Online News Popularity

1.1 Summary

- Problem we are trying to solve: predict the number of shares of an online news article, developing an understanding of the attributes that would lead an article to be shared frequently. <u>How can we write a more popular article?</u>
    - We need to come up with a criteria for the article to be classifies as popular or not based on the number of shares

As increasingly more people turn to online digital media to access news, online platforms such as Buzefeed, Medium, and Mashable are becoming more competitive in their goal of attracting a larger share of the online news audience. The Mashable dataset, created by Kelwin Fernandes, Pedro Vinagre, Paulo Cortez, and Pedro Sernadela on January 2015 draws on data collected from 39k articles published by the online platform Mashable in order to predict which articles will have the most shares among the website's viewership.

- Models used
    - Random Forests
    - Boosted Tree
    - ANN

Median shares by data channel

1.2 Data Dictionary

Data Structure and Dictionary

We will be using the Mashable dataset which can be found on the UC Irvine repository here: https://archive.ics.uci.edu/ml/datasets/Online+News+Popularity

Report: https://repositorium.sdum.uminho.pt/bitstream/1822/39169/1/main.pdf

Number of attributes: 61 (58 predictive attributes, 2 non-predictive, 1 goal field)

Variable Name	**Unit Type**	**Description**
url	chr	URL of the article
timedelta	num	Days between the article publication and the dataset acquisition
n_tokens_title	num	Number of words in the title

n_tokens_content	num	Number of words in the content
n_unique_tokens	num	Ratio of unique words in the content to the total number of words in the content
n_non_stop_words	num	Ratio of non-stop words in the content to the total number of words in the content
n_non_stop_unique_tokens	num	Ratio of unique non-stop words to the total number of words in the content
num_hrefs	num	Number of links
num_self_hrefs	num	Number of links to other articles published by Mashable
num_imgs	num	Number of images
num_videos	num	Number of videos
average_token_length	num	Average length of the words in the content
num_keywords	num	Number of keywords in the metadata
data_channel_is_lifestyle	num	Is data channel 'Lifestyle'?
data_channel_is_entertainment	num	Is data channel 'Entertainment'?
data_channel_is_bus	num	Is data channel 'Business'?
data_channel_is_socmed	num	Is data channel 'Social Media'?
data_channel_is_tech	num	Is data channel 'Tech'?
data_channel_is_world	num	Is data channel 'World'?

kw_min_min	num	Worst keyword (min. shares)
kw_max_min	num	Worst keyword (max. shares)
kw_avg_min	num	Worst keyword (avg. shares)
kw_min_max	num	Best keyword (min. shares)
kw_max_max	num	Best keyword (max. shares)
kw_avg_max	num	Best keyword (avg. shares)
kw_min_avg	num	Avg. keyword (min. shares)
kw_max_avg	num	Avg. keyword (max. shares)
kw_avg_avg	num	Avg. keyword (avg. shares)
self_reference_min_shares	num	Min. shares of referenced articles in Mashable
self_reference_max_shares	num	Max. shares of referenced articles in Mashable
self_reference_avg_sharess	num	Avg. shares of referenced articles in Mashable
weekday_is_monday	num	Was the article published on a Monday?
weekday_is_tuesday	num	Was the article published on a Tuesday?
weekday_is_wednesday	num	Was the article published on a Wednesday?
weekday_is_thursday	num	Was the article published on a Thursday?
weekday_is_friday	num	Was the article published on a Friday?
weekday_is_saturday	num	Binary indicator (0 or 1) for whether the day of the week is Saturday.

weekday_is_sunday	num	Binary indicator (0 or 1) for whether the day of the week is Sunday.
is_weekend	num	Binary indicator (0 or 1) for whether the day of the week is a weekend day (either Saturday or Sunday).
LDA_00	num	A numeric value representing the weight of the first topic in a set of topics generated by a Latent Dirichlet Allocation (LDA) model. - Has high correlation with entertainment so it probably contains words that pertain to entertainment
LDA_01	num	A numeric value representing the weight of the second topic in a set of topics generated by a Latent Dirichlet Allocation (LDA) model. - lifestyle
LDA_02	num	A numeric value representing the weight of the third topic in a set of topics generated by a Latent Dirichlet Allocation (LDA) model. - tech
LDA_03	num	A numeric value representing the weight of the fourth topic in a set of topics generated by a Latent Dirichlet Allocation (LDA) model. - No strong correlation with data channel
LDA_04	num	A numeric value representing the weight of the fifth topic in a set of topics generated by a Latent Dirichlet Allocation (LDA) model. - Socmed (social media)
global_subjectivity	num	A numeric value representing the degree of subjectivity of the text in the article. This value ranges from 0 to 1, with 0 indicating that the text is completely objective and 1 indicating that the text is completely subjective.
global_sentiment_polarity	num	A numeric value representing the overall sentiment of the text in the article. This value ranges from -1 to 1, with -1 indicating that the text has a highly negative sentiment and 1 indicating that the text has a highly positive sentiment.
global_rate_positive_words	num	A numeric value representing the proportion of positive words in the text of the article.
global_rate_negative_words	num	A numeric value representing the proportion of negative words in the text of the article.

rate_positive_words	num	A numeric value representing the proportion of positive words in the non-neutral words of the text of the article.
rate_negative_words	num	A numeric value representing the proportion of negative words in the non-neutral words of the text of the article.
avg_positive_polarity	num	A numeric value representing the average polarity (i.e., positivity or negativity) of the positive words in the text of the article.
min_positive_polarity	num	A numeric value representing the minimum polarity of the positive words in the text of the article.
max_positive_polarity	num	A numeric value representing the maximum polarity of the positive words in the text of the article.
avg_negative_polarity	num	A numeric value representing the average polarity (i.e., positivity or negativity) of the negative words in the text of the article.
avg_negative_polarity	num	The average negative polarity of the article's text
min_negative_polarity	num	The minimum negative polarity of the article's text
max_negative_polarity	num	The maximum negative polarity of the article's text
title_subjectivity	num	The subjectivity of the article's title
title_sentiment_polarity	num	The sentiment polarity of the article's title
abs_title_subjectivity	num	The absolute value of the subjectivity of the article's title
abs_title_sentiment_polarity	num	The absolute value of the sentiment polarity of the title
shares	int	The number of shares the article received on social media - Facebook, Twitter, Google+, LinkedIn, StumbleUpon and Pinterest

2 Exploratory Data Analysis

2.1 Analyzing the data structure of each dataset
The Mashable dataset is a data frame, with 39,644 observations of 61 variables including the number of shares, the absolute polarity level, and various variables representing the data channel or category in which the article was published. The 61 variables in this dataset describe attributes of the articles or observations collected by the data collectors. All variables are of numerical type, with the exception of the “shares” variable and “url” which is simply the URL of the article and is a non-predictive variable. The data channel variables, such as data_channel_is_lifestyle or data_channel_is_tech are valued at either 0 or 1 to represent Yes or No to the corresponding question of interest (being whether or not an article is of category Lifestyle, Tech, etc.). The shares/polarity variables such as "shares" and "abs_title_polarity" have values on a scale of continuous numerical values. The "shares" variable represents the number of times an article was shared on social media, while "abs_title_polarity" is a measure of the polarity or sentiment of the article's title, with positive values indicating a positive sentiment and negative values indicating a negative sentiment. We determined this information using the str() function to find the number of observations, a list of the variables and their types, and the class of the dataset.

2.2 Determining if there are missing values
There are no missing values for any of the variables of this dataset. We determined this by using the is.na() and incomplete cases function to determine if there were any missing values in the first few thousand rows and if there were any missing values in any of the rows of the dataset by column, respectively.

Will null non-predictive variables: url and timedelta

Models Chosen: Random Forest, Boosted tree, ANN

Beginning of model implementation

WHY RANDOM FORESTS

QUESTIONS
1. Should I keep outliers considering that Random Forests are a great fit considering outliers?
2. Successful code for XGBoost… Should I try to predict something other than number of shares, possibly category of article to see classification?

RESULTS FOR RANDOM FOREST:
```
[1] "MAPE for Testing Set Is: 69.62"
[1] "RMSE for Testing Set Is: 1056.59"
```

CODE FOR RANDOM FORESTS

```
#Code for Random Forests
#Online News Popularity

options(scipen=999)
library(randomForest)
library(pdp) #to get the partial dependence plots on probability scale for
#classification problems
library(gmodels)
library(ggplot2)
install.packages("ggcorrplot")
library("ggcorrplot")

# Read the CSV file
mydata = read.csv("C:/Docs/BABSON/QTM2623/Data/OnlineNewsPopularity.csv")

#Data Structure
str(mydata) #dataset is a date.frame; 39644 obs of 61 variables

#Find the number of variables
length(mydata)          #61 variables

#Summary
summary(mydata)

#Class
class(mydata) #dataframe is indeed a data.frame

#Determine if there are missing values
# Find missing values
is.na(mydata) #no missing values
mydata[!complete.cases(mydata),] #there are 0 rows with missing values

#Defining response variable
mydata$myresponse=mydata$shares
mydata$shares=NULL

#Determining Colinearity
mydata$url<-NULL

#can only use numerical predictors
corr<-round(cor(mydata), 1)
head(corr[, 1:5])
ggcorrplot(corr)
```

```
##Removing Outliers
Q1<-quantile(mydata$myresponse,.25)
Q3<-quantile(mydata$myresponse,.75)
IQR<-IQR(mydata$myresponse)

mydata_no_outliers<-subset(mydata,mydata$myresponse> (Q1 - 1.5*IQR) &
mydata$myresponse< ( Q3 + 1.5*IQR))

mydata<-mydata_no_outliers
mydata_no_outliers<-NULL

##Removing Redundant Variables
colnames(mydata)
mydata$is_weekend<-NULL #Redundant, info captured in day specific predictors
mydata$n_unique_tokens<-NULL
mydata$n_non_stop_words<-NULL

mydata$kw_min_min<-NULL
mydata$kw_max_min<-NULL
mydata$kw_avg_min<-NULL

mydata$kw_min_max<-NULL
mydata$kw_max_max<-NULL
mydata$kw_avg_max<-NULL

mydata$kw_min_avg<-NULL
mydata$kw_max_avg<-NULL
#mydata$kw_avg_avg<-NULL

mydata$self_reference_max_shares<-NULL
mydata$self_reference_min_shares<-NULL

mydata$rate_positive_words<-NULL
mydata$rate_negative_words<-NULL

mydata$min_positive_polarity<-NULL
mydata$max_positive_polarity<-NULL
mydata$min_negative_polarity<-NULL
mydata$max_negative_polarity<-NULL

mydata$abs_title_sentiment_polarity<-NULL
mydata$abs_title_subjectivity<-NULL
```

```
mydata$num_videos<-NULL
mydata$num_imgs<-NULL

##Reassessment of Colinearity
corr<-round(cor(mydata), 1)
head(corr[, 1:5])
ggcorrplot(corr)

##Predictor Transformation
str(mydata)

mydata$data_channel_is_lifestyle<-as.factor(mydata$data_channel_is_lifestyle)
mydata$data_channel_is_entertainment<-as.factor(mydata$data_channel_is_entertainment)
mydata$data_channel_is_bus<-as.factor(mydata$data_channel_is_bus)
mydata$data_channel_is_socmed<-as.factor(mydata$data_channel_is_socmed)
mydata$data_channel_is_tech<-as.factor(mydata$data_channel_is_tech)
mydata$data_channel_is_world<-as.factor(mydata$data_channel_is_world)

mydata$weekday_is_monday<-as.factor(mydata$weekday_is_monday)
mydata$weekday_is_tuesday<-as.factor(mydata$weekday_is_tuesday)
mydata$weekday_is_wednesday<-as.factor(mydata$weekday_is_wednesday)
mydata$weekday_is_thursday<-as.factor(mydata$weekday_is_thursday)
mydata$weekday_is_friday<-as.factor(mydata$weekday_is_friday)
mydata$weekday_is_saturday<-as.factor(mydata$weekday_is_saturday)
mydata$weekday_is_sunday<-as.factor(mydata$weekday_is_sunday)

str(mydata)

#START OF SETUP

tree_type="R" #regression

num.tree = 500 #maximum allowable number of trees in the forest

#END OF SETUP

#START DATA BREAKDOWN FOR HOLDOUT METHOD

#Start finding the categorical predictors

numpredictors=dim(mydata)[2]-1

numfac=0
```

```
for (i in 1:numpredictors) {
  if ((is.factor(mydata[,i]))){
    numfac=numfac+1}
}

#End finding the number of categorical predictors

nobs=dim(mydata)[1]

if (tree_type=="R") {

  train_size=floor(0.8*nobs)
  test_size=nobs-train_size

} else {

  prop = prop.table(table(mydata$myresponse))
  length.vector = round(nobs*0.8*prop)
  train_size=sum(length.vector)
  test_size=nobs-train_size
  class.names = as.data.frame(prop)[,1]
  numb.class = length(class.names)}


resample=1
RNGkind(sample.kind = "Rejection")
set.seed(1) #setting the seed for random sampling

while (resample==1) {


  if (tree_type=="C") {

    train_index = c()

    for(i in 1:numb.class){
      index_temp = which(mydata$myresponse==class.names[i])
      train_index_temp = sample(index_temp, length.vector[i], replace = F)
      train_index = c(train_index, train_index_temp)
    }} else {
      train_index=sample(nobs,train_size, replace=F)
    }
  mydata_train=mydata[train_index,] #randomly selecting the data for training set using the row
numbers generated above
  mydata_test=mydata[-train_index,]#everything not in the training set should go into testing set

  right_fac=0 #denotes the number of factors with "right" distributions (i.e. - the unique levels
match across mydata, test, and train data sets)

  for (i in 1:numpredictors) {
    if (is.factor(mydata_train[,i])) {
      if (sum(as.vector(unique(mydata_test[,i])) %in%
as.vector(unique(mydata_train[,i])))==length(unique(mydata_test[,i])))
        right_fac=right_fac+1
    }
  }

  if (right_fac==numfac) (resample=0) else (resample=1)

}

dim(mydata_test) #confirms that testing data has only 20% of observations
dim(mydata_train) #confirms that training data has 80% of observations

#END DATA BREAKDOWN FOR HOLDOUT METHOD

#START FOREST SIZE FINDER
set.seed(123)#don't modify the seed
rf.train=randomForest(myresponse~.,
                      data=mydata_train,
                      ntree=num.tree,
                      mtry = floor(sqrt(ncol(mydata_train))),
                      replace = TRUE)



ylim.ceiling=max(plot(rf.train))+0.20*(max(plot(rf.train))-min(plot(rf.train)))
ylim.floor=min(plot(rf.train))
plot(rf.train, main="Error Rate vs Number of Trees In the Forest",ylim=c(ylim.floor,ylim.ceiling))

if (tree_type=="C"){
  rndF1.legend <- colnames(rf.train$err.rate)
  legend("top",cex =0.7, legend=rndF1.legend, lty=rep.int(2,length(rndF1.legend)),
col=c(1:length(rndF1.legend)), horiz=T)}

#END FOREST SIZE FINDER
#START FINAL CONFIGURATION
```

```
################################################################################
#####
num.tree.final=150 #After inspecting "Error Rate vs Number of Trees In the Forest"
#the point on the horizontal axis where the error rate tends to stabilize is 150

#END FINAL CONFIGURATION

################################################################################
#####
####################DO NOT MODIFY BEYOND THIS
POINT############################
################################################################################
#####
set.seed(123)#don't modify the seed
rf.train.final=randomForest(myresponse~.,
                       data=mydata_train,
                       ntree=num.tree.final,
                       importance=TRUE, na.action = na.omit)

varImpPlot(rf.train.final, type=1, scale=FALSE)#mean decrease in accuracy variable importance
plot

#Finding the most important predictors for which partial dependence plots will be plotted
importance.tbl=as.data.frame(unlist(rf.train.final$importance))
if (tree_type=="C") (which.col="MeanDecreaseAccuracy") else (which.col="%IncMSE")
q.09=quantile(importance.tbl[,which.col], 0.9)
most.important.predictors=rownames(importance.tbl)[which(importance.tbl[,which.col]>=q.09)]

#creating partial dependence plots
if (tree_type=="C"){
  class.to.plot=rf.train.final$classes[1]
  title=paste("PD Plot for Class", class.to.plot)
  y.legend="Average Probability"}else
  {title="PD Plot"
  y.legend="Average Value of Outcome"}

pd.plot <-function (x) {partial(rf.train.final, x, plot = TRUE, prob=TRUE, quantiles=F,
                          plot.engine = "ggplot2")+ggtitle(title)+ylab(y.legend)}

lapply(most.important.predictors, pd.plot)

#START PREDICTING THE RESPONSE IN THE TESTING SET (20 % SUBSET)
predictions=predict(rf.train.final, newdata = mydata_test)
```

```
mydata_test_w_predictions=cbind(mydata_test, predictions)

#Measuring predictive accuracy

if (tree_type=="R") {


abs.diff=abs(mydata_test_w_predictions$predictions-mydata_test_w_predictions$myresponse)
  mape=100*mean(abs.diff/abs(mydata_test_w_predictions$myresponse))
  rmse=sqrt(mean(abs.diff^2))

  print(paste("MAPE for Testing Set Is:",
              round(mape,2)))

  print(paste("RMSE for Testing Set Is:",
              round(rmse,2)))

} else {
  print("Confusion Matrix Is:")

CrossTable(mydata_test_w_predictions$myresponse,mydata_test_w_predictions$predictions,pr
op.chisq=F,prop.t=F) }
```

CODE FOR BOOSTED TREES

```
#####DATA CLEANING#####
###Data Management
colnames(mydata)

mydata$myresponse=mydata$shares
  #specify response variable
mydata$shares=NULL
  #Nullify redundant variable

##Removing Incomplete Records
mydata<-na.omit(mydata)

##Determining Colinearity
str(mydata)
```

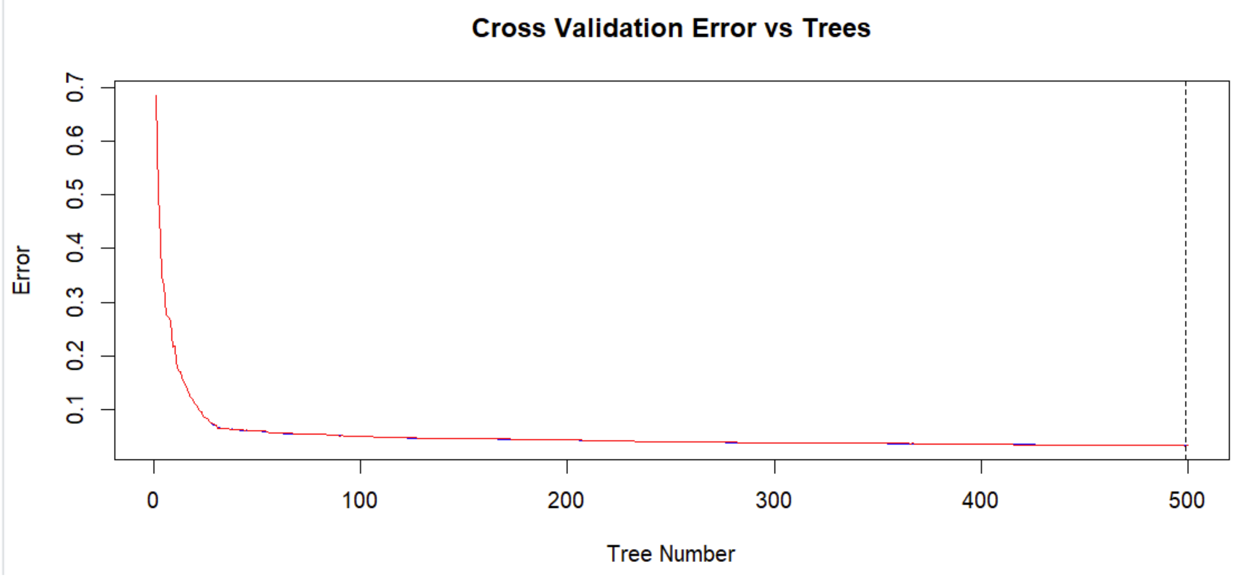
```
mydata$url<-NULL
  #can only use numerical predictors
corr<-round(cor(mydata), 1)
head(corr[, 1:5])
ggcorrplot(corr)

##Removing Outliers
Q1<-quantile(mydata$myresponse,.25)
Q3<-quantile(mydata$myresponse,.75)
IQR<-IQR(mydata$myresponse)

mydata_no_outliers<-subset(mydata,mydata$myresponse> (Q1 - 1.5*IQR) &
mydata$myresponse< ( Q3 + 1.5*IQR))

mydata<-mydata_no_outliers
mydata_no_outliers<-NULL

##Removing Redundant Variables
colnames(mydata)
mydata$is_weekend<-NULL
  #Redundant, info captured in day specific predictors
mydata$n_unique_tokens<-NULL
mydata$n_non_stop_words<-NULL

mydata$kw_min_min<-NULL
mydata$kw_max_min<-NULL
mydata$kw_avg_min<-NULL

mydata$kw_min_max<-NULL
mydata$kw_max_max<-NULL
mydata$kw_avg_max<-NULL

mydata$kw_min_avg<-NULL
mydata$kw_max_avg<-NULL
#mydata$kw_avg_avg<-NULL

mydata$self_reference_max_shares<-NULL
mydata$self_reference_min_shares<-NULL

mydata$rate_positive_words<-NULL
mydata$rate_negative_words<-NULL

mydata$min_positive_polarity<-NULL
mydata$max_positive_polarity<-NULL
```

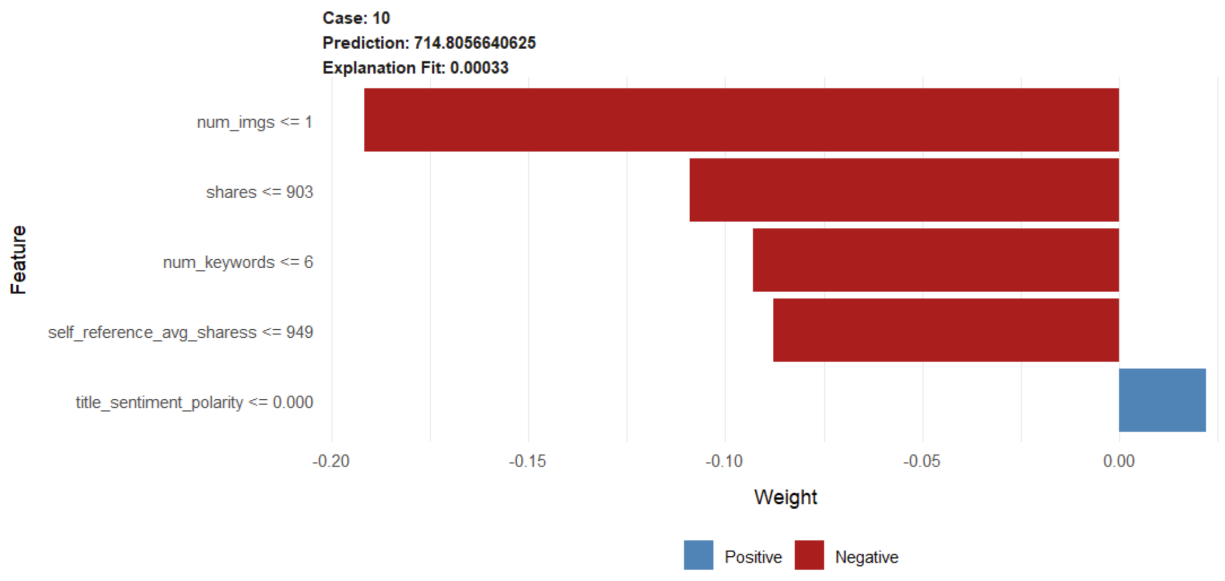
```
mydata$min_negative_polarity<-NULL
mydata$max_negative_polarity<-NULL

mydata$abs_title_sentiment_polarity<-NULL
mydata$abs_title_subjectivity<-NULL

##Reassessment of Colinearity
corr<-round(cor(mydata), 1)
head(corr[, 1:5])
ggcorrplot(corr)

##Predictor Transformation
str(mydata)

mydata$data_channel_is_lifestyle<-as.factor(mydata$data_channel_is_lifestyle)
mydata$data_channel_is_entertainment<-as.factor(mydata$data_channel_is_entertainment)
mydata$data_channel_is_bus<-as.factor(mydata$data_channel_is_bus)
mydata$data_channel_is_socmed<-as.factor(mydata$data_channel_is_socmed)
mydata$data_channel_is_tech<-as.factor(mydata$data_channel_is_tech)
mydata$data_channel_is_world<-as.factor(mydata$data_channel_is_world)

mydata$weekday_is_monday<-as.factor(mydata$weekday_is_monday)
mydata$weekday_is_tuesday<-as.factor(mydata$weekday_is_tuesday)
mydata$weekday_is_wednesday<-as.factor(mydata$weekday_is_wednesday)
mydata$weekday_is_thursday<-as.factor(mydata$weekday_is_thursday)
mydata$weekday_is_friday<-as.factor(mydata$weekday_is_friday)
mydata$weekday_is_saturday<-as.factor(mydata$weekday_is_saturday)
mydata$weekday_is_sunday<-as.factor(mydata$weekday_is_sunday)

str(mydata)
```

RESULTS FOR XGBoost:

```
[1] "MAPE for Testing Set Is: 2.9"
[1] "RMSE for Testing Set Is: 48.71"
```

Plots for XGBoost:





CODE FOR XGBoost:

```
library(xgboost)
library(gmodels)
library(mltools) #for one-hot encoding
library(data.table) #for one-hot encoding
library(lime)

#Import the data
mydata=read.csv("C:/Users/Jmoschella1/Desktop/RStudio/rawdata/OnlineNewsPopularity.csv")

#Type "C" for classification and "R" for regression

model.type="R"
```

```
#Number of rounds. This can be set to a lower or higher value, if you wish, example: 150 or 250 or 300
cv.nround = 500


#END OF DATA IMPORT

#START OF RESPONSE REDEFINITION

mydata$myresponse=mydata$shares #Substitute "Status" with the name of your response variable
mydata$shares=NULL #Substitute "Status" with the name of your response variable

if (model.type=="C") (table(mydata$myresponse))

#The code below redefines the levels of categorical outcome to 0,1,...
if (model.type=="C"){

  if (is.character(mydata$myresponse)){
    mydata$myresponse=as.factor(mydata$myresponse)}

  mydata$myresponse=as.numeric(mydata$myresponse)

  if (min(mydata$myresponse)!=0){
    if (min(mydata$myresponse)>0)
      (mydata$myresponse=mydata$myresponse-min(mydata$myresponse)) else
        (mydata$myresponse=mydata$myresponse+abs(min(mydata$myresponse)))}


  unique.vals=unique(mydata$myresponse)
  unique.vals=unique.vals[order(unique.vals)]

  if (sum(unique.vals==seq(0,length(unique.vals)-1))!=length(unique(unique.vals))){

    j=0

    for (i in unique.vals){
      mydata$myresponse[mydata$myresponse==i]=j
      j=j+1
    }
  }
}

if (model.type=="C") (table(mydata$myresponse))


#END OF RESPONSE REDEFINITION

#In the following statements substitute the names after "$" sign with the names of predictors
#in your data that are categorical but are read into R in a different format. If there are no such
#variables in your data, then ignore.

#START OF PREDICTOR TRANSFORMATION

mydata$weekday_is_tuesday=as.factor(mydata$weekday_is_tuesday)
mydata$weekday_is_monday=as.factor(mydata$weekday_is_monday)
mydata$weekday_is_wednesday=as.factor(mydata$weekday_is_wednesday)
mydata$weekday_is_thursday=as.factor(mydata$weekday_is_thursday)
```

```
mydata$weekday_is_friday=as.factor(mydata$weekday_is_friday)
mydata$weekday_is_saturday=as.factor(mydata$weekday_is_saturday)
mydata$weekday_is_sunday=as.factor(mydata$weekday_is_sunday)
mydata$is_weekend=as.factor(mydata$is_weekend)


mydata$data_channel_is_lifestyle=as.factor(mydata$data_channel_is_lifestyle)
mydata$data_channel_is_entertainment=as.factor(mydata$data_channel_is_entertainment)
mydata$data_channel_is_bus=as.factor(mydata$data_channel_is_bus)
mydata$data_channel_is_socmed=as.factor(mydata$data_channel_is_socmed)
mydata$data_channel_is_tech=as.factor(mydata$data_channel_is_tech)
mydata$data_channel_is_world=as.factor(mydata$data_channel_is_world)

#add statements similar to above as needed


#END OF PREDICTOR TRANSFORMATION

#The statements below remove all the variables that will not be passed to the tree algorithm
#as predictors. If no such redundant variables exist in your dataset, then the statements
#in the "REDUNDANT VARIABLE REMOVAL" section should be ignored.

#START OF REDUNDANT VARIABLE REMOVAL
##Removing Outliers
Q1<-quantile(mydata$myresponse,.25)
Q3<-quantile(mydata$myresponse,.75)
IQR<-IQR(mydata$myresponse)

mydata_no_outliers<-subset(mydata,mydata$myresponse> (Q1 - 1.5*IQR) & mydata$myresponse< ( Q3
+ 1.5*IQR))

mydata<-mydata_no_outliers
mydata_no_outliers<-NULL

##Removing Redundant Variables
colnames(mydata)
mydata$is_weekend<-NULL #Redundant, info captured in day specific predictors
mydata$n_unique_tokens<-NULL
mydata$n_non_stop_words<-NULL

mydata$kw_min_min<-NULL
mydata$kw_max_min<-NULL
mydata$kw_avg_min<-NULL

mydata$kw_min_max<-NULL
mydata$kw_max_max<-NULL
mydata$kw_avg_max<-NULL

mydata$kw_min_avg<-NULL
mydata$kw_max_avg<-NULL
#mydata$kw_avg_avg<-NULL

mydata$self_reference_max_shares<-NULL
mydata$self_reference_min_shares<-NULL

mydata$rate_positive_words<-NULL
```

```
mydata$rate_negative_words<-NULL

mydata$min_positive_polarity<-NULL
mydata$max_positive_polarity<-NULL
mydata$min_negative_polarity<-NULL
mydata$max_negative_polarity<-NULL

mydata$abs_title_sentiment_polarity<-NULL
mydata$abs_title_subjectivity<-NULL



mydata$url=NULL
mydata$timedelta=NULL
#Substitute "xyz" with the name of the variable in your data that
#will not be passed to the tree algorithm. Add as many statements similar
#to this as needed.

#END OF REDUNDANT VARIABLE REMOVAL

####################################################################################
#########
#####################################ATTENTION#####################################
##########
####################################################################################
#########

#######################IF THE ABOVE MODIFICATIONS ARE MADE
CORRECTLY,########################
####AT THIS POINT "MYDATA" DATA FRAME SHOULD CONTAIN ONLY THE PREDICTORS AND THE
OUTCOME.####
####IN CASE IT CONTAINS ANYTHING MORE OR LESS, THE CODE BELOW WILL NOT FUNCTION
PROPERLY.####
####################################################################################
#########

str(mydata) #make sure the structure of your data reflects all the modifications made above

#Start 80-20 partition
numpredictors=dim(mydata)[2]-1

numfac=0

for (i in 1:numpredictors) {
  if ((is.factor(mydata[,i]))){
    numfac=numfac+1}
}

#End finding the number of categorical predictors

nobs=dim(mydata)[1]


if (model.type=="R") {

  #Below is the setup for stratified 80-20 holdout sampling for a Regression Tree
  train_size=floor(0.8*nobs)
  test_size=nobs-train_size

} else {

  #Below is the setup for stratified 80-20 holdout sampling for a Classification Tree

  prop = prop.table(table(mydata$myresponse))
  length.vector = round(nobs*0.8*prop)
  train_size=sum(length.vector)
  test_size=nobs-train_size
  class.names = as.data.frame(prop)[,1]
  numb.class = length(class.names)}


resample=1
RNGkind(sample.kind = "Rejection")
set.seed(1) #sets the seed for random sampling

while (resample==1) {


  if (model.type=="C") {

    train_index = c()

    for(i in 1:numb.class){
      index_temp = which(mydata$myresponse==class.names[i])
      train_index_temp = sample(index_temp, length.vector[i], replace = F)
      train_index = c(train_index, train_index_temp)
    }} else {
      train_index=sample(nobs,train_size, replace=F)
    }

  mydata_train=mydata[train_index,] #randomly select the data for training set using the row numbers
generated above
  mydata_test=mydata[-train_index,]#everything not in the training set should go into testing set

  right_fac=0 #denotes the number of factors with "right" distributions (i.e. - the unique levels match across
mydata, test, and train data sets)


  for (i in 1:numpredictors) {
    if (is.factor(mydata_train[,i])) {
      if (sum(as.vector(unique(mydata_test[,i])) %in%
as.vector(unique(mydata_train[,i])))==length(unique(mydata_test[,i])))
        right_fac=right_fac+1
    }
  }

  if (right_fac==numfac) (resample=0) else (resample=1)

}

dim(mydata_test) #confirms that testing data has only 20% of observations
dim(mydata_train) #confirms that training data has 80% of observations
#End 80-20 partition


#One-hot encoding of factor predictors for training and test sets separately

if (sum(sapply(mydata_train, is.factor))>0) {

  only.fac.train=mydata_train[,sapply(mydata_train, is.factor), drop=FALSE]
  only.fac.test=mydata_test[,sapply(mydata_test, is.factor), drop=FALSE]

  non.fac.train=mydata_train[,setdiff(colnames(mydata_train), colnames(only.fac.train))]
  non.fac.test=mydata_test[,setdiff(colnames(mydata_test), colnames(only.fac.test))]

  if (length(only.fac.train)>0) {
    dummy.train <- one_hot(as.data.table(only.fac.train))
    dummy.test  <- one_hot(as.data.table(only.fac.test))
    mydata_train=cbind(non.fac.train, dummy.train)
    mydata_test=cbind(non.fac.test, dummy.test)}
}


#Set the parameters for cross-validation and xgboost.
#You can try different values for nthread, max_depth, eta, gamma, etc., and see if you get lower
prediction error.

#Start of Hyperparameter Setup

if ((model.type=="C") & (length(unique(mydata$myresponse))==2)) {
  cl.obj="binary:logistic"
  cl.eval.metric="error"} else {
    if ((model.type=="C") & (length(unique(mydata$myresponse))>2)) {

      cl.obj="multi:softmax"
      cl.eval.metric="merror"}}

if (model.type=="C") {

  if ((length(unique(mydata$myresponse))==2)) {

    param.cl       = list("objective" = cl.obj,
                          #"num_class"= length(unique(mydata$myresponse)),
                          "eval_metric" = cl.eval.metric,
                          "nthread" = 8,                    # number of threads to be used
                          "max_depth" = 1,                  # maximum depth of tree
                          "eta" = 0.3,                      # step size shrinkage
                          "gamma" = 0,                      # minimum loss reduction
                          "subsample" = 1,                  # part of data instances to grow tree
                          "colsample_bytree" = 1,                   # subsample ratio of columns when
constructing each tree
                          "min_child_weight" = 5,                   # minimum sum of instance weight needed in a
child
                          "early_stopping_rounds"=10
    )
  } else {
    param.cl       = list("objective" = cl.obj,
                          "num_class"= length(unique(mydata$myresponse)),
                          "eval_metric" = cl.eval.metric,
                          "nthread" = 8,                    # number of threads to be used
                          "max_depth" = 1,                  # maximum depth of tree
                          "eta" = 0.3,                      # step size shrinkage
                          "gamma" = 0,                      # minimum loss reduction
                          "subsample" = 1,                  # part of data instances to grow tree
                          "colsample_bytree" = 1,                   # subsample ratio of columns when
constructing each tree
                          "min_child_weight" = 5,                   # minimum sum of instance weight needed in a
child
                          "early_stopping_rounds"=10
    )
  }
} else {

  param.reg       = list("objective" = "reg:squarederror",
                     "eval_metric" = "mape",
                     "nthread" = 8,                     # number of threads to be used
                     "max_depth" = 1,                   # maximum depth of tree
                     "eta" = 0.3,                       # step size shrinkage
                     "gamma" = 0,                       # minimum loss reduction
                     "subsample" = 1,                   # part of data instances to grow tree
                     "colsample_bytree" = 1,            # subsample ratio of columns when constructing each
tree
                     "min_child_weight" = 5,            # minimum sum of instance weight needed in a child
                     "early_stopping_rounds"=10
  )
}

#End of Hyperparameter Setup


#Identify the Predictors and the dependent variable, aka label.
predictors = setdiff(colnames(mydata_train), "myresponse")

#Save the training output variable into "label"
label=mydata_train$myresponse

###################################################################################
#####################
# Step 1: Run a Cross-Validation to identify the round with the minimum loss or error.
#         Note: xgboost expects the data in the form of a numeric matrix.
###################################################################################
#########################################

set.seed(100)

if (model.type=="C"){
  bst.cv = xgb.cv(
    param=param.cl,
    data = as.matrix(mydata_train[,predictors]),
    label = label,
    nfold = 10,
    nrounds=cv.nround,
    prediction=T,
    verbose=F)
} else {

  bst.cv = xgb.cv(
    param=param.reg,
    data = as.matrix(mydata_train[,predictors]),
    label = label,
    nfold = 10,
    nrounds=cv.nround,
    prediction=T,
    verbose=F)}

#Find where the minimum loss occurred

if (model.type=="C"){
  if ((length(unique(mydata$myresponse))==2)) {

    min.error.idx = bst.cv$evaluation_log$iter[which.min(bst.cv$evaluation_log$test_error_mean)]} else {
      min.error.idx = bst.cv$evaluation_log$iter[which.min(bst.cv$evaluation_log$test_merror_mean)]}} else
{

        min.error.idx = bst.cv$evaluation_log$iter[which.min(bst.cv$evaluation_log$test_mape_mean)]}

cat ("Minimum error occurred in round : ", min.error.idx, "\n")

# Minimum error
print(min(bst.cv$evaluation_log[,4]))

#Plot the CV error vs tree order

dd=as.data.frame(bst.cv$evaluation_log)
ylim.min=min(c(dd[,2], dd[,4]))
ylim.max=max(c(dd[,2], dd[,4]))

plot(dd[,2], col="blue", type="l", ylab="Error", xlab="Tree Number", main="Cross Validation Error vs
Trees", ylim=c(ylim.min, ylim.max))
lines(dd[,4], col="red")
abline(v=min.error.idx, col="black", lwd=1, lty=2)


##########################################################################################
############################################
# Step 2: Train the xgboost model using min.error.idx found above.
#        Note, we have to stop at the round where we get the minimum error.

##########################################################################################
############################################

set.seed(100)

if (model.type=="C"){

  bst = xgboost(
    param=param.cl,
    data =as.matrix(mydata_train[,predictors]),
    label = label,
    #early_stopping_rounds=10,
    nrounds=min.error.idx,
    verbose=F)} else {

      bst = xgboost(
        param=param.reg,
        data =as.matrix(mydata_train[,predictors]),
        label = label,
        #early_stopping_rounds=10,
        nrounds=min.error.idx,
        verbose = F)}


# Make prediction on the testing data.
mydata_test$prediction = predict(bst, as.matrix(mydata_test[,predictors]))

if ((model.type=="C") & length(unique(mydata$myresponse))==2) {
  mydata_test$prediction=0*(mydata_test$prediction<=0.5)+1*(mydata_test$prediction>0.5)
}


#LIME plots below
#For details see resources below for excellent explanations:
#https://uc-r.github.io/lime
#https://christophm.github.io/interpretable-ml-book/lime.html


#Below provide the row number of the testing set which you would like LIME to explain
row.to.explain=2

set.seed(100)
lime_to_explain <-mydata_test[row.to.explain,predictors] #Explaining the row stored in 'row.to.explain'
explainer <- suppressWarnings(lime(mydata_train, model = bst)) #specifying that the training model will
be used for explanations


if (model.type=="R"){
  explanation <- explain(
    lime_to_explain, #the explanations will be given for these observations
    explainer,
    n_features=5, #the top 5 features based on forward selection will be used in explanations
    feature_select = "forward_selection", #see above

    dist_fun = "euclidean",

    kernel_width = 0.5,


    n_permutations = 500 #for each obs in "lime_to_explain" there will be 500 permutations created
    #based on the data contained in "explainer", i.e. based on the variables
    #of training set. In other words, for each test set observation, 500 obs
    #are created using the distributions of the training data columns. Those then
    #are used to fit the local model in the vicinity of the test set obs in question.
  )
```

```
} else {

  explanation <- explain(
    lime_to_explain, #the explanations will be given for these observations
    explainer,
    n_labels = length(unique(mydata_test$myresponse)),
    n_features=5, #the top 5 features based on forward selection will be used in explanations
    feature_select = "forward_selection", #see above

    dist_fun = "euclidean",

    kernel_width = 0.5,


    n_permutations = 500 #for each obs in "lime_to_explain" there will be 5,00 permutations created
    #based on the data contained in "explainer", i.e. based on the variables
    #of training set. In other words, for each test set observation, 5K obs
    #are created using the distributions of the training data columns. Those then
    #are used to fit the local model in the vicinity of the test set obs in question.
  )


}

plot_features(explanation)


#Summarize the predictive accuracy

if (model.type=="C"){
  #Compute the accuracy of predictions.
  CrossTable(mydata_test$myresponse,mydata_test$prediction,prop.chisq=F,prop.t=F)} else {

    rmse=sqrt(mean((mydata_test$myresponse-mydata_test$prediction)^2))
    mape=100*mean(abs(mydata_test$myresponse-mydata_test$prediction)/mydata_test$myresponse)

    print(paste("RMSE: ", round(rmse,2)))
    print(paste("MAPE: ", round(mape,2)))
  }
```

Code for Data Management

- All outliers (across all variables) and missing records removed
- Histograms for all variables in aggregate and individually
- Box plots for all variables in aggregate
- Correlation plots
- Plots will take a long time to run, they are for all variables

```
#####Onine_News Data_Management_v2#####
```

```
###Loading Packages
#install.packages("ggcorrplot")
#install.packages("caret")
#install.packages("dplyr")
#install.packages("data.table")
#install.packages("ggplot2")
#install.pachages("tidyr")

library("ggcorrplot")
library("caret")
library("dplyr")
library("data.table")
library("ggplot2")
library("tidyr")

###Loading Data
#setwd("D:/Data")
  #Desktop WD
setwd("E:/Data")
  #Laptop WD
#set WD, change to appropriate WD or replace w/ pathway

mydata<-read.csv("OnlineNewsPopularity.csv")
  #load OnlineNewsPopularity as df

###Problem Type
problem.type = "R"
  #C for classification
  #R for regression


###Data Management

##Removing Incomplete Records
mydata<-na.omit(mydata)

mydata$url<-NULL
  #subsequent lines cannot accept chr

##Data visualization

mydata %>% gather() %>% head()

ggplot(gather(mydata), aes(value)) +
  geom_histogram() +
  facet_wrap(~key, scales = "free_x")
  #matrix of histograms

ggplot(gather(mydata), aes(value)) +
  geom_boxplot() +
  facet_wrap(~key, scales = "free_x")
  #matrix of boxplots

for (col in 1:ncol(mydata)) {
  hist(mydata[,col], main = names(mydata[col]) )
}
  #creates histogrm for each column

##Removing Outliers
str(mydata)

mydata$ID = seq_along(mydata[,1])
  #create ID variable, relevant later

head(mydata)

scalable.preds<-c(
  'n_tokens_title',
  'n_tokens_content',
  'n_unique_tokens',
  'n_non_stop_words',
  'n_non_stop_unique_tokens',
  'num_hrefs',
  'num_self_hrefs',
  'num_imgs',
  'num_videos',
  'average_token_length',
  'num_keywords',
  'kw_min_min',
  'kw_max_min',
  'kw_avg_min',
  'kw_min_max',
  'kw_max_max',
  'kw_avg_max',
  'kw_min_avg',
  'kw_max_avg',
  'kw_avg_avg',
  'self_reference_min_shares',
  'self_reference_max_shares',
  'self_reference_avg_sharess',
  'LDA_00',
  'LDA_01',
  'LDA_02',
  'LDA_03',
  'LDA_04',
  'global_subjectivity',
  'global_sentiment_polarity',
  'global_rate_positive_words',
  'global_rate_negative_words',
  'rate_positive_words',
  'rate_negative_words',
  'avg_positive_polarity',
  'min_positive_polarity',
  'max_positive_polarity',
  'avg_negative_polarity',
  'min_negative_polarity',
  'max_negative_polarity',
  'title_subjectivity',
  'title_sentiment_polarity',
  'abs_title_subjectivity',
  'abs_title_sentiment_polarity',
  'timedelta'
)
  #predictors which can be standardized, numeric and not part of target
  #remove time delta if it is being used to find shares over time

mydata_scalable_preds<-mydata[,c('ID' ,scalable.preds)]
head(mydata_scalable_preds)

mydata_no_scale<-data.table(mydata)
mydata_no_scale<-mydata_no_scale[,-c(scalable.preds), with=F]
mydata_no_scale<-as.data.frame(mydata_no_scale)

head(mydata_no_scale)

mydata_z_scores<-as.data.frame(sapply(mydata_scalable_preds,function(mydata_scalable_pr
eds)(abs(mydata_scalable_preds
-mean(mydata_scalable_preds))/sd(mydata_scalable_preds))))

head(mydata_z_scores)
mydata_z_scores$ID<-NULL
mydata_z_scores$ID=seq_along(mydata_z_scores[,1])
```

```
head(mydata_z_scores)

mydata_no_outliers<-mydata_z_scores[!rowSums(mydata_z_scores[,1:43]>3),]
head(mydata_no_outliers)

mydata2<-merge(mydata_no_outliers,mydata_no_scale, by="ID", all=TRUE)
is.na(mydata2)
mydata2<-na.omit(mydata2)

str(mydata2)

mydata<-mydata2
  #df w/ numeric outliers removed
mydata2<-NULL
mydata_no_outliers<-NULL
mydata_no_scale<-NULL
mydata_z_scores<-NULL
mydata_scalable_preds<-NULL

##Establishing Response Variable

#mydata$target=mydata$shares/mydata$timedelta
#mydata$shares<-NULL
#mydata$timedelta<-NULL
#need to activate if finding shares over time

mydata$target=mydata$shares
  #activate if predicting for raw shares
mydata$shares<-NULL

mydata$myresponse=mydata$target
mydata$target=NULL

Q1<-quantile(mydata$myresponse,.25)
Q3<-quantile(mydata$myresponse,.75)
IQR<-IQR(mydata$myresponse)

mydata_no_outliers<-subset(mydata,mydata$myresponse> (Q1 - 1.5*IQR) &
mydata$myresponse< ( Q3 + 1.5*IQR))

mydata<-mydata_no_outliers
  #df w/ outliers for target removed
mydata_no_outliers<-NULL
```

```
##Determining Colinearity
str(mydata)
mydata$url<-NULL
  #can only use numerical predictors
corr<-round(cor(mydata), 1)
head(corr[, 1:5])
ggcorrplot(corr)
  #correlation heatmap between variables

##Removing Redundant Variables
colnames(mydata)

mydata$is_weekend<-NULL
  #Redundant, info captured in day specific predictors

mydata$ID<-NULL
  #no longer necessary

mydata$n_unique_tokens<-NULL
mydata$n_non_stop_words<-NULL

mydata$kw_min_min<-NULL
mydata$kw_max_min<-NULL
mydata$kw_avg_min<-NULL

mydata$kw_min_max<-NULL
mydata$kw_max_max<-NULL
mydata$kw_avg_max<-NULL

mydata$kw_min_avg<-NULL
mydata$kw_max_avg<-NULL
#mydata$kw_avg_avg<-NULL

mydata$self_reference_max_shares<-NULL
mydata$self_reference_min_shares<-NULL

mydata$rate_positive_words<-NULL
mydata$rate_negative_words<-NULL

mydata$min_positive_polarity<-NULL
mydata$max_positive_polarity<-NULL
mydata$min_negative_polarity<-NULL
mydata$max_negative_polarity<-NULL
```

```r
mydata$abs_title_sentiment_polarity<-NULL
mydata$abs_title_subjectivity<-NULL

##Reassesment of Colinearity
corr<-round(cor(mydata), 1)
head(corr[, 1:5])
ggcorrplot(corr)

##Predictor Transformation
str(mydata)

mydata$data_channel_is_lifestyle<-as.factor(mydata$data_channel_is_lifestyle)
mydata$data_channel_is_entertainment<-as.factor(mydata$data_channel_is_entertainment)
mydata$data_channel_is_bus<-as.factor(mydata$data_channel_is_bus)
mydata$data_channel_is_socmed<-as.factor(mydata$data_channel_is_socmed)
mydata$data_channel_is_tech<-as.factor(mydata$data_channel_is_tech)
mydata$data_channel_is_world<-as.factor(mydata$data_channel_is_world)

mydata$weekday_is_monday<-as.factor(mydata$weekday_is_monday)
mydata$weekday_is_tuesday<-as.factor(mydata$weekday_is_tuesday)
mydata$weekday_is_wednesday<-as.factor(mydata$weekday_is_wednesday)
mydata$weekday_is_thursday<-as.factor(mydata$weekday_is_thursday)
mydata$weekday_is_friday<-as.factor(mydata$weekday_is_friday)
mydata$weekday_is_saturday<-as.factor(mydata$weekday_is_saturday)
mydata$weekday_is_sunday<-as.factor(mydata$weekday_is_sunday)

str(mydata)

cat.vars<-c(
  "data_channel_is_lifestyle",
  "data_channel_is_entertainment",
  "data_channel_is_bus",
  "data_channel_is_socmed",
  "data_channel_is_tech",
  "data_channel_is_world",
  "weekday_is_monday",
  "weekday_is_tuesday",
  "weekday_is_wednesday",
  "weekday_is_thursday",
  "weekday_is_friday",
  "weekday_is_saturday",
  "weekday_is_sunday"
)
```