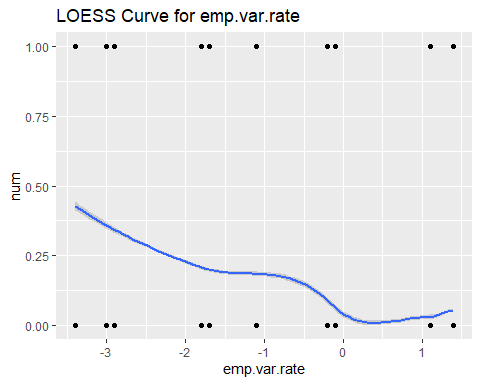
## Warning in simpleLoess(y, x, w, span, degree = degree, parametric = parametric,  
## : There are other near singularities as well. 0.09

## Warning in predLoess(object$y, object$x, newx = if (is.null(newdata)) object$x  
## else if (is.data.frame(newdata))  
## as.matrix(model.frame(delete.response(terms(object)), : pseudoinverse used at  
## 1.424

## Warning in predLoess(object$y, object$x, newx = if (is.null(newdata)) object$x  
## else if (is.data.frame(newdata))  
## as.matrix(model.frame(delete.response(terms(object)), : neighborhood radius  
## 0.324

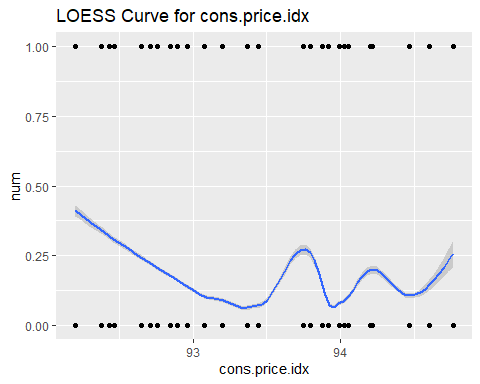
## Warning in predLoess(object$y, object$x, newx = if (is.null(newdata)) object$x  
## else if (is.data.frame(newdata))  
## as.matrix(model.frame(delete.response(terms(object)), : reciprocal condition  
## number 1.0203e-26

## Warning in predLoess(object$y, object$x, newx = if (is.null(newdata)) object$x  
## else if (is.data.frame(newdata))  
## as.matrix(model.frame(delete.response(terms(object)), : There are other near  
## singularities as well. 0.09



# LOESS for cons.price.idx   
train\_data %>% ggplot(aes(x=cons.price.idx,y=num)) +   
 geom\_point() + geom\_smooth(method="loess",span=.5) +   
 ggtitle('LOESS Curve for cons.price.idx')

## `geom\_smooth()` using formula = 'y ~ x'

 The LOESS plots for Age and Campaign show that No becomes more likely as Age and Campaign increase, and then less likely. However, the LOESS plots for nr.employed, emp.var.rate, and cons.price.idx show a general downward trend toward a higher likelihood of No as those values increase.

# LOESS for nr.employed by Month  
train\_data %>% ggplot(aes(x=nr.employed,y=num,color = month)) +   
 geom\_point() + geom\_smooth(method="loess", size = 1, span=1.1) +   
 ggtitle('LOESS Curve for nr.employed by Month') +   
 ylim(c(-.1,1.1))

## `geom\_smooth()` using formula = 'y ~ x'

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric = parametric,  
## : pseudoinverse used at 5008.2

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric = parametric,  
## : neighborhood radius 95.286

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric = parametric,  
## : reciprocal condition number 2.775e-15

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric = parametric,  
## : There are other near singularities as well. 9079.5

## Warning in predLoess(object$y, object$x, newx = if (is.null(newdata)) object$x  
## else if (is.data.frame(newdata))  
## as.matrix(model.frame(delete.response(terms(object)), : pseudoinverse used at  
## 5008.2

## Warning in predLoess(object$y, object$x, newx = if (is.null(newdata)) object$x  
## else if (is.data.frame(newdata))  
## as.matrix(model.frame(delete.response(terms(object)), : neighborhood radius  
## 95.286

## Warning in predLoess(object$y, object$x, newx = if (is.null(newdata)) object$x  
## else if (is.data.frame(newdata))  
## as.matrix(model.frame(delete.response(terms(object)), : reciprocal condition  
## number 2.775e-15

## Warning in predLoess(object$y, object$x, newx = if (is.null(newdata)) object$x  
## else if (is.data.frame(newdata))  
## as.matrix(model.frame(delete.response(terms(object)), : There are other near  
## singularities as well. 9079.5

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric = parametric,  
## : pseudoinverse used at 5008.2

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric = parametric,  
## : neighborhood radius 95.286

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric = parametric,  
## : reciprocal condition number 5.8467e-15

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric = parametric,  
## : There are other near singularities as well. 9079.5

## Warning in predLoess(object$y, object$x, newx = if (is.null(newdata)) object$x  
## else if (is.data.frame(newdata))  
## as.matrix(model.frame(delete.response(terms(object)), : pseudoinverse used at  
## 5008.2

## Warning in predLoess(object$y, object$x, newx = if (is.null(newdata)) object$x  
## else if (is.data.frame(newdata))  
## as.matrix(model.frame(delete.response(terms(object)), : neighborhood radius  
## 95.286

## Warning in predLoess(object$y, object$x, newx = if (is.null(newdata)) object$x  
## else if (is.data.frame(newdata))  
## as.matrix(model.frame(delete.response(terms(object)), : reciprocal condition  
## number 5.8467e-15

## Warning in predLoess(object$y, object$x, newx = if (is.null(newdata)) object$x  
## else if (is.data.frame(newdata))  
## as.matrix(model.frame(delete.response(terms(object)), : There are other near  
## singularities as well. 9079.5

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric = parametric,  
## : Chernobyl! trL<k 3  
  
## Warning in simpleLoess(y, x, w, span, degree = degree, parametric = parametric,  
## : Chernobyl! trL<k 3

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric = parametric,  
## : pseudoinverse used at 4963.3

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric = parametric,  
## : neighborhood radius 56.813

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric = parametric,  
## : reciprocal condition number 3.1301e-15

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric = parametric,  
## : There are other near singularities as well. 3227.8

## Warning in predLoess(object$y, object$x, newx = if (is.null(newdata)) object$x  
## else if (is.data.frame(newdata))  
## as.matrix(model.frame(delete.response(terms(object)), : pseudoinverse used at  
## 4963.3

## Warning in predLoess(object$y, object$x, newx = if (is.null(newdata)) object$x  
## else if (is.data.frame(newdata))  
## as.matrix(model.frame(delete.response(terms(object)), : neighborhood radius  
## 56.813

## Warning in predLoess(object$y, object$x, newx = if (is.null(newdata)) object$x  
## else if (is.data.frame(newdata))  
## as.matrix(model.frame(delete.response(terms(object)), : reciprocal condition  
## number 3.1301e-15

## Warning in predLoess(object$y, object$x, newx = if (is.null(newdata)) object$x  
## else if (is.data.frame(newdata))  
## as.matrix(model.frame(delete.response(terms(object)), : There are other near  
## singularities as well. 3227.8

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric = parametric,  
## : Chernobyl! trL<k 3  
  
## Warning in simpleLoess(y, x, w, span, degree = degree, parametric = parametric,  
## : Chernobyl! trL<k 3  
  
## Warning in simpleLoess(y, x, w, span, degree = degree, parametric = parametric,  
## : Chernobyl! trL<k 3  
  
## Warning in simpleLoess(y, x, w, span, degree = degree, parametric = parametric,  
## : Chernobyl! trL<k 3

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric = parametric,  
## : pseudoinverse used at 5022.7

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric = parametric,  
## : neighborhood radius 161.06

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric = parametric,  
## : reciprocal condition number 0

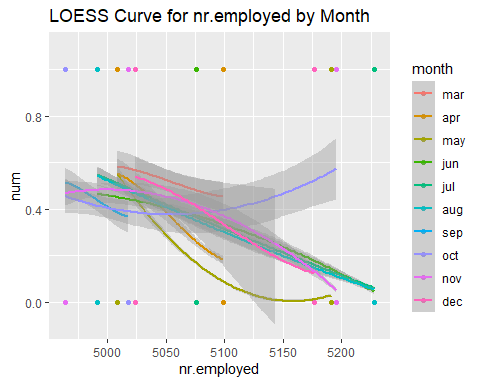
## Warning in simpleLoess(y, x, w, span, degree = degree, parametric = parametric,  
## : There are other near singularities as well. 25940

## Warning in predLoess(object$y, object$x, newx = if (is.null(newdata)) object$x  
## else if (is.data.frame(newdata))  
## as.matrix(model.frame(delete.response(terms(object)), : pseudoinverse used at  
## 5022.7

## Warning in predLoess(object$y, object$x, newx = if (is.null(newdata)) object$x  
## else if (is.data.frame(newdata))  
## as.matrix(model.frame(delete.response(terms(object)), : neighborhood radius  
## 161.06

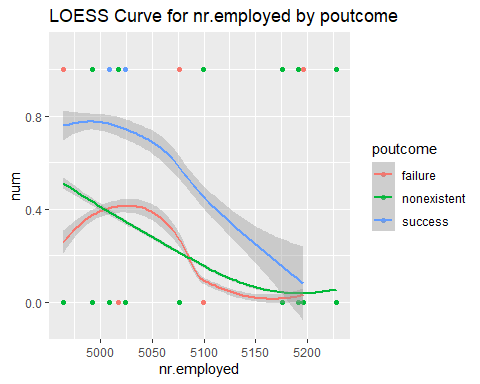
## Warning in predLoess(object$y, object$x, newx = if (is.null(newdata)) object$x  
## else if (is.data.frame(newdata))  
## as.matrix(model.frame(delete.response(terms(object)), : reciprocal condition  
## number 0

## Warning in predLoess(object$y, object$x, newx = if (is.null(newdata)) object$x  
## else if (is.data.frame(newdata))  
## as.matrix(model.frame(delete.response(terms(object)), : There are other near  
## singularities as well. 25940



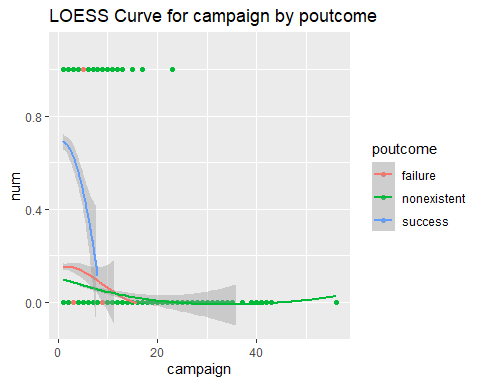
# LOESS for nr.employed by poutcome  
train\_data %>% ggplot(aes(x=nr.employed,y=num,color = poutcome)) +   
 geom\_point() + geom\_smooth(method="loess", size = 1, span = 1) +   
 ggtitle('LOESS Curve for nr.employed by poutcome') +   
 ylim(c(-.1,1.1))

## `geom\_smooth()` using formula = 'y ~ x'



# LOESS for campaign by poutcome  
train\_data %>% ggplot(aes(x=campaign,y=num,color = poutcome)) +   
 geom\_point() + geom\_smooth(method="loess", size = 1, span = 1.1) +   
 ggtitle('LOESS Curve for campaign by poutcome') +   
 ylim(c(-.1,1.1))

## `geom\_smooth()` using formula = 'y ~ x'

 The LOESS curves for these variable combinations show that using these variables in the same model or even as an interaction between these two variables could be useful.

# LOESS for nr.employed squared  
train\_data %>% ggplot(aes(x=(nr.employed)^2,y=num)) +   
 geom\_point() + geom\_smooth(method="loess",span=.5) +   
 ggtitle('LOESS Curve for nr.employed Squared') +   
 ylim(c(-.1,1.1))

## `geom\_smooth()` using formula = 'y ~ x'

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric = parametric,  
## : pseudoinverse used at 2.7333e+07

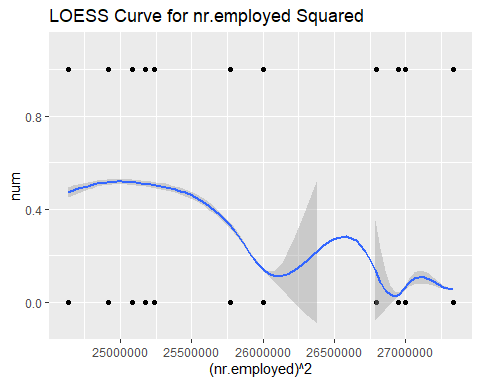
## Warning in simpleLoess(y, x, w, span, degree = degree, parametric = parametric,  
## : neighborhood radius 3.8655e+05

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric = parametric,  
## : reciprocal condition number 2.0272e-14

## Warning in predLoess(object$y, object$x, newx = if (is.null(newdata)) object$x  
## else if (is.data.frame(newdata))  
## as.matrix(model.frame(delete.response(terms(object)), : pseudoinverse used at  
## 2.7333e+07

## Warning in predLoess(object$y, object$x, newx = if (is.null(newdata)) object$x  
## else if (is.data.frame(newdata))  
## as.matrix(model.frame(delete.response(terms(object)), : neighborhood radius  
## 3.8655e+05

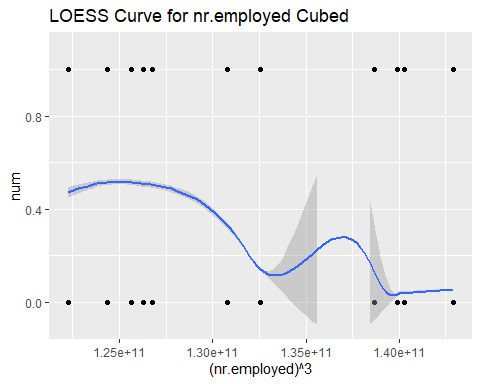
## Warning in predLoess(object$y, object$x, newx = if (is.null(newdata)) object$x  
## else if (is.data.frame(newdata))  
## as.matrix(model.frame(delete.response(terms(object)), : reciprocal condition  
## number 2.0272e-14



# LOESS for nr.employed cubed  
train\_data %>% ggplot(aes(x=(nr.employed)^3,y=num)) +   
 geom\_point() + geom\_smooth(method="loess",span=.5) +   
 ggtitle('LOESS Curve for nr.employed Cubed') +   
 ylim(c(-.1,1.1))

## `geom\_smooth()` using formula = 'y ~ x'

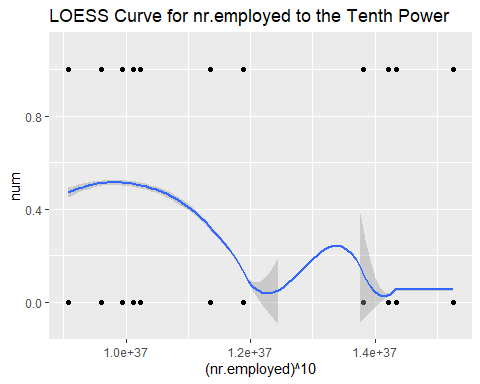
## Warning: Removed 10 rows containing missing values (`geom\_smooth()`).



# LOESS for nr.employed tenth power  
train\_data %>% ggplot(aes(x=(nr.employed)^10,y=num)) +   
 geom\_point() + geom\_smooth(method="loess",span=.5) +   
 ggtitle('LOESS Curve for nr.employed to the Tenth Power') +   
 ylim(c(-.1,1.1))

## `geom\_smooth()` using formula = 'y ~ x'

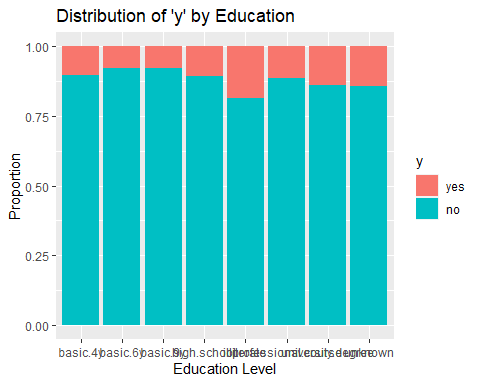
## Warning: Removed 11 rows containing missing values (`geom\_smooth()`).

 The LOESS curves for powers of nr.employed (particularly comparing the tenth power to the first or second power) seem to improve as the power increases. This points to adding polynomial complexity terms to the model could be useful.

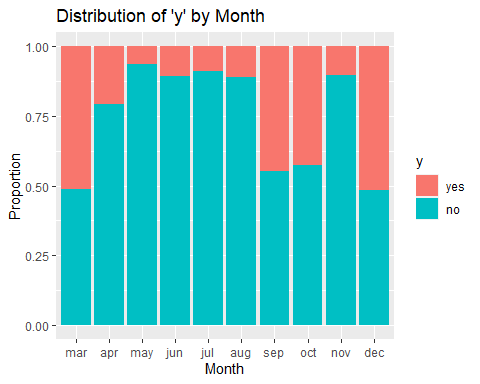
### Percentage Plots

It can be helpful to look at plots that sum up to 100% for Yes and No results for categorical variables. This can show that certain values have a higher or lower percentage of Yes results.

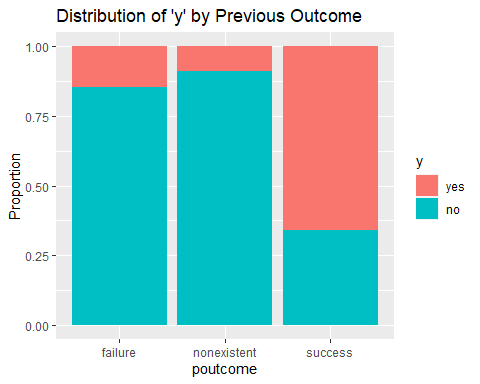
# Percentage plot for education  
ggplot(train\_data, aes(x = education, fill = y)) +   
 geom\_bar(position = "fill") +   
 ggtitle("Distribution of 'y' by Education") +   
 ylab("Proportion") +   
 xlab("Education Level")



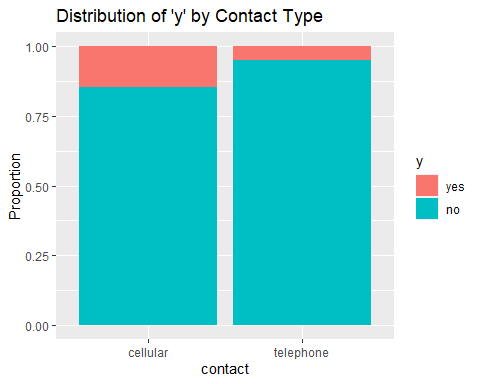
# Percentage plot for month  
ggplot(train\_data, aes(x = month, fill = y)) +   
 geom\_bar(position = "fill") +   
 ggtitle("Distribution of 'y' by Month") +   
 ylab("Proportion") +   
 xlab("Month")



# Percentage plot for poutcome  
ggplot(train\_data, aes(x = poutcome, fill = y)) +   
 geom\_bar(position = "fill") +   
 ggtitle("Distribution of 'y' by Previous Outcome") +   
 ylab("Proportion") +   
 xlab("poutcome")



# Percentage plot for contact  
ggplot(train\_data, aes(x = contact, fill = y)) +   
 geom\_bar(position = "fill") +   
 ggtitle("Distribution of 'y' by Contact Type") +   
 ylab("Proportion") +   
 xlab("contact")

 Education shows very similar Yes/No percentages across the different Education Types, which indicates that it won’t be a useful variable for model building. Month and poutcome have 1 or more values with very high values of yes, which could be useful for model building. Contact seems to have one category with roughly double the number of yes, so it might be slightlty useful in model building.

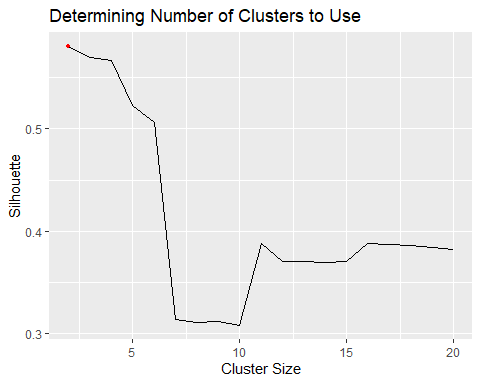
## Clustering

We wanted to see if a clustering analysis of the data would be useful. We only looked at the numeric variables. The purpose of a clustering analysis is to see if splitting the data into clusters allows for additional insight, particularly into classification.

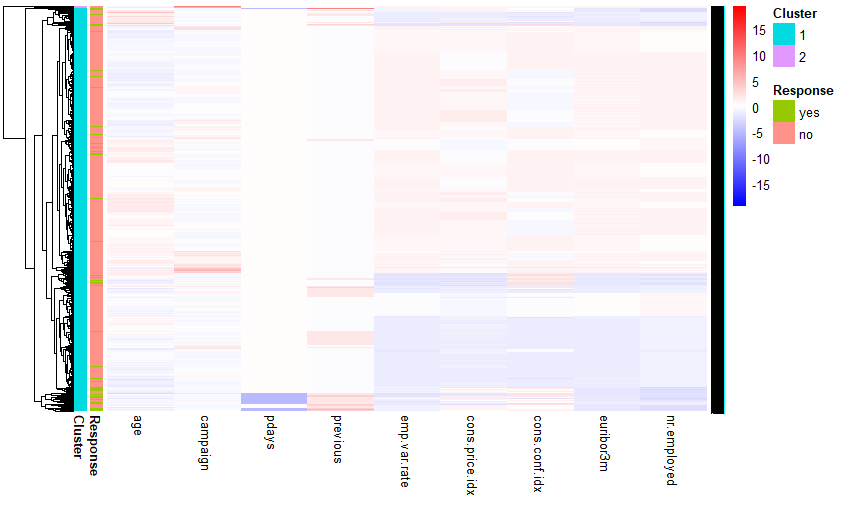
First, we will try to determine the number of clusters to use. The metric we used for comparing cluster sizes is the Silhoutte Statistic. Below is the code I ran, but it has trouble knitting. The heat maps are in the PPT though.

library(RColorBrewer)  
library(pheatmap)  
library(cluster)  
df.numeric <- train\_data[ , sapply(train\_data, is.numeric)]  
center.scale=scale(df.numeric)  
mydist<-dist(center.scale)  
sim.clust<-hclust(mydist,method="complete")  
max\_clusters <- 20  
my.sil<-c()  
for (i in 2:max\_clusters){  
 print(i)  
 sil.result<-silhouette(cutree(sim.clust,i),mydist)  
 my.sil[i-1]<-summary(sil.result)$avg.width  
}

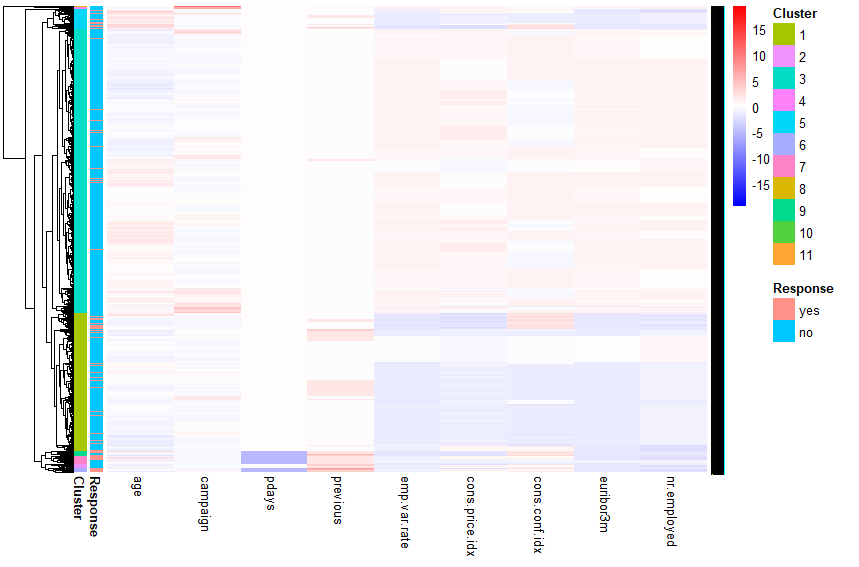
max\_clusters <- 20  
my.sil <- read.csv('https://raw.githubusercontent.com/stedua22/6372-Project-2/main/my\_sil.csv')  
my.sil <- my.sil$x  
ggplot(data = data.frame(x=2:max\_clusters, y=my.sil),aes(x=x,y=y)) + geom\_line() +   
 ylab('Silhouette') + xlab('Cluster Size') + ggtitle('Determining Number of Clusters to Use') +   
 geom\_point(data = data.frame(x = 2, y = my.sil[1]), aes(x = x, y = y), size = 1, color = "red", fill = "red", shape = 21)

 It looks like 2 clusters is the highest, so we will try that.

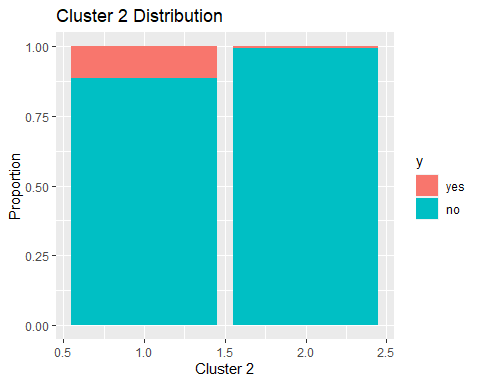
num\_clusters <- 2  
rownames(df.numeric)<-paste("R",1:nrow(df.numeric),sep="")  
annotation\_row<-data.frame(Response=factor(train\_data$y),Cluster=factor(cutree(sim.clust,num\_clusters)))  
rownames(annotation\_row)<-rownames(df.numeric)  
pheatmap(df.numeric,annotation\_row=annotation\_row,cluster\_cols=F,scale="column",fontsize\_row=3,legend=T  
 ,color=colorRampPalette(c("blue","white", "red"), space = "rgb")(100))

 One of the clusters is huge and the other is tiny. Let’s try with 11 clusters.

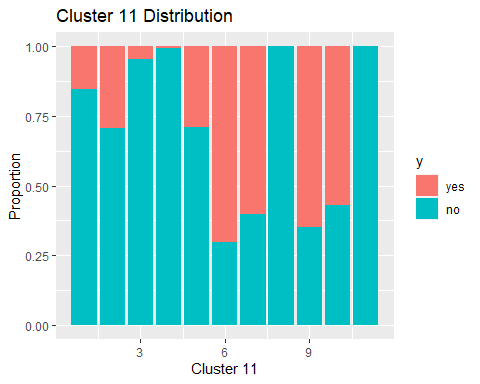
num\_clusters <- 11  
rownames(df.numeric)<-paste("R",1:nrow(df.numeric),sep="")  
annotation\_row<-data.frame(Response=factor(train\_data$y),Cluster=factor(cutree(sim.clust,num\_clusters)))  
rownames(annotation\_row)<-rownames(df.numeric)  
pheatmap(df.numeric,annotation\_row=annotation\_row,cluster\_cols=F,scale="column",fontsize\_row=3,legend=T  
 ,color=colorRampPalette(c("blue","white", "red"), space = "rgb")(100))

 This is a little bit more interesting. Some clusters had a lot of ‘yes’ results. Others seem to be the same distribution as before.

cluster2 <- read.csv('https://raw.githubusercontent.com/stedua22/6372-Project-2/main/cluster2.csv')  
cluster11 <- read.csv('https://raw.githubusercontent.com/stedua22/6372-Project-2/main/cluster11.csv')  
train\_data$cluster2 <- cluster2$x  
train\_data$cluster11 <- cluster11$x  
  
ggplot(train\_data, aes(x = cluster2, fill = y)) +   
 geom\_bar(position = "fill") +   
 ggtitle("Cluster 2 Distribution") +   
 ylab("Proportion") +   
 xlab("Cluster 2")



ggplot(train\_data, aes(x = cluster11, fill = y)) +   
 geom\_bar(position = "fill") +   
 ggtitle("Cluster 11 Distribution") +   
 ylab("Proportion") +   
 xlab("Cluster 11")

 The 11 cluster example does seem like it does a reasonable job of picking out clusters with

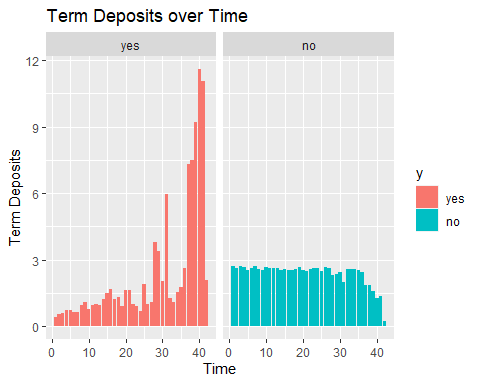
## Variables over time

We want to see if there was any variation in term deposits over time. The file explaining the dataset mentioned that the data was in order of date.

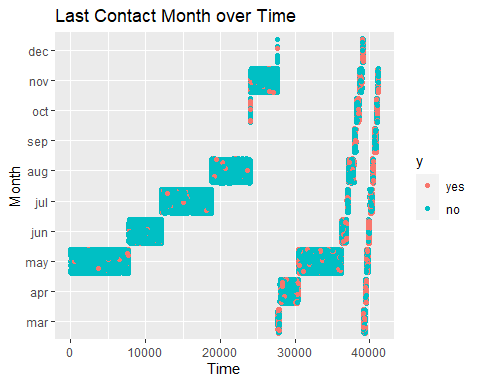
# Sorting data by row number  
train\_data\_sorted <- train\_data  
train\_data\_sorted$num <- as.numeric(rownames(train\_data\_sorted))  
train\_data\_sorted$group <- ceiling(train\_data\_sorted$num / 1000)  
summary <- train\_data\_sorted %>%  
 group\_by(group,y) %>%  
 summarize(count=n())

## `summarise()` has grouped output by 'group'. You can override using the  
## `.groups` argument.

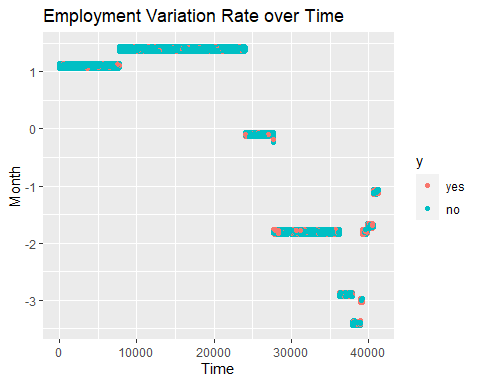
summary$perc <- 0  
summary$perc[summary$y == 'no'] <- summary$count[summary$y == 'no'] / nrow(train\_data\_sorted[train\_data\_sorted$y == 'no',]) \* 100  
summary$perc[summary$y == 'yes'] <- summary$count[summary$y == 'yes'] / nrow(train\_data\_sorted[train\_data\_sorted$y == 'yes',]) \* 100  
  
# Term deposit over time  
summary %>% ggplot(aes(x=group,y=perc,fill=y)) + geom\_bar(stat="identity") + facet\_wrap(~y) +   
 ylab('Term Deposits') + xlab('Time') + ggtitle('Term Deposits over Time')



# Month over time  
train\_data\_sorted %>% ggplot(aes(x=num, y=month, color=y)) + geom\_jitter() +   
 xlab('Time') + ylab('Month') + ggtitle('Last Contact Month over Time')



# Employment Variation Rate over time  
train\_data\_sorted %>% ggplot(aes(x=num, y=emp.var.rate, color=y)) + geom\_jitter() +   
 xlab('Time') + ylab('Month') + ggtitle('Employment Variation Rate over Time')



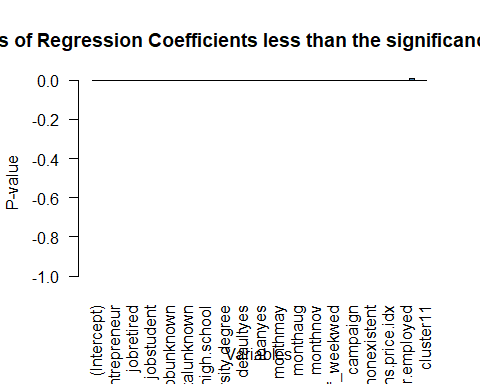
#### LOOKING AT THE PVALUE DISTRIBUTIONS

Looking at how each variable in the model, significantly impacts our response variable

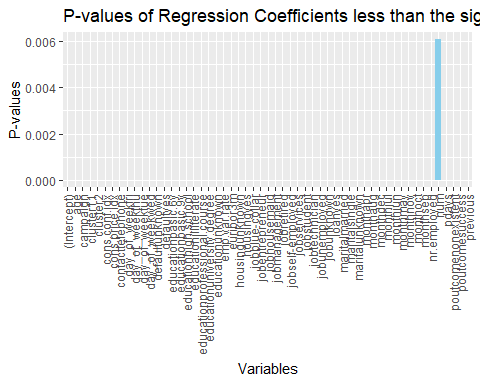
log.model <-glm(y ~ . ,data = train\_data,family="binomial")

## Warning: glm.fit: algorithm did not converge

# Extract variable names  
variable\_names <- rownames(summary(log.model)$coefficients)  
  
# Setting the levels back  
data$y <- relevel(data$y, ref="yes")  
  
# getting the p-values from the model3  
p\_values <- summary(log.model)$coefficients[, 4] # Assuming p-values are in the 4th column of the summary table  
p\_values <- data.frame(p\_values)$p\_values  
  
df <- data.frame(variable\_names, p\_values) #combining the pvalues and variable names into a dataframe  
  
df <- df[!df$p\_value == 0 , ] #removing varaiables with pvalue = 0  
df$p\_values <- log(df$p\_values) \* -1  
  
barplot(df$p\_values,   
 main = "P-values of Regression Coefficients less than the significance level 0.05",   
 xlab = "Variables",   
 ylab = "P-value",  
 names.arg = df$variable\_names,  
 las = 2, # Rotate x-axis labels vertically for better readability  
 col = "steelblue", # Set color of bars  
 ylim = c(exp(0.05) \* -1, max(df$p\_values) \* 1.2) # Set ylim from the significance level to the max p-values  
   
)



library(ggplot2)  
ggplot(df,aes(variable\_names,p\_values, fill = ifelse(p\_values > (exp(0.05) \* -1), "Positive", "Negative"))) + #filtering just the highly significant p-values  
 geom\_bar(stat="identity", fill = "skyblue") +   
 #geom\_text(aes(label = variable\_names), vjust = -0.5) + # Add text labels on top of bars  
 scale\_fill\_manual(values = c("Positive" = "skyblue", "Negative" = "salmon")) +  
 labs(x = "Variables", y = "P-values", title = "P-values of Regression Coefficients less than the significance level 0.05") +  
 theme(axis.text.x = element\_text(angle = 90, vjust = 0.5, hjust=1)) # Rotate x-axis labels for better readability

 From the plot above, we can see that the top 5 highly significant values with respoct to the response variable y are months(most of them), poutcome, emp.var.rate, contact, cons.price.idx, which are similar to our selected simple logistic model

# PCA models

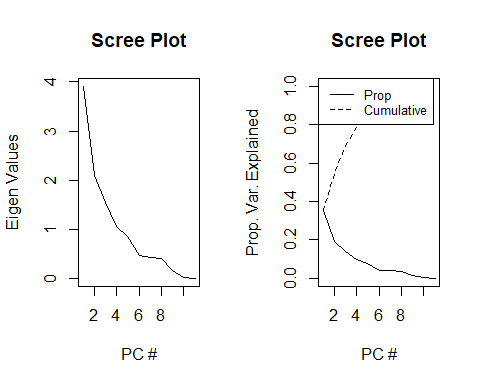
#make sure "success" level is defined as "yes"  
str(train\_data$y)

## Factor w/ 2 levels "yes","no": 1 2 2 2 2 2 1 2 2 2 ...

train\_data$num <- c()  
  
  
#PCA  
df.numericPC <- train\_data[ , sapply(train\_data, is.numeric)]  
pc.result<-prcomp(df.numericPC,scale.=TRUE)  
pc.scores<-pc.result$x  
pc.scores<-data.frame(pc.scores)  
pc.scores$y<-train\_data$y  
  
  
  
#Eignenvector Matrix  
View(pc.result$rotation)  
  
#Scree plot  
eigenvals<-(pc.result$sdev)^2  
eigenvals

## [1] 3.90935320 2.10748012 1.55067031 1.06927372 0.86991295 0.46536151  
## [7] 0.42482392 0.40536507 0.16265162 0.02466925 0.01043832

par(mfrow=c(1,2))  
plot(eigenvals,type="l",main="Scree Plot",ylab="Eigen Values",xlab="PC #")  
plot(eigenvals/sum(eigenvals),type="l",main="Scree Plot",ylab="Prop. Var. Explained",xlab="PC #",ylim=c(0,1))  
cumulative.prop<-cumsum(eigenvals/sum(eigenvals))  
lines(cumulative.prop,lty=2)  
legend("topleft", legend=c("Prop","Cumulative"),  
 lty=1:2, cex=0.8)



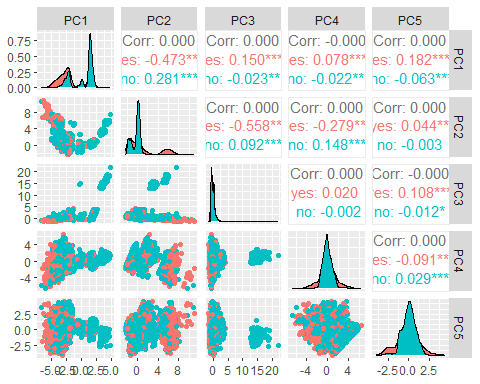
data.frame(PC=1:length(eigenvals),Prop=eigenvals/sum(eigenvals),Cumulative=cumulative.prop)

## PC Prop Cumulative  
## 1 1 0.3553957456 0.3553957  
## 2 2 0.1915891022 0.5469848  
## 3 3 0.1409700280 0.6879549  
## 4 4 0.0972067021 0.7851616  
## 5 5 0.0790829955 0.8642446  
## 6 6 0.0423055922 0.9065502  
## 7 7 0.0386203567 0.9451705  
## 8 8 0.0368513697 0.9820219  
## 9 9 0.0147865105 0.9968084  
## 10 10 0.0022426592 0.9990511  
## 11 11 0.0009489382 1.0000000

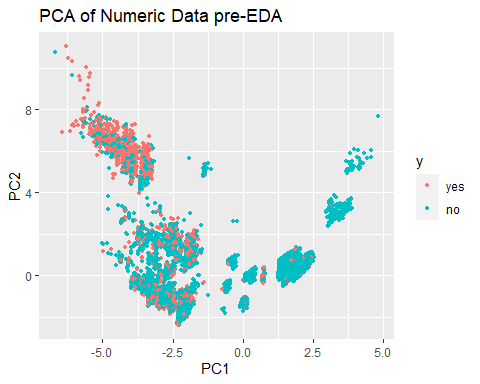
# Calculate the variance explained by each principal component  
var\_explained <- pc.result$sdev^2 / sum(pc.result$sdev^2)  
cum\_var\_explained <- cumsum(var\_explained)  
  
# Find the number of components that explain at least 90% of the variance  
num\_comp\_90 <- which(cum\_var\_explained >= 0.9)[1]  
  
# Print the number of components  
print(num\_comp\_90) #We would need 5 to retain approximately 90%

## [1] 6

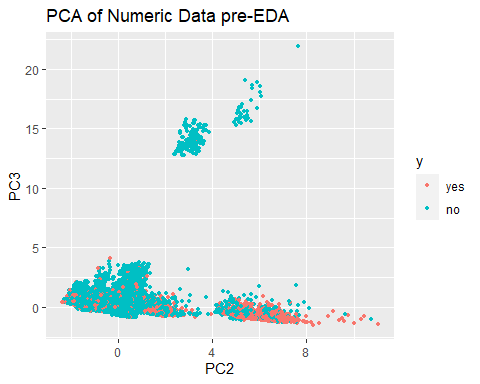
#Plotting PCA variables with the two colors:  
  
  
pc.result<-prcomp(df.numericPC,scale.=TRUE)  
PC <- data.frame(diagnosis = train\_data$y)  
PC$PC1 <- pc.result$x[,1]  
PC$PC2 <- pc.result$x[,2]  
PC$PC3 <- pc.result$x[,3]  
PC$PC4 <- pc.result$x[,4]  
PC$PC5 <- pc.result$x[,5]  
ggpairs(PC[,-1],aes(color=PC[,1]))



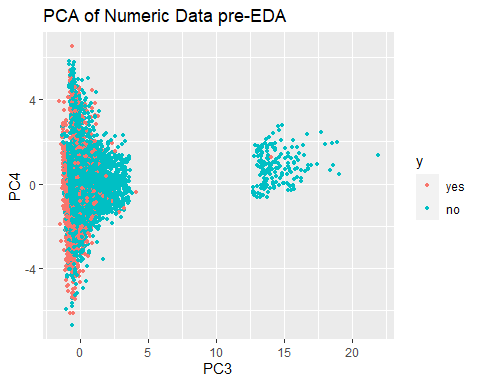
#Use ggplot2 to plot the first few pc's  
ggplot(data = pc.scores, aes(x = PC1, y = PC2)) +  
 geom\_point(aes(col=y), size=1)+  
 ggtitle("PCA of Numeric Data pre-EDA")



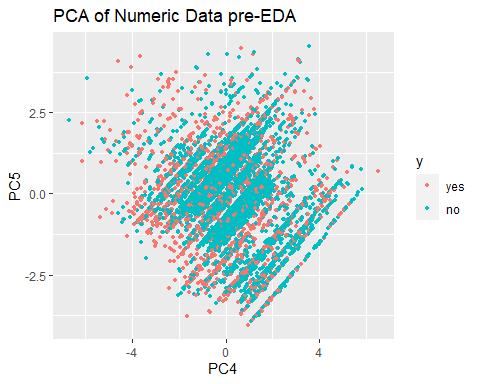
ggplot(data = pc.scores, aes(x = PC2, y = PC3)) +  
 geom\_point(aes(col=y), size=1)+  
 ggtitle("PCA of Numeric Data pre-EDA")



ggplot(data = pc.scores, aes(x = PC3, y = PC4)) +  
 geom\_point(aes(col=y), size=1)+  
 ggtitle("PCA of Numeric Data pre-EDA")



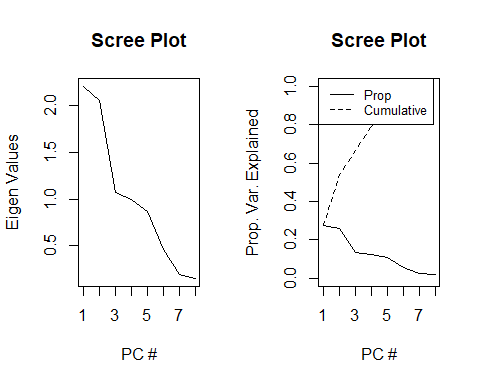
ggplot(data = pc.scores, aes(x = PC4, y = PC5)) +  
 geom\_point(aes(col=y), size=1)+  
 ggtitle("PCA of Numeric Data pre-EDA")



# PCA without campaign, euribor3m,and nr.employed as they are more like factors and not continuous  
  
  
#Performing PCA on predictors  
  
df.numeric2 <- df.numericPC[,-c(2,8,9)]  
pc.result2<-prcomp(df.numeric2,scale.=TRUE)  
pc.scores2<-pc.result2$x  
pc.scores2<-data.frame(pc.scores2)  
pc.scores2$y<-train\_data$y  
#pc.scores2  
  
  
#Eignenvector Matrix  
View(pc.result2$rotation)  
  
#Scree plot  
eigenvals2<-(pc.result2$sdev)^2  
eigenvals2

## [1] 2.2091986 2.0617699 1.0752687 0.9925676 0.8622411 0.4611968 0.1881167  
## [8] 0.1496406

par(mfrow=c(1,2))  
plot(eigenvals2,type="l",main="Scree Plot",ylab="Eigen Values",xlab="PC #")  
plot(eigenvals2/sum(eigenvals2),type="l",main="Scree Plot",ylab="Prop. Var. Explained",xlab="PC #",ylim=c(0,1))  
cumulative.prop<-cumsum(eigenvals2/sum(eigenvals2))  
lines(cumulative.prop,lty=2)  
legend("topleft", legend=c("Prop","Cumulative"),  
 lty=1:2, cex=0.8)



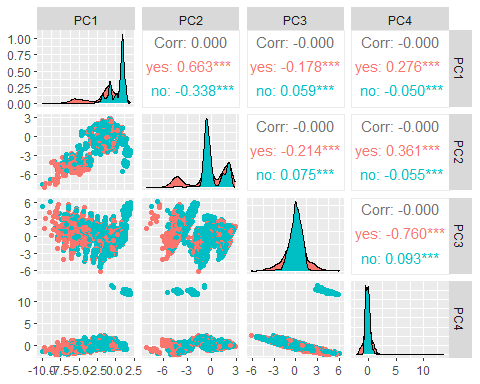
data.frame(PC=1:length(eigenvals2),Prop=eigenvals2/sum(eigenvals2),Cumulative=cumulative.prop)

## PC Prop Cumulative  
## 1 1 0.27614982 0.2761498  
## 2 2 0.25772124 0.5338711  
## 3 3 0.13440859 0.6682796  
## 4 4 0.12407095 0.7923506  
## 5 5 0.10778013 0.9001307  
## 6 6 0.05764961 0.9577803  
## 7 7 0.02351459 0.9812949  
## 8 8 0.01870507 1.0000000

# Calculate the variance explained by each principal component  
var\_explained <- pc.result2$sdev^2 / sum(pc.result2$sdev^2)  
cum\_var\_explained <- cumsum(var\_explained)  
  
# Find the number of components that explain at least 90% of the variance  
num\_comp\_90 <- which(cum\_var\_explained >= 0.9)[1]  
  
# Print the number of components  
print(num\_comp\_90) #We would need 4 to retain approximately 90%

## [1] 5

#Plotting PCA variables with the two colors:  
  
pc.result<-prcomp(df.numeric2[,-c(2,8,9)],scale.=TRUE)  
PC <- data.frame(diagnosis = train\_data$y)  
PC$PC1 <- pc.result2$x[,1]  
PC$PC2 <- pc.result2$x[,2]  
PC$PC3 <- pc.result2$x[,3]  
PC$PC4 <- pc.result2$x[,4]  
ggpairs(PC[,-1],aes(color=PC[,1]))



Final Project - Bank Dataset

Aaron Abromowitz, Stephanie Duarte, Dammy Owolabi

2024-04-20

# Before Part 2

library(tidyverse)

## ── Attaching core tidyverse packages ──────────────────────── tidyverse 2.0.0 ──  
## ✔ dplyr 1.1.1 ✔ readr 2.1.4  
## ✔ forcats 1.0.0 ✔ stringr 1.5.0  
## ✔ ggplot2 3.4.1 ✔ tibble 3.2.1  
## ✔ lubridate 1.9.2 ✔ tidyr 1.3.0  
## ✔ purrr 1.0.1   
## ── 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

# Pull in data  
data<-read.csv('https://raw.githubusercontent.com/stedua22/6372-Project-2/main/bank-additional-full.csv',stringsAsFactors = T, sep=";")  
  
# Set levels to use for later  
data$y <- relevel(data$y, ref="yes")  
data$month <- factor(data$month, levels=c('mar','apr','may','jun','jul','aug','sep','oct','nov','dec'))  
data$day\_of\_week <- factor(data$day\_of\_week, levels=c('mon','tue','wed','thu','fri'))  
  
# Duration was removed since the dataset explanation file said that it was created after y variable was known, so shouldn't be used for prediction.  
data$duration <- c()  
data$default <- c()  
  
# Create the train and test split  
train\_perc <- .8  
set.seed(1234)  
train\_indices <- sample(nrow(data), floor(train\_perc \* nrow(data)))  
train\_data <- data[train\_indices, ]  
nrow(train\_data)

## [1] 32950

test\_data <- data[-train\_indices, ]   
nrow(test\_data)

## [1] 8238

#GLMNET Model

library(readr)  
library(GGally)

## Registered S3 method overwritten by 'GGally':  
## method from   
## +.gg ggplot2

library(caret)

## Loading required package: lattice

##   
## Attaching package: 'caret'

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

library(ggcorrplot)  
# Prepare the matrix of predictors  
x <- model.matrix(~ . - 1 - y, data = train\_data) # Excludes the intercept and the response variable  
  
# Define the trainControl with classProbs enabled  
fitControl <- trainControl(method = "cv",   
 number = 10,   
 classProbs = TRUE, # Enable class probability predictions  
 summaryFunction = twoClassSummary) # Use a summary function for classification  
  
# Run the glmnet model  
set.seed(1234) # for reproducibility  
glmnet\_fit <- train(x, y = train\_data$y,   
 method = "glmnet",  
 trControl = fitControl,  
 tuneLength = 10, # Number of lambda values to test  
 metric = "ROC") # Optimize the model based on ROC curve  
  
# View the best model's lambda value and corresponding coefficients  
best\_lambda <- glmnet\_fit$bestTune$lambda  
coef(glmnet\_fit$finalModel, s = best\_lambda)

## 52 x 1 sparse Matrix of class "dgCMatrix"  
## s1  
## (Intercept) 1.271357e+02  
## age 6.288258e-04  
## jobadmin. -5.537285e-02  
## jobblue-collar 1.042755e-01  
## jobentrepreneur -3.234962e-03  
## jobhousemaid -1.306103e-02  
## jobmanagement .   
## jobretired -3.078555e-01  
## jobself-employed 3.618493e-02  
## jobservices 5.156474e-02  
## jobstudent -2.873845e-01  
## jobtechnician .   
## jobunemployed 5.206854e-02  
## jobunknown 7.563443e-02  
## maritalmarried -2.397840e-02  
## maritalsingle -2.912019e-02  
## maritalunknown -4.058666e-01  
## educationbasic.6y .   
## educationbasic.9y 2.706680e-02  
## educationhigh.school -1.043184e-02  
## educationilliterate -3.876507e-01  
## educationprofessional.course -6.937688e-02  
## educationuniversity.degree -1.123063e-01  
## educationunknown -6.533999e-02  
## housingunknown 8.105645e-02  
## housingyes 3.384646e-02  
## loanunknown 1.362202e-02  
## loanyes 4.709627e-02  
## contacttelephone 6.154443e-01  
## monthapr 1.206473e+00  
## monthmay 1.744218e+00  
## monthjun 1.548731e+00  
## monthjul 1.116250e+00  
## monthaug 8.789533e-01  
## monthsep 1.219657e+00  
## monthoct 1.305819e+00  
## monthnov 1.619636e+00  
## monthdec 7.376210e-01  
## day\_of\_weektue -2.501461e-01  
## day\_of\_weekwed -3.222477e-01  
## day\_of\_weekthu -2.604626e-01  
## day\_of\_weekfri -2.037529e-01  
## campaign 4.271160e-02  
## pdays 9.320162e-04  
## previous 6.642481e-02  
## poutcomenonexistent -3.905654e-01  
## poutcomesuccess -9.262589e-01  
## emp.var.rate 1.023679e+00  
## cons.price.idx -1.356221e+00  
## cons.conf.idx -2.066279e-02  
## euribor3m -1.918785e-01  
## nr.employed .

There definitely is a change in the data over time. We may re-visit this later on.

# Objective 1: Simple Logistic Regerssion Model

We used a combination of forward and backward selection to determine a simple logistic regression model that performed well on the data. We did this by adding each variable to a model, and then doing Cross Validation to test the out of sample data on AUROC (Area under the ROC curve). We used 10 folds. After we chose the first variable, we then did this to add more variables to see if those increased the AUROC. We also tried removing variables (once we got three of them) to see if that improved the score. Below, I included example code for this logic. It takes several minutes to run though, so it is not set to evaluate the code. One caveat is that since the Default variable only has 2 Yes values, it can cause errors sometimes during the cross validation. This happens when the training data (a randomly chosen 90%) doesn’t have either of those values, and the test data (the other 10%) has both of them. This only happens 1% of the time, but when you are running 100s of tests, this happens regularly. And since Default didn’t increase the AUROC much anyways, we decided to drop the variable.

train\_data$default <- c()

# Forward Selection Example  
set.seed(21)  
vars <- names(train\_data)  
vars <- vars[vars!="y"]  
num\_vars <- length(vars)  
var\_aucs <- data.frame("vars" = vars)  
num\_folds <- 10  
for (j in 1:num\_vars) {  
 var <- vars[j]  
 print(var)  
 folds <- createFolds(train\_data$y, k = num\_folds)  
 auc\_scores <- numeric(num\_folds)  
 for (i in 1:num\_folds) {  
 train\_indices <- unlist(folds[-i])  
 test\_indices <- unlist(folds[i])  
 train <- train\_data[train\_indices, ]  
 test <- train\_data[test\_indices, ]  
 form <- as.formula(paste("y ~ ",var,sep=""))  
 model <- glm(form, data = train, family = "binomial")  
 predictions <- predict(model, newdata = test, type = "response")  
 roc <- roc(response=test$y,predictor=predictions,levels=c("no", "yes"),direction = ">")  
 auc\_scores[i] <- auc(roc)  
 }  
 var\_aucs$auc[var\_aucs$var == var] <- mean(auc\_scores)  
}  
  
# Backward selection example  
set.seed(24)  
start\_form\_str <- 'y ~ nr.employed + month + poutcome'  
vars <- c('nr.employed','month','poutcome')  
num\_vars <- length(vars)  
var\_aucs <- data.frame("vars" = vars)  
num\_folds <- 10  
for (j in 1:num\_vars) {  
 var <- vars[j]  
 print(var)  
 folds <- createFolds(train\_data$y, k = num\_folds)  
 auc\_scores <- numeric(num\_folds)  
 for (i in 1:num\_folds) {  
 train\_indices <- unlist(folds[-i])  
 test\_indices <- unlist(folds[i])  
 train <- train\_data[train\_indices, ]  
 test <- train\_data[test\_indices, ]  
 form <- as.formula(paste(start\_form\_str," -",var,sep=""))  
 model <- glm(form, data = train, family = "binomial")  
 predictions <- predict(model, newdata = test, type = "response")  
 roc <- roc(response=test$y,predictor=predictions,levels=c("no", "yes"),direction = ">")  
 auc\_scores[i] <- auc(roc)  
 }  
 var\_aucs$auc[var\_aucs$var == var] <- mean(auc\_scores)  
}

After we did this until the AUROC didn’t increase any more, the resulting model was y ~ month + poutcome + emp.var.rate + euribor3m + contact + cons.price.idx. We then checked the p values and VIR to see if it made sense to keep all of those variables.

library(car)

## Loading required package: carData

##   
## Attaching package: 'car'

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

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

model <- glm(y ~ month + poutcome + emp.var.rate + euribor3m + contact + cons.price.idx, data = train\_data, family = "binomial")  
summary(model)

##   
## Call:  
## glm(formula = y ~ month + poutcome + emp.var.rate + euribor3m +   
## contact + cons.price.idx, family = "binomial", data = train\_data)  
##   
## Deviance Residuals:   
## Min 1Q Median 3Q Max   
## -2.7180 0.2565 0.3225 0.3652 2.0610   
##   
## Coefficients:  
## Estimate Std. Error z value Pr(>|z|)   
## (Intercept) 165.34420 9.72777 16.997 < 2e-16 \*\*\*  
## monthapr 1.48149 0.11493 12.891 < 2e-16 \*\*\*  
## monthmay 1.95366 0.10893 17.936 < 2e-16 \*\*\*  
## monthjun 1.99307 0.13976 14.261 < 2e-16 \*\*\*  
## monthjul 1.34672 0.12631 10.662 < 2e-16 \*\*\*  
## monthaug 0.77035 0.11695 6.587 4.48e-11 \*\*\*  
## monthsep 1.23224 0.14671 8.399 < 2e-16 \*\*\*  
## monthoct 1.48029 0.15076 9.819 < 2e-16 \*\*\*  
## monthnov 1.93888 0.13899 13.950 < 2e-16 \*\*\*  
## monthdec 0.88523 0.21637 4.091 4.29e-05 \*\*\*  
## poutcomenonexistent -0.41002 0.06036 -6.793 1.10e-11 \*\*\*  
## poutcomesuccess -1.81692 0.08686 -20.919 < 2e-16 \*\*\*  
## emp.var.rate 1.50678 0.10080 14.948 < 2e-16 \*\*\*  
## euribor3m -0.53296 0.07687 -6.933 4.12e-12 \*\*\*  
## contacttelephone 0.63484 0.06856 9.260 < 2e-16 \*\*\*  
## cons.price.idx -1.73621 0.10191 -17.037 < 2e-16 \*\*\*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## (Dispersion parameter for binomial family taken to be 1)  
##   
## Null deviance: 23269 on 32949 degrees of freedom  
## Residual deviance: 18332 on 32934 degrees of freedom  
## AIC: 18364  
##   
## Number of Fisher Scoring iterations: 6

vif(model)

## GVIF Df GVIF^(1/(2\*Df))  
## month 6.253389 9 1.107207  
## poutcome 1.288981 2 1.065520  
## emp.var.rate 76.465448 1 8.744452  
## euribor3m 51.138803 1 7.151140  
## contact 1.990009 1 1.410677  
## cons.price.idx 11.218806 1 3.349449

The p values were all significant at the 0.05 level, but there was a high amount of correlation between emp.var.rate and euribor3m. So we removed euribor3m to see if that didn’t make the model too much worse.

library(caret)  
library(pROC)

## Type 'citation("pROC")' for a citation.

##   
## Attaching package: 'pROC'

## The following objects are masked from 'package:stats':  
##   
## cov, smooth, var

set.seed(80)  
form <- as.formula('y ~ month + poutcome + emp.var.rate + contact + cons.price.idx')  
num\_folds <- 10  
folds <- createFolds(train\_data$y, k = num\_folds)  
accuracy\_scores <- numeric(num\_folds)  
auc\_scores <- numeric(num\_folds)  
for (i in 1:num\_folds) {  
 train\_indices <- unlist(folds[-i])  
 test\_indices <- unlist(folds[i])  
 train <- train\_data[train\_indices, ]  
 test <- train\_data[test\_indices, ]  
 model <- glm(form, data = train, family = "binomial")  
 predictions <- predict(model, newdata = test, type = "response")  
 roc <- roc(response=test$y,predictor=predictions,levels=c("no", "yes"),direction = ">")  
 auc\_scores[i] <- auc(roc)  
}  
mean(auc\_scores)

## [1] 0.7917643

It wasn’t too much worse. And removing the correlation between those two variables makes the model easier to interpret.

model <- glm(y ~ month + poutcome + emp.var.rate + contact + cons.price.idx, data = train\_data, family = "binomial")  
summary(model)

##   
## Call:  
## glm(formula = y ~ month + poutcome + emp.var.rate + contact +   
## cons.price.idx, family = "binomial", data = train\_data)  
##   
## Deviance Residuals:   
## Min 1Q Median 3Q Max   
## -2.7139 0.2257 0.3265 0.3616 1.9460   
##   
## Coefficients:  
## Estimate Std. Error z value Pr(>|z|)   
## (Intercept) 108.53001 5.14668 21.087 < 2e-16 \*\*\*  
## monthapr 1.36684 0.11411 11.979 < 2e-16 \*\*\*  
## monthmay 1.88593 0.10846 17.388 < 2e-16 \*\*\*  
## monthjun 1.47245 0.11859 12.416 < 2e-16 \*\*\*  
## monthjul 1.02367 0.11750 8.712 < 2e-16 \*\*\*  
## monthaug 0.69115 0.11645 5.935 2.94e-09 \*\*\*  
## monthsep 1.01545 0.14351 7.076 1.49e-12 \*\*\*  
## monthoct 1.03319 0.13677 7.554 4.21e-14 \*\*\*  
## monthnov 1.44456 0.11940 12.098 < 2e-16 \*\*\*  
## monthdec 0.55440 0.21012 2.638 0.00833 \*\*   
## poutcomenonexistent -0.43221 0.05976 -7.232 4.75e-13 \*\*\*  
## poutcomesuccess -1.81937 0.08629 -21.083 < 2e-16 \*\*\*  
## emp.var.rate 0.82587 0.02213 37.325 < 2e-16 \*\*\*  
## contacttelephone 0.43524 0.06009 7.244 4.37e-13 \*\*\*  
## cons.price.idx -1.14550 0.05491 -20.863 < 2e-16 \*\*\*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## (Dispersion parameter for binomial family taken to be 1)  
##   
## Null deviance: 23269 on 32949 degrees of freedom  
## Residual deviance: 18379 on 32935 degrees of freedom  
## AIC: 18409  
##   
## Number of Fisher Scoring iterations: 6

vif(model)

## GVIF Df GVIF^(1/(2\*Df))  
## month 2.394736 9 1.049711  
## poutcome 1.274785 2 1.062574  
## emp.var.rate 3.656646 1 1.912236  
## contact 1.506607 1 1.227439  
## cons.price.idx 3.284604 1 1.812348

Now the VIFs are much more resonable without euribor3m. So we chose the simpler model for our Simple Logistic Regression Model.

Now that we have a model, we look at the model coefficients for interpretations.

train\_data$y <- relevel(train\_data$y, ref="no")  
mod <- glm(y ~ month + poutcome + emp.var.rate + contact + cons.price.idx, data = train\_data, family = "binomial")  
summary(mod)

##   
## Call:  
## glm(formula = y ~ month + poutcome + emp.var.rate + contact +   
## cons.price.idx, family = "binomial", data = train\_data)  
##   
## Deviance Residuals:   
## Min 1Q Median 3Q Max   
## -1.9460 -0.3616 -0.3265 -0.2257 2.7139   
##   
## Coefficients:  
## Estimate Std. Error z value Pr(>|z|)   
## (Intercept) -108.53001 5.14668 -21.087 < 2e-16 \*\*\*  
## monthapr -1.36684 0.11411 -11.979 < 2e-16 \*\*\*  
## monthmay -1.88593 0.10846 -17.388 < 2e-16 \*\*\*  
## monthjun -1.47245 0.11859 -12.416 < 2e-16 \*\*\*  
## monthjul -1.02367 0.11750 -8.712 < 2e-16 \*\*\*  
## monthaug -0.69115 0.11645 -5.935 2.94e-09 \*\*\*  
## monthsep -1.01545 0.14351 -7.076 1.49e-12 \*\*\*  
## monthoct -1.03319 0.13677 -7.554 4.21e-14 \*\*\*  
## monthnov -1.44456 0.11940 -12.098 < 2e-16 \*\*\*  
## monthdec -0.55440 0.21012 -2.638 0.00833 \*\*   
## poutcomenonexistent 0.43221 0.05976 7.232 4.75e-13 \*\*\*  
## poutcomesuccess 1.81937 0.08629 21.083 < 2e-16 \*\*\*  
## emp.var.rate -0.82587 0.02213 -37.325 < 2e-16 \*\*\*  
## contacttelephone -0.43524 0.06009 -7.244 4.37e-13 \*\*\*  
## cons.price.idx 1.14550 0.05491 20.863 < 2e-16 \*\*\*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## (Dispersion parameter for binomial family taken to be 1)  
##   
## Null deviance: 23269 on 32949 degrees of freedom  
## Residual deviance: 18379 on 32935 degrees of freedom  
## AIC: 18409  
##   
## Number of Fisher Scoring iterations: 6

odds\_ratio <- exp(mod$coefficients)  
confident\_interval <- exp(confint(mod))

## Waiting for profiling to be done...

train\_data$y <- relevel(train\_data$y, ref="yes")

Interpretations (interpreting individually):

Monthapr The odds of getting the clients subscribing to a term deposit in the month of April is 0.255 times lower than that of subscribing in the month of March with a CI of (0.204, 0.319)

Monthmay The odds of getting the clients subscribing to a term deposit in the month of May is 0.152 times lower than that of subscribing in the month of March with a CI of (0.123, 0.188)

Monthjun The odds of getting the clients subscribing to a term deposit in the month of June is 0.229 times lower than that of subscribing in the month of March with a CI of (0.182, 0.289).

Monthjul The odds of getting the clients subscribing to a term deposit in the month of July is 0.359 times lower than that of subscribing in the month of March with a CI of (0.285, 0.452).

Monthaug The odds of getting the clients subscribing to a term deposit in the month of August is 0.500 times lower than that of subscribing in the month of March with a CI of (0.399, 0.629)

Monthsep The odds of getting the clients subscribing to a term deposit in the month of September is 0.362 times lower than that of subscribing in the month of March with a CI of (0.273, 0.479)

Monthoct The odds of getting the clients subscribing to a term deposit in the month of October is 0.356 times lower than that of subscribing in the month of March with a CI of (0.272, 0.465)

Monthnov The odds of getting the clients subscribing to a term deposit in the month of November is 0.236 times lower than that of subscribing in the month of March with a CI of (0.187, 0.298)

Monthdec The odds of getting the clients subscribing to a term deposit in the month of December is 0.574 times lower than that of subscribing in the month of March with a CI of (0.380, 0.868)

poutcomenonexistent The odds of getting the clients subscribing to a term deposit based on a nonexistent outcome of the previous campaign is 1.54 times higher than that of a failed outcome with a CI of (1.37, 1.73)

poutcomesuccess The odds of getting the clients subscribing to a term deposit based on a succesful outcome of the previous campaign is 6.17 times higher than that of a failed outcome with a CI of (5.21, 7.31)

emp.var.rate For every 1 unit increase in customer subscription to a term deposit, the odds of the customer subscribing based on the employment variation rate decreases by 56.2% with a CI of (58.1%, 54.3%)

contacttelephone The odds of getting the clients subscribing to a term deposit based on the contact communication type is 0.647 times lower than that of subscribing by cell phone with a CI of (0.575, 0.728)

cons,price.idx For every 1 unit increase in customer subscription to a term deposit, the odds of the customer subscribing based on the consumers price index increases by a factor of 3.14 with a CI of (2.82, 3.50)

# Objective 2: Complex Logistic Regerssion Model

For the second model, we looked into adding polynomial terms and/or interaction terms to the regression model.

## Polynomial Terms

To see if it made sense to add some polynomial terms we looked at what happened with adding those for Number of Employees, since that was the first variable added during Forward Selection (although it got removed later).

library(tidyverse)  
  
# Make a nr.employed^2 variable  
train\_data$ne2 = (train\_data$nr.employed)^2  
  
# Plot nr.employed  
summary <- train\_data %>%  
 group\_by(nr.employed,y) %>%  
 summarize(count=n())

## `summarise()` has grouped output by 'nr.employed'. You can override using the  
## `.groups` argument.

summary$perc <- 0  
summary$perc[summary$y == 'no'] <- summary$count[summary$y == 'no'] / nrow(train\_data[train\_data$y == 'no',]) \* 100  
summary$perc[summary$y == 'yes'] <- summary$count[summary$y == 'yes'] / nrow(train\_data[train\_data$y == 'yes',]) \* 100  
summary %>% ggplot(aes(x=nr.employed,y=perc,fill=y)) + geom\_bar(stat="identity") + facet\_wrap(~y) +   
 ylab('Percentage') + xlab('Number Employed') + ggtitle('Number Employed Based on Y Value')

A graph of number employed

Description automatically generated

# Plot nr.employed^2  
summary <- train\_data %>%  
 group\_by(ne2,y) %>%  
 summarize(count=n())

## `summarise()` has grouped output by 'ne2'. You can override using the `.groups`  
## argument.

summary$perc <- 0  
summary$perc[summary$y == 'no'] <- summary$count[summary$y == 'no'] / nrow(train\_data[train\_data$y == 'no',]) \* 100  
summary$perc[summary$y == 'yes'] <- summary$count[summary$y == 'yes'] / nrow(train\_data[train\_data$y == 'yes',]) \* 100  
summary %>% ggplot(aes(x=ne2,y=perc,fill=y)) + geom\_bar(stat="identity") + facet\_wrap(~y) +   
 ylab('Percentage') + xlab('Number Employed') + ggtitle('Number Employed Squared Based on Y Value')

A graph of number employed based on y value

Description automatically generated These plots look very hard to distinguish. So instead of plotting polynomial terms, we tried creating simple models and then calculating out of sample AUC. If this increases as polynomial degree increases, then it could make sense to include polynomial terms.

# Maybe just plot the improvement of out of sample AUC  
set.seed(120)  
vars <- 'nr.employed'  
allVars <- vars  
num\_poly <- 10  
for (i in 1:length(vars)){  
 for (j in 2:num\_poly){  
 if (class(train\_data[,vars[i]]) != "factor") {  
 allVars <- c(allVars,paste('poly(',vars[i],',',j,')',sep=""))  
 }  
 }  
}  
num\_vars <- length(allVars)  
var\_aucs <- data.frame("vars" = allVars)  
num\_folds <- 10  
for (j in 1:num\_vars) {  
 var <- allVars[j]  
 print(var)  
 folds <- createFolds(train\_data$y, k = num\_folds)  
 auc\_scores <- numeric(num\_folds)  
 for (i in 1:num\_folds) {  
 train\_indices <- unlist(folds[-i])  
 test\_indices <- unlist(folds[i])  
 train <- train\_data[train\_indices, ]  
 test <- train\_data[test\_indices, ]  
 form <- as.formula(paste("y ~ ",var,sep=""))  
 model <- glm(form, data = train, family = "binomial")  
 predictions <- predict(model, newdata = test, type = "response")  
 roc <- roc(response=test$y,predictor=predictions,levels=c("no", "yes"),direction = ">")  
 auc\_scores[i] <- auc(roc)  
 }  
 var\_aucs$auc[var\_aucs$var == var] <- mean(auc\_scores)  
}

## [1] "nr.employed"  
## [1] "poly(nr.employed,2)"  
## [1] "poly(nr.employed,3)"  
## [1] "poly(nr.employed,4)"  
## [1] "poly(nr.employed,5)"  
## [1] "poly(nr.employed,6)"  
## [1] "poly(nr.employed,7)"  
## [1] "poly(nr.employed,8)"  
## [1] "poly(nr.employed,9)"  
## [1] "poly(nr.employed,10)"

# Get max val  
maxAUC <- max(var\_aucs$auc, na.rm = TRUE)  
maxDeg <- which.max(var\_aucs$auc)  
  
# Looking at the p values for the highest degree model  
model <- glm(form, data = train\_data, family = "binomial")  
summary(model)

##   
## Call:  
## glm(formula = form, family = "binomial", data = train\_data)  
##   
## Deviance Residuals:   
## Min 1Q Median 3Q Max   
## -2.6432 0.2485 0.3321 0.3578 1.2860   
##   
## Coefficients:  
## Estimate Std. Error z value Pr(>|z|)   
## (Intercept) 2.44760 0.02338 104.676 < 2e-16 \*\*\*  
## poly(nr.employed, 10)1 156.34441 3.19323 48.961 < 2e-16 \*\*\*  
## poly(nr.employed, 10)2 -44.45416 3.14126 -14.152 < 2e-16 \*\*\*  
## poly(nr.employed, 10)3 -51.90203 4.32051 -12.013 < 2e-16 \*\*\*  
## poly(nr.employed, 10)4 4.92844 2.83743 1.737 0.082398 .   
## poly(nr.employed, 10)5 26.51619 2.34879 11.289 < 2e-16 \*\*\*  
## poly(nr.employed, 10)6 30.20002 3.42973 8.805 < 2e-16 \*\*\*  
## poly(nr.employed, 10)7 8.50557 3.68814 2.306 0.021100 \*   
## poly(nr.employed, 10)8 -2.79958 2.14882 -1.303 0.192628   
## poly(nr.employed, 10)9 11.69125 3.00755 3.887 0.000101 \*\*\*  
## poly(nr.employed, 10)10 -7.20951 2.02859 -3.554 0.000379 \*\*\*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## (Dispersion parameter for binomial family taken to be 1)  
##   
## Null deviance: 23269 on 32949 degrees of freedom  
## Residual deviance: 19131 on 32939 degrees of freedom  
## AIC: 19153  
##   
## Number of Fisher Scoring iterations: 6

# Plot the AUC improving  
var\_aucs %>% ggplot(aes(x=1:10, y=auc)) + geom\_point() + ylim(c(0.7,0.8)) +   
 ylab('AUC') + xlab('Polynomial Degree') + ggtitle('Out of Sample AUC for nr.employed') +   
 geom\_point(data = data.frame(x = maxDeg, y = maxAUC),   
 aes(x = x, y = y), size = 2, color = "red", fill = "red", shape = 21)

A graph with black dots and red dots

Description automatically generated You can see from the plot that there is a pretty substantial increase in performance after 4 polynomial terms. It levels off after that, but the maximum AUC is technically at n = 9. And even the model for n = 10 shows that many of the higher degree terms are statistically significant, including the 10th term.

We will proceed with trying forward selection using polynomial terms. Using up to the 10th degree term seems like it should be sufficient.

## Variable Interactions

Variable interactions are often present that affect numeric response variables. Usually to show that, you can plot. To see if there are possible interactions, let’s try with a coloring a couple of interactions.

# Try plotting month by contact  
train\_data %>% ggplot(aes(x=month,y=contact,color=y)) + geom\_jitter() +   
 ylab('Month') + xlab('Contact') + ggtitle('Term Deposit for Month and Contact')

A graph of blue and red dots

Description automatically generated

# Looking at model  
model <- glm('y ~ month\*contact', data = train\_data, family = "binomial")  
summary(model)

##   
## Call:  
## glm(formula = "y ~ month\*contact", family = "binomial", data = train\_data)  
##   
## Deviance Residuals:   
## Min 1Q Median 3Q Max   
## -2.6172 0.2573 0.4524 0.4794 1.2668   
##   
## Coefficients:  
## Estimate Std. Error z value Pr(>|z|)   
## (Intercept) -0.09097 0.10061 -0.904 0.36588   
## monthapr 1.44852 0.11528 12.565 < 2e-16 \*\*\*  
## monthmay 2.19672 0.11155 19.692 < 2e-16 \*\*\*  
## monthjun 0.37338 0.12758 2.927 0.00343 \*\*   
## monthjul 2.31893 0.11155 20.788 < 2e-16 \*\*\*  
## monthaug 2.19520 0.11095 19.786 < 2e-16 \*\*\*  
## monthsep 0.16392 0.14336 1.143 0.25287   
## monthoct 0.31811 0.13822 2.301 0.02137 \*   
## monthnov 2.27145 0.11756 19.322 < 2e-16 \*\*\*  
## monthdec -0.11667 0.21208 -0.550 0.58224   
## contacttelephone 0.36541 0.32055 1.140 0.25431   
## monthapr:contacttelephone -0.57305 0.37493 -1.528 0.12640   
## monthmay:contacttelephone 0.92049 0.33152 2.777 0.00549 \*\*   
## monthjun:contacttelephone 2.34176 0.33919 6.904 5.06e-12 \*\*\*  
## monthjul:contacttelephone 0.39625 0.36151 1.096 0.27305   
## monthaug:contacttelephone -0.59236 0.38226 -1.550 0.12123   
## monthsep:contacttelephone 0.70108 0.44236 1.585 0.11300   
## monthoct:contacttelephone -0.03293 0.38399 -0.086 0.93166   
## monthnov:contacttelephone -0.50233 0.36799 -1.365 0.17223   
## monthdec:contacttelephone 0.60437 0.58918 1.026 0.30499   
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## (Dispersion parameter for binomial family taken to be 1)  
##   
## Null deviance: 23269 on 32949 degrees of freedom  
## Residual deviance: 20600 on 32930 degrees of freedom  
## AIC: 20640  
##   
## Number of Fisher Scoring iterations: 6

# Try plotting month by day of week  
train\_data %>% ggplot(aes(x=month,y=day\_of\_week,color=y)) + geom\_jitter() +   
 ylab('Month') + xlab('Day of Week') + ggtitle('Term Deposit for Month and Day of Week')

A graph of a number of red dots

Description automatically generated with medium confidence

# Looking at model  
model <- glm('y ~ month\*day\_of\_week', data = train\_data, family = "binomial")  
summary(model)

##   
## Call:  
## glm(formula = "y ~ month\*day\_of\_week", family = "binomial", data = train\_data)  
##   
## Deviance Residuals:   
## Min 1Q Median 3Q Max   
## -2.4220 0.3623 0.4223 0.4727 1.4132   
##   
## Coefficients:  
## Estimate Std. Error z value Pr(>|z|)   
## (Intercept) 0.312872 0.187972 1.664 0.096020 .   
## monthapr 1.827194 0.233674 7.819 5.31e-15 \*\*\*  
## monthmay 2.278239 0.206389 11.039 < 2e-16 \*\*\*  
## monthjun 1.730202 0.213119 8.118 4.72e-16 \*\*\*  
## monthjul 2.162129 0.216775 9.974 < 2e-16 \*\*\*  
## monthaug 1.979662 0.218099 9.077 < 2e-16 \*\*\*  
## monthsep 0.337715 0.314228 1.075 0.282489   
## monthoct 0.435845 0.278242 1.566 0.117249   
## monthnov 1.977134 0.234519 8.431 < 2e-16 \*\*\*  
## monthdec -0.158722 0.372167 -0.426 0.669758   
## day\_of\_weektue -0.696831 0.266651 -2.613 0.008968 \*\*   
## day\_of\_weekwed -0.772405 0.321428 -2.403 0.016259 \*   
## day\_of\_weekthu -0.055043 0.295758 -0.186 0.852359   
## day\_of\_weekfri -0.430655 0.307183 -1.402 0.160930   
## monthapr:day\_of\_weektue -0.659116 0.336886 -1.956 0.050407 .   
## monthmay:day\_of\_weektue 0.983982 0.295241 3.333 0.000860 \*\*\*  
## monthjun:day\_of\_weektue 0.506141 0.303830 1.666 0.095739 .   
## monthjul:day\_of\_weektue 0.521733 0.304277 1.715 0.086407 .   
## monthaug:day\_of\_weektue 0.427297 0.304207 1.405 0.160132   
## monthsep:day\_of\_weektue 0.112202 0.422507 0.266 0.790577   
## monthoct:day\_of\_weektue 0.148785 0.383220 0.388 0.697832   
## monthnov:day\_of\_weektue 0.496488 0.325825 1.524 0.127562   
## monthdec:day\_of\_weektue 0.003684 0.632825 0.006 0.995355   
## monthapr:day\_of\_weekwed -0.514429 0.377348 -1.363 0.172796   
## monthmay:day\_of\_weekwed 0.872195 0.343141 2.542 0.011028 \*   
## monthjun:day\_of\_weekwed 0.797344 0.355250 2.244 0.024803 \*   
## monthjul:day\_of\_weekwed 0.608365 0.353901 1.719 0.085610 .   
## monthaug:day\_of\_weekwed 0.493582 0.354500 1.392 0.163821   
## monthsep:day\_of\_weekwed 0.041774 0.454737 0.092 0.926805   
## monthoct:day\_of\_weekwed 0.233983 0.428858 0.546 0.585344   
## monthnov:day\_of\_weekwed 0.670189 0.372352 1.800 0.071879 .   
## monthdec:day\_of\_weekwed 0.243561 0.599916 0.406 0.684749   
## monthapr:day\_of\_weekthu -1.284675 0.338186 -3.799 0.000145 \*\*\*  
## monthmay:day\_of\_weekthu 0.193431 0.321553 0.602 0.547473   
## monthjun:day\_of\_weekthu 0.282541 0.335666 0.842 0.399938   
## monthjul:day\_of\_weekthu -0.028636 0.330096 -0.087 0.930869   
## monthaug:day\_of\_weekthu -0.018694 0.331765 -0.056 0.955064   
## monthsep:day\_of\_weekthu -0.470381 0.438986 -1.072 0.283936   
## monthoct:day\_of\_weekthu -0.442359 0.401626 -1.101 0.270714   
## monthnov:day\_of\_weekthu -0.033120 0.350090 -0.095 0.924630   
## monthdec:day\_of\_weekthu -0.159732 0.558543 -0.286 0.774894   
## monthapr:day\_of\_weekfri 0.426120 0.367607 1.159 0.246385   
## monthmay:day\_of\_weekfri 0.449520 0.329336 1.365 0.172275   
## monthjun:day\_of\_weekfri 0.759834 0.343928 2.209 0.027155 \*   
## monthjul:day\_of\_weekfri -0.007922 0.343393 -0.023 0.981596   
## monthaug:day\_of\_weekfri 0.061498 0.342213 0.180 0.857382   
## monthsep:day\_of\_weekfri 0.239600 0.450654 0.532 0.594953   
## monthoct:day\_of\_weekfri -0.208862 0.415835 -0.502 0.615476   
## monthnov:day\_of\_weekfri 0.209000 0.361374 0.578 0.563030   
## monthdec:day\_of\_weekfri 0.681970 0.637079 1.070 0.284411   
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## (Dispersion parameter for binomial family taken to be 1)  
##   
## Null deviance: 23269 on 32949 degrees of freedom  
## Residual deviance: 21314 on 32900 degrees of freedom  
## AIC: 21414  
##   
## Number of Fisher Scoring iterations: 5

It looks like there is some promise for variable interactions. When looking at Month vs Day of Week, certain months have days that have a different distribution of yes and no. And certain days have months that have a different distribution. Some of the interaction terms also have significant p values. We will also include interaction terms in the Forward Selection, along with the polynomial terms. Below is some example code (not being evaluate, because it takes over an hour to finish) of adding all the single variables, polynomial variables, and interaction terms to the model.

set.seed(70)  
vars <- colnames(train\_data)  
vars <- vars[vars!="y"]  
allVars <- vars  
num\_poly <- 10  
for (i in 1:length(vars)){  
 for (j in 2:num\_poly){  
 if (class(train\_data[,vars[i]]) != "factor") {  
 allVars <- c(allVars,paste('poly(',vars[i],',',j,')',sep=""))  
 }  
 }  
}  
for (i in 1:length(vars)){  
 for (j in 1:i){  
 if(vars[i]!=vars[j]) {  
 allVars <- c(allVars,paste(vars[i],'\*',vars[j],sep=''))  
 }  
 }  
}  
# These variables need to be removed so the code doesn't error out  
allVars <- allVars[allVars!="poly(pdays,6)"]   
allVars <- allVars[allVars!="poly(pdays,7)"]  
allVars <- allVars[allVars!="poly(pdays,8)"]  
allVars <- allVars[allVars!="poly(pdays,9)"]  
allVars <- allVars[allVars!="poly(pdays,10)"]  
allVars <- allVars[allVars!="poly(previous,7)"]  
allVars <- allVars[allVars!="poly(previous,8)"]  
allVars <- allVars[allVars!="poly(previous,9)"]  
allVars <- allVars[allVars!="poly(previous,10)"]  
allVars <- allVars[allVars!="poly(emp.var.rate,10)"]  
var\_aucs <- data.frame("vars" = allVars)  
num\_vars <- length(allVars)  
num\_folds <- 10  
start\_num <- 124  
for (j in start\_num:num\_vars) {  
 var <- allVars[j]  
 print(paste(j,'/',num\_vars,': ',var,sep=''))  
 folds <- createFolds(train\_data$y, k = num\_folds)  
 auc\_scores <- numeric(num\_folds)  
 for (i in 1:num\_folds) {  
 train\_indices <- unlist(folds[-i])  
 test\_indices <- unlist(folds[i])  
 train <- train\_data[train\_indices, ]  
 test <- train\_data[test\_indices, ]  
 form <- as.formula(paste("y ~ ",var,sep=""))  
 model <- glm(form, data = train, family = "binomial")  
 predictions <- predict(model, newdata = test, type = "response")  
 roc <- roc(response=test$y,predictor=predictions,levels=c("no", "yes"),direction = ">")  
 auc\_scores[i] <- auc(roc)  
 }  
 var\_aucs$auc[var\_aucs$var == var] <- mean(auc\_scores)  
}

After several iterations of this (plus Backwards Selection), we arrived at the final model: y ~ poly(cons.conf.idx,10) + pdays + day\_of\_week \* month + month \* contact + cons.conf.idx \* housing + poutcome \* previous + poly(campaign,5) + poly(euribor3m,8) + campaign \* month + cons.conf.idx \* age + poly(previous,6) + campaign \* contact + poly(age,3).

## Adding PCA

Since our earlier PCA analysis looked promising, I tried adding PC1 to the model.

# PCA  
df.numeric <- train\_data[ , sapply(train\_data, is.numeric)]  
pc.result<-prcomp(df.numeric,scale.=TRUE)  
pc.scores<-pc.result$x  
pc.scores<-data.frame(pc.scores)  
  
# Trying out adding PC1  
train\_data$PC1 <- pc.scores$PC1  
set.seed(134)  
form <- as.formula('y ~ PC1 + poly(cons.conf.idx,10) + pdays + day\_of\_week\*month + month\*contact + cons.conf.idx\*housing + poutcome\*previous + poly(campaign,5) + poly(euribor3m,8) + campaign\*month + cons.conf.idx\*age + poly(previous,6) + campaign\*contact + poly(age,3)')  
num\_folds <- 10  
folds <- createFolds(train\_data$y, k = num\_folds)  
accuracy\_scores <- numeric(num\_folds)  
auc\_scores <- numeric(num\_folds)  
for (i in 1:num\_folds) {  
 train\_indices <- unlist(folds[-i])  
 test\_indices <- unlist(folds[i])  
 train <- train\_data[train\_indices, ]  
 test <- train\_data[test\_indices, ]  
 model <- glm(form, data = train, family = "binomial")  
 predictions <- predict(model, newdata = test, type = "response")  
 roc <- roc(response=test$y,predictor=predictions,levels=c("no", "yes"),direction = ">")  
 auc\_scores[i] <- auc(roc)  
}  
mean(auc\_scores)

## [1] 0.8026035

The AUC value was slightly worse than it was when PC1 wasn’t there. So we’ll just use the formula derived above.

## Support Vector Machine (SVM)

For a non-parametric model, we tried Support Vector Machines (SVM). These are a type of non-parametric model that try to form a line, plane, hyper-plan between datapoints.

They had three important hyper parameters: the kernal, gamma, and the cost. The kernal would be the type of fit. Possible values are linear, polynomial, radial, etc. Gamma sets the curvature of the separating hyper-plane. A higher Gamma values means more curvature. The Cost parameter sets how much error points it wants to classify on. The higher the Cost, the worse it predicts on the training set errors (and hope isn’t overfitting).

Here is some example code for training the SVM. It takes over half an hour to train even fairly simple models, so it isn’t set to execute.

form <- as.formula("y ~ euribor3m + month + poutcome + contact + cons.conf.idx + campaign + previous + age + housing + day\_of\_week")  
svm\_model <- svm(form, data = train\_data, kernel = "radial", gamma = 1, cost = 1, probability = TRUE, decision.values = TRUE)  
predictions <- predict(svm\_model, newdata = train\_data, probability = TRUE, decision.values = TRUE)  
probs <- attr(predictions,"probabilities")[,'yes']  
predicted\_classes <- ifelse(probs > .083, 'yes','no')   
confusion\_matrix <- confusionMatrix(as.factor(predicted\_classes), as.factor(train\_data$y))  
confusion\_matrix # PPV = 0.69, Sens = 0.62  
confusion\_matrix$byClass['F1'] # 0.65  
roc <- roc(response=train\_data$y,predictor=probs,levels=c("yes", "no"),direction = ">")  
auc(roc) # 0.8513  
plot(roc,print.thres="best",col="red")  
title(main = 'ROC Curve for SVM', line = 3)

And here is the sample code for testing.

predictions <- predict(svm\_model, newdata = test\_data, probability = TRUE)  
probs <- attr(predictions,"probabilities")[,'yes']  
predicted\_classes <- ifelse(probs > .083, 'yes','no')   
confusion\_matrix <- confusionMatrix(as.factor(predicted\_classes), as.factor(test\_data$y))  
confusion\_matrix # Sensitivity = 0.5236, Specificity = 0.8705, PPV = 0.3345, NPV = 0.9363  
confusion\_matrix$byClass['F1'] # 0.4082157  
roc <- roc(response=test\_data$y,predictor=probs,levels=c("yes", "no"),direction = ">")  
auc(roc) # 0.6979

# Calculating Metrics

We are calculating Sensitivity (Correctly Predicted Positive / All Positive), Specificity (Correctly Predicted Negative / All Negative), Prevalence (All Positive / All Observations), PPV (Correctly Predicted Positive / Predicted Positive), NPV (Correctly Predicted Negative / Predicted Negative), and AUROC (Area under the ROC Curve).

In addition, we are also calculating the F1 score. This is equal to 2 \* Sensitivity \* PPV / (Sensitivity + PPV). Since it is more important that we do a good job of predicting whether people will get a term deposit, and the F1 score is a nice metric to make sure that there is a good balance between Sensitivity and PPV, we decided to track this metric as well.

## Simple Logisitc Regression Model

First we determined the threshold to use for the Simple Logistic Regression Model in order to maximize the F1 metric.

# Get threshold  
metrics = data.frame(thresh=seq(0, 1, by = 0.0001))  
num\_thresh <- nrow(metrics)  
metrics$sensitivity <- 0  
metrics$specificity <- 0  
metrics$ppv <- 0  
metrics$npv <- 0  
metrics$accuracy <- 0  
metrics$f1 <- 0  
form <- as.formula(y ~ month + poutcome + emp.var.rate + contact + cons.price.idx)  
model <- glm(form, data = train\_data, family = "binomial")  
predicted <- predict(model, newdata = train\_data, type = "response")

for (i in 1:num\_thresh){  
   
 # Confusion Matrix  
 predicted\_classes <- ifelse(predicted > metrics$thresh[i], 'no','yes')  
 confusion\_matrix <- confusionMatrix(as.factor(predicted\_classes), as.factor(train\_data$y))  
   
 # Metrics  
 metrics$sensitivity[i] <- as.numeric(confusion\_matrix$byClass['Sensitivity'])  
 metrics$specificity[i] <- as.numeric(confusion\_matrix$byClass['Specificity'])  
 metrics$ppv[i] <- as.numeric(confusion\_matrix$byClass['Pos Pred Value'])  
 metrics$npv[i] <- as.numeric(confusion\_matrix$byClass['Neg Pred Value'])  
 metrics$accuracy[i] <- as.numeric(confusion\_matrix$overall['Accuracy'])  
 metrics$f1[i] <- as.numeric(confusion\_matrix$byClass['F1'])  
}

# Get threshold value that maximizes F1  
metrics <- read.csv('https://raw.githubusercontent.com/stedua22/6372-Project-2/main/metrics\_simple.csv')  
maxF1 <- max(metrics$f1, na.rm = TRUE)  
maxF1

## [1] 0.4798184

theshF1 <- metrics$thresh[which.max(metrics$f1)]   
theshF1

## [1] 0.7703

# Plots  
metrics %>% ggplot(aes(x = thresh, y = sensitivity)) + geom\_point() +   
 ylab('Sensitivity') + xlab('Thresholds') + ggtitle('Sensitivity for Training Data')

A graph with a line drawn on it

Description automatically generated

metrics %>% ggplot(aes(x = thresh, y = specificity)) + geom\_point() +   
 ylab('Specificity') + xlab('Thresholds') + ggtitle('Specificity for Training Data')

A graph with a line drawn on it

Description automatically generated

metrics %>% ggplot(aes(x = thresh, y = ppv)) + geom\_point() +   
 ylab('PPV') + xlab('Thresholds') + ggtitle('PPV for Training Data')

## Warning: Removed 1506 rows containing missing values (`geom\_point()`).

A graph of a graph showing the growth of a training data

Description automatically generated

metrics %>% ggplot(aes(x = thresh, y = npv)) + geom\_point() +   
 ylab('NPV') + xlab('Thresholds') + ggtitle('NPV for Training Data')

## Warning: Removed 252 rows containing missing values (`geom\_point()`).

A graph of a graph showing the growth of a number of individuals

Description automatically generated

metrics %>% ggplot(aes(x = thresh, y = f1)) + geom\_point() +   
 ylab('F1 Score') + xlab('Thresholds') + ggtitle('F1 Scores for Training Data') +   
 geom\_point(data = data.frame(x = theshF1, y = maxF1), aes(x = x, y = y), size = 3, color = "red", fill = "red", shape = 21)

## Warning: Removed 1506 rows containing missing values (`geom\_point()`).

A graph of a graph showing the number of scores

Description automatically generated

# Get Confusion Matrix for threshold value  
predicted\_classes <- ifelse(predicted > theshF1, 'no','yes')  
confusion\_matrix <- confusionMatrix(as.factor(predicted\_classes), as.factor(train\_data$y))

## Warning in confusionMatrix.default(as.factor(predicted\_classes),  
## as.factor(train\_data$y)): Levels are not in the same order for reference and  
## data. Refactoring data to match.

confusion\_matrix

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction yes no  
## yes 1902 2297  
## no 1827 26924  
##   
## Accuracy : 0.8748   
## 95% CI : (0.8712, 0.8784)  
## No Information Rate : 0.8868   
## P-Value [Acc > NIR] : 1   
##   
## Kappa : 0.409   
##   
## Mcnemar's Test P-Value : 2.81e-13   
##   
## Sensitivity : 0.51006   
## Specificity : 0.92139   
## Pos Pred Value : 0.45296   
## Neg Pred Value : 0.93645   
## Prevalence : 0.11317   
## Detection Rate : 0.05772   
## Detection Prevalence : 0.12744   
## Balanced Accuracy : 0.71572   
##   
## 'Positive' Class : yes   
##

confusion\_matrix$byClass['F1']

## F1   
## 0.4798184

After we got the thresholds to use, then we calculated the metrics on the test dataset.

# Test data  
predicted <- predict(model, newdata = test\_data, type = "response")  
predicted\_classes <- ifelse(predicted > theshF1, 'no','yes')  
confusion\_matrix <- confusionMatrix(as.factor(predicted\_classes), as.factor(test\_data$y))

## Warning in confusionMatrix.default(as.factor(predicted\_classes),  
## as.factor(test\_data$y)): Levels are not in the same order for reference and  
## data. Refactoring data to match.

confusion\_matrix

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction yes no  
## yes 449 571  
## no 462 6756  
##   
## Accuracy : 0.8746   
## 95% CI : (0.8673, 0.8817)  
## No Information Rate : 0.8894   
## P-Value [Acc > NIR] : 0.9999882   
##   
## Kappa : 0.3943   
##   
## Mcnemar's Test P-Value : 0.0007787   
##   
## Sensitivity : 0.4929   
## Specificity : 0.9221   
## Pos Pred Value : 0.4402   
## Neg Pred Value : 0.9360   
## Prevalence : 0.1106   
## Detection Rate : 0.0545   
## Detection Prevalence : 0.1238   
## Balanced Accuracy : 0.7075   
##   
## 'Positive' Class : yes   
##

confusion\_matrix$byClass['F1']

## F1   
## 0.465044

# AUC  
roc <- roc(response=test\_data$y,predictor=predicted,levels=c("no", "yes"),direction = ">")  
auc(roc)

## Area under the curve: 0.7868

plot(roc,print.thres="best",col="red")

A red line graph with numbers

Description automatically generated This code takes a while to run, so we won’t include the code to maximize the threshold value going forward. Given the thresholds though, here is the code to get the metrics for the complicated logistic regression model.

## Complex Logistic Regression

# Get threshold value that maximizes F1  
metrics <- read.csv('https://raw.githubusercontent.com/stedua22/6372-Project-2/main/metrics\_complex\_logistic.csv')  
maxF1 <- max(metrics$f1, na.rm = TRUE)  
maxF1

## [1] 0.5073269

theshF1 <- metrics$thresh[which.max(metrics$f1)]   
theshF1

## [1] 0.7354

# Get Confusion Matrix for threshold value  
form <- as.formula(y ~ poly(cons.conf.idx,10) + pdays + day\_of\_week\*month + month\*contact + cons.conf.idx\*housing + poutcome\*previous + poly(campaign,5) + poly(euribor3m,8) + campaign\*month + cons.conf.idx\*age + poly(previous,6) + campaign\*contact + poly(age,3))  
model <- glm(form, data = train\_data, family = "binomial")  
predicted <- predict(model, newdata = train\_data, type = "response")

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :  
## prediction from a rank-deficient fit may be misleading

predicted\_classes <- ifelse(predicted > theshF1, 'no','yes')  
confusion\_matrix <- confusionMatrix(as.factor(predicted\_classes), as.factor(train\_data$y))

## Warning in confusionMatrix.default(as.factor(predicted\_classes),  
## as.factor(train\_data$y)): Levels are not in the same order for reference and  
## data. Refactoring data to match.

confusion\_matrix

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction yes no  
## yes 2008 2179  
## no 1721 27042  
##   
## Accuracy : 0.8816   
## 95% CI : (0.8781, 0.8851)  
## No Information Rate : 0.8868   
## P-Value [Acc > NIR] : 0.9985   
##   
## Kappa : 0.4403   
##   
## Mcnemar's Test P-Value : 2.52e-13   
##   
## Sensitivity : 0.53848   
## Specificity : 0.92543   
## Pos Pred Value : 0.47958   
## Neg Pred Value : 0.94017   
## Prevalence : 0.11317   
## Detection Rate : 0.06094   
## Detection Prevalence : 0.12707   
## Balanced Accuracy : 0.73196   
##   
## 'Positive' Class : yes   
##

confusion\_matrix$byClass['F1']

## F1   
## 0.5073269

# Test data  
predicted <- predict(model, newdata = test\_data, type = "response")

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :  
## prediction from a rank-deficient fit may be misleading

predicted\_classes <- ifelse(predicted > theshF1, 'no','yes')  
confusion\_matrix <- confusionMatrix(as.factor(predicted\_classes), as.factor(test\_data$y))

## Warning in confusionMatrix.default(as.factor(predicted\_classes),  
## as.factor(test\_data$y)): Levels are not in the same order for reference and  
## data. Refactoring data to match.

confusion\_matrix

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction yes no  
## yes 481 550  
## no 430 6777  
##   
## Accuracy : 0.881   
## 95% CI : (0.8739, 0.888)  
## No Information Rate : 0.8894   
## P-Value [Acc > NIR] : 0.9922347   
##   
## Kappa : 0.4282   
##   
## Mcnemar's Test P-Value : 0.0001439   
##   
## Sensitivity : 0.52799   
## Specificity : 0.92494   
## Pos Pred Value : 0.46654   
## Neg Pred Value : 0.94034   
## Prevalence : 0.11059   
## Detection Rate : 0.05839   
## Detection Prevalence : 0.12515   
## Balanced Accuracy : 0.72646   
##   
## 'Positive' Class : yes   
##

confusion\_matrix$byClass['F1']

## F1   
## 0.4953656

# AUC  
roc <- roc(response=test\_data$y,predictor=predicted,levels=c("no", "yes"),direction = ">")  
auc(roc)

## Area under the curve: 0.8013

plot(roc,print.thres="best",col="red")

A red line graph with numbers

Description automatically generated

## Comparing old vs new data

We noticed as part of EDA that the newer data (data was in order of when it was received) had a different Yes/No distribution than older data. We thought it would be interesting to see how training on old data and testing on newer data would perform.

# Simple logistic model  
metrics <- read.csv('https://raw.githubusercontent.com/stedua22/6372-Project-2/main/metrics\_simple\_date.csv')  
  
# Train simple model  
form <- as.formula(y ~ month + poutcome + emp.var.rate + contact + cons.price.idx)  
model <- glm(form, data = train\_data, family = "binomial")  
  
# Get threshold value that maximizes F1  
maxF1 <- max(metrics$f1, na.rm = TRUE)  
maxF1 # 0.2623695

## [1] 0.2623695

theshF1 <- metrics$thresh[which.max(metrics$f1)]   
theshF1 # 0.8486

## [1] 0.8486

# Try filtering data to get it to work  
test\_data <- test\_data[test\_data$month != "sep",]  
  
# Get the confusion matrix  
predicted <- predict(model, newdata = test\_data, type = "response")  
predicted\_classes <- ifelse(predicted > theshF1, 'no','yes')  
confusion\_matrix <- confusionMatrix(as.factor(predicted\_classes), as.factor(test\_data$y))

## Warning in confusionMatrix.default(as.factor(predicted\_classes),  
## as.factor(test\_data$y)): Levels are not in the same order for reference and  
## data. Refactoring data to match.

confusion\_matrix # Sensitivity = 0.5661, Specificity = 0.7740, PPV = 0.5151, NPV = 0.8079, Prevalence = 0.2979

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction yes no  
## yes 474 819  
## no 382 6443  
##   
## Accuracy : 0.8521   
## 95% CI : (0.8441, 0.8597)  
## No Information Rate : 0.8946   
## P-Value [Acc > NIR] : 1   
##   
## Kappa : 0.3599   
##   
## Mcnemar's Test P-Value : <2e-16   
##   
## Sensitivity : 0.55374   
## Specificity : 0.88722   
## Pos Pred Value : 0.36659   
## Neg Pred Value : 0.94403   
## Prevalence : 0.10544   
## Detection Rate : 0.05839   
## Detection Prevalence : 0.15928   
## Balanced Accuracy : 0.72048   
##   
## 'Positive' Class : yes   
##

confusion\_matrix$byClass['F1'] # 0.5394243

## F1   
## 0.4411354

# AUC  
roc <- roc(response=test\_data$y,predictor=predicted,levels=c("no", "yes"),direction = ">")  
auc(roc) # 0.7095

## Area under the curve: 0.7788

# Complex model  
metrics <- read.csv('https://raw.githubusercontent.com/stedua22/6372-Project-2/main/metrics\_complex\_logistic\_date.csv')  
  
# Train simple model  
form <- as.formula(y ~ poly(cons.conf.idx,10) + pdays + day\_of\_week\*month + month\*contact + cons.conf.idx\*housing + poutcome\*previous + poly(campaign,5) + poly(euribor3m,8) + campaign\*month + cons.conf.idx\*age + poly(previous,6) + campaign\*contact + poly(age,3))  
model <- glm(form, data = train\_data, family = "binomial")  
  
# Error about poly(cons.conf.idx,10) having too high of a degree  
form <- as.formula(y ~ poly(cons.conf.idx,9) + pdays + day\_of\_week\*month + month\*contact + cons.conf.idx\*housing + poutcome\*previous + poly(campaign,5) + poly(euribor3m,8) + campaign\*month + cons.conf.idx\*age + poly(previous,3) + campaign\*contact + poly(age,3))  
model <- glm(form, data = train\_data, family = "binomial")  
  
# Get threshold value that maximizes F1  
maxF1 <- max(metrics$f1, na.rm = TRUE)  
maxF1 # 0.2431846

## [1] 0.2431846

theshF1 <- metrics$thresh[which.max(metrics$f1)]   
theshF1 # 0.7308

## [1] 0.7308

# Try filtering data to get it to work  
test\_data <- test\_data[test\_data$month != "sep",]  
  
# Get the confusion matrix  
predicted <- predict(model, newdata = test\_data, type = "response")

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :  
## prediction from a rank-deficient fit may be misleading

predicted\_classes <- ifelse(predicted > theshF1, 'no','yes')  
confusion\_matrix <- confusionMatrix(as.factor(predicted\_classes), as.factor(test\_data$y))

## Warning in confusionMatrix.default(as.factor(predicted\_classes),  
## as.factor(test\_data$y)): Levels are not in the same order for reference and  
## data. Refactoring data to match.

confusion\_matrix # Sensitivity = 0.6287, Specificity = 0.6716, PPV = 0.4482, NPV = 0.8100, Prevalence = 0.2979

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction yes no  
## yes 430 513  
## no 426 6749  
##   
## Accuracy : 0.8843   
## 95% CI : (0.8772, 0.8912)  
## No Information Rate : 0.8946   
## P-Value [Acc > NIR] : 0.998562   
##   
## Kappa : 0.4132   
##   
## Mcnemar's Test P-Value : 0.005008   
##   
## Sensitivity : 0.50234   
## Specificity : 0.92936   
## Pos Pred Value : 0.45599   
## Neg Pred Value : 0.94063   
## Prevalence : 0.10544   
## Detection Rate : 0.05297   
## Detection Prevalence : 0.11616   
## Balanced Accuracy : 0.71585   
##   
## 'Positive' Class : yes   
##

confusion\_matrix$byClass['F1'] # 0.5233236

## F1   
## 0.4780434

# AUC  
roc <- roc(response=test\_data$y,predictor=predicted,levels=c("no", "yes"),direction = ">")  
auc(roc) # 0.6502

## Area under the curve: 0.7936

We compared the simple model and complex model to how they did previously. To even be able to test against the newer data, we first had to filter out month = Sep, since that didn’t exist in the training data. Also, we had to lower the order of some of the polynomials for the complex model.

The training metrics were poor, with an F1 score of around .25 for both models. However, the F1 scores for the test data were both above 0.5. The extra Yes results in the data seemed to help out. However, the AUC scores were worse, as well as the Specificity and NPV. This makes some sense, since there were less No results.

#QDA/LDA Model

library(caret) # CreateFolds  
library(pROC)  
library(car) # VIF  
library(tidyverse)  
  
  
#LDA Model--- simple model month + poutcome + emp.var.rate + contact + cons.price.idx  
  
# Convert the binary outcome to a factor  
train\_data$y <- as.factor(train\_data$y)  
  
  
fitControl<-trainControl(method="repeatedcv",number=5,repeats=1,classProbs=TRUE, summaryFunction=mnLogLoss)  
set.seed(1234)  
  
  
lda.fit<-train(y~ month + poutcome + emp.var.rate + contact + cons.price.idx,  
 data=train\_data,  
 method="lda",  
 trControl=fitControl,  
 metric="logLoss")  
  
# Get threshold  
metrics = data.frame(thresh=seq(0, 1, by = 0.0001))  
num\_thresh <- nrow(metrics)  
metrics$sensitivity <- 1  
metrics$specificity <- 1  
metrics$ppv <- 1  
metrics$npv <- 1  
metrics$accuracy <- 1  
metrics$f1 <- 1  
predicted <- predict(lda.fit, newdata = train\_data, type = "prob")  
for (i in 1:num\_thresh){  
 if(i %% 100 == 0) {  
 print(paste(i,'/',num\_thresh,sep=''))  
 }  
   
 # Confusion Matrix  
 predicted\_classes <- ifelse(predicted[, "yes"] > metrics$thresh[i], 'yes', 'no')  
 predicted\_classes\_factor <- factor(predicted\_classes, levels = levels(train\_data$y))  
 confusion\_matrix <- confusionMatrix(predicted\_classes\_factor, train\_data$y)  
   
   
 # Metrics  
 metrics$sensitivity[i] <- as.numeric(confusion\_matrix$byClass['Sensitivity'])  
 metrics$specificity[i] <- as.numeric(confusion\_matrix$byClass['Specificity'])  
 metrics$ppv[i] <- as.numeric(confusion\_matrix$byClass['Pos Pred Value'])  
 metrics$npv[i] <- as.numeric(confusion\_matrix$byClass['Neg Pred Value'])  
 metrics$accuracy[i] <- as.numeric(confusion\_matrix$overall['Accuracy'])  
 metrics$f1[i] <- as.numeric(confusion\_matrix$byClass['F1'])  
}

## [1] "100/10001"  
## [1] "200/10001"  
## [1] "300/10001"  
## [1] "400/10001"  
## [1] "500/10001"  
## [1] "600/10001"  
## [1] "700/10001"  
## [1] "800/10001"  
## [1] "900/10001"  
## [1] "1000/10001"  
## [1] "1100/10001"  
## [1] "1200/10001"  
## [1] "1300/10001"  
## [1] "1400/10001"  
## [1] "1500/10001"  
## [1] "1600/10001"  
## [1] "1700/10001"  
## [1] "1800/10001"  
## [1] "1900/10001"  
## [1] "2000/10001"  
## [1] "2100/10001"  
## [1] "2200/10001"  
## [1] "2300/10001"  
## [1] "2400/10001"  
## [1] "2500/10001"  
## [1] "2600/10001"  
## [1] "2700/10001"  
## [1] "2800/10001"  
## [1] "2900/10001"  
## [1] "3000/10001"  
## [1] "3100/10001"  
## [1] "3200/10001"  
## [1] "3300/10001"  
## [1] "3400/10001"  
## [1] "3500/10001"  
## [1] "3600/10001"  
## [1] "3700/10001"  
## [1] "3800/10001"  
## [1] "3900/10001"  
## [1] "4000/10001"  
## [1] "4100/10001"  
## [1] "4200/10001"  
## [1] "4300/10001"  
## [1] "4400/10001"  
## [1] "4500/10001"  
## [1] "4600/10001"  
## [1] "4700/10001"  
## [1] "4800/10001"  
## [1] "4900/10001"  
## [1] "5000/10001"  
## [1] "5100/10001"  
## [1] "5200/10001"  
## [1] "5300/10001"  
## [1] "5400/10001"  
## [1] "5500/10001"  
## [1] "5600/10001"  
## [1] "5700/10001"  
## [1] "5800/10001"  
## [1] "5900/10001"  
## [1] "6000/10001"  
## [1] "6100/10001"  
## [1] "6200/10001"  
## [1] "6300/10001"  
## [1] "6400/10001"  
## [1] "6500/10001"  
## [1] "6600/10001"  
## [1] "6700/10001"  
## [1] "6800/10001"  
## [1] "6900/10001"  
## [1] "7000/10001"  
## [1] "7100/10001"  
## [1] "7200/10001"  
## [1] "7300/10001"  
## [1] "7400/10001"  
## [1] "7500/10001"  
## [1] "7600/10001"  
## [1] "7700/10001"  
## [1] "7800/10001"  
## [1] "7900/10001"  
## [1] "8000/10001"  
## [1] "8100/10001"  
## [1] "8200/10001"  
## [1] "8300/10001"  
## [1] "8400/10001"  
## [1] "8500/10001"  
## [1] "8600/10001"  
## [1] "8700/10001"  
## [1] "8800/10001"  
## [1] "8900/10001"  
## [1] "9000/10001"  
## [1] "9100/10001"  
## [1] "9200/10001"  
## [1] "9300/10001"  
## [1] "9400/10001"  
## [1] "9500/10001"  
## [1] "9600/10001"  
## [1] "9700/10001"  
## [1] "9800/10001"  
## [1] "9900/10001"  
## [1] "10000/10001"

# Get threshold value that maximizes F1  
# Get F1 thresholds  
maxF1 <- max(metrics$f1, na.rm = TRUE) #   
maxF1 # 0.4847

## [1] 0.4847892

theshF1 <- metrics$thresh[which.max(metrics$f1)]   
theshF1 # 0.1313

## [1] 0.1313

# Test data  
predicted <- predict(lda.fit, newdata = test\_data, type = "prob")  
predicted\_classes <- ifelse(predicted[, "yes"] > theshF1, 'yes', 'no')  
confusion\_matrix <- confusionMatrix(as.factor(predicted\_classes), as.factor(test\_data$y))

## Warning in confusionMatrix.default(as.factor(predicted\_classes),  
## as.factor(test\_data$y)): Levels are not in the same order for reference and  
## data. Refactoring data to match.

confusion\_matrix # Sensitivity = 0.5159, Specificity = 0.9179, PPV = 0.4388, NPV = 0.9385, Prevalence = 0.11059

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction yes no  
## yes 415 536  
## no 441 6726  
##   
## Accuracy : 0.8797   
## 95% CI : (0.8724, 0.8867)  
## No Information Rate : 0.8946   
## P-Value [Acc > NIR] : 0.999992   
##   
## Kappa : 0.3918   
##   
## Mcnemar's Test P-Value : 0.002636   
##   
## Sensitivity : 0.48481   
## Specificity : 0.92619   
## Pos Pred Value : 0.43638   
## Neg Pred Value : 0.93847   
## Prevalence : 0.10544   
## Detection Rate : 0.05112   
## Detection Prevalence : 0.11715   
## Balanced Accuracy : 0.70550   
##   
## 'Positive' Class : yes   
##

confusion\_matrix$byClass['F1'] # 0.4743

## F1   
## 0.4593248

#AUC  
  
  
# Assuming `predicted` contains the predicted probabilities for the 'yes' class  
roc <- roc(response = test\_data$y,   
 predictor = as.numeric(as.character(predicted[, "yes"])),  
 levels = rev(levels(test\_data$y))) # Ensure correct ordering of levels if needed

## Setting direction: controls < cases

# Print the AUROC  
auc(roc) # 0.7846

## Area under the curve: 0.7768

#LDA Model--- using numeric variables only campaign + pdays + previous + emp.var.rate + cons.price.idx + euribor3m + nr.employed  
  
  
fitControl<-trainControl(method="repeatedcv",number=5,repeats=1,classProbs=TRUE, summaryFunction=mnLogLoss)  
set.seed(1234)  
  
  
lda.fit<-train(y~ campaign + pdays + previous + emp.var.rate + cons.price.idx + euribor3m + nr.employed,  
 data=train\_data,  
 method="lda",  
 trControl=fitControl,  
 metric="logLoss")  
  
# Get threshold  
metrics = data.frame(thresh=seq(0, 1, by = 0.0001))  
num\_thresh <- nrow(metrics)  
metrics$sensitivity <- 1  
metrics$specificity <- 1  
metrics$ppv <- 1  
metrics$npv <- 1  
metrics$accuracy <- 1  
metrics$f1 <- 1  
predicted <- predict(lda.fit, newdata = train\_data, type = "prob")  
for (i in 1:num\_thresh){  
 if(i %% 100 == 0) {  
 print(paste(i,'/',num\_thresh,sep=''))  
 }  
   
 # Confusion Matrix  
 predicted\_classes <- ifelse(predicted[, "yes"] > metrics$thresh[i], 'yes', 'no')  
 predicted\_classes\_factor <- factor(predicted\_classes, levels = levels(train\_data$y))  
 confusion\_matrix <- confusionMatrix(predicted\_classes\_factor, train\_data$y)  
   
   
 # Metrics  
 metrics$sensitivity[i] <- as.numeric(confusion\_matrix$byClass['Sensitivity'])  
 metrics$specificity[i] <- as.numeric(confusion\_matrix$byClass['Specificity'])  
 metrics$ppv[i] <- as.numeric(confusion\_matrix$byClass['Pos Pred Value'])  
 metrics$npv[i] <- as.numeric(confusion\_matrix$byClass['Neg Pred Value'])  
 metrics$accuracy[i] <- as.numeric(confusion\_matrix$overall['Accuracy'])  
 metrics$f1[i] <- as.numeric(confusion\_matrix$byClass['F1'])  
}

## [1] "100/10001"  
## [1] "200/10001"  
## [1] "300/10001"  
## [1] "400/10001"  
## [1] "500/10001"  
## [1] "600/10001"  
## [1] "700/10001"  
## [1] "800/10001"  
## [1] "900/10001"  
## [1] "1000/10001"  
## [1] "1100/10001"  
## [1] "1200/10001"  
## [1] "1300/10001"  
## [1] "1400/10001"  
## [1] "1500/10001"  
## [1] "1600/10001"  
## [1] "1700/10001"  
## [1] "1800/10001"  
## [1] "1900/10001"  
## [1] "2000/10001"  
## [1] "2100/10001"  
## [1] "2200/10001"  
## [1] "2300/10001"  
## [1] "2400/10001"  
## [1] "2500/10001"  
## [1] "2600/10001"  
## [1] "2700/10001"  
## [1] "2800/10001"  
## [1] "2900/10001"  
## [1] "3000/10001"  
## [1] "3100/10001"  
## [1] "3200/10001"  
## [1] "3300/10001"  
## [1] "3400/10001"  
## [1] "3500/10001"  
## [1] "3600/10001"  
## [1] "3700/10001"  
## [1] "3800/10001"  
## [1] "3900/10001"  
## [1] "4000/10001"  
## [1] "4100/10001"  
## [1] "4200/10001"  
## [1] "4300/10001"  
## [1] "4400/10001"  
## [1] "4500/10001"  
## [1] "4600/10001"  
## [1] "4700/10001"  
## [1] "4800/10001"  
## [1] "4900/10001"  
## [1] "5000/10001"  
## [1] "5100/10001"  
## [1] "5200/10001"  
## [1] "5300/10001"  
## [1] "5400/10001"  
## [1] "5500/10001"  
## [1] "5600/10001"  
## [1] "5700/10001"  
## [1] "5800/10001"  
## [1] "5900/10001"  
## [1] "6000/10001"  
## [1] "6100/10001"  
## [1] "6200/10001"  
## [1] "6300/10001"  
## [1] "6400/10001"  
## [1] "6500/10001"  
## [1] "6600/10001"  
## [1] "6700/10001"  
## [1] "6800/10001"  
## [1] "6900/10001"  
## [1] "7000/10001"  
## [1] "7100/10001"  
## [1] "7200/10001"  
## [1] "7300/10001"  
## [1] "7400/10001"  
## [1] "7500/10001"  
## [1] "7600/10001"  
## [1] "7700/10001"  
## [1] "7800/10001"  
## [1] "7900/10001"  
## [1] "8000/10001"  
## [1] "8100/10001"  
## [1] "8200/10001"  
## [1] "8300/10001"  
## [1] "8400/10001"  
## [1] "8500/10001"  
## [1] "8600/10001"  
## [1] "8700/10001"  
## [1] "8800/10001"  
## [1] "8900/10001"  
## [1] "9000/10001"  
## [1] "9100/10001"  
## [1] "9200/10001"  
## [1] "9300/10001"  
## [1] "9400/10001"  
## [1] "9500/10001"  
## [1] "9600/10001"  
## [1] "9700/10001"  
## [1] "9800/10001"  
## [1] "9900/10001"  
## [1] "10000/10001"

# Get threshold value that maximizes F1  
# Get F1 thresholds  
maxF1 <- max(metrics$f1, na.rm = TRUE)   
maxF1 #

## [1] 0.4744277

theshF1 <- metrics$thresh[which.max(metrics$f1)]   
theshF1 #

## [1] 0.1082

# Test data  
predicted <- predict(lda.fit, newdata = test\_data, type = "prob")  
predicted\_classes <- ifelse(predicted[, "yes"] > theshF1, 'yes', 'no')  
confusion\_matrix <- confusionMatrix(as.factor(predicted\_classes), as.factor(test\_data$y))

## Warning in confusionMatrix.default(as.factor(predicted\_classes),  
## as.factor(test\_data$y)): Levels are not in the same order for reference and  
## data. Refactoring data to match.

confusion\_matrix

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction yes no  
## yes 410 563  
## no 446 6699  
##   
## Accuracy : 0.8757   
## 95% CI : (0.8683, 0.8828)  
## No Information Rate : 0.8946   
## P-Value [Acc > NIR] : 1.0000000   
##   
## Kappa : 0.3786   
##   
## Mcnemar's Test P-Value : 0.0002604   
##   
## Sensitivity : 0.47897   
## Specificity : 0.92247   
## Pos Pred Value : 0.42138   
## Neg Pred Value : 0.93758   
## Prevalence : 0.10544   
## Detection Rate : 0.05051   
## Detection Prevalence : 0.11986   
## Balanced Accuracy : 0.70072   
##   
## 'Positive' Class : yes   
##

confusion\_matrix$byClass['F1'] # 0.4640

## F1   
## 0.4483324

#AUC  
  
# Assuming `predicted` contains the predicted probabilities for the 'yes' class  
roc <- roc(response = test\_data$y,   
 predictor = as.numeric(as.character(predicted[, "yes"])),  
 levels = rev(levels(test\_data$y))) # Ensure correct ordering of levels if needed

## Setting direction: controls < cases

# Print the AUROC  
auc(roc)

## Area under the curve: 0.7502

#LDA Model--- using numeric variables pdays + previous + emp.var.rate + cons.price.idx + nr.employed  
  
# Convert the binary outcome to a factor  
train\_data$y <- as.factor(train\_data$y)  
  
  
fitControl<-trainControl(method="repeatedcv",number=5,repeats=1,classProbs=TRUE, summaryFunction=mnLogLoss)  
set.seed(1234)  
  
  
lda.fit<-train(y~ pdays + previous + emp.var.rate + cons.price.idx + nr.employed,  
 data=train\_data,  
 method="lda",  
 trControl=fitControl,  
 metric="logLoss")  
  
# Get threshold  
metrics = data.frame(thresh=seq(0, 1, by = 0.0001))  
num\_thresh <- nrow(metrics)  
metrics$sensitivity <- 1  
metrics$specificity <- 1  
metrics$ppv <- 1  
metrics$npv <- 1  
metrics$accuracy <- 1  
metrics$f1 <- 1  
predicted <- predict(lda.fit, newdata = train\_data, type = "prob")  
for (i in 1:num\_thresh){  
 if(i %% 100 == 0) {  
 print(paste(i,'/',num\_thresh,sep=''))  
 }  
   
 # Confusion Matrix  
 predicted\_classes <- ifelse(predicted[, "yes"] > metrics$thresh[i], 'yes', 'no')  
 predicted\_classes\_factor <- factor(predicted\_classes, levels = levels(train\_data$y))  
 confusion\_matrix <- confusionMatrix(predicted\_classes\_factor, train\_data$y)  
   
   
 # Metrics  
 metrics$sensitivity[i] <- as.numeric(confusion\_matrix$byClass['Sensitivity'])  
 metrics$specificity[i] <- as.numeric(confusion\_matrix$byClass['Specificity'])  
 metrics$ppv[i] <- as.numeric(confusion\_matrix$byClass['Pos Pred Value'])  
 metrics$npv[i] <- as.numeric(confusion\_matrix$byClass['Neg Pred Value'])  
 metrics$accuracy[i] <- as.numeric(confusion\_matrix$overall['Accuracy'])  
 metrics$f1[i] <- as.numeric(confusion\_matrix$byClass['F1'])  
}

## [1] "100/10001"  
## [1] "200/10001"  
## [1] "300/10001"  
## [1] "400/10001"  
## [1] "500/10001"  
## [1] "600/10001"  
## [1] "700/10001"  
## [1] "800/10001"  
## [1] "900/10001"  
## [1] "1000/10001"  
## [1] "1100/10001"  
## [1] "1200/10001"  
## [1] "1300/10001"  
## [1] "1400/10001"  
## [1] "1500/10001"  
## [1] "1600/10001"  
## [1] "1700/10001"  
## [1] "1800/10001"  
## [1] "1900/10001"  
## [1] "2000/10001"  
## [1] "2100/10001"  
## [1] "2200/10001"  
## [1] "2300/10001"  
## [1] "2400/10001"  
## [1] "2500/10001"  
## [1] "2600/10001"  
## [1] "2700/10001"  
## [1] "2800/10001"  
## [1] "2900/10001"  
## [1] "3000/10001"  
## [1] "3100/10001"  
## [1] "3200/10001"  
## [1] "3300/10001"  
## [1] "3400/10001"  
## [1] "3500/10001"  
## [1] "3600/10001"  
## [1] "3700/10001"  
## [1] "3800/10001"  
## [1] "3900/10001"  
## [1] "4000/10001"  
## [1] "4100/10001"  
## [1] "4200/10001"  
## [1] "4300/10001"  
## [1] "4400/10001"  
## [1] "4500/10001"  
## [1] "4600/10001"  
## [1] "4700/10001"  
## [1] "4800/10001"  
## [1] "4900/10001"  
## [1] "5000/10001"  
## [1] "5100/10001"  
## [1] "5200/10001"  
## [1] "5300/10001"  
## [1] "5400/10001"  
## [1] "5500/10001"  
## [1] "5600/10001"  
## [1] "5700/10001"  
## [1] "5800/10001"  
## [1] "5900/10001"  
## [1] "6000/10001"  
## [1] "6100/10001"  
## [1] "6200/10001"  
## [1] "6300/10001"  
## [1] "6400/10001"  
## [1] "6500/10001"  
## [1] "6600/10001"  
## [1] "6700/10001"  
## [1] "6800/10001"  
## [1] "6900/10001"  
## [1] "7000/10001"  
## [1] "7100/10001"  
## [1] "7200/10001"  
## [1] "7300/10001"  
## [1] "7400/10001"  
## [1] "7500/10001"  
## [1] "7600/10001"  
## [1] "7700/10001"  
## [1] "7800/10001"  
## [1] "7900/10001"  
## [1] "8000/10001"  
## [1] "8100/10001"  
## [1] "8200/10001"  
## [1] "8300/10001"  
## [1] "8400/10001"  
## [1] "8500/10001"  
## [1] "8600/10001"  
## [1] "8700/10001"  
## [1] "8800/10001"  
## [1] "8900/10001"  
## [1] "9000/10001"  
## [1] "9100/10001"  
## [1] "9200/10001"  
## [1] "9300/10001"  
## [1] "9400/10001"  
## [1] "9500/10001"  
## [1] "9600/10001"  
## [1] "9700/10001"  
## [1] "9800/10001"  
## [1] "9900/10001"  
## [1] "10000/10001"

# Get threshold value that maximizes F1  
# Get F1 thresholds  
maxF1 <- max(metrics$f1, na.rm = TRUE) #   
maxF1 #0.4744

## [1] 0.4647335

theshF1 <- metrics$thresh[which.max(metrics$f1)]   
theshF1 # 0.1082

## [1] 0.1254

# Test data  
predicted <- predict(lda.fit, newdata = test\_data, type = "prob")  
predicted\_classes <- ifelse(predicted[, "yes"] > theshF1, 'yes', 'no')  
confusion\_matrix <- confusionMatrix(as.factor(predicted\_classes), as.factor(test\_data$y))

## Warning in confusionMatrix.default(as.factor(predicted\_classes),  
## as.factor(test\_data$y)): Levels are not in the same order for reference and  
## data. Refactoring data to match.

confusion\_matrix # Sensitivity = 0.51043, Specificity = 0.9143, PPV = 0.4254, NPV = 0.9376, Prevalence = 0.1106

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction yes no  
## yes 376 470  
## no 480 6792  
##   
## Accuracy : 0.883   
## 95% CI : (0.8758, 0.8899)  
## No Information Rate : 0.8946   
## P-Value [Acc > NIR] : 0.9996   
##   
## Kappa : 0.3765   
##   
## Mcnemar's Test P-Value : 0.7703   
##   
## Sensitivity : 0.43925   
## Specificity : 0.93528   
## Pos Pred Value : 0.44444   
## Neg Pred Value : 0.93399   
## Prevalence : 0.10544   
## Detection Rate : 0.04632   
## Detection Prevalence : 0.10421   
## Balanced Accuracy : 0.68727   
##   
## 'Positive' Class : yes   
##

confusion\_matrix$byClass['F1'] # 0.4641

## F1   
## 0.4418331

#AUC  
  
# Assuming `predicted` contains the predicted probabilities for the 'yes' class  
roc <- roc(response = test\_data$y,   
 predictor = as.numeric(as.character(predicted[, "yes"])),  
 levels = rev(levels(test\_data$y))) # Ensure correct ordering of levels if needed

## Setting direction: controls < cases

# Print the AUROC  
auc(roc) # 0.7599

## Area under the curve: 0.7439

#LDA Model--- using numeric variables emp.var.rate + cons.price.idx + euribor3m   
  
  
fitControl<-trainControl(method="repeatedcv",number=5,repeats=1,classProbs=TRUE, summaryFunction=mnLogLoss)  
set.seed(1234)  
  
  
lda.fit<-train(y~ emp.var.rate + cons.price.idx + euribor3m ,  
 data=train\_data,  
 method="lda",  
 trControl=fitControl,  
 metric="logLoss")  
  
# Get threshold  
metrics = data.frame(thresh=seq(0, 1, by = 0.0001))  
num\_thresh <- nrow(metrics)  
metrics$sensitivity <- 1  
metrics$specificity <- 1  
metrics$ppv <- 1  
metrics$npv <- 1  
metrics$accuracy <- 1  
metrics$f1 <- 1  
predicted <- predict(lda.fit, newdata = train\_data, type = "prob")  
for (i in 1:num\_thresh){  
 if(i %% 100 == 0) {  
 print(paste(i,'/',num\_thresh,sep=''))  
 }  
   
 # Confusion Matrix  
 predicted\_classes <- ifelse(predicted[, "yes"] > metrics$thresh[i], 'yes', 'no')  
 predicted\_classes\_factor <- factor(predicted\_classes, levels = levels(train\_data$y))  
 confusion\_matrix <- confusionMatrix(predicted\_classes\_factor, train\_data$y)  
   
   
 # Metrics  
 metrics$sensitivity[i] <- as.numeric(confusion\_matrix$byClass['Sensitivity'])  
 metrics$specificity[i] <- as.numeric(confusion\_matrix$byClass['Specificity'])  
 metrics$ppv[i] <- as.numeric(confusion\_matrix$byClass['Pos Pred Value'])  
 metrics$npv[i] <- as.numeric(confusion\_matrix$byClass['Neg Pred Value'])  
 metrics$accuracy[i] <- as.numeric(confusion\_matrix$overall['Accuracy'])  
 metrics$f1[i] <- as.numeric(confusion\_matrix$byClass['F1'])  
}

## [1] "100/10001"  
## [1] "200/10001"  
## [1] "300/10001"  
## [1] "400/10001"  
## [1] "500/10001"  
## [1] "600/10001"  
## [1] "700/10001"  
## [1] "800/10001"  
## [1] "900/10001"  
## [1] "1000/10001"  
## [1] "1100/10001"  
## [1] "1200/10001"  
## [1] "1300/10001"  
## [1] "1400/10001"  
## [1] "1500/10001"  
## [1] "1600/10001"  
## [1] "1700/10001"  
## [1] "1800/10001"  
## [1] "1900/10001"  
## [1] "2000/10001"  
## [1] "2100/10001"  
## [1] "2200/10001"  
## [1] "2300/10001"  
## [1] "2400/10001"  
## [1] "2500/10001"  
## [1] "2600/10001"  
## [1] "2700/10001"  
## [1] "2800/10001"  
## [1] "2900/10001"  
## [1] "3000/10001"  
## [1] "3100/10001"  
## [1] "3200/10001"  
## [1] "3300/10001"  
## [1] "3400/10001"  
## [1] "3500/10001"  
## [1] "3600/10001"  
## [1] "3700/10001"  
## [1] "3800/10001"  
## [1] "3900/10001"  
## [1] "4000/10001"  
## [1] "4100/10001"  
## [1] "4200/10001"  
## [1] "4300/10001"  
## [1] "4400/10001"  
## [1] "4500/10001"  
## [1] "4600/10001"  
## [1] "4700/10001"  
## [1] "4800/10001"  
## [1] "4900/10001"  
## [1] "5000/10001"  
## [1] "5100/10001"  
## [1] "5200/10001"  
## [1] "5300/10001"  
## [1] "5400/10001"  
## [1] "5500/10001"  
## [1] "5600/10001"  
## [1] "5700/10001"  
## [1] "5800/10001"  
## [1] "5900/10001"  
## [1] "6000/10001"  
## [1] "6100/10001"  
## [1] "6200/10001"  
## [1] "6300/10001"  
## [1] "6400/10001"  
## [1] "6500/10001"  
## [1] "6600/10001"  
## [1] "6700/10001"  
## [1] "6800/10001"  
## [1] "6900/10001"  
## [1] "7000/10001"  
## [1] "7100/10001"  
## [1] "7200/10001"  
## [1] "7300/10001"  
## [1] "7400/10001"  
## [1] "7500/10001"  
## [1] "7600/10001"  
## [1] "7700/10001"  
## [1] "7800/10001"  
## [1] "7900/10001"  
## [1] "8000/10001"  
## [1] "8100/10001"  
## [1] "8200/10001"  
## [1] "8300/10001"  
## [1] "8400/10001"  
## [1] "8500/10001"  
## [1] "8600/10001"  
## [1] "8700/10001"  
## [1] "8800/10001"  
## [1] "8900/10001"  
## [1] "9000/10001"  
## [1] "9100/10001"  
## [1] "9200/10001"  
## [1] "9300/10001"  
## [1] "9400/10001"  
## [1] "9500/10001"  
## [1] "9600/10001"  
## [1] "9700/10001"  
## [1] "9800/10001"  
## [1] "9900/10001"  
## [1] "10000/10001"

# Get threshold value that maximizes F1  
# Get F1 thresholds  
maxF1 <- max(metrics$f1, na.rm = TRUE) #   
maxF1 # 0.4561

## [1] 0.4560976

theshF1 <- metrics$thresh[which.max(metrics$f1)]   
theshF1 # 0.2306

## [1] 0.2306

# Test data  
predicted <- predict(lda.fit, newdata = test\_data, type = "prob")  
predicted\_classes <- ifelse(predicted[, "yes"] > theshF1, 'yes', 'no')  
confusion\_matrix <- confusionMatrix(as.factor(predicted\_classes), as.factor(test\_data$y))

## Warning in confusionMatrix.default(as.factor(predicted\_classes),  
## as.factor(test\_data$y)): Levels are not in the same order for reference and  
## data. Refactoring data to match.

confusion\_matrix # Sensitivity = 0.4522, Specificity = 0.9335, PPV = 0.4583, NPV = 0.9320, Prevalence = 0.1106

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction yes no  
## yes 357 422  
## no 499 6840  
##   
## Accuracy : 0.8865   
## 95% CI : (0.8794, 0.8934)  
## No Information Rate : 0.8946   
## P-Value [Acc > NIR] : 0.99051   
##   
## Kappa : 0.3738   
##   
## Mcnemar's Test P-Value : 0.01227   
##   
## Sensitivity : 0.41706   
## Specificity : 0.94189   
## Pos Pred Value : 0.45828   
## Neg Pred Value : 0.93201   
## Prevalence : 0.10544   
## Detection Rate : 0.04398   
## Detection Prevalence : 0.09596   
## Balanced Accuracy : 0.67947   
##   
## 'Positive' Class : yes   
##

confusion\_matrix$byClass['F1'] # 0.4552

## F1   
## 0.4366972

# Assuming `predicted` contains the predicted probabilities for the 'yes' class  
roc <- roc(response = test\_data$y,   
 predictor = as.numeric(as.character(predicted[, "yes"])),  
 levels = rev(levels(test\_data$y))) # Ensure correct ordering of levels if needed

## Setting direction: controls < cases

# Print the AUROC  
auc(roc) # 0.7506

## Area under the curve: 0.7414

# Training data  
predicted <- predict(lda.fit, newdata = train\_data, type = "prob")  
predicted\_classes <- ifelse(predicted[, "yes"] > theshF1, 'yes', 'no')  
confusion\_matrix <- confusionMatrix(as.factor(predicted\_classes), as.factor(train\_data$y))

## Warning in confusionMatrix.default(as.factor(predicted\_classes),  
## as.factor(train\_data$y)): Levels are not in the same order for reference and  
## data. Refactoring data to match.

confusion\_matrix

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction yes no  
## yes 1683 1968  
## no 2046 27253  
##   
## Accuracy : 0.8782   
## 95% CI : (0.8746, 0.8817)  
## No Information Rate : 0.8868   
## P-Value [Acc > NIR] : 1.0000   
##   
## Kappa : 0.3875   
##   
## Mcnemar's Test P-Value : 0.2242   
##   
## Sensitivity : 0.45133   
## Specificity : 0.93265   
## Pos Pred Value : 0.46097   
## Neg Pred Value : 0.93017   
## Prevalence : 0.11317   
## Detection Rate : 0.05108   
## Detection Prevalence : 0.11080   
## Balanced Accuracy : 0.69199   
##   
## 'Positive' Class : yes   
##

confusion\_matrix$byClass['F1'] #0.4561

## F1   
## 0.4560976

# Assuming `predicted` contains the predicted probabilities for the 'yes' class  
roc <- roc(response = train\_data$y,   
 predictor = as.numeric(as.character(predicted[, "yes"])),  
 levels = rev(levels(train\_data$y))) # Ensure correct ordering of levels if needed

## Setting direction: controls < cases

# Print the AUROC  
auc(roc) # 0.7555

## Area under the curve: 0.7555

plot(roc,print.thres="best",col="red")  
title(main = 'ROC Curve for LDA', line = 3)

A graph with a red line

Description automatically generated

# QDA Model--simple model  
  
fitControl<-trainControl(method="repeatedcv",number=5,repeats=1,classProbs=TRUE, summaryFunction=mnLogLoss)  
set.seed(1234)  
  
qda.fit <- train(y~ month + poutcome + emp.var.rate + contact + cons.price.idx,  
 data = train\_data,  
 method = "qda",  
 trControl = fitControl,  
 metric = "logLoss")  
  
# Get threshold  
metrics = data.frame(thresh=seq(0, 1, by = 0.0001))  
num\_thresh <- nrow(metrics)  
metrics$sensitivity <- 1  
metrics$specificity <- 1  
metrics$ppv <- 1  
metrics$npv <- 1  
metrics$accuracy <- 1  
metrics$f1 <- 1  
predicted <- predict(qda.fit, newdata = train\_data, type = "prob")  
for (i in 1:num\_thresh){  
 if(i %% 100 == 0) {  
 print(paste(i,'/',num\_thresh,sep=''))  
 }  
   
 # Confusion Matrix  
 predicted\_classes <- ifelse(predicted[, "yes"] > metrics$thresh[i], 'yes', 'no')  
 predicted\_classes\_factor <- factor(predicted\_classes, levels = levels(train\_data$y))  
 confusion\_matrix <- confusionMatrix(predicted\_classes\_factor, train\_data$y)  
   
   
 # Metrics  
 metrics$sensitivity[i] <- as.numeric(confusion\_matrix$byClass['Sensitivity'])  
 metrics$specificity[i] <- as.numeric(confusion\_matrix$byClass['Specificity'])  
 metrics$ppv[i] <- as.numeric(confusion\_matrix$byClass['Pos Pred Value'])  
 metrics$npv[i] <- as.numeric(confusion\_matrix$byClass['Neg Pred Value'])  
 metrics$accuracy[i] <- as.numeric(confusion\_matrix$overall['Accuracy'])  
 metrics$f1[i] <- as.numeric(confusion\_matrix$byClass['F1'])  
}

## [1] "100/10001"  
## [1] "200/10001"  
## [1] "300/10001"  
## [1] "400/10001"  
## [1] "500/10001"  
## [1] "600/10001"  
## [1] "700/10001"  
## [1] "800/10001"  
## [1] "900/10001"  
## [1] "1000/10001"  
## [1] "1100/10001"  
## [1] "1200/10001"  
## [1] "1300/10001"  
## [1] "1400/10001"  
## [1] "1500/10001"  
## [1] "1600/10001"  
## [1] "1700/10001"  
## [1] "1800/10001"  
## [1] "1900/10001"  
## [1] "2000/10001"  
## [1] "2100/10001"  
## [1] "2200/10001"  
## [1] "2300/10001"  
## [1] "2400/10001"  
## [1] "2500/10001"  
## [1] "2600/10001"  
## [1] "2700/10001"  
## [1] "2800/10001"  
## [1] "2900/10001"  
## [1] "3000/10001"  
## [1] "3100/10001"  
## [1] "3200/10001"  
## [1] "3300/10001"  
## [1] "3400/10001"  
## [1] "3500/10001"  
## [1] "3600/10001"  
## [1] "3700/10001"  
## [1] "3800/10001"  
## [1] "3900/10001"  
## [1] "4000/10001"  
## [1] "4100/10001"  
## [1] "4200/10001"  
## [1] "4300/10001"  
## [1] "4400/10001"  
## [1] "4500/10001"  
## [1] "4600/10001"  
## [1] "4700/10001"  
## [1] "4800/10001"  
## [1] "4900/10001"  
## [1] "5000/10001"  
## [1] "5100/10001"  
## [1] "5200/10001"  
## [1] "5300/10001"  
## [1] "5400/10001"  
## [1] "5500/10001"  
## [1] "5600/10001"  
## [1] "5700/10001"  
## [1] "5800/10001"  
## [1] "5900/10001"  
## [1] "6000/10001"  
## [1] "6100/10001"  
## [1] "6200/10001"  
## [1] "6300/10001"  
## [1] "6400/10001"  
## [1] "6500/10001"  
## [1] "6600/10001"  
## [1] "6700/10001"  
## [1] "6800/10001"  
## [1] "6900/10001"  
## [1] "7000/10001"  
## [1] "7100/10001"  
## [1] "7200/10001"  
## [1] "7300/10001"  
## [1] "7400/10001"  
## [1] "7500/10001"  
## [1] "7600/10001"  
## [1] "7700/10001"  
## [1] "7800/10001"  
## [1] "7900/10001"  
## [1] "8000/10001"  
## [1] "8100/10001"  
## [1] "8200/10001"  
## [1] "8300/10001"  
## [1] "8400/10001"  
## [1] "8500/10001"  
## [1] "8600/10001"  
## [1] "8700/10001"  
## [1] "8800/10001"  
## [1] "8900/10001"  
## [1] "9000/10001"  
## [1] "9100/10001"  
## [1] "9200/10001"  
## [1] "9300/10001"  
## [1] "9400/10001"  
## [1] "9500/10001"  
## [1] "9600/10001"  
## [1] "9700/10001"  
## [1] "9800/10001"  
## [1] "9900/10001"  
## [1] "10000/10001"

# Get threshold value that maximizes F1  
# Get F1 thresholds  
maxF1 <- max(metrics$f1, na.rm = TRUE) #   
maxF1 # 0.4692

## [1] 0.4692364

theshF1 <- metrics$thresh[which.max(metrics$f1)]   
theshF1 # 0.0227

## [1] 0.0227

# Test data  
predicted <- predict(qda.fit, newdata = test\_data, type = "prob")  
predicted\_classes <- ifelse(predicted[, "yes"] > theshF1, 'yes', 'no')  
confusion\_matrix <- confusionMatrix(as.factor(predicted\_classes), as.factor(test\_data$y))

## Warning in confusionMatrix.default(as.factor(predicted\_classes),  
## as.factor(test\_data$y)): Levels are not in the same order for reference and  
## data. Refactoring data to match.

confusion\_matrix # Sensitivity = 0.5917, Specificity = 0.8781, PPV = 0.3764, NPV = 0.9453, Prevalence =0.1106

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction yes no  
## yes 484 828  
## no 372 6434  
##   
## Accuracy : 0.8522   
## 95% CI : (0.8443, 0.8598)  
## No Information Rate : 0.8946   
## P-Value [Acc > NIR] : 1   
##   
## Kappa : 0.3655   
##   
## Mcnemar's Test P-Value : <2e-16   
##   
## Sensitivity : 0.56542   
## Specificity : 0.88598   
## Pos Pred Value : 0.36890   
## Neg Pred Value : 0.94534   
## Prevalence : 0.10544   
## Detection Rate : 0.05962   
## Detection Prevalence : 0.16162   
## Balanced Accuracy : 0.72570   
##   
## 'Positive' Class : yes   
##

confusion\_matrix$byClass['F1'] # 0.4600

## F1   
## 0.4464945

# Assuming `predicted` contains the predicted probabilities for the 'yes' class  
roc.qda <- roc(response = test\_data$y,   
 predictor = as.numeric(as.character(predicted[, "yes"])),  
 levels = rev(levels(test\_data$y))) # Ensure correct ordering of levels if needed

## Setting direction: controls < cases

# Print the AUROC  
auc(roc.qda) # 0.7811

## Area under the curve: 0.7741

# QDA --- using numeric variables pdays + previous + emp.var.rate + cons.price.idx + nr.employed  
  
fitControl<-trainControl(method="repeatedcv",number=5,repeats=1,classProbs=TRUE, summaryFunction=mnLogLoss)  
set.seed(1234)  
  
qda.fit <- train(y~ pdays + previous + emp.var.rate + cons.price.idx + nr.employed,  
 data = train\_data,  
 method = "qda",  
 trControl = fitControl,  
 metric = "logLoss")  
  
# Get threshold  
metrics = data.frame(thresh=seq(0, 1, by = 0.0001))  
num\_thresh <- nrow(metrics)  
metrics$sensitivity <- 1  
metrics$specificity <- 1  
metrics$ppv <- 1  
metrics$npv <- 1  
metrics$accuracy <- 1  
metrics$f1 <- 1  
predicted <- predict(qda.fit, newdata = train\_data, type = "prob")  
for (i in 1:num\_thresh){  
 if(i %% 100 == 0) {  
 print(paste(i,'/',num\_thresh,sep=''))  
 }  
   
 # Confusion Matrix  
 predicted\_classes <- ifelse(predicted[, "yes"] > metrics$thresh[i], 'yes', 'no')  
 predicted\_classes\_factor <- factor(predicted\_classes, levels = levels(train\_data$y))  
 confusion\_matrix <- confusionMatrix(predicted\_classes\_factor, train\_data$y)  
   
   
 # Metrics  
 metrics$sensitivity[i] <- as.numeric(confusion\_matrix$byClass['Sensitivity'])  
 metrics$specificity[i] <- as.numeric(confusion\_matrix$byClass['Specificity'])  
 metrics$ppv[i] <- as.numeric(confusion\_matrix$byClass['Pos Pred Value'])  
 metrics$npv[i] <- as.numeric(confusion\_matrix$byClass['Neg Pred Value'])  
 metrics$accuracy[i] <- as.numeric(confusion\_matrix$overall['Accuracy'])  
 metrics$f1[i] <- as.numeric(confusion\_matrix$byClass['F1'])  
}

## [1] "100/10001"  
## [1] "200/10001"  
## [1] "300/10001"  
## [1] "400/10001"  
## [1] "500/10001"  
## [1] "600/10001"  
## [1] "700/10001"  
## [1] "800/10001"  
## [1] "900/10001"  
## [1] "1000/10001"  
## [1] "1100/10001"  
## [1] "1200/10001"  
## [1] "1300/10001"  
## [1] "1400/10001"  
## [1] "1500/10001"  
## [1] "1600/10001"  
## [1] "1700/10001"  
## [1] "1800/10001"  
## [1] "1900/10001"  
## [1] "2000/10001"  
## [1] "2100/10001"  
## [1] "2200/10001"  
## [1] "2300/10001"  
## [1] "2400/10001"  
## [1] "2500/10001"  
## [1] "2600/10001"  
## [1] "2700/10001"  
## [1] "2800/10001"  
## [1] "2900/10001"  
## [1] "3000/10001"  
## [1] "3100/10001"  
## [1] "3200/10001"  
## [1] "3300/10001"  
## [1] "3400/10001"  
## [1] "3500/10001"  
## [1] "3600/10001"  
## [1] "3700/10001"  
## [1] "3800/10001"  
## [1] "3900/10001"  
## [1] "4000/10001"  
## [1] "4100/10001"  
## [1] "4200/10001"  
## [1] "4300/10001"  
## [1] "4400/10001"  
## [1] "4500/10001"  
## [1] "4600/10001"  
## [1] "4700/10001"  
## [1] "4800/10001"  
## [1] "4900/10001"  
## [1] "5000/10001"  
## [1] "5100/10001"  
## [1] "5200/10001"  
## [1] "5300/10001"  
## [1] "5400/10001"  
## [1] "5500/10001"  
## [1] "5600/10001"  
## [1] "5700/10001"  
## [1] "5800/10001"  
## [1] "5900/10001"  
## [1] "6000/10001"  
## [1] "6100/10001"  
## [1] "6200/10001"  
## [1] "6300/10001"  
## [1] "6400/10001"  
## [1] "6500/10001"  
## [1] "6600/10001"  
## [1] "6700/10001"  
## [1] "6800/10001"  
## [1] "6900/10001"  
## [1] "7000/10001"  
## [1] "7100/10001"  
## [1] "7200/10001"  
## [1] "7300/10001"  
## [1] "7400/10001"  
## [1] "7500/10001"  
## [1] "7600/10001"  
## [1] "7700/10001"  
## [1] "7800/10001"  
## [1] "7900/10001"  
## [1] "8000/10001"  
## [1] "8100/10001"  
## [1] "8200/10001"  
## [1] "8300/10001"  
## [1] "8400/10001"  
## [1] "8500/10001"  
## [1] "8600/10001"  
## [1] "8700/10001"  
## [1] "8800/10001"  
## [1] "8900/10001"  
## [1] "9000/10001"  
## [1] "9100/10001"  
## [1] "9200/10001"  
## [1] "9300/10001"  
## [1] "9400/10001"  
## [1] "9500/10001"  
## [1] "9600/10001"  
## [1] "9700/10001"  
## [1] "9800/10001"  
## [1] "9900/10001"  
## [1] "10000/10001"

# Get threshold value that maximizes F1  
# Get F1 thresholds  
maxF1 <- max(metrics$f1, na.rm = TRUE) #   
maxF1 # 0.4382

## [1] 0.4381683

theshF1 <- metrics$thresh[which.max(metrics$f1)]   
theshF1 # 0.0635

## [1] 0.0635

# Test data  
predicted <- predict(qda.fit, newdata = test\_data, type = "prob")  
predicted\_classes <- ifelse(predicted[, "yes"] > theshF1, 'yes', 'no')  
confusion\_matrix <- confusionMatrix(as.factor(predicted\_classes), as.factor(test\_data$y))

## Warning in confusionMatrix.default(as.factor(predicted\_classes),  
## as.factor(test\_data$y)): Levels are not in the same order for reference and  
## data. Refactoring data to match.

confusion\_matrix # accuracy = 0.8546, Sensitivity = 0.4951, Specificity = 0.8993, PPV = 0.3793, NPV = 0.9347, Prevalence =0.1106

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction yes no  
## yes 396 673  
## no 460 6589  
##   
## Accuracy : 0.8604   
## 95% CI : (0.8527, 0.8679)  
## No Information Rate : 0.8946   
## P-Value [Acc > NIR] : 1   
##   
## Kappa : 0.3334   
##   
## Mcnemar's Test P-Value : 3.01e-10   
##   
## Sensitivity : 0.46262   
## Specificity : 0.90733   
## Pos Pred Value : 0.37044   
## Neg Pred Value : 0.93474   
## Prevalence : 0.10544   
## Detection Rate : 0.04878   
## Detection Prevalence : 0.13168   
## Balanced Accuracy : 0.68497   
##   
## 'Positive' Class : yes   
##

confusion\_matrix$byClass['F1'] # 0.4295

## F1   
## 0.4114286

# Assuming `predicted` contains the predicted probabilities for the 'yes' class  
roc.qda <- roc(response = test\_data$y,   
 predictor = as.numeric(as.character(predicted[, "yes"])),  
 levels = rev(levels(test\_data$y))) # Ensure correct ordering of levels if needed

## Setting direction: controls < cases

# Print the AUROC  
auc(roc.qda) # 0.7527

## Area under the curve: 0.7428

# QDA --- using numeric variables emp.var.rate + cons.price.idx + euribor3m  
  
fitControl<-trainControl(method="repeatedcv",number=5,repeats=1,classProbs=TRUE, summaryFunction=mnLogLoss)  
set.seed(1234)  
  
qda.fit <- train(y~ emp.var.rate + cons.price.idx + euribor3m,  
 data = train\_data,  
 method = "qda",  
 trControl = fitControl,  
 metric = "logLoss")  
  
# Get threshold  
metrics = data.frame(thresh=seq(0, 1, by = 0.0001))  
num\_thresh <- nrow(metrics)  
metrics$sensitivity <- 1  
metrics$specificity <- 1  
metrics$ppv <- 1  
metrics$npv <- 1  
metrics$accuracy <- 1  
metrics$f1 <- 1  
predicted <- predict(qda.fit, newdata = train\_data, type = "prob")  
for (i in 1:num\_thresh){  
 if(i %% 100 == 0) {  
 print(paste(i,'/',num\_thresh,sep=''))  
 }  
   
 # Confusion Matrix  
 predicted\_classes <- ifelse(predicted[, "yes"] > metrics$thresh[i], 'yes', 'no')  
 predicted\_classes\_factor <- factor(predicted\_classes, levels = levels(train\_data$y))  
 confusion\_matrix <- confusionMatrix(predicted\_classes\_factor, train\_data$y)  
   
   
 # Metrics  
 metrics$sensitivity[i] <- as.numeric(confusion\_matrix$byClass['Sensitivity'])  
 metrics$specificity[i] <- as.numeric(confusion\_matrix$byClass['Specificity'])  
 metrics$ppv[i] <- as.numeric(confusion\_matrix$byClass['Pos Pred Value'])  
 metrics$npv[i] <- as.numeric(confusion\_matrix$byClass['Neg Pred Value'])  
 metrics$accuracy[i] <- as.numeric(confusion\_matrix$overall['Accuracy'])  
 metrics$f1[i] <- as.numeric(confusion\_matrix$byClass['F1'])  
}

## [1] "100/10001"  
## [1] "200/10001"  
## [1] "300/10001"  
## [1] "400/10001"  
## [1] "500/10001"  
## [1] "600/10001"  
## [1] "700/10001"  
## [1] "800/10001"  
## [1] "900/10001"  
## [1] "1000/10001"  
## [1] "1100/10001"  
## [1] "1200/10001"  
## [1] "1300/10001"  
## [1] "1400/10001"  
## [1] "1500/10001"  
## [1] "1600/10001"  
## [1] "1700/10001"  
## [1] "1800/10001"  
## [1] "1900/10001"  
## [1] "2000/10001"  
## [1] "2100/10001"  
## [1] "2200/10001"  
## [1] "2300/10001"  
## [1] "2400/10001"  
## [1] "2500/10001"  
## [1] "2600/10001"  
## [1] "2700/10001"  
## [1] "2800/10001"  
## [1] "2900/10001"  
## [1] "3000/10001"  
## [1] "3100/10001"  
## [1] "3200/10001"  
## [1] "3300/10001"  
## [1] "3400/10001"  
## [1] "3500/10001"  
## [1] "3600/10001"  
## [1] "3700/10001"  
## [1] "3800/10001"  
## [1] "3900/10001"  
## [1] "4000/10001"  
## [1] "4100/10001"  
## [1] "4200/10001"  
## [1] "4300/10001"  
## [1] "4400/10001"  
## [1] "4500/10001"  
## [1] "4600/10001"  
## [1] "4700/10001"  
## [1] "4800/10001"  
## [1] "4900/10001"  
## [1] "5000/10001"  
## [1] "5100/10001"  
## [1] "5200/10001"  
## [1] "5300/10001"  
## [1] "5400/10001"  
## [1] "5500/10001"  
## [1] "5600/10001"  
## [1] "5700/10001"  
## [1] "5800/10001"  
## [1] "5900/10001"  
## [1] "6000/10001"  
## [1] "6100/10001"  
## [1] "6200/10001"  
## [1] "6300/10001"  
## [1] "6400/10001"  
## [1] "6500/10001"  
## [1] "6600/10001"  
## [1] "6700/10001"  
## [1] "6800/10001"  
## [1] "6900/10001"  
## [1] "7000/10001"  
## [1] "7100/10001"  
## [1] "7200/10001"  
## [1] "7300/10001"  
## [1] "7400/10001"  
## [1] "7500/10001"  
## [1] "7600/10001"  
## [1] "7700/10001"  
## [1] "7800/10001"  
## [1] "7900/10001"  
## [1] "8000/10001"  
## [1] "8100/10001"  
## [1] "8200/10001"  
## [1] "8300/10001"  
## [1] "8400/10001"  
## [1] "8500/10001"  
## [1] "8600/10001"  
## [1] "8700/10001"  
## [1] "8800/10001"  
## [1] "8900/10001"  
## [1] "9000/10001"  
## [1] "9100/10001"  
## [1] "9200/10001"  
## [1] "9300/10001"  
## [1] "9400/10001"  
## [1] "9500/10001"  
## [1] "9600/10001"  
## [1] "9700/10001"  
## [1] "9800/10001"  
## [1] "9900/10001"  
## [1] "10000/10001"

# Get threshold value that maximizes F1  
# Get F1 thresholds  
maxF1 <- max(metrics$f1, na.rm = TRUE) #   
maxF1 # 0.4681

## [1] 0.4680797

theshF1 <- metrics$thresh[which.max(metrics$f1)]   
theshF1 # 0.1264

## [1] 0.1264

# Test data  
predicted <- predict(qda.fit, newdata = test\_data, type = "prob")  
predicted\_classes <- ifelse(predicted[, "yes"] > theshF1, 'yes', 'no')  
confusion\_matrix <- confusionMatrix(as.factor(predicted\_classes), as.factor(test\_data$y))

## Warning in confusionMatrix.default(as.factor(predicted\_classes),  
## as.factor(test\_data$y)): Levels are not in the same order for reference and  
## data. Refactoring data to match.

confusion\_matrix # accuracy = 0.8765, Sensitivity = 0.4829, Specificity = 0.9255, PPV = 0.4463, NPV = 0.9351, Prevalence = 0.1106

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction yes no  
## yes 385 481  
## no 471 6781  
##   
## Accuracy : 0.8827   
## 95% CI : (0.8755, 0.8897)  
## No Information Rate : 0.8946   
## P-Value [Acc > NIR] : 0.9997   
##   
## Kappa : 0.3816   
##   
## Mcnemar's Test P-Value : 0.7705   
##   
## Sensitivity : 0.44977   
## Specificity : 0.93376   
## Pos Pred Value : 0.44457   
## Neg Pred Value : 0.93505   
## Prevalence : 0.10544   
## Detection Rate : 0.04743   
## Detection Prevalence : 0.10668   
## Balanced Accuracy : 0.69177   
##   
## 'Positive' Class : yes   
##

confusion\_matrix$byClass['F1'] # 0.4639

## F1   
## 0.4471545

# Assuming `predicted` contains the predicted probabilities for the 'yes' class  
roc.qda <- roc(response = test\_data$y,   
 predictor = as.numeric(as.character(predicted[, "yes"])),  
 levels = rev(levels(test\_data$y))) # Ensure correct ordering of levels if needed

## Setting direction: controls < cases

# Print the AUROC  
auc(roc.qda) # 0.7623

## Area under the curve: 0.754

# Training data  
predicted <- predict(qda.fit, newdata = train\_data, type = "prob")  
predicted\_classes <- ifelse(predicted[, "yes"] > theshF1, 'yes', 'no')  
confusion\_matrix <- confusionMatrix(as.factor(predicted\_classes), as.factor(train\_data$y))

## Warning in confusionMatrix.default(as.factor(predicted\_classes),  
## as.factor(train\_data$y)): Levels are not in the same order for reference and  
## data. Refactoring data to match.

confusion\_matrix

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction yes no  
## yes 1833 2270  
## no 1896 26951  
##   
## Accuracy : 0.8736   
## 95% CI : (0.8699, 0.8771)  
## No Information Rate : 0.8868   
## P-Value [Acc > NIR] : 1   
##   
## Kappa : 0.3965   
##   
## Mcnemar's Test P-Value : 7.517e-09   
##   
## Sensitivity : 0.49155   
## Specificity : 0.92232   
## Pos Pred Value : 0.44675   
## Neg Pred Value : 0.93427   
## Prevalence : 0.11317   
## Detection Rate : 0.05563   
## Detection Prevalence : 0.12452   
## Balanced Accuracy : 0.70693   
##   
## 'Positive' Class : yes   
##

confusion\_matrix$byClass['F1'] #0.4681

## F1   
## 0.4680797

# Assuming `predicted` contains the predicted probabilities for the 'yes' class  
roc <- roc(response = train\_data$y,   
 predictor = as.numeric(as.character(predicted[, "yes"])),  
 levels = rev(levels(train\_data$y))) # Ensure correct ordering of levels if needed

## Setting direction: controls < cases

# Print the AUROC  
auc(roc) # 0.7694

## Area under the curve: 0.7694

plot(roc,print.thres="best",col="red")  
title(main = 'ROC Curve for QDA', line = 3)

A graph with a red line

Description automatically generated

Clasification

Oluwadamilola Owolabi

2024-04-20

## RANDOM FOREST MODEL 1: SIMPLE LOGISTIC MODEL

For our non-parametric model, we plan on using Random forest on our Simple Logistic Model. It is an ensemble learning method that combines the predictions of multiple individual decision trees to improve the overall performance and robustness of the model.

#### SLM : CARET PACKAGE

We plan on using the caret package, which iterates through different mtry and ntree values, to find the optimum one

set.seed(1234) #setting the seed  
library(caret) # Loading the caret package  
  
# Running Random Forest  
# Specify the training control parameters  
train\_control <- trainControl(method = "cv", # Cross-validation method  
 number = 5,) # Number of folds  
  
# Define the random forest model  
fitted\_rf <- train(y ~ month + poutcome + emp.var.rate + contact + cons.price.idx, # formulae  
 data = train\_data, # Response variable  
 method = "rf", # Random forest method  
 trControl = train\_control) # Training control parameters  
  
#fitted\_rf  
  
#plotting the importance plot  
plot(varImp(fitted\_rf, horizontal = TRUE))

A graph with text overlay

Description automatically generated From the plot above, we can see that the 3 most impactful variables in the SLM are poutcome, emp.var.rate and cons.price.idx.

#### Making predictions and getting the metrics

# Getting the threshold  
metrics = data.frame(thresh=seq(0, 1, by = 0.0001))   
num\_thresh <- nrow(metrics)  
  
#initializing the new metrics to 0  
metrics$sensitivity <- 0  
metrics$specificity <- 0  
metrics$ppv <- 0  
metrics$npv <- 0  
metrics$accuracy <- 0  
metrics$f1 <- 0  
predicted <- predict(fitted\_rf, newdata = train\_data, type = "prob")['yes']  
#Getting the threshold  
  
#Running a for loop to find the optimum threshold  
for (i in 1:num\_thresh){  
 if(i %% 100 == 0) {  
 #print(paste(i,'/',num\_thresh,sep=''))  
 }  
   
 # Confusion Matrix  
 predicted\_classes <- ifelse(predicted[, "yes"] > metrics$thresh[i], 'yes', 'no')  
 predicted\_classes\_factor <- factor(predicted\_classes, levels = levels(train\_data$y))  
 confusion\_matrix <- confusionMatrix(predicted\_classes\_factor, train\_data$y)  
   
   
 # Metrics  
 metrics$sensitivity[i] <- as.numeric(confusion\_matrix$byClass['Sensitivity'])  
 metrics$specificity[i] <- as.numeric(confusion\_matrix$byClass['Specificity'])  
 metrics$ppv[i] <- as.numeric(confusion\_matrix$byClass['Pos Pred Value'])  
 metrics$npv[i] <- as.numeric(confusion\_matrix$byClass['Neg Pred Value'])  
 metrics$accuracy[i] <- as.numeric(confusion\_matrix$overall['Accuracy'])  
 metrics$f1[i] <- as.numeric(confusion\_matrix$byClass['F1'])  
}  
  
# Get threshold value that maximizes F1  
# Get F1 thresholds  
maxF1 <- max(metrics$f1, na.rm = TRUE) #   
maxF1 # 0.4593

## [1] 0.9468008

theshF1 <- metrics$thresh[which.max(metrics$f1)]   
theshF1 # 0.004

## [1] 0.534

# Test data  
predicted <- predict(fitted\_rf, newdata = test\_data, type = "prob")['yes']  
predicted\_classes <- ifelse(predicted[, "yes"] > theshF1, 'yes', 'no')  
CM <- confusionMatrix(as.factor(predicted\_classes), as.factor(test\_data$y))

## Warning in confusionMatrix.default(as.factor(predicted\_classes),  
## as.factor(test\_data$y)): Levels are not in the same order for reference and  
## data. Refactoring data to match.

CM

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction yes no  
## yes 215 125  
## no 696 7202  
##   
## Accuracy : 0.9003   
## 95% CI : (0.8937, 0.9067)  
## No Information Rate : 0.8894   
## P-Value [Acc > NIR] : 0.0007196   
##   
## Kappa : 0.3018   
##   
## Mcnemar's Test P-Value : < 2.2e-16   
##   
## Sensitivity : 0.23600   
## Specificity : 0.98294   
## Pos Pred Value : 0.63235   
## Neg Pred Value : 0.91188   
## Prevalence : 0.11059   
## Detection Rate : 0.02610   
## Detection Prevalence : 0.04127   
## Balanced Accuracy : 0.60947   
##   
## 'Positive' Class : yes   
##

#Printing out the metric  
Sensitivity <- CM$byClass["Sensitivity"]  
Specificity <- CM$byClass["Specificity"]  
Prevalence <- CM$byClass["Prevalence"]  
PPV <- CM$byClass["Pos Pred Value"]  
NPV <- CM$byClass["Neg Pred Value"]  
F1 <- (2 \* Sensitivity \* PPV)/(Sensitivity + PPV)  
  
#AUROC  
predicted\_classes <- factor(predicted\_classes, levels = c("yes", "no"))  
roc\_rf <- roc(response=test\_data$y,predictor= as.numeric(predicted\_classes),levels=c("no","yes"),direction = ">")  
auroc <- auc(roc\_rf)  
  
#printing merics  
cat("F1: ", F1, "\n") # 0.4380

## F1: 0.343725

cat("Sensitivity: ", Sensitivity, "\n") #0.3820

## Sensitivity: 0.2360044

cat("Specificity: ", Specificity, "\n") #0.9550

## Specificity: 0.9829398

cat("Prevalence: ", Prevalence, "\n") #0.1106

## Prevalence: 0.1105851

cat("PPV: ", PPV, "\n") #0.5133

## PPV: 0.6323529

cat("NPV: ", NPV, "\n") #0.9255

## NPV: 0.9118764

cat("AUROC: ", auroc, "\n") #0.6685

## AUROC: 0.6094721

#### GETTING THE ROC CURVE

# Print the AUROC  
auroc <- auc(roc\_rf) # 0.6685  
  
plot(roc\_rf,print.thres="best",col="red")  
title(main = 'ROC Curve for the Random Forest Model', line = 3)

A graph with a red line

Description automatically generated

The graphs looks different. It looks like the Random Forest model is confident of its predictions. Its most likely due to overfitting due to the high complexity of the Random Forest model.

#### SLM : RANDOM FOREST PACKAGE

Running the random forest again using the Random Forest Package this time. I am looking for the one to produce the best AUC value. For my hyperparameters, I chose mtry = 2 and ntree = 6000.We plan on using the caret package, which iterates through different mtry and ntree values, to find the optimum one

set.seed(1234) #setting the seed  
  
# Convert the binary outcome to a factor  
train\_data$y <- as.factor(train\_data$y)

# Define the random forest model  
fitted\_rf1.2 <-randomForest(y ~ month + poutcome + emp.var.rate + contact + cons.price.idx, data=train\_data, ntree=5000, importance = TRUE, keep.forest=TRUE, mtry=3)

#### CONTRIBUTION PLOTS FROM THE RANDOM FOREST

Looking at the contribution plots of our RF results to visualize the data to see which variables contributed the most

importance\_data <- as.data.frame(importance(fitted\_rf1.2))

plot\_data <- data.frame(

Variable = row.names(importance\_data),

no = importance\_data$no,

yes = importance\_data$yes,

accuracy = importance\_data$MeanDecreaseAccuracy, #impact of each variable on the overall accuracy of the model

Impurity = importance\_data$MeanDecreaseGini # reduction in impurity (how well a variable separates the classes) achieved by each variable.

)

plot\_data <- plot\_data[order(plot\_data$accuracy, decreasing = TRUE), ]

*# Make a contribution plot*

ggplot(plot\_data, aes(x = Variable, y = accuracy)) +

geom\_bar(stat = "identity", fill = "skyblue", width = 0.7) +

labs(title = "Accuracy Contribution Plot - Random Forest",

x = "Variable",

y = "accuracy") +

theme\_minimal() +

theme(axis.text.x = element\_text(angle = 45, hjust = 1))

ggplot(plot\_data, aes(x = Variable, y = Impurity)) +

geom\_bar(stat = "identity", fill = "skyblue", width = 0.7) +

labs(title = "Impurity Contribution Plot - Random Forest",

x = "Variable",

y = "Impurity") +

theme\_minimal() +

theme(axis.text.x = element\_text(angle = 45, hjust = 1))

A graph with blue rectangles

Description automatically generated

A graph with blue rectangles

Description automatically generated

Based on the contribution plot, there is evidence that the 3 most influential variables within the dataset are Poutcome, cons.price.idx & emp.var.rate, which is similar to the previous caret package.

#### MAKING PREDICTIOND AND GETTONG METRICS

# Get threshold

metrics = data.frame(thresh=seq(0, 1, by = 0.0001))

num\_thresh <- nrow(metrics)

metrics$sensitivity <- 0

metrics$specificity <- 0

metrics$ppv <- 0

metrics$npv <- 0

metrics$accuracy <- 0

metrics$f1 <- 0

predicted <- data.frame(predict(fitted\_rf1.2, newdata = train\_data, type = "prob"))['yes']

#Getting the threshold

for (i in 1:num\_thresh){

if(i %% 100 == 0) {

#print(paste(i,'/',num\_thresh,sep=''))

}

# Confusion Matrix

predicted\_classes <- ifelse(predicted[, "yes"] > metrics$thresh[i], 'yes', 'no')

predicted\_classes\_factor <- factor(predicted\_classes, levels = levels(train\_data$y))

confusion\_matrix <- confusionMatrix(predicted\_classes\_factor, train\_data$y)

# Metrics

metrics$sensitivity[i] <- as.numeric(confusion\_matrix$byClass['Sensitivity'])

metrics$specificity[i] <- as.numeric(confusion\_matrix$byClass['Specificity'])

metrics$ppv[i] <- as.numeric(confusion\_matrix$byClass['Pos Pred Value'])

metrics$npv[i] <- as.numeric(confusion\_matrix$byClass['Neg Pred Value'])

metrics$accuracy[i] <- as.numeric(confusion\_matrix$overall['Accuracy'])

metrics$f1[i] <- as.numeric(confusion\_matrix$byClass['F1'])

}

# Get threshold value that maximizes F1

# Get F1 thresholds

maxF1 <- max(metrics$f1, na.rm = TRUE) #

maxF1 #0.4605

[1] 0.4605124

theshF1 <- metrics$thresh[which.max(metrics$f1)]

theshF1 # 0.0034

[1] 0.0034

# Test data

predicted <- data.frame(predict(fitted\_rf1.2, newdata = test\_data, type = "prob"))['yes']

predicted\_classes <- ifelse(predicted[, "yes"] > theshF1, 'yes', 'no')

CM <- confusionMatrix(as.factor(predicted\_classes), as.factor(test\_data$y))

CM

Reference

Prediction yes no

yes 352 335

no 559 6992

Accuracy : 0.8915

95% CI : (0.8846, 0.8981)

No Information Rate : 0.8894

P-Value [Acc > NIR] : 0.2821

Kappa : 0.3818

Mcnemar's Test P-Value : 8.769e-14

Sensitivity : 0.38639

Specificity : 0.95428

Pos Pred Value : 0.51237

Neg Pred Value : 0.92597

Prevalence : 0.11059

Detection Rate : 0.04273

Detection Prevalence : 0.08339

Balanced Accuracy : 0.67033

'Positive' Class : yes

#Printing out the metric

Sensitivity <- CM$byClass["Sensitivity"]

Specificity <- CM$byClass["Specificity"]

Prevalence <- CM$byClass["Prevalence"]

PPV <- CM$byClass["Pos Pred Value"]

NPV <- CM$byClass["Neg Pred Value"]

F1 <- (2 \* Sensitivity \* PPV)/(Sensitivity + PPV)

#AUROC

predicted\_classes <- factor(predicted\_classes, levels = c("yes", "no"))

roc\_rf <- roc(response=test\_data$y,predictor= as.numeric(predicted\_classes),levels=c("no","yes"),direction = ">")

auroc <- auc(roc\_rf)

cat("F1: ", F1, "\n")

cat("Sensitivity: ", Sensitivity, "\n")

cat("Specificity: ", Specificity, "\n")

cat("Prevalence: ", Prevalence, "\n")

cat("PPV: ", PPV, "\n")

cat("NPV: ", NPV, "\n")

cat("AUROC: ", auroc, "\n")

# Results:

# F1: 0.4405507

# Sensitivity: 0.3863886

# Specificity: 0.9542787

# Prevalence: 0.1105851

# PPV: 0.5123726

# NPV: 0.9259701

# AUROC: 0.6703336

The RF model for the randomForest package is greater than the caret package by 0.5%. Hence i will go ahead with this model

#### GETTING THE ROC CURVE

# Print the AUROC

auroc <- auc(roc\_rf) # 0.7044

plot(roc\_rf,print.thres="best",col="red")

title(main = 'ROC Curve for the Random Forest Model', line = 3)

A graph with a red line

Description automatically generated

The ROC curve is is similar to the previous model. Hence i can infer that this is the roc curve of what a random forest model would predict. Most likely due to the complexity.