UF Foundation

Team 9 Final Model Report

Audrey Geoffroy
Project Sponsor
ageoffroy@ufl.edu

Alex Real, *Project Lead*
Data Scientist
alexreal@ufl.edu

Maria Camila Arias
Statistician
marias1@ufl.edu

Madison Kaplan
Data Engineer
madisonkaplan@ufl.edu

Walter Ogozaly
Data Analyst
walter.ogozaly@ufl.edu

Model Versions

MODEL 1.1 **First logistic regression for *madeDonationLastMonth***

- Included independent variables *weeksSinceLastDonation, weeksSinceFirstDonation,* and *GIFT_COUNT*.
- Very simple, yielded no relevant results.

MODEL 1.2 **Logistic regression for variable reduction**

- This was the first model to include all variables intended for the final product.
- In conjunction with a confusion matrix and our Model 1.3 random forest importance chart, this logistic regression was used to identify variables for elimination.
- Built after Model 1.3, but logically a precursor.

MODEL 1.3 **Regression, Decision Tree, and RM on *madeDonationLastMonth***

- First models to use SMOTE, k-fold cross validation, and centering/scaling of data.
- High correlation between variables diminished these models' effectiveness.
- RM model still yielded valuable results for variable reduction in Model 1.2.

MODEL 2.1 **Regression, Decision Tree, and RM on *likelyToDonateThisMonth***

- The random forest ran on only half the training data (split on the dependent variable) due to machine limitations. This was still ~180k observations.
- The newly developed dependent variable *likelyToDonateThisMonth* was the largest update to these models. These results reflect real people who should be reached out to instead of just those who donated very recently.
- All three models yielded significant results with high accuracy.

FINAL MODEL **Decision tree predicting *likelyToDonateThisMonth***

- This model had the highest accuracy of predicting *likelyToDonateThisMonth*, accompanied by the highest TP and lowest FN rates.

Decisions On Important Variables

The team wanted a measure of a donor's responsiveness to contact. To make one, we compared every contact date with every gift date and summed those that occured within 31 days after contact. This variable *donationsWithinMonth* was assigned this sum.

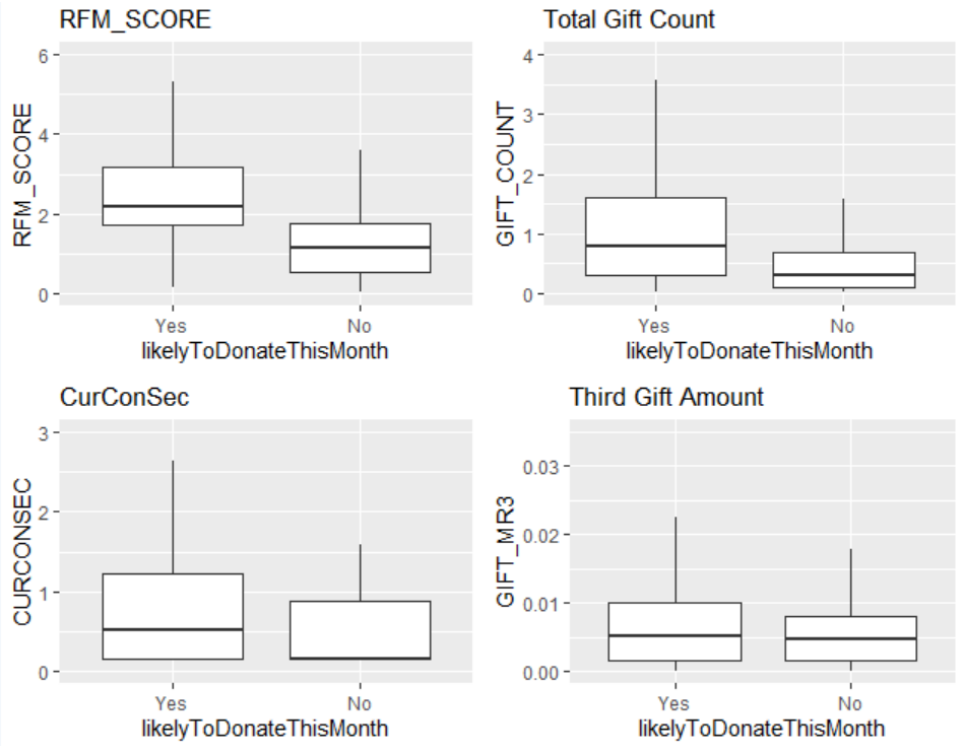
The data's gift date columns were clearly relevant but contained many NAs that were not substitutable. To get around this challenge, the team decided to designate time intervals of 1 year since 2015 and mark a simple 1 or 0 if a donor gifted in that interval. This removed all NAs while preserving important information.

We also converted all date columns to integers that represented the weeks since the event. While all Date types are stored in R as integers, we preferred to have a direct integer representation centered on today().

An iterative process was used to determine which variables to remove, whether for insignificant p-values or high correlation with other variables. For any two columns with a correlation above .7, we eliminated the one with the less significant p-value in our logistic regression, and then reran the regression. When eliminating variables that were simply insignificant, we also consulted the variable importance chart from the Model 1.3 random forest (and found no conflicts).

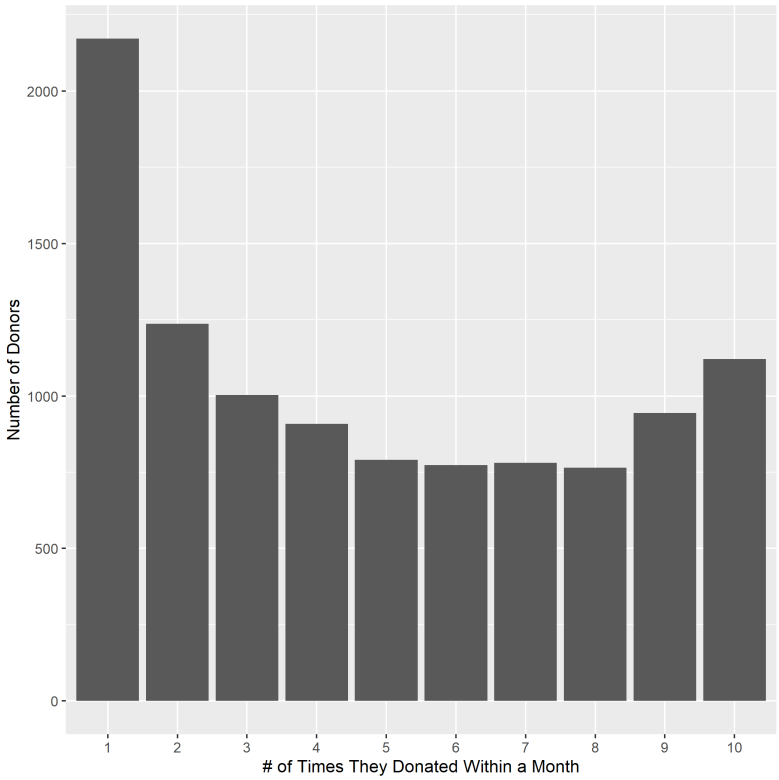
We still judge that a PCA analysis would be valuable in further reducing correlation without losing information. It would be especially useful for ranges of columns such as gift amounts 1 through 10.

Visualizing the data

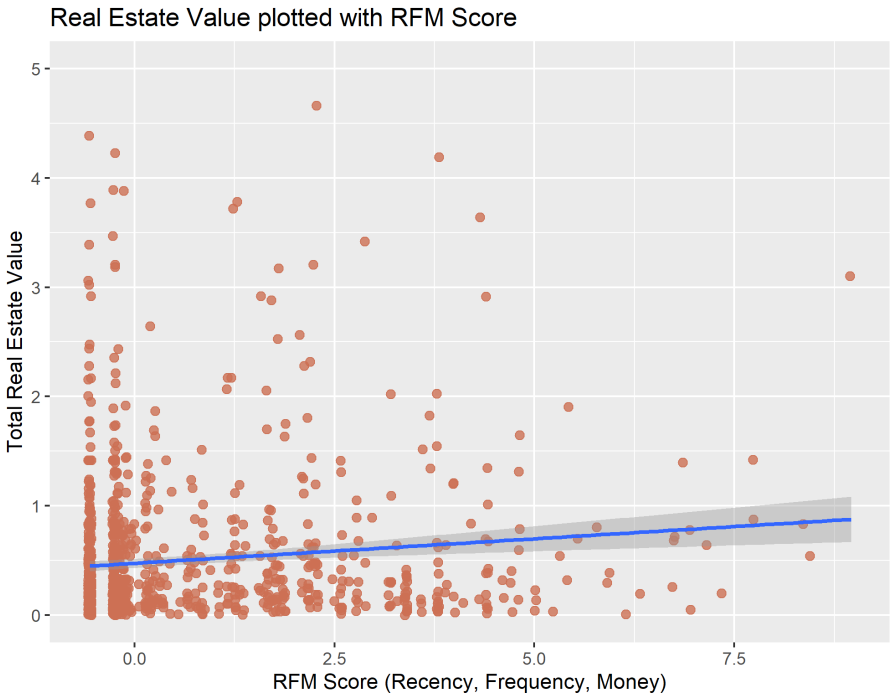


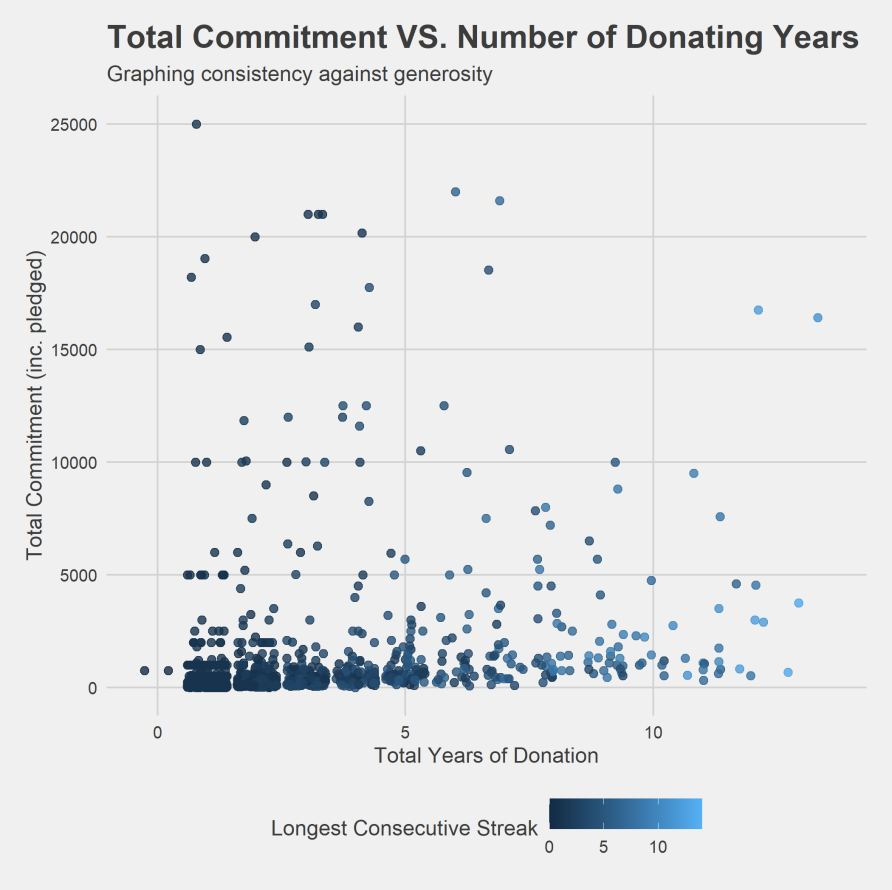
These box plots illustrate the differences in four significant variables from our model.

A histogram showing the distribution of how many times donors made gifts within a month of contact.



We can see how real estate valuations correlate positively with RFM Score, identified by the random forest as one of the most important variables.





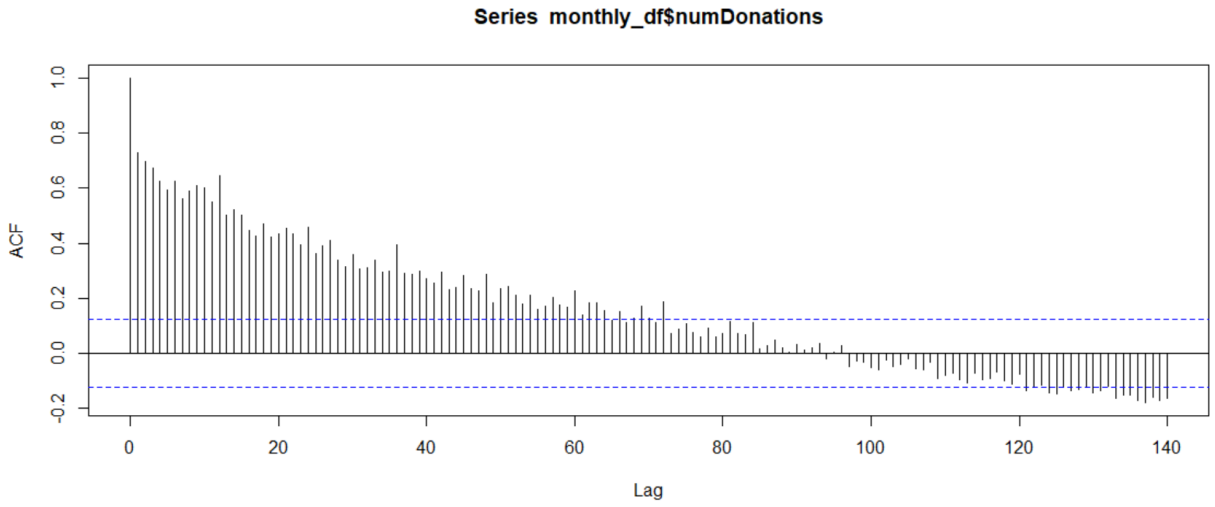
Large donors do not necessarily give many times or over many years, as this graph shows.

Developing the Dependent Variable

In order to contact donors with the highest likelihood of giving, we first created two lagged date variables. Methodology for choosing the lags is described below. If either lagged date was within 4 weeks of today, we assigned “Yes” to our dependent variable *likelyToDonateThisMonth*.

The first lag was determined by measuring autocorrelation of donation numbers by date. Lags were applied to 9 data frames (donations summed by day, by week, and by month, with data subsets starting in 2000, 2015, and 2019). A trivial weekly trend was observed on donations split by day in all three time subsets.

The most significant autocorrelation was seen by year. When data was grouped by month, it showed strong and persisting autocorrelation on the year mark.



In the autocorrelation plot for monthly donations since 2000, the 1 year autocorrelation is clearly strongest (see lags 12, 24, 36, etc) and persists for over 5 years.

Based on this lag of 1 year, we moved all donation dates forward one year and checked if any were within 4 weeks of today. If so, we assigned the respective donor “Yes” to the variable *laggedYearDonationNow*.

The second lag was determined by an individual’s average gift interval. This was defined as the period of weeks between their first and last MFOS gift divided by total gift count. We then lagged the last MFOS gift date by the average gift interval and checked if the result was within 4 weeks of today. If so, we assigned “Yes” to the variable *laggedAverageIntervalNow*.

Finally, to create our dependent variable *likelyToDonateThisMonth*, we simply marked “Yes” on any donor who had a “Yes” in either of the two lagged columns above.

Transforming data for our models

- To ensure that our model worked well on out-of-sample data, we used 10 fold cross validation repeated twice. For our last run of the models, we had to reduce the folds to 2 due to machine limitations. Since our data is so large, that should mitigate the risk of using fewer folds.

- Because our success condition was met by only 3% of the observations, our data was clearly unbalanced. We used SMOTE in our model training controls to ensure a better ratio of successes and failures.

- To protect against outsized influences from variables with large numbers (for example the gift amount columns), we centered and scaled all of our data. With this complete, our model’s criterion will only be how much variation each variable explains.

Evaluation metrics

The team aimed to develop a model with high sensitivity and a low number of false negatives. Accuracy was also a consideration.

Logistic Regression:

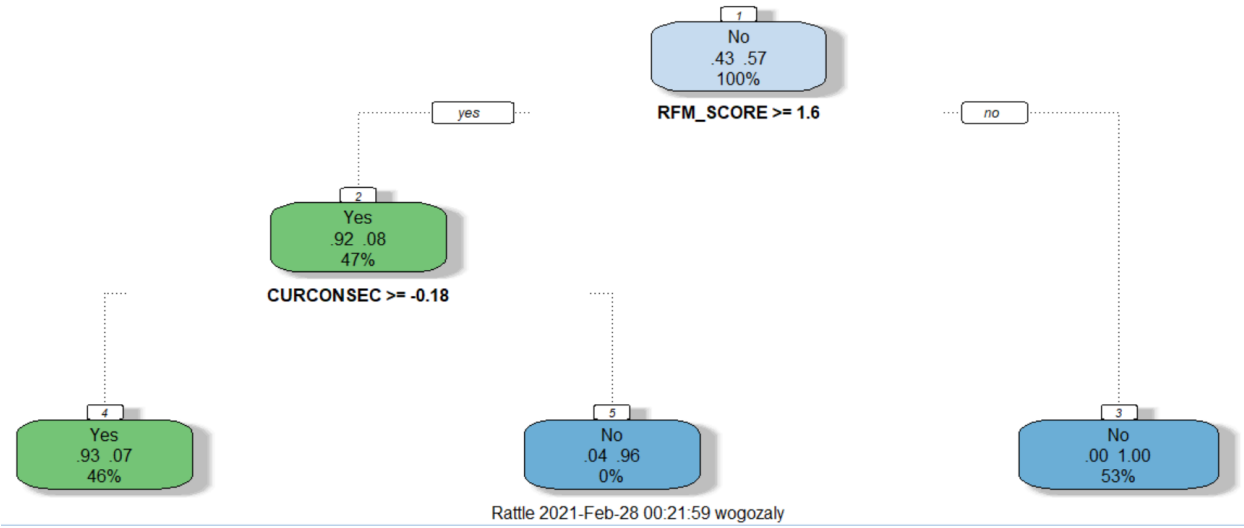
- Accuracy of **93.65%**. The model captured 4184 of the true positives. Moreover, 1006 values were classified as false negatives.

$$Sensitivity = \frac{4184}{1006+4184} x100 = 80.61\%$$

Decision Tree:

- Accuracy of **94.07%**. The model captured 5170 of the true positives. Moreover, 20 values were classified as false negatives.

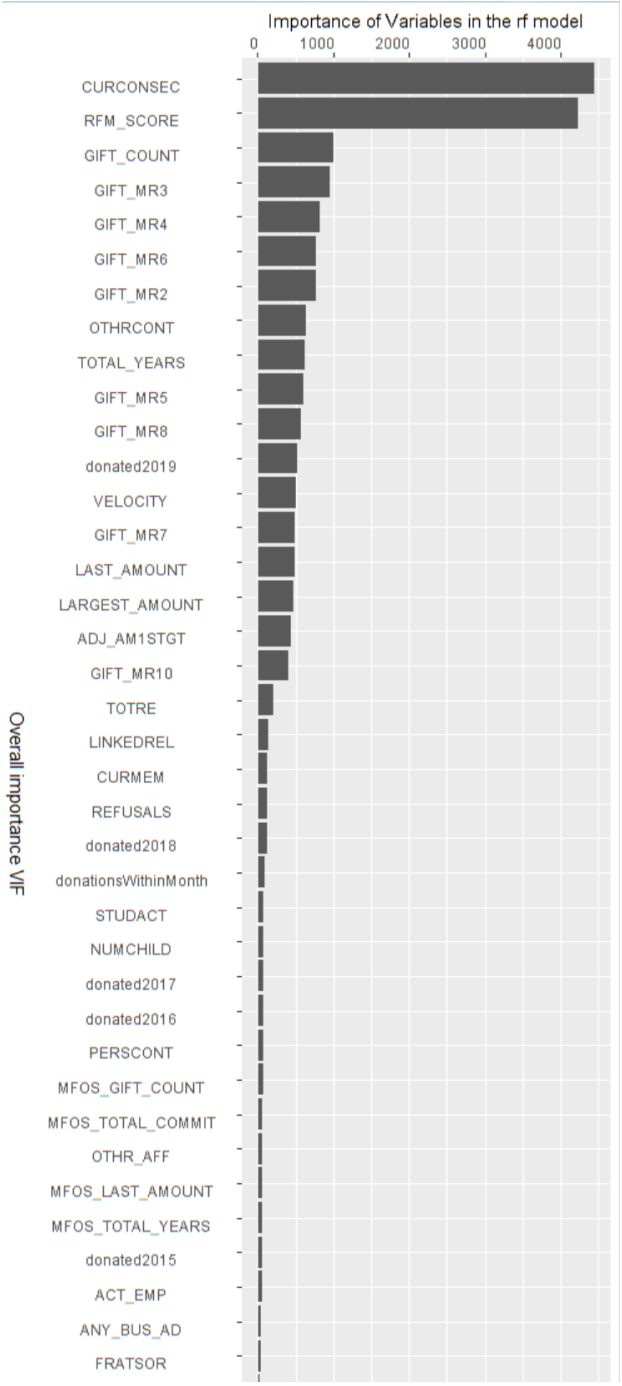
$$Sensitivity = \frac{5170}{5170+20} x100 = 99.61\%$$



Random Forest:

- Accuracy of **94.73%**. The model captured 5166 of the true positives. Moreover, 24 values were classified as false negatives.

$$Sensitivity = \frac{5166}{5166+24} x100 = 99.53\%$$




```
Confusion Matrix and Statistics

          Reference
Prediction    Yes     No
       Yes   5166   8209
       No      24 142738

               Accuracy : 0.9473
                 95% CI : (0.9462, 0.9484)
    No Information Rate : 0.9668
    P-Value [Acc > NIR] : 1

                  Kappa : 0.5342

 Mcnemar's Test P-Value : <0.0000000000000002

            Sensitivity : 0.99538
            Specificity : 0.94562
         Pos Pred Value : 0.38624
         Neg Pred Value : 0.99983
             Prevalence : 0.03324
         Detection Rate : 0.03309
   Detection Prevalence : 0.08566
      Balanced Accuracy : 0.97050

       'Positive' Class : Yes

>
```

Even though the random forest has greater accuracy than the decision tree model, the latter was able to predict more true positives and had fewer false negatives. Our sponsor is interested in having a high number of TP and a low number of FN. Therefore, among the three models, the decision tree is the one that performs best.

Final Model Selection

Our random forest and decision tree models both performed well and had nearly equivalent statistics. Seeking the most true positives and fewest false positives, however, tipped the scale in favor of the decision tree. Since they were so close, I would be eager to see if the random forest performed better under different circumstances. In theory a random forest should be superior.

Source Code

The source code can be read on Github here. It has also been pasted below.

Backup link: https://github.com/walter97/UFF_Models/blob/main/UFF%20Models.R

```
# Use 64bit RGUI.

#          ---------------------------
##################     UF Foundation     ##################
#          ---------------------------


##################     Setup     ##################

### Gets rid of scientific notation
options(scipen=999)

### Libraries
library(tidyverse)
library(data.table) # version 1.13.4
library(dplyr)
library(lubridate)
library(fastDummies)
library(gridExtra)
library(caret)
library(ISLR)
library(pROC)
library(plotROC)
library(ROCit)
library(precrec)
library(rattle)
library(DMwR) # Smote
library(ROSE) # Sampling
library(AppliedPredictiveModeling)
library(e1071)
```

```
library(cranlogs)
library(stringr)
library(forcats)
library(tidyquant)
library(forecast)


### Set Memory To Maximum Speed
memory.limit(size=34000)

### Set WD
setwd("J:/Data Services/Practicum Student Folder/Data")

##################      Loading Data      ##################

# Imports as data table. Make sure to convert to data frame.
# The fread command is being used here becuase read_excel had memory issues.
myData = fread("JOINED_EXPORT_20210215.csv", header = TRUE)

# The above returns Warning: package 'bit64' may be needed to prevent some integer columns from printing as nonsense.
# The integers print just fine when I run the head() command, so I am not taking their advice for the time being.

backup = myData
#myData = backup

# fread returns a data table, so we convert it to a data frame.
myData <- as.data.frame(myData)

# Informational commands for reference
dim(myData) ## 291 variables,  520,458 observations
#head(myData)
#str(myData, list.len = ncol(myData))
colnames(myData)


##################      Data Wrangling      ##################


##### COVERT DATE COLUMNS TO APPROPRIATE TYPE  #####

# Confirming we have real dates here.
sum(myData$TOUCH_DATE_MR1 != "") # 302,000 date observations NOT empty
sum(myData$TOUCH_DATE_MR1 == "") # 218,000 date observations empty

# Create vectors for the column numbers of
# related dates -- touches, gifts, and miscellaneous.
touchDateColumnNums = c(197, 199, 201, 203, 205, 207, 209, 211, 213, 215)
giftDateColumnNums = c(157,161,165,169,173,177,181,185,189, 193)
```

```
miscDateColumnNums = c(243, 245, 256, 263, 270) #First date +MFOS, last date +MFOS, largest
date

dateColumnNumbers = c(touchDateColumnNums, giftDateColumnNums,
 miscDateColumnNums)

# Converts all date columns to type Date.
myData[,dateColumnNumbers] <- lapply(myData[,dateColumnNumbers],
 as.Date, format = "%m/%d/%Y")

# Check if successful.
sapply(myData[,dateColumnNumbers], is.Date) # should all be TRUE

#str(myData[,dateColumnNumbers])


##### CONVERT CHAR COLUMNS TO FACTOR  #####

# List all columns that are type character.
str(myData[, sapply(myData, class) == 'character'])

# Convert to factor.
myData[, sapply(myData, class) == 'character'] <- lapply(myData[, sapply(myData, class) == 'character'],
 as.factor)

# Check if successful.
str(myData[, sapply(myData, class) == 'factor']) # should return columns from above
str(myData[, sapply(myData, class) == 'character']) # should return 0 variable data frame


#####  DROPPING UNIMPORTANT COLUMNS PRE-MODELING  #####

# Creating dummy variables at first crashed because some of our factors
# have thousands of levels but don't appear to be useful in any way.
# Getting rid of some of the unnecessary ones.
drops <- c("BAR_INDEX", "FIRST_DEPT_DESC", "FIRST_FUND",
 "FIRST_FUND_NAME", "LAST_DEPT_DESC", "LAST_FUND", "LAST_FUND_NAME",
 "LARGEST_DEPT_DESC", "LARGEST_FUND", "LARGEST_FUND_NAME", "ESTHHI",
 "COUNTY" )
myData = myData[, !(names(myData) %in% drops )]

# I'm also dropping columns with a 50-100 factors
# if I don't know what they mean or they seem unimportant.
# There is a very low chance that one of these would be important.
iDontKnowWhatTheseMean <- c("FIRST_UNIT", "LAST_UNIT", "LARGEST_UNIT",
 "PRIMARY_UNIT_OF_GIVING", "INDUSTRY", "PRIM_UNIT", "PREF_SCHOOL_CODE",
 "STATE_CODE")
myData = myData[, !(names(myData) %in% iDontKnowWhatTheseMean )]

# I'll also drop columns where the vast (95+%) of values are null.
```

```
# These columns are also not very useful.
# They were purchased data from a 3rd organization.
moreToDrop = c("OPG_ART", "OPG_EDU", "OPG_ENVIR", "OPG_ANIMAL",
  "OPG_HEALTH", "OPG_MNTL_HLTH", "OPG_VOL_HLTH_ASSOC",
  "OPG_MED_RES", "OPG_LEGAL", "OPG_EMPL", "OPG_FOOD_AG",
  "OPG_HOUSING", "OPG_PUB_SAFE", "OPG_REC_SPRTS", "OPG_YOUTH_DEV",
  "OPG_HUM_SERV", "OPG_INTERNTNL", "OPG_CIVIL_RHGTS", "OPG_COM_IMPR",
  "OPG_PHILAN", "OPG_SCI_TECH", "OPG_SOC_SCI", "OPG_PUB_BEN", "OPG_RELIGION",
  "OPG_MUT_MEMBER", "OPG_UNKN", "TG_TENPLUS_YRS", "TG_ANN", "TG_CHARTER",
  "TG_CAMPAIGN", "TG_CUMM", "TG_EVENT", "TG_ENDOW", "TG_EVENT_SPON", "TG_HON",
  "TG_INKIND", "TG_MATCH", "TG_MEM", "TG_MONTHLY", "TG_NAMED",
  "TG_PLANNEDGT", "TG_RESTRICTED", "TG_SCHOLAR", "TG_UNKNOWN",
  "TG_VEHICLE", "TG_TWO_FOUR_YRS", "TG_FIVE_NINE_YRS")
myData = myData[, !(names(myData) %in% moreToDrop )]

str(myData, list.len = ncol(myData))
colnames(myData)

# Removing more variables outlined as unimportant
# in the Variable Dictionary excel file.
unimportantPerExcel = c("PERSON_OR_ORG", "RECORD_STATUS_CODE",
  "BUSINT", "BUS_MILES", "GGAAF", "GGAGC", "GGAMG", "GGANL",
  "GGAPG", "GTR_CLUB", "HIGH_PTR", "JNTMEM", "MAIDENNAME",
  "MANAGR", "MAXYRCON", "MEDALUM", "MEDICAL", "MRKGUIDE", "NICKNAME",
  "NUMCOMM", "OTHER_AFF", "OTHRSPRT", "PENSION", "PRIMUF", "PRODUCT",
  "RCCODE", "REUNION", "RSCODE", "RUCODE", "SP_ID", "SURVEY",
  "TOTSEC", "TOTYRS", "TRAVEL", "UNITPOOL", "VOLUNT",
  "GIVING_STATUS_CODE", "SOFT_GIVING_STATUS_CODE")
myData = myData[, !(names(myData) %in% unimportantPerExcel )]

# Removing variables with zero variance.
zeroVarianceVars = c("DNB", "FAMWEAL", "HH_ID_NUMBER", "DR_PERFECT_YEARS")
myData = myData[, !(names(myData) %in% zeroVarianceVars )]

#####  CREATE DUMMY VARIABLES AND REMOVE FACTORS FROM MYDATA #####
### FOR LATER REJOINING AFTER CENTERING AND SCALING ###

# This will make dummies out of all the factor columns.
# It also returns all original columns in myData.
myDataDummied <- dummy_cols(myData, select_columns =
  as.vector(colnames(myData[, sapply(myData, class) == "factor"])),
  remove_first_dummy = TRUE )
str(myDataDummied, list.len = ncol(myDataDummied))

# I want to isolate the dummied variables for passing on later,
# so I'm removing all the non-dummy columns besides ID_NUMBER.
myDataDummied = myDataDummied[,-c(2:183)]

# In preparation for centering and scaling the data,
# I'm going to remove the 45+ columns
```

```
# that have been dummied and only center and scale the rest. This will
# hopefully help that preProces function not crash my computer.

myDataWithoutFactors = myData[, sapply(myData, class) !='factor']
myData = myDataWithoutFactors


#####  CREATE CONTACT SUCCESS VARIABLE  #####

numDaysToRespond = 31
# This code will mark a touch as successfull is the donor
# made a gift within the # of days specified above.
myData$touchOneSuccessful = myData$TOUCH_DATE_MR1 + days(numDaysToRespond) <
myData$GIFT_DATE_MR1
myData$touchTwoSuccessful = myData$TOUCH_DATE_MR2 + days(numDaysToRespond) <
myData$GIFT_DATE_MR2
myData$touchThreeSuccessful = myData$TOUCH_DATE_MR3 + days(numDaysToRespond) <
myData$GIFT_DATE_MR3
myData$touchFourSuccessful = myData$TOUCH_DATE_MR4 + days(numDaysToRespond) <
myData$GIFT_DATE_MR4
myData$touchFiveSuccessful = myData$TOUCH_DATE_MR5 + days(numDaysToRespond) <
myData$GIFT_DATE_MR5
myData$touchSixSuccessful = myData$TOUCH_DATE_MR6 + days(numDaysToRespond) <
myData$GIFT_DATE_MR6
myData$touchSevenSuccessful = myData$TOUCH_DATE_MR7 + days(numDaysToRespond) <
myData$GIFT_DATE_MR7
myData$touchEightSuccessful = myData$TOUCH_DATE_MR8 + days(numDaysToRespond) <
myData$GIFT_DATE_MR8
myData$touchNineSuccessful = myData$TOUCH_DATE_MR9 + days(numDaysToRespond) <
myData$GIFT_DATE_MR9
myData$touchTenSuccessful = myData$TOUCH_DATE_MR10 + days(numDaysToRespond) <
myData$GIFT_DATE_MR10

# Sums the number of above columns that contain TRUE.
colnames(myData)
myData$donationsWithinMonth = rowSums( myData[,137:146])

# Now that we've summed the successful contacts, we can remove
# the columns touchOneSuccessful through touchTenSuccessful
myData = myData[,-c(137:146)]

# For visualization of who responds to contact.
regularDonors = as.data.frame(table(myData$donationsWithinMonth))
regularDonors = regularDonors[-1,]

# Bar plots number of donors by how many times they've donated
# within a month.
ggplot(regularDonors, aes(x=as.factor(Var1), y=Freq)) +
  geom_bar(stat='identity') +
  labs(y="Number of Donors", x="# of Times They Donated Within a Month")
```

```
ggsave("donorsByNumberOfDonationsWithinMonth.png") # Saves above graph.


#####  CREATE INTERVAL VARIABLES  #####

# I'd like to convert all the remaining date columns
# from the type DATE to an int reflecting the weeks since
# the event occured.
str(myData[,sapply(myData, class) == "Date"])

# Create a list of all the date columns I want to select as well as ID_NUMBER.
dateColumnNames = names(myData[,sapply(myData, class) == "Date"])
dateColumnNames = append(dateColumnNames, "ID_NUMBER")

dateColumns = myData[,dateColumnNames]

convertToWeeksSince <- function(x) {
  return(floor(as.double(difftime(today(), x, unit = "weeks"))))
}

# This will run every date through the function above
# and return the rounded # of weeks from today to the event.
transformedDates = data.frame(dateColumns[26], lapply(dateColumns[1:25], convertToWeeksSince))

# Create new column names for these transformed date columns.
newColumnNames = c("ID_NUMBER", "weeksSinceGiftDate1", "weeksSinceGiftDate2",
  "weeksSinceGiftDate3", "weeksSinceGiftDate4", "weeksSinceGiftDate5",
  "weeksSinceGiftDate6", "weeksSinceGiftDate7", "weeksSinceGiftDate8",
  "weeksSinceGiftDate9", "weeksSinceGiftDate10", "weeksSinceTouch1", "weeksSinceTouch2",
  "weeksSinceTouch3", "weeksSinceTouch4", "weeksSinceTouch5", "weeksSinceTouch6",
  "weeksSinceTouch7", "weeksSinceTouch8", "weeksSinceTouch9", "weeksSinceTouch10",
  "weeksSinceFirstMFOSGift", "weeksSinceLastMFOSGift", "weeksSinceFirstGift",
  "weeksSinceLastGift", "weeksSinceLargestGift")
colnames(transformedDates) = newColumnNames

# Remove the old date columns from myData in anticipation of a merge.
myData = myData[, sapply(myData, class) != "Date"]

# Left join transformed dates with myData
myData <- merge(x=myData,y=transformedDates,by="ID_NUMBER",all.x=FALSE)

colnames(myData)
str(myData, list.len = ncol(myData))

# All columns are now int or num. Success!

##### REMOVING NA VALUES WITH TIME INTERVAL YEAR BUCKETS #####

colnames(dateColumns)
```

```
# This is poor code, but it marks a 1 if a donor
# donated in a particular year and 0 otherwise.
# It does this for 2021 through 2015.
dateColumns$donated2021 <- ifelse(year(dateColumns$GIFT_DATE_MR1) == 2021, 1,
 ifelse(year(dateColumns$GIFT_DATE_MR2) == 2021, 1,
 ifelse(year(dateColumns$GIFT_DATE_MR3) == 2021, 1,
 ifelse(year(dateColumns$GIFT_DATE_MR4) == 2021, 1,
 ifelse(year(dateColumns$GIFT_DATE_MR5) == 2021, 1,
 ifelse(year(dateColumns$GIFT_DATE_MR6) == 2021, 1,
 ifelse(year(dateColumns$GIFT_DATE_MR7) == 2021, 1,
 ifelse(year(dateColumns$GIFT_DATE_MR8) == 2021, 1,
 ifelse(year(dateColumns$GIFT_DATE_MR9) == 2021, 1,
 ifelse(year(dateColumns$GIFT_DATE_MR10) == 2021, 1, 0)))))))))

table(dateColumns$donated2021)

dateColumns$donated2020 <- ifelse(year(dateColumns$GIFT_DATE_MR1) == 2020, 1,
 ifelse(year(dateColumns$GIFT_DATE_MR2) == 2020, 1,
 ifelse(year(dateColumns$GIFT_DATE_MR3) == 2020, 1,
 ifelse(year(dateColumns$GIFT_DATE_MR4) == 2020, 1,
 ifelse(year(dateColumns$GIFT_DATE_MR5) == 2020, 1,
 ifelse(year(dateColumns$GIFT_DATE_MR6) == 2020, 1,
 ifelse(year(dateColumns$GIFT_DATE_MR7) == 2020, 1,
 ifelse(year(dateColumns$GIFT_DATE_MR8) == 2020, 1,
 ifelse(year(dateColumns$GIFT_DATE_MR9) == 2020, 1,
 ifelse(year(dateColumns$GIFT_DATE_MR10) == 2020, 1, 0)))))))))

table(dateColumns$donated2020)

dateColumns$donated2019 <- ifelse(year(dateColumns$GIFT_DATE_MR1) == 2019, 1,
 ifelse(year(dateColumns$GIFT_DATE_MR2) == 2019, 1,
 ifelse(year(dateColumns$GIFT_DATE_MR3) == 2019, 1,
 ifelse(year(dateColumns$GIFT_DATE_MR4) == 2019, 1,
 ifelse(year(dateColumns$GIFT_DATE_MR5) == 2019, 1,
 ifelse(year(dateColumns$GIFT_DATE_MR6) == 2019, 1,
 ifelse(year(dateColumns$GIFT_DATE_MR7) == 2019, 1,
 ifelse(year(dateColumns$GIFT_DATE_MR8) == 2019, 1,
 ifelse(year(dateColumns$GIFT_DATE_MR9) == 2019, 1,
 ifelse(year(dateColumns$GIFT_DATE_MR10) == 2019, 1, 0)))))))))

table(dateColumns$donated2019)

dateColumns$donated2018 <- ifelse(year(dateColumns$GIFT_DATE_MR1) == 2018, 1,
 ifelse(year(dateColumns$GIFT_DATE_MR2) == 2018, 1,
 ifelse(year(dateColumns$GIFT_DATE_MR3) == 2018, 1,
 ifelse(year(dateColumns$GIFT_DATE_MR4) == 2018, 1,
 ifelse(year(dateColumns$GIFT_DATE_MR5) == 2018, 1,
 ifelse(year(dateColumns$GIFT_DATE_MR6) == 2018, 1,
 ifelse(year(dateColumns$GIFT_DATE_MR7) == 2018, 1,
```

```
 ifelse(year(dateColumns$GIFT_DATE_MR8) == 2018, 1,
 ifelse(year(dateColumns$GIFT_DATE_MR9) == 2018, 1,
 ifelse(year(dateColumns$GIFT_DATE_MR10) == 2018, 1, 0)))))))))

table(dateColumns$donated2018)

dateColumns$donated2017 <- ifelse(year(dateColumns$GIFT_DATE_MR1) == 2017, 1,
 ifelse(year(dateColumns$GIFT_DATE_MR2) == 2017, 1,
 ifelse(year(dateColumns$GIFT_DATE_MR3) == 2017, 1,
 ifelse(year(dateColumns$GIFT_DATE_MR4) == 2017, 1,
 ifelse(year(dateColumns$GIFT_DATE_MR5) == 2017, 1,
 ifelse(year(dateColumns$GIFT_DATE_MR6) == 2017, 1,
 ifelse(year(dateColumns$GIFT_DATE_MR7) == 2017, 1,
 ifelse(year(dateColumns$GIFT_DATE_MR8) == 2017, 1,
 ifelse(year(dateColumns$GIFT_DATE_MR9) == 2017, 1,
 ifelse(year(dateColumns$GIFT_DATE_MR10) == 2017, 1, 0)))))))))

table(dateColumns$donated2017)

dateColumns$donated2016 <- ifelse(year(dateColumns$GIFT_DATE_MR1) == 2016, 1,
 ifelse(year(dateColumns$GIFT_DATE_MR2) == 2016, 1,
 ifelse(year(dateColumns$GIFT_DATE_MR3) == 2016, 1,
 ifelse(year(dateColumns$GIFT_DATE_MR4) == 2016, 1,
 ifelse(year(dateColumns$GIFT_DATE_MR5) == 2016, 1,
 ifelse(year(dateColumns$GIFT_DATE_MR6) == 2016, 1,
 ifelse(year(dateColumns$GIFT_DATE_MR7) == 2016, 1,
 ifelse(year(dateColumns$GIFT_DATE_MR8) == 2016, 1,
 ifelse(year(dateColumns$GIFT_DATE_MR9) == 2016, 1,
 ifelse(year(dateColumns$GIFT_DATE_MR10) == 2016, 1, 0)))))))))

table(dateColumns$donated2016)

dateColumns$donated2015 <- ifelse(year(dateColumns$GIFT_DATE_MR1) == 2015, 1,
 ifelse(year(dateColumns$GIFT_DATE_MR2) == 2015, 1,
 ifelse(year(dateColumns$GIFT_DATE_MR3) == 2015, 1,
 ifelse(year(dateColumns$GIFT_DATE_MR4) == 2015, 1,
 ifelse(year(dateColumns$GIFT_DATE_MR5) == 2015, 1,
 ifelse(year(dateColumns$GIFT_DATE_MR6) == 2015, 1,
 ifelse(year(dateColumns$GIFT_DATE_MR7) == 2015, 1,
 ifelse(year(dateColumns$GIFT_DATE_MR8) == 2015, 1,
 ifelse(year(dateColumns$GIFT_DATE_MR9) == 2015, 1,
 ifelse(year(dateColumns$GIFT_DATE_MR10) == 2015, 1, 0)))))))))

table(dateColumns$donated2015)


# Merging these new columns back with myData
oldDateColumns = dateColumns
dateColumns = dateColumns[,c(26:33)]
myData <- merge(x=myData,y=dateColumns,by="ID_NUMBER",all.x=TRUE)
```

```
colnames(myData)

#####  AUTOCORRELATION TO DETERMINE APPROPRIATE LAG  #####

set.seed(365)

# This creates a tibble for each date that had a donation, and the number of donations on that date.
dateTibble = oldDateColumns %>%
 gather("key", "value", GIFT_DATE_MR1, GIFT_DATE_MR2, GIFT_DATE_MR3, GIFT_DATE_MR4,
 GIFT_DATE_MR5, GIFT_DATE_MR6, GIFT_DATE_MR7, GIFT_DATE_MR8, GIFT_DATE_MR9,
GIFT_DATE_MR10) %>%
 group_by(value) %>%
 summarize(n=n())

# Convert to data frame and changes column names.
datesCounts_df = data.frame(dateTibble)
colnames(datesCounts_df) = c("donationDate", "numDonations")

# Creating a data frame of all days (we want to include days with no donation).
allDatesSince2000 = data.frame(seq(as.Date("2000/1/1"), today(), "days"))
colnames(allDatesSince2000) = c("donationDate")

# Left joining donation date data frame to the one with all dates, that way we can lag by day.
donationsPerDaySince2000 <- merge(x=allDatesSince2000,y=datesCounts_df,by="donationDate",all.x=TRUE)
dim(donationsPerDaySince2000)

# Setting number of donations to 0 for days with NAs currently.
donationsPerDaySince2000$numDonations = replace_na(donationsPerDaySince2000$numDonations, 0)

head(donationsPerDaySince2000)

#### AUTOCORRELATIONS for data FROM 2000-onwards

## BY DAY
# You actually don't see much yearly seasonality as I would have expected.
yearAutoCorByDay.2000 <- acf(donationsPerDaySince2000$numDonations, lag.max = 370)

# There is a trivial weekly trend.
monthAutoCorByDay.2000 <- acf(donationsPerDaySince2000$numDonations, lag.max = 31)

## BY WEEK

# Sum the daily counts to create numdonations by week.
xtsData <- as.xts(donationsPerDaySince2000$numDonations, order.by =
as.Date(donationsPerDaySince2000$donationDate))
weekly <- apply.weekly(xtsData, sum)
weekly_df <- data.frame(weekly)

# Weeks currently stored in rownames. We'll change that now.
```

```
weekly_df <- cbind(rownames(weekly_df), data.frame(weekly_df, row.names=NULL))
colnames(weekly_df) <- c("week", "numDonations")

# Once again we don't see any seasonality. Next I will try using data from only 2015 onwards.
yearAutoCorWeekly.2000 <- acf(weekly_df$numDonations, lag.max = 55)

## BY MONTH

# Sum the daily counts to create numdonations by month.
xtsDataM <- as.xts(donationsPerDaySince2000$numDonations, order.by =
as.Date(donationsPerDaySince2000$donationDate))
monthly <- apply.monthly(xtsDataM, sum)
monthly_df <- data.frame(monthly)

# Months currently stored in rownames. We'll change that now.
monthly_df <- cbind(rownames(monthly_df), data.frame(monthly_df, row.names=NULL))
colnames(monthly_df) <- c("month", "numDonations")

# You can see yearly correlation persists for many lags at the monthly level. ## THIS IS THE BEST
## GRAPH MAKING THE CASE FOR A YEAR LONG LAG. It's most predictive for longest, and on the largest data
set.
yearAutoCorMonthly.2000 <- acf(monthly_df$numDonations, lag.max = 140)

#### AUTOCORRELATION FROM 2015 ONWARD

# Filtering day, week, and month data frames to include only 2015-onward.
# More recent dates are GREATER than older dates.
donationsPerDaySince2015 = donationsPerDaySince2000[ donationsPerDaySince2000$donationDate >
as.Date("2014-12-31"),]
weeklySince2015_df = weekly_df[ as.Date(weekly_df$week) > as.Date("2014-12-31"),]
monthlySince2015_df = monthly_df[ as.Date(monthly_df$month) > as.Date("2014-12-31"),]

## BY DAY

# Still only a mild weekly trend in data.
yearAutoCorDaily.2015 <- acf(donationsPerDaySince2015$numDonations, lag.max = 370)
monthAutoCorDaily.2015 <- acf(donationsPerDaySince2015$numDonations, lag.max = 31)

## BY WEEK

# You do see pretty high autocorrelation at the year mark in the post-2015 data. Let's try post-2019.
yearAutoCorWeekly.2015 <- acf(weeklySince2015_df$numDonations, lag.max = 55)

## BY MONTH

# Here the year autocorrelation is weaker and decays fast.
yearAutoCorMonthly.2015 <- acf(monthlySince2015_df$numDonations, lag.max = 55)


#### AUTOCORRELATION FROM 2019 ONWARD
```

```
weeklySince2019_df = weekly_df[ as.Date(weekly_df$week) > as.Date("2018-12-31"),]
monthlySince2019_df = monthly_df[ as.Date(monthly_df$month) > as.Date("2018-12-31"),]

## BY WEEKS

# By weeks, the yearly autocorrelation is clearly the highest, suggesting we should use that.
# Because this autocorrelation decays into "noise" and hits negative values within 2 lags, that suggests
# this is a stationary variable.
yearAutoCorWeekly.2019 <- acf(weeklySince2019_df$numDonations, lag.max = 55)

## BY MONTHS

# You don't see a yearly autocorrelation in months since 2019, but you see it in most other graphs, so
# I don't think it changes my final conclusion.
yearAutoCorMonthly.2019 <- acf(monthlySince2019_df$numDonations, lag.max = 24)


######## MUST ALSO REMOVE NAS FROM ALL SUCCESS VARIABLES (SET NA TO FALSE)

# Create three variables representing weeks since lastMFOSgift.
myData$madeDonationLastMonth = ifelse(
 myData$weeksSinceLastMFOSGift < 5,
 1,0)
myData$madeDonationLastMonth = replace_na(myData$madeDonationLastMonth, 0)

# Let's also make two more for within the past six months and year.
myData$madeDonationLastSixMonths = ifelse(
 myData$weeksSinceLastMFOSGift < 27,
 1,0)
myData$madeDonationLastSixMonths = replace_na(myData$madeDonationLastSixMonths, 0)

myData$madeDonationLastYear = ifelse(
 myData$weeksSinceLastMFOSGift < 53,
 1,0)
myData$madeDonationLastYear = replace_na(myData$madeDonationLastYear , 0)

##### REPLACING NAs

# A lot of columns have NAs that can be accurately represented
# by a real value. I replace those NAs here.
myData$madeDonationLastYear = replace_na(myData$madeDonationLastYear, FALSE)
myData$donated2021 = replace_na(myData$donated2021, 0)
myData$donated2020 = replace_na(myData$donated2020, 0)
myData$donated2019 = replace_na(myData$donated2019, 0)
myData$donated2018 = replace_na(myData$donated2018, 0)
myData$donated2017 = replace_na(myData$donated2017, 0)
myData$donated2016 = replace_na(myData$donated2016, 0)
myData$donated2015 = replace_na(myData$donated2015, 0)
```

```
myData$donationsWithinMonth = replace_na(myData$donationsWithinMonth, 0)
myData$TOTAL_YEARS = replace_na(myData$TOTAL_YEARS, 0)
myData$GIFT_COUNT = replace_na(myData$GIFT_COUNT, 0)

myData$FIRST_AMOUNT = replace_na(myData$FIRST_AMOUNT, 0)
myData$LAST_AMOUNT = replace_na(myData$LAST_AMOUNT, 0)
myData$LONGEST_CONSECUTIVE_STREAK = replace_na(myData$LONGEST_CONSECUTIVE_STREAK, 0)
myData$MFOS_FIRST_AMOUNT = replace_na(myData$MFOS_FIRST_AMOUNT, 0)
myData$MFOS_GIFT_COUNT = replace_na(myData$MFOS_GIFT_COUNT, 0)
myData$MFOS_TOTAL_COMMIT = replace_na(myData$MFOS_TOTAL_COMMIT, 0)
myData$MFOS_LAST_AMOUNT = replace_na(myData$MFOS_LAST_AMOUNT, 0)
myData$GIFT_MR1 = replace_na(myData$GIFT_MR1, 0)
myData$GIFT_MR2 = replace_na(myData$GIFT_MR2, 0)
myData$GIFT_MR3 = replace_na(myData$GIFT_MR3, 0)
myData$GIFT_MR4 = replace_na(myData$GIFT_MR4, 0)
myData$GIFT_MR5 = replace_na(myData$GIFT_MR5, 0)
myData$GIFT_MR6 = replace_na(myData$GIFT_MR6, 0)
myData$GIFT_MR7 = replace_na(myData$GIFT_MR7, 0)
myData$GIFT_MR8 = replace_na(myData$GIFT_MR8, 0)
myData$GIFT_MR9 = replace_na(myData$GIFT_MR9, 0)
myData$GIFT_MR10 = replace_na(myData$GIFT_MR10, 0)
myData$LARGEST_AMOUNT = replace_na(myData$LARGEST_AMOUNT, 0)
myData$MFOS_TOTAL_YEARS = replace_na(myData$MFOS_TOTAL_YEARS, 0)
myData$TOTALUF = replace_na(myData$TOTALUF, 0)
myData$REFUSALS = replace_na(myData$REFUSALS, 0)
myData$TOTRE = replace_na(myData$TOTRE, 0)
myData$CURCONSEC = replace_na(myData$CURCONSEC, 0)
myData$ADJ_AM1STGT = replace_na(myData$ADJ_AM1STGT, 0)

# This code was used to find columns for discarding that were mostly empty.
#na_count = sapply(myData, function(y) sum(length(which(is.na(y)))))
#na_count_backup = na_count
#na_count = data.frame(na_count)
#na_count %>% filter ( na_count == 0)

#ggplot(myData, aes(x=GIFT_MR1)) + geom_histogram() + xlim(0,1000)

# The dummied variables have no NAs as confirmed below.
na_count = sapply(myDataDummied, function(y) sum(length(which(is.na(y)))))
na_count_backup = na_count
na_count = data.frame(na_count)
na_count %>% filter ( na_count == 0)

#####  CREATING AVG DONATION FREQUENCY  #####

# Calculate how frequently they donate by dividing
# the time between the first and last gift by the number of gifts.
myData$donationFrequencyInWeeks = (myData$weeksSinceFirstMFOSGift -
  myData$weeksSinceLastMFOSGift) / myData$GIFT_COUNT
```

```
# View frequency by year
table(floor(myData$donationFrequencyInWeeks / 4))


#####  CREATING LAGGED DEPENDENT VARIABLE BUILDING BLOCK  #####

# Marks YES if their last MFOS gift date + average weekly interval is within 4 weeks of today.
myData$laggedAverageIntervalNow = ifelse( myData$weeksSinceLastMFOSGift -
myData$donationFrequencyInWeeks < 5, "Yes", "No")
myData$laggedAverageIntervalNow = replace_na(myData$laggedAverageIntervalNow, "No")

table(myData$laggedAverageIntervalNow) ## 351 yes as of Feb 27


# Returns a data frame with only those who've donated within a month of this time last year.
# Mutates a new variable laggedYearDonationNow that is YES for all these filtered rows.
donatedAYearAgo = oldDateColumns %>%
  select(ID_NUMBER, GIFT_DATE_MR1, GIFT_DATE_MR2, GIFT_DATE_MR3, GIFT_DATE_MR4,
  GIFT_DATE_MR5, GIFT_DATE_MR6, GIFT_DATE_MR7, GIFT_DATE_MR8, GIFT_DATE_MR9,
GIFT_DATE_MR10) %>%
  filter ( between( GIFT_DATE_MR1 + days(365), today() - days(28), today() + days(28)) |
    between( GIFT_DATE_MR2 + days(365), today() - days(28), today() + days(28)) |
    between( GIFT_DATE_MR3 + days(365), today() - days(28), today() + days(28)) |
    between( GIFT_DATE_MR4 + days(365), today() - days(28), today() + days(28)) |
    between( GIFT_DATE_MR5 + days(365), today() - days(28), today() + days(28)) |
    between( GIFT_DATE_MR6 + days(365), today() - days(28), today() + days(28)) |
    between( GIFT_DATE_MR7 + days(365), today() - days(28), today() + days(28)) |
    between( GIFT_DATE_MR8 + days(365), today() - days(28), today() + days(28)) |
    between( GIFT_DATE_MR9 + days(365), today() - days(28), today() + days(28)) |
    between( GIFT_DATE_MR10 + days(365), today() - days(28), today() + days(28)) ) %>%
    mutate(laggedYearDonationNow = "Yes")

# Left join the people who donated a year ago with all people.
myData <- merge(x=myData, y=donatedAYearAgo[,c(1,12)], by="ID_NUMBER",all.x=TRUE)
myData$laggedYearDonationNow = replace_na(myData$laggedYearDonationNow, "No")
table(myData$laggedYearDonationNow) # 17k yes at the moment.


#####  CREATING THE COMBINED DEPENDENT VARIABLE  #####

# If either the yearly autocorrelation or an individual's average interval suggests
# a donation this month, we'll mark them YES as likelyToDonateThisMonth.
myData$likelyToDonateThisMonth = ifelse( myData$laggedYearDonationNow == "Yes" |
  myData$laggedAverageIntervalNow == "Yes", "Yes", "No")

table(myData$likelyToDonateThisMonth) # 17,300 YES at this point

myData$likelyToDonateThisMonth = factor(myData$likelyToDonateThisMonth, levels=c("Yes", "No"))

## Removing the predecessors.
```

```
## I don't want to feed the model the building blocks of this dependent variable.
myData = myData %>% select(-laggedYearDonationNow, -laggedAverageIntervalNow)

### The idea is that false positives would be people you should still contact,
### because UFF wants a wide net and because many people have a "NO" in
### the dependent column because of some lack of information that caused
### them to fail the above ifelse statements. If the model says that
### based on the independent variables those people DO have that they're
### likely to donate, we could see that as exposing the aforementioned
### incomplete-data people. Hope this makes sense.


#####  BOXPLOTS  #####

#plot01 <- ggplot(data = trainingData %>% filter(!is.na(madeDonationLastMonth))) +
#  geom_boxplot(aes(x = madeDonationLastMonth, y = GIFT_COUNT), outlier.shape=NA) +
#  labs(title = "Number of Donations", subtitle = "Made Donation Last Month vs. No Donations") +
#  ylim(0,10)
#plot02 <- ggplot(data = trainingData %>% filter(!is.na(madeDonationLastMonth))) +
#  geom_boxplot(aes(x = madeDonationLastMonth, y = TOTALUF), outlier.shape=NA) +
#  labs(title = "Total Donated ($)", subtitle = "Made Donation Last Month vs. No Donations") +
#  ylim(0,2000)
#plot03 <- ggplot(data = trainingData %>% filter(!is.na(madeDonationLastMonth))) +
#  geom_boxplot(aes(x = madeDonationLastMonth, y = weeksSinceFirstDonation), outlier.shape=NA) +
#  labs(title = "Weeks since First Donation", subtitle = "Made Donation Last Month vs. No Donations") +
#  ylim(0,1000)
#plot04 <- ggplot(data = trainingData %>% filter(!is.na(madeDonationLastMonth))) +
#  geom_boxplot(aes(x = madeDonationLastMonth, y = weeksSinceLastDonation), outlier.shape=NA) +
#  labs(title = "Weeks since Last Donation", subtitle = "Made Donation Last Month vs. No Donations") +
#  ylim(0,1000)
#grid.arrange(plot01, plot02, plot03, plot04, ncol = 2)


########   MODEL TRAINING AND FINAL SUBSET    ########

# Training control setup
trn_ctrl <- trainControl(summaryFunction = twoClassSummary,
  savePredictions = TRUE,
  sampling = "smote",
  method = "repeatedcv",
  number = 2, # Reduced from 10 to 2 in order to make models run.
  repeats = 2,
  classProbs = TRUE,
  allowParallel = FALSE)


# We're not going to feed the model any columns with NAs.
# Here we identify those.
na_count = sapply(myData, function(y) sum(length(which(is.na(y)))))
na_count_backup = na_count
```

```
na_count = data.frame(na_count)
na_count %>% filter ( na_count == 0)

# Subsetting for columns with 0 NAs.
dataForModel = myData[, colSums(is.na(myData)) == 0]


#####  PREPROCESSING  ######

# This line of code crashed my shit before.
preProcValues <- preProcess(dataForModel, method = c("center", "scale"), na.remove=TRUE)
myDataTransformed <- predict(preProcValues, dataForModel)

#myDataBackup = myData

dataForModelScaled = myDataTransformed


########   REMOVING CORRELATED & UNIMPORTANT (PER GLM+RF) VARS    ########

dataForCorrelation = dataForModelScaled

# If two variables have too much correlation, I compare their p-value
# and remove the less important one.
removedForCor = c("GIFT_MR1", "LONGEST_CONSECUTIVE_STREAK",
 "GIFT_MR9", "ANY_EMP", "FIRST_AMOUNT", "TOTALUF", "MFOS_FIRST_AMOUNT",
 "donated2020")

# If the p-value is miserable or the random forest model said it was
# unimportant, it's removed below.
removedForUnimportance = c("VARSPORT", "CHILDALUM", "CREDCARD",
 "GATRTAG", "ETHNALUM", "EVENT", "EMAIL", "UF_EMP", "UFF_BRD",
 "SEC_HOME", "FFF", "LIFEMEM", "madeDonationLastMonth",
 "madeDonationLastYear")

removed = c(removedForCor, removedForUnimportance)

dataForCorrelation = dataForCorrelation[, !(names(dataForCorrelation) %in% removed )]

## Calculate correlations among variables and print the correlations above .7.
## All correlations above .7 were removed.
res <- cor(dataForCorrelation)
res_df <- as.data.frame(as.table(res))
#colnames(res_df)
res_df[ res_df$Freq > .7 & res_df$Freq < 1.00,]

## This code ran a simple model that I used to compare two correlated vars p-values
# and choose one for removal. I also consulted our first iteration random forest model's
# variable importance graph.
dataForModelOne = dataForModelScaled[, !(names(dataForModelScaled) %in% removed)]
```

```
modelOne <- glm(likelyToDonateThisMonth ~.-ID_NUMBER, family = binomial(link="logit"), data =
dataForModelOne ,
  maxit = 100)
summary(modelOne)
with(summary(modelOne), 1 - deviance / null.deviance) # prints R, currently .45

# Finally we'll remove the identified columns from our data for the models.
dataForModelScaled = dataForModelScaled[, !(names(dataForModelScaled) %in% removed)]

#####  SPLITTING TEST AND TRAIN DATA  #####

set.seed(365)
lastMonth_idx <- createDataPartition(dataForModelScaled$likelyToDonateThisMonth,
 p = 0.7, list = FALSE)
lastMonth_trn <- dataForModelScaled[lastMonth_idx,]
lastMonth_tst <- dataForModelScaled[-lastMonth_idx,]

table(lastMonth_trn$likelyToDonateThisMonth)
table(lastMonth_tst$likelyToDonateThisMonth)
# Proportions look good. Let's continue.


########   LOGISTIC REGRESSION MODEL    ########

# GLM model on the likelyToDonateThisMonth training data.
## Took 15 minutes to run.
logreg_model <- train(likelyToDonateThisMonth~.-ID_NUMBER, data=lastMonth_trn,
  method = "glm",
  family = "binomial",
  metric = "ROC",
  trControl = trn_ctrl )

logreg_model
summary(logreg_model)
# All the p values are significant.
#View(logreg_model$pred)

lvs <- c("Yes", "No")
truth <- factor(logreg_model$pred$obs)
pred <- factor(logreg_model$pred$pred)
xtab <- table(pred, truth)
confusionMatrix(xtab)

# Applying the trained model to the test data lastMonth_tst
pred_logreg <- predict.train(object=logreg_model,
  newdata = lastMonth_tst, type = "raw")
prob_logreg <- predict.train(object=logreg_model,
  newdata = lastMonth_tst, type = "prob"

# Evaluate results
```

```
confusionMatrix(pred_logreg, lastMonth_tst$likelyToDonateThisMonth)

# Save the logistic model results in new data frame.
lastMonth_tst_logreg <- lastMonth_tst
lastMonth_tst_logreg$pred <- pred_logreg
lastMonth_tst_logreg$prob <- prob_logreg


########   DECISION TREE MODEL    ########

# Now we'll do a decision tree model.
# Took 15 minutes to run.
dtree_model <- train(likelyToDonateThisMonth~.-ID_NUMBER, data=lastMonth_trn,
 method="rpart",
 metric="ROC",
 trControl = trn_ctrl)

dtree_model
summary(dtree_model$finalModel)


# Predict the test data using the model
pred_dtree <- predict.train(object = dtree_model, newdata = lastMonth_tst, type = "raw")
prob_dtree <- predict.train(object = dtree_model, newdata = lastMonth_tst, type = "prob")
# Evaluate the model performance on the test data
confusionMatrix(pred_dtree, lastMonth_tst$likelyToDonateThisMonth)

# Save the decision tree model results in a new data frame
lastMonth_tst_dtree <- lastMonth_tst
lastMonth_tst_dtree$pred <- pred_dtree
lastMonth_tst_dtree$prob <- prob_dtree

fancyRpartPlot(dtree_model$finalModel)


########   RANDOM FOREST MODEL    ########

mtry <- 1:3
tunegrid <- expand.grid(.mtry=mtry)

# The RF model was still not running (crashed computer)
# so I am going to feed it a subset of half of the training data.
lastMonth_trn_backup = lastMonth_trn
further_idx <- createDataPartition(lastMonth_trn$likelyToDonateThisMonth,
 p = 0.5, list = FALSE)
lastMonth_trn <- lastMonth_trn[further_idx,]

# Now running RF model with half the observations.
# It ran successfully with half in 15 minutes.
rf_model <- train(likelyToDonateThisMonth~.-ID_NUMBER, data = lastMonth_trn,
```

```
                method="rf",
                metric="ROC",
                tuneGrid = tunegrid,
                trControl = trn_ctrl)

# We want to restore the training data to its full size
# in case we run further models.
lastMonth_trn = lastMonth_trn_backup

rf_model
summary(rf_model$results)


# predict the test data using the model
pred_rf <- predict.train(object = rf_model, newdata = lastMonth_tst, type = "raw")
prob_rf <- predict.train(object = rf_model, newdata = lastMonth_tst, type = "prob")
# and let's evaluate the model performance on the test data
confusionMatrix(pred_rf, lastMonth_tst$likelyToDonateThisMonth)

# lets find the important variables with VIF
rfVarImp <- varImp(rf_model$finalModel)
# rfVarImp <- arrange(rfVarImp, desc(Overall))

# Graphs each variable's importance.
rfVarImp %>%
  mutate(Variable = factor(rownames(rfVarImp))) %>%
  ggplot(aes(x = reorder(Variable,desc(Overall)), y = Overall))+
    geom_bar(stat = "identity")+
    ylab("Importance of Variables in the rf model")+
    xlab("Overall importance VIF")+
    coord_flip()+
    scale_x_discrete(limits=rev)

# This just flips the previous graph for a different look.
rfVarImp %>%
  mutate(Variable = factor(rownames(rfVarImp))) %>%
  ggplot(aes(x = reorder(Variable,(Overall)), y = Overall))+
    geom_bar(stat = "identity")+
    ylab("Importance of Variables in the rf model")+
    xlab("Overall importance VIF")+
    #coord_flip()+
    scale_x_discrete(limits=rev)+
    theme(axis.text = element_text(angle = 90))


table(rfVarImp$Overall)
```