ECON 573 - Final Proejct

```
library(readr)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(stringr)
library(ggplot2)
library(lubridate)
```

```
##
## Attaching package: 'lubridate'
```

```
## The following objects are masked from 'package:base':
##
##     date, intersect, setdiff, union
```

```
library(basictabler)
```

```
## Warning: package 'basictabler' was built under R version 4.2.2
```

```
library(boot)
library(caret)
```

```
## Loading required package: lattice
```

```
##
## Attaching package: 'lattice'
```

```
## The following object is masked from 'package:boot':
##
##     melanoma
```

```
library(glmnet)
```

```
## Loading required package: Matrix
```

```
## Loaded glmnet 4.1-4
```

```
library(rpart)
library(rpart.plot)
```

```
## Warning: package 'rpart.plot' was built under R version 4.2.2
```

```
library(tree)
```

```
## Warning: package 'tree' was built under R version 4.2.2
```

```
library(ipred)
library(randomForest)
```

```
## randomForest 4.7-1.1
```

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

```
##
## Attaching package: 'randomForest'
```

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

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

```
library(gbm)
```

```
## Loaded gbm 2.1.8.1
```

```
KS_Raw = read.csv("Data/kickstarter_data_full.csv", stringsAsFactors = TRUE)
```

```
# Conversion to USD for FX, Adjusting to Feb 2017 CPI
monthly_cpi = read.table("http://research.stlouisfed.org/fred2/data/CPIAUCSL.txt",
             skip = 54, header = TRUE)
monthly_cpi$DATE = as.Date(monthly_cpi$DATE)

KS_Adjusted = KS_Raw %>%
  mutate(goal_usd = goal*static_usd_rate,
         launched_at = as.Date(launched_at),
         yr_mth_launch = floor_date(launched_at, "month")) %>%
  left_join(monthly_cpi, by = c("yr_mth_launch" = "DATE")) %>%
  rename(CPI = VALUE) %>%
  mutate(goal_adj = goal_usd * max(CPI) / CPI)
```

```r
# Cleaning
KS_Clean = KS_Adjusted %>%
  select(state, goal_adj, country, category, name_len, name_len_clean, blurb_len,
         blurb_len_clean, deadline_weekday, created_at_weekday, launched_at_weekday,
         deadline_month, deadline_yr, created_at_month, created_at_yr,
         launched_at_month, launched_at_yr, create_to_launch_days,
         launch_to_deadline_days) %>%
  filter(state == "successful" | state == "failed") %>%
  mutate(success = ifelse(state == "successful", 1, 0),
         country = relevel(country, ref = "US"),
         deadline_month = as.factor(deadline_month),
         deadline_yr = as.factor(deadline_yr),
         created_at_month = as.factor(created_at_month),
         created_at_yr = as.factor(created_at_yr),
         launched_at_month = as.factor(launched_at_month),
         launched_at_yr = as.factor(launched_at_yr)) %>%
   filter(category != "Comedy", #1
          country != "LU", #2
          ) %>% # Removed for CV issues
  select(success, everything(), -state)

KS_Clean$category = as.character(KS_Clean$category)
KS_Clean$category[KS_Clean$category == ""] = "Other"
KS_Clean$category = relevel(as.factor(KS_Clean$category), ref = "Other")
KS_Clean$country = droplevels(KS_Clean$country)

write.csv(KS_Clean, "KS_Clean.csv")
```

```r
# Model Accuracy Function
CMQ = function(Model, Test_Data, p = .5) {
  pred_test = ifelse(predict(Model, Test_Data, type = "response")>p, 1, 0)
  pred_table = table(pred_test, Test_Data$success)
  pred_df = as.data.frame(pred_table)
  pred_df$class = paste0(pred_df$pred_test,":", pred_df$Var2)

  check_vec = c("0:0","0:1","1:1","1:0")
  for (i in check_vec) {
    if (!(i %in% pred_df$class)) {
      pred_df = rbind(pred_df, data.frame(pred_test = NA, Var2 = NA,
                                          class = i, Freq = 0))}}
  total_obs = sum(pred_df$Freq)
  t_neg = subset(pred_df, class == "0:0")$Freq
  t_pos = subset(pred_df, class == "1:1")$Freq
  f_neg = subset(pred_df, class == "0:1")$Freq
  f_pos = subset(pred_df, class == "1:0")$Freq

  t_neg_p = t_neg/total_obs
  t_pos_p = t_pos/total_obs
  f_neg_p = f_neg/total_obs
  f_pos_p = f_pos/total_obs

  f_class = f_neg_p + f_pos_p
  t_class = 1 - f_class
```

```
  data.frame()

  output_list = list(t_class = t_class, f_class = f_class,
                     t_neg = t_neg, t_pos = t_pos, f_neg = f_neg, f_pos = f_pos,
                     t_neg_p = t_neg_p, t_pos_p = t_pos_p,
                     f_neg_p = f_neg_p, f_pos_p = f_pos_p, total_obs = total_obs)
  return(output_list)
}
```

```
# Table Function
table_it = function(CMQ, prop = FALSE, misc = FALSE, plain = TRUE){

  tbl = BasicTable$new()
  tbl$cells$setCell(1, 1, cellType="root")
  tbl$cells$setCell(1, 2, cellType="columnHeader", rawValue="Actual TRUE")
  tbl$cells$setCell(1, 3, cellType="columnHeader", rawValue="Actual FALSE")
  tbl$cells$setCell(2, 1, cellType="rowHeader", rawValue="Predicted TRUE")
  tbl$cells$setCell(3, 1, cellType="rowHeader", rawValue="Predicted FALSE")
  tbl$cells$setCell(2, 2, cellType="cell", rawValue=CMQ$t_pos)
  tbl$cells$setCell(2, 3, cellType="cell", rawValue=CMQ$f_pos)
  tbl$cells$setCell(3, 2, cellType="cell", rawValue=CMQ$f_neg)
  tbl$cells$setCell(3, 3, cellType="cell", rawValue=CMQ$t_neg)
  if (prop == TRUE) {
    tbl$cells$setCell(2, 2, cellType="cell",
                      rawValue=paste0(round(100*CMQ$t_pos/CMQ$total_obs, 3),"%"))
    tbl$cells$setCell(2, 3, cellType="cell",
                      rawValue=paste0(round(100*CMQ$f_pos/CMQ$total_obs, 3),"%"))
    tbl$cells$setCell(3, 2, cellType="cell",
                      rawValue=paste0(round(100*CMQ$f_neg/CMQ$total_obs, 3),"%"))
    tbl$cells$setCell(3, 3, cellType="cell",
                      rawValue=paste0(round(100*CMQ$t_neg/CMQ$total_obs, 3),"%"))
  }
  if (misc == TRUE) {
    tbl$cells$setCell(4, 1, cellType="rowHeader", rawValue="Misclassification")
    tbl$cells$setCell(4, 2, cellType="cell", rawValue=paste0(round(100*CMQ$f_class, 3),"%"))
  }
  if (plain == FALSE) {
    return(tbl$renderTable())
  } else {return(tbl$print())}
}
```

```
# Sampling of Test / Train
set.seed("10302022")

Train_Ind = sample(1:nrow(KS_Clean),
                   round(.80*nrow(KS_Clean))) # 80:20 Train/Test Split
KS_Train = KS_Clean[Train_Ind,]
KS_Test = KS_Clean[-Train_Ind,]
```

```
# Logistic Regression
Logistic_Model = glm(success ~ ., family = binomial, data = KS_Train)
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
summary(Logistic_Model)
```

```
##
## Call:
## glm(formula = success ~ ., family = binomial, data = KS_Train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.1397  -0.8585  -0.4219   0.9671   5.4785
##
## Coefficients:
##                               Estimate Std. Error z value Pr(>|z|)
## (Intercept)                  6.409e-01  5.432e-01   1.180 0.238047
## goal_adj                    -7.316e-06  5.322e-07 -13.749  < 2e-16 ***
## countryAT                   -4.324e-01  4.417e-01  -0.979 0.327669
## countryAU                   -4.776e-01  1.375e-01  -3.472 0.000516 ***
## countryBE                   -1.863e+00  7.640e-01  -2.439 0.014734 *
## countryCA                   -3.399e-01  9.937e-02  -3.420 0.000626 ***
## countryCH                    8.043e-02  3.151e-01   0.255 0.798515
## countryDE                   -8.257e-02  1.608e-01  -0.513 0.607607
## countryDK                   -3.514e-01  3.117e-01  -1.127 0.259566
## countryES                   -9.434e-01  2.890e-01  -3.264 0.001098 **
## countryFR                   -1.617e-01  1.858e-01  -0.870 0.384031
## countryGB                    1.034e-01  6.339e-02   1.632 0.102738
## countryHK                    2.712e-01  5.143e-01   0.527 0.598021
## countryIE                    3.692e-01  3.162e-01   1.167 0.243051
## countryIT                   -1.425e+00  2.987e-01  -4.770 1.85e-06 ***
## countryMX                   -8.850e-01  4.763e-01  -1.858 0.063132 .
## countryNL                   -2.469e-01  1.798e-01  -1.373 0.169670
## countryNO                   -9.015e-01  4.483e-01  -2.011 0.044345 *
## countryNZ                   -9.094e-02  2.748e-01  -0.331 0.740673
## countrySE                   -7.085e-01  3.581e-01  -1.978 0.047901 *
## countrySG                    1.821e-02  5.713e-01   0.032 0.974573
## categoryAcademic            -1.529e+01  3.750e+02  -0.041 0.967478
## categoryApps                -5.768e-01  9.997e-02  -5.769 7.96e-09 ***
## categoryBlues                1.533e+01  3.420e+02   0.045 0.964251
## categoryExperimental         7.285e-01  1.559e-01   4.672 2.99e-06 ***
## categoryFestivals            6.826e-01  1.328e-01   5.138 2.77e-07 ***
## categoryFlight              -9.975e-01  1.753e-01  -5.690 1.27e-08 ***
## categoryGadgets             -4.185e-01  8.411e-02  -4.975 6.51e-07 ***
## categoryHardware            -4.438e-01  7.803e-02  -5.688 1.28e-08 ***
## categoryImmersive            5.345e-01  1.691e-01   3.161 0.001571 **
## categoryMakerspaces         -8.926e-02  1.991e-01  -0.448 0.653955
## categoryMusical              3.767e-01  1.067e-01   3.529 0.000417 ***
## categoryPlaces              -1.533e+01  1.577e+02  -0.097 0.922574
## categoryPlays                6.181e-01  9.508e-02   6.501 7.98e-11 ***
## categoryRobots               8.264e-02  1.337e-01   0.618 0.536509
## categoryShorts               1.532e+01  2.484e+02   0.062 0.950819
## categorySoftware            -1.645e+00  9.284e-02 -17.714  < 2e-16 ***
## categorySound                1.242e-01  1.292e-01   0.961 0.336652
## categorySpaces               4.841e-01  1.977e-01   2.448 0.014357 *
## categoryThrillers           -1.501e+01  3.545e+02  -0.042 0.966223
## categoryWearables           -9.290e-02  1.087e-01  -0.855 0.392663
```

```
## categoryWeb                 -2.076e+00  1.008e-01 -20.593  < 2e-16 ***
## categoryWebseries           -1.573e+01  3.318e+02  -0.047 0.962196
## name_len                     7.663e-02  2.133e-02   3.593 0.000327 ***
## name_len_clean               5.838e-02  2.511e-02   2.325 0.020050 *
## blurb_len                   -4.861e-02  7.382e-03  -6.585 4.56e-11 ***
## blurb_len_clean              5.228e-02  1.045e-02   5.004 5.61e-07 ***
## deadline_weekdayMonday       5.860e-02  8.390e-02   0.698 0.484918
## deadline_weekdaySaturday    -8.266e-02  8.031e-02  -1.029 0.303412
## deadline_weekdaySunday      -8.910e-02  7.837e-02  -1.137 0.255558
## deadline_weekdayThursday    -7.291e-02  7.716e-02  -0.945 0.344714
## deadline_weekdayTuesday      1.774e-01  8.854e-02   2.003 0.045141 *
## deadline_weekdayWednesday    3.462e-02  7.814e-02   0.443 0.657733
## created_at_weekdayMonday     3.761e-02  7.503e-02   0.501 0.616211
## created_at_weekdaySaturday  -1.465e-01  8.609e-02  -1.701 0.088883 .
## created_at_weekdaySunday    -1.578e-01  8.403e-02  -1.879 0.060300 .
## created_at_weekdayThursday  -5.107e-02  7.659e-02  -0.667 0.504889
## created_at_weekdayTuesday    3.669e-02  7.459e-02   0.492 0.622777
## created_at_weekdayWednesday  4.159e-03  7.599e-02   0.055 0.956357
## launched_at_weekdayMonday    1.966e-01  8.261e-02   2.380 0.017296 *
## launched_at_weekdaySaturday  5.918e-02  1.114e-01   0.531 0.595155
## launched_at_weekdaySunday    2.097e-01  1.081e-01   1.940 0.052352 .
## launched_at_weekdayThursday  2.061e-01  8.518e-02   2.419 0.015556 *
## launched_at_weekdayTuesday   5.124e-01  8.059e-02   6.358 2.05e-10 ***
## launched_at_weekdayWednesday 2.463e-01  8.115e-02   3.034 0.002410 **
## deadline_month2              1.810e-01  1.959e-01   0.924 0.355523
## deadline_month3              2.565e-01  2.725e-01   0.941 0.346540
## deadline_month4              2.788e-01  3.375e-01   0.826 0.408646
## deadline_month5              4.255e-01  3.970e-01   1.072 0.283773
## deadline_month6              3.295e-01  4.553e-01   0.724 0.469298
## deadline_month7              2.156e-01  5.118e-01   0.421 0.673509
## deadline_month8              1.156e-01  5.668e-01   0.204 0.838381
## deadline_month9             -1.878e-01  6.245e-01  -0.301 0.763604
## deadline_month10            -2.532e-01  6.859e-01  -0.369 0.712034
## deadline_month11            -4.959e-02  7.406e-01  -0.067 0.946618
## deadline_month12            -1.310e-01  7.991e-01  -0.164 0.869817
## deadline_yr2010             -5.639e-01  1.205e+00  -0.468 0.639705
## deadline_yr2011             -1.151e+00  2.016e+00  -0.571 0.568029
## deadline_yr2012             -1.626e+00  2.816e+00  -0.577 0.563650
## deadline_yr2013             -1.405e+00  3.629e+00  -0.387 0.698645
## deadline_yr2014             -1.326e+00  4.446e+00  -0.298 0.765477
## deadline_yr2015             -1.474e+00  5.269e+00  -0.280 0.779720
## deadline_yr2016             -1.316e+00  6.101e+00  -0.216 0.829182
## deadline_yr2017             -1.520e+00  6.938e+00  -0.219 0.826545
## created_at_month2           -6.801e-02  1.287e-01  -0.529 0.597135
## created_at_month3            1.755e-01  1.624e-01   1.081 0.279816
## created_at_month4            1.571e-01  2.090e-01   0.751 0.452429
## created_at_month5            2.243e-01  2.561e-01   0.876 0.380980
## created_at_month6            1.961e-01  3.062e-01   0.640 0.521953
## created_at_month7           -2.273e-02  3.602e-01  -0.063 0.949691
## created_at_month8            6.435e-02  4.136e-01   0.156 0.876363
## created_at_month9            2.149e-01  4.670e-01   0.460 0.645390
## created_at_month10          -7.186e-02  5.206e-01  -0.138 0.890217
## created_at_month11          -1.028e-01  5.741e-01  -0.179 0.857872
## created_at_month12          -2.026e-01  6.308e-01  -0.321 0.748066
```

```
## created_at_yr2010           -1.681e+01  7.705e+02  -0.022 0.982592
## created_at_yr2011           -1.825e+01  7.705e+02  -0.024 0.981101
## created_at_yr2012           -1.875e+01  7.705e+02  -0.024 0.980580
## created_at_yr2013           -1.887e+01  7.705e+02  -0.024 0.980466
## created_at_yr2014           -1.898e+01  7.705e+02  -0.025 0.980345
## created_at_yr2015           -1.901e+01  7.705e+02  -0.025 0.980316
## created_at_yr2016           -1.894e+01  7.705e+02  -0.025 0.980386
## created_at_yr2017           -2.027e+01  7.705e+02  -0.026 0.979015
## launched_at_month2          -2.568e-03  1.900e-01  -0.014 0.989217
## launched_at_month3          -4.749e-02  2.613e-01  -0.182 0.855775
## launched_at_month4          -3.684e-01  3.316e-01  -1.111 0.266668
## launched_at_month5          -2.052e-01  3.928e-01  -0.522 0.601371
## launched_at_month6          -1.674e-01  4.552e-01  -0.368 0.713136
## launched_at_month7          -5.366e-02  5.140e-01  -0.104 0.916846
## launched_at_month8           2.730e-01  5.751e-01   0.475 0.635003
## launched_at_month9           2.695e-01  6.378e-01   0.423 0.672617
## launched_at_month10          3.194e-01  6.960e-01   0.459 0.646239
## launched_at_month11          3.958e-01  7.561e-01   0.523 0.600638
## launched_at_month12          1.058e-01  8.115e-01   0.130 0.896291
## launched_at_yr2010           1.676e+01  7.705e+02   0.022 0.982649
## launched_at_yr2011           1.918e+01  7.705e+02   0.025 0.980141
## launched_at_yr2012           2.011e+01  7.705e+02   0.026 0.979179
## launched_at_yr2013           1.989e+01  7.705e+02   0.026 0.979401
## launched_at_yr2014           1.912e+01  7.705e+02   0.025 0.980203
## launched_at_yr2015           1.931e+01  7.705e+02   0.025 0.980002
## launched_at_yr2016           1.923e+01  7.705e+02   0.025 0.980086
## launched_at_yr2017           2.001e+01  7.705e+02   0.026 0.979285
## create_to_launch_days        2.701e-05  1.869e-03   0.014 0.988471
## launch_to_deadline_days     -1.328e-02  2.998e-03  -4.429 9.48e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 17976  on 13944  degrees of freedom
## Residual deviance: 14413  on 13821  degrees of freedom
## AIC: 14661
##
## Number of Fisher Scoring iterations: 14
```

```
# Logistic Regression Performance
CMQ(Logistic_Model, KS_Train)$f_class # Train Misclassification
```

```
## [1] 0.270061
```

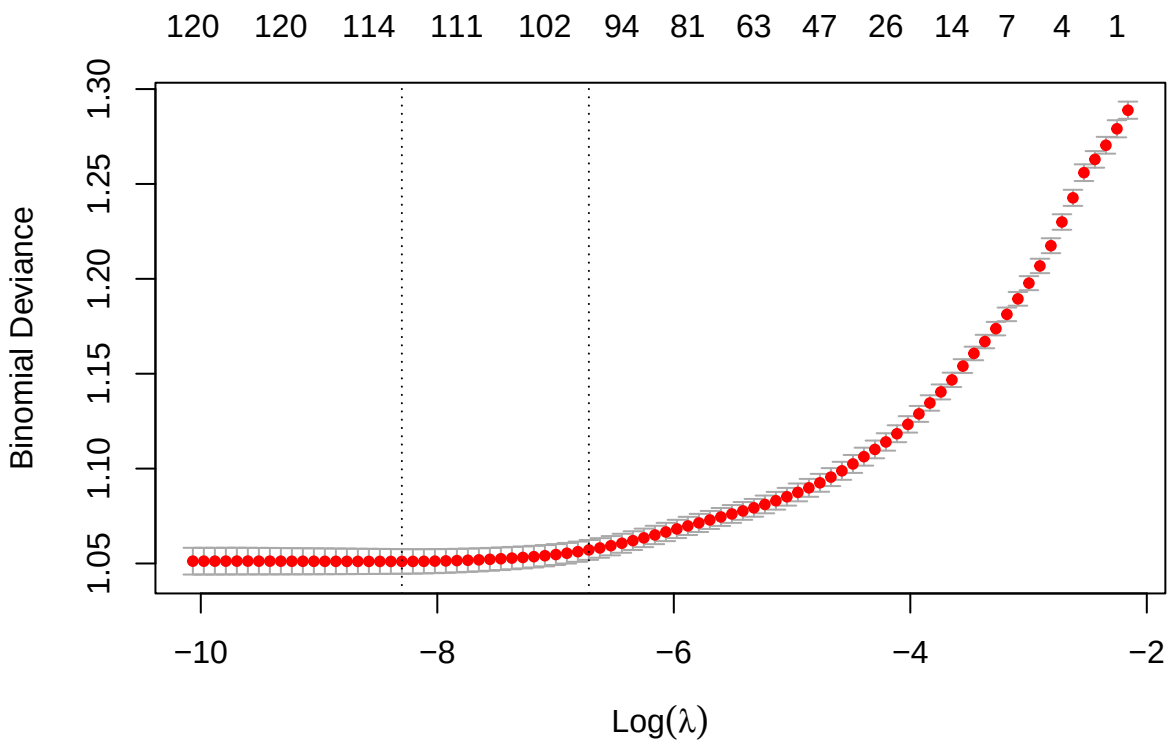
```
Logistic_Misc = CMQ(Logistic_Model, KS_Test)$f_class # Test Misclassification
Logistic_Misc
```

```
## [1] 0.288296
```

```
# Lasso Regression
x_Train = model.matrix(success~., KS_Train)[,-1]
```

```r
y_Train = KS_Train$success

set.seed("10302022")
CV_Lasso = cv.glmnet(x_Train, y_Train, alpha = 1, family = "binomial", nfolds = 10)
plot(CV_Lasso)
```




```r
# Lasso Min
Lasso_Model_min = glmnet(x_Train, y_Train, alpha = 1, family = "binomial",
                         lambda = CV_Lasso$lambda.min)

Lasso_Min_Coef = coef(Lasso_Model_min)
```

```r
# Lasso 1se
Lasso_Model_1se = glmnet(x_Train, y_Train, alpha = 1, family = "binomial",
                         lambda = CV_Lasso$lambda.1se)

Lasso_1se_Coef = coef(Lasso_Model_1se)
```

```r
# Comparing Lasso min and Lasso 1se
Lasso_Comparison = data.frame(Vars = Lasso_Min_Coef@Dimnames[[1]],
                              lambda_min_coef = as.numeric(as.character(Lasso_Min_Coef))!=0,
                              lambda_1se_coef = as.numeric(as.character(Lasso_1se_Coef))!=0)

sum(Lasso_Comparison$lambda_min_coef)
```

```
## [1] 113
```

```
sum(Lasso_Comparison$lambda_1se_coef)
```

```
## [1] 99
```

```
Lasso_Diff = Lasso_Comparison[which(
  Lasso_Comparison$lambda_min_coef != Lasso_Comparison$lambda_1se_coef),]
Lasso_Diff$Vars
```

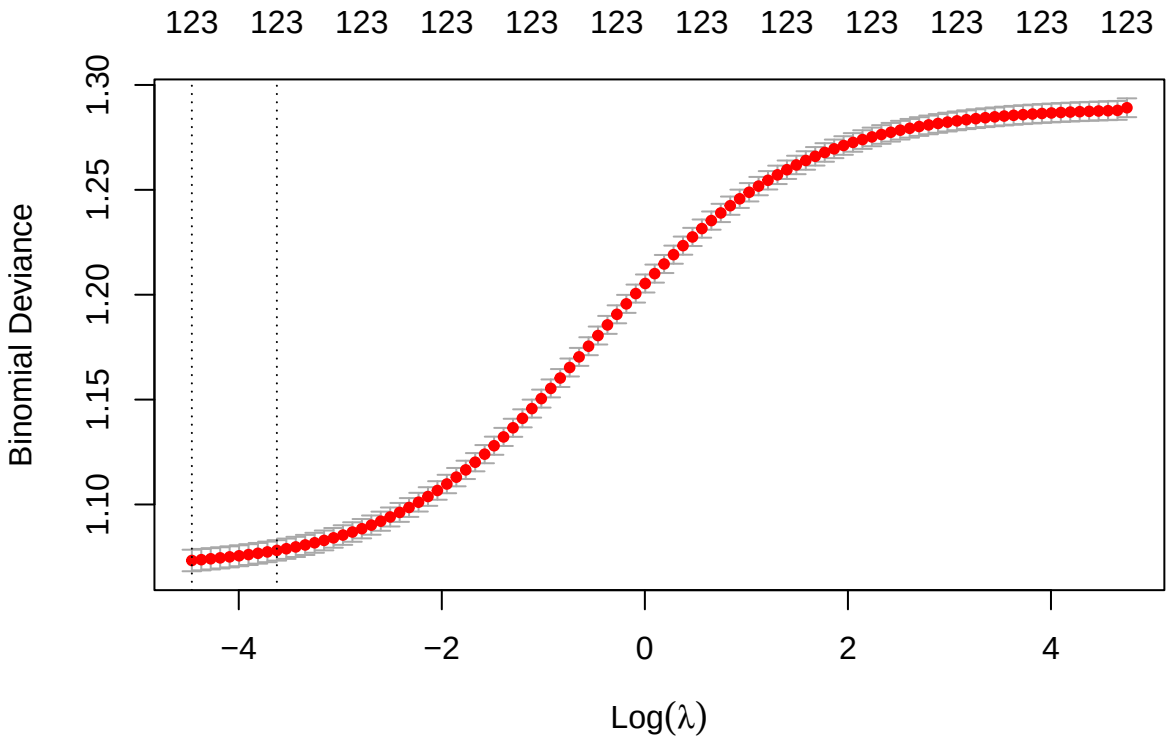
```
##  [1] "countryCH"                   "countryNZ"
##  [3] "categoryMakerspaces"         "created_at_weekdayWednesday"
##  [5] "launched_at_weekdaySaturday" "deadline_month2"
##  [7] "deadline_month4"             "deadline_month6"
##  [9] "deadline_month7"             "deadline_month11"
## [11] "deadline_month12"            "deadline_yr2012"
## [13] "created_at_month7"           "created_at_month10"
## [15] "created_at_yr2010"           "created_at_yr2011"
## [17] "created_at_yr2013"           "launched_at_month5"
## [19] "launched_at_month9"          "launched_at_month10"
## [21] "launched_at_yr2015"          "launched_at_yr2016"
```

```
# Lasso Model Performance
x_Test = model.matrix(success~., KS_Test)[,-1]

Lasso_Pred = ifelse(predict(Lasso_Model_min, x_Test, type = "response")>.5, 1, 0)
Lasso_Table = table(Lasso_Pred, KS_Test$success)
Lasso_Table_Prop = prop.table(Lasso_Table)
Lasso_Misc = 1-sum(diag(Lasso_Table_Prop))
Lasso_Misc
```

```
## [1] 0.2900172
```

```
# Ridge Regression
set.seed("10302022")
CV_Ridge = cv.glmnet(x_Train, y_Train, alpha = 0, family = "binomial", nfolds = 10)
plot(CV_Ridge)
```




```
# Ridge Min
Ridge_Model_min = glmnet(x_Train, y_Train, alpha = 1, family = "binomial",
                         lambda = CV_Ridge$lambda.min)
```

```
# Ridge 1se
Ridge_Model_1se = glmnet(x_Train, y_Train, alpha = 1, family = "binomial",
                         lambda = CV_Ridge$lambda.1se)
```

```
# Ridge Model Performance
Ridge_Pred = ifelse(predict(Ridge_Model_min, x_Test, type = "response")>.5, 1, 0)
Ridge_Table = table(Ridge_Pred, KS_Test$success)
Ridge_Table_Prop = prop.table(Ridge_Table)
Ridge_Misc = 1-sum(diag(Ridge_Table_Prop))
Ridge_Misc
```

```
## [1] 0.3141136
```

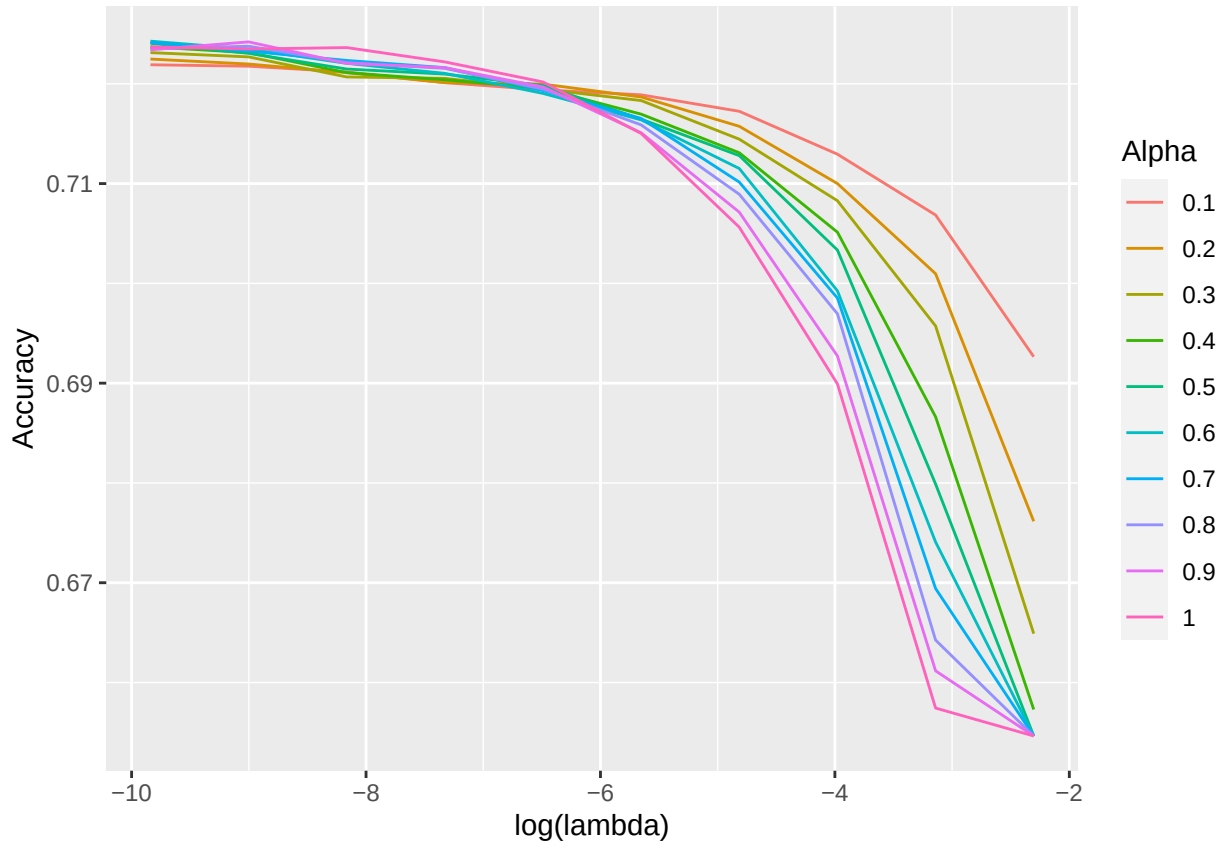
```
# Elastic Net Model
set.seed("10302022")
CV = trainControl(method = "cv", number = 10)
CV_ENet = train(as.factor(success) ~ ., data = KS_Train,
                method = "glmnet", trControl = CV,
                tuneLength = 10)
CV_ENet
```

```
## glmnet
##
## 13945 samples
##    18 predictor
##     2 classes: '0', '1'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 12551, 12551, 12550, 12551, 12551, 12551, ...
## Resampling results across tuning parameters:
##
##   alpha  lambda        Accuracy   Kappa
##   0.1    5.331286e-05  0.7219069  0.35449428
##   0.1    1.231596e-04  0.7217634  0.35408991
##   0.1    2.845147e-04  0.7211896  0.35088990
##   0.1    6.572658e-04  0.7201136  0.34715413
##   0.1    1.518369e-03  0.7193961  0.34331379
##   0.1    3.507630e-03  0.7188944  0.33966763
##   0.1    8.103080e-03  0.7172456  0.33164728
##   0.1    1.871917e-02  0.7129435  0.31138752
##   0.1    4.324370e-02  0.7068484  0.27514180
##   0.1    9.989856e-02  0.6926491  0.19400702
##   0.2    5.331286e-05  0.7224807  0.35662552
##   0.2    1.231596e-04  0.7219787  0.35455654
##   0.2    2.845147e-04  0.7211180  0.35081930
##   0.2    6.572658e-04  0.7201135  0.34714500
##   0.2    1.518369e-03  0.7199697  0.34457494
##   0.2    3.507630e-03  0.7186793  0.33877918
##   0.2    8.103080e-03  0.7157399  0.32554023
##   0.2    1.871917e-02  0.7100034  0.29838917
##   0.2    4.324370e-02  0.7009677  0.24425902
##   0.2    9.989856e-02  0.6761569  0.11239808
##   0.3    5.331286e-05  0.7231262  0.35841097
##   0.3    1.231596e-04  0.7226956  0.35648666
##   0.3    2.845147e-04  0.7206878  0.34995357
##   0.3    6.572658e-04  0.7205435  0.34794153
##   0.3    1.518369e-03  0.7194679  0.34323190
##   0.3    3.507630e-03  0.7183206  0.33714470
##   0.3    8.103080e-03  0.7144485  0.32017497
##   0.3    1.871917e-02  0.7082822  0.28698345
##   0.3    4.324370e-02  0.6957328  0.21535107
##   0.3    9.989856e-02  0.6648982  0.05225420
##   0.4    5.331286e-05  0.7236999  0.35993406
##   0.4    1.231596e-04  0.7231258  0.35753795
##   0.4    2.845147e-04  0.7211179  0.35129010
##   0.4    6.572658e-04  0.7204003  0.34755401
##   0.4    1.518369e-03  0.7195396  0.34303238
##   0.4    3.507630e-03  0.7169581  0.33326189
##   0.4    8.103080e-03  0.7130862  0.31391791
##   0.4    1.871917e-02  0.7051271  0.27187514
##   0.4    4.324370e-02  0.6866261  0.17450817
##   0.4    9.989856e-02  0.6572967  0.01281396
##   0.5    5.331286e-05  0.7240586  0.36066678
##   0.5    1.231596e-04  0.7230542  0.35754617
```

```
##   0.5    2.845147e-04  0.7214764  0.35242458
##   0.5    6.572658e-04  0.7209740  0.34937823
##   0.5    1.518369e-03  0.7198268  0.34371290
##   0.5    3.507630e-03  0.7164561  0.33105187
##   0.5    8.103080e-03  0.7127994  0.31088950
##   0.5    1.871917e-02  0.7033341  0.26177795
##   0.5    4.324370e-02  0.6798860  0.13810824
##   0.5    9.989856e-02  0.6546433  0.00000000
##   0.6    5.331286e-05  0.7242736  0.36138820
##   0.6    1.231596e-04  0.7234845  0.35873940
##   0.6    2.845147e-04  0.7220501  0.35402389
##   0.6    6.572658e-04  0.7210459  0.34950451
##   0.6    1.518369e-03  0.7190379  0.34176047
##   0.6    3.507630e-03  0.7163845  0.33015726
##   0.6    8.103080e-03  0.7115083  0.30500660
##   0.6    1.871917e-02  0.6992462  0.24387524
##   0.6    4.324370e-02  0.6740773  0.10501665
##   0.6    9.989856e-02  0.6546433  0.00000000
##   0.7    5.331286e-05  0.7241302  0.36132145
##   0.7    1.231596e-04  0.7232693  0.35866614
##   0.7    2.845147e-04  0.7223371  0.35528797
##   0.7    6.572658e-04  0.7216197  0.35097678
##   0.7    1.518369e-03  0.7193247  0.34188483
##   0.7    3.507630e-03  0.7165279  0.32958480
##   0.7    8.103080e-03  0.7101460  0.29923804
##   0.7    1.871917e-02  0.6985284  0.23466913
##   0.7    4.324370e-02  0.6694159  0.07883610
##   0.7    9.989856e-02  0.6546433  0.00000000
##   0.8    5.331286e-05  0.7234848  0.36005586
##   0.8    1.231596e-04  0.7237714  0.36011164
##   0.8    2.845147e-04  0.7221219  0.35468208
##   0.8    6.572658e-04  0.7215478  0.35110037
##   0.8    1.518369e-03  0.7194679  0.34223552
##   0.8    3.507630e-03  0.7158825  0.32679355
##   0.8    8.103080e-03  0.7089274  0.29329190
##   0.8    1.871917e-02  0.6969516  0.22360703
##   0.8    4.324370e-02  0.6642521  0.05225637
##   0.8    9.989856e-02  0.6546433  0.00000000
##   0.9    5.331286e-05  0.7234132  0.36037567
##   0.9    1.231596e-04  0.7242018  0.36125382
##   0.9    2.845147e-04  0.7220502  0.35481119
##   0.9    6.572658e-04  0.7216197  0.35142334
##   0.9    1.518369e-03  0.7196829  0.34262822
##   0.9    3.507630e-03  0.7150937  0.32368031
##   0.9    8.103080e-03  0.7071347  0.28607130
##   0.9    1.871917e-02  0.6927212  0.20548934
##   0.9    4.324370e-02  0.6611691  0.03505549
##   0.9    9.989856e-02  0.6546433  0.00000000
##   1.0    5.331286e-05  0.7237001  0.36156751
##   1.0    1.231596e-04  0.7234849  0.36043369
##   1.0    2.845147e-04  0.7236284  0.35966584
##   1.0    6.572658e-04  0.7221937  0.35415531
##   1.0    1.518369e-03  0.7201850  0.34443813
##   1.0    3.507630e-03  0.7150221  0.32252761
```

```
##   1.0    8.103080e-03  0.7056286  0.27875719
##   1.0    1.871917e-02  0.6899248  0.19076200
##   1.0    4.324370e-02  0.6574403  0.01461201
##   1.0    9.989856e-02  0.6546433  0.00000000
##
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were alpha = 0.6 and lambda = 5.331286e-05.
```

```
# Elastic Net Visualization
CV_ENet_Results = CV_ENet$results
ggplot(data = CV_ENet_Results,
       aes(x = log(lambda), y = Accuracy,
           group = alpha, color = as.factor(alpha))) +
  geom_line() + labs(color = "Alpha")
```




```
# Elastic Net Model
CV_ENet$bestTune$alpha
```

```
## [1] 0.6
```

```
CV_ENet$bestTune$lambda
```

```
## [1] 5.331286e-05
```

```
ENet_Model_min = glmnet(x_Train, y_Train, alpha = CV_ENet$bestTune$alpha,
                        family = "binomial", lambda = CV_ENet$bestTune$lambda)

# Elastic Net Performance
ENet_Pred = ifelse(predict(ENet_Model_min, x_Test, type = "response")>.5, 1, 0)
ENet_Table = table(ENet_Pred, KS_Test$success)
ENet_Table_Prop = prop.table(ENet_Table)
ENet_Misc = 1-sum(diag(ENet_Table_Prop))
ENet_Misc
```

```
## [1] 0.2880092
```

```
# Decision Tree
set.seed("10302022")
Basic_Tree = rpart(success ~ ., data = KS_Train, method = "class",
                   control = rpart.control(cp = 0))

Tree_Min_Error_Ind = which.min(Basic_Tree$cptable[,4])
Tree_Min_Error_Cp = Basic_Tree$cptable[Tree_Min_Error_Ind,1]
```

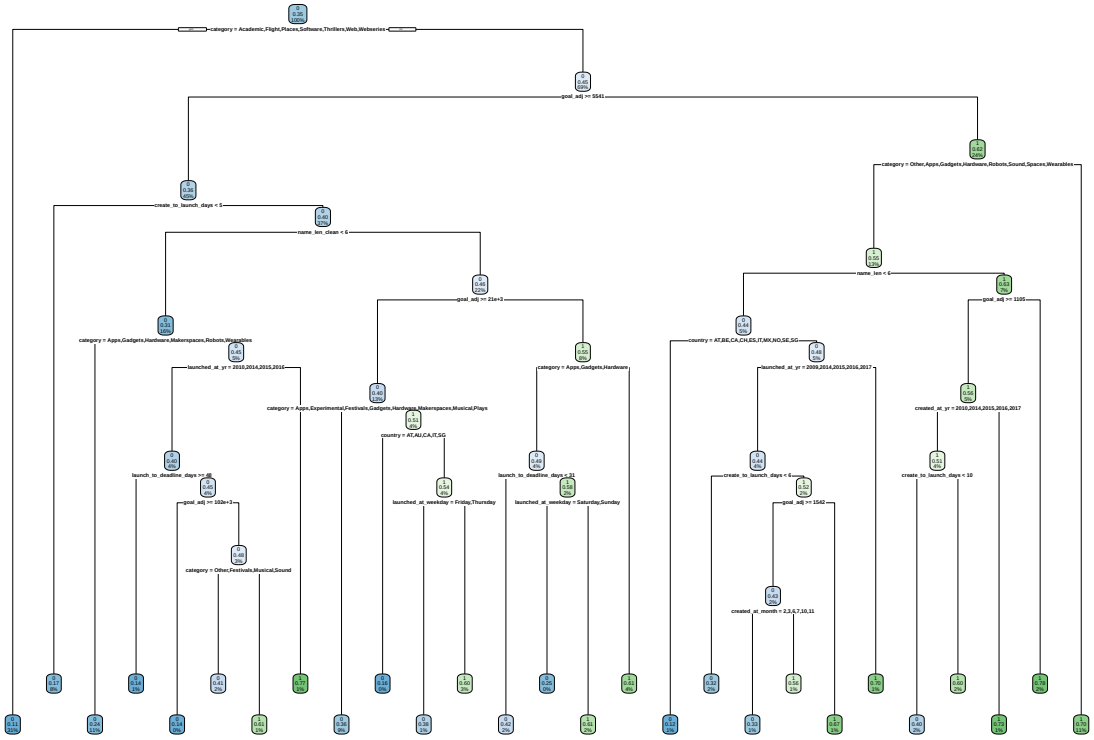
```
# Basic Tree Performance
Tree_Pred = predict(Basic_Tree, KS_Test, type = "class")

Tree_Table = table(Tree_Pred, KS_Test$success)
Tree_Table_Prop = prop.table(Tree_Table)
Tree_Misc = 1-sum(diag(Tree_Table_Prop))
Tree_Misc
```

```
## [1] 0.3218589
```

```
# Pruning
Pruned_Tree = prune(Basic_Tree, cp = Tree_Min_Error_Cp)
rpart.plot(Pruned_Tree)
```

```
## Warning: labs do not fit even at cex 0.15, there may be some overplotting
```



```
# Pruned Decision Tree Performance
Pruned_Tree_Pred = predict(Pruned_Tree, KS_Test, type = "class")

Pruned_Tree_Table = table(Pruned_Tree_Pred, KS_Test$success)
Pruned_Tree_Table_Prop = prop.table(Pruned_Tree_Table)
Pruned_Tree_Misc = 1-sum(diag(Pruned_Tree_Table_Prop))
Pruned_Tree_Misc
```

```
## [1] 0.275961
```

```
# Bagging
set.seed("10302022")
Bag_Model = bagging(factor(success) ~ ., data = KS_Train,
  nbagg = 100,
  coob = TRUE,
  control = rpart.control(minsplit = 2, cp = 0))

Bag_Model
```

```
##
## Bagging classification trees with 100 bootstrap replications
##
## Call: bagging.data.frame(formula = factor(success) ~ ., data = KS_Train,
##     nbagg = 100, coob = TRUE, control = rpart.control(minsplit = 2,
##         cp = 0))
```

```
##
## Out-of-bag estimate of misclassification error:  0.2654
```

```
# Bagged Model Variable Importance
VI = varImp(Bag_Model)
VI$var = rownames(VI)
VI_Order = order(VI$Overall, decreasing = TRUE)
VI = VI[VI_Order,]
write.csv(VI, "BaggedVI.csv")
```

```
# Bagged Model Performance
Bag_Pred = predict(Bag_Model, KS_Test, type = "class")

Bag_Table = table(Bag_Pred, KS_Test$success)
Bag_Table_Prop = prop.table(Bag_Table)
Bag_Misc = 1-sum(diag(Bag_Table_Prop))
Bag_Misc
```

```
## [1] 0.2644865
```

```
# Random Forest Model
RF_Model = randomForest(factor(success) ~ ., data = KS_Train,
                        ntree = 1000)
RF_Model
```

```
##
## Call:
##  randomForest(formula = factor(success) ~ ., data = KS_Train,      ntree = 1000)
##                Type of random forest: classification
##                      Number of trees: 1000
## No. of variables tried at each split: 4
##
##         OOB estimate of  error rate: 25.31%
## Confusion matrix:
##      0    1 class.error
## 0 7441 1688   0.1849052
## 1 1842 2974   0.3824751
```

```
# Bagged Model Variable Importance
VI_RF = varImp(RF_Model)
VI_RF$var = rownames(VI_RF)
VI_RF_Order = order(VI_RF$Overall, decreasing = TRUE)
VI_RF = VI[VI_RF_Order,]
write.csv(VI, "RFVI.csv")
```

```
# Random Forest Performance
RF_Pred = predict(RF_Model, KS_Test, type = "class")

RF_Table = table(RF_Pred, KS_Test$success)
RF_Table_Prop = prop.table(RF_Table)
RF_Misc = 1-sum(diag(RF_Table_Prop))
RF_Misc
```

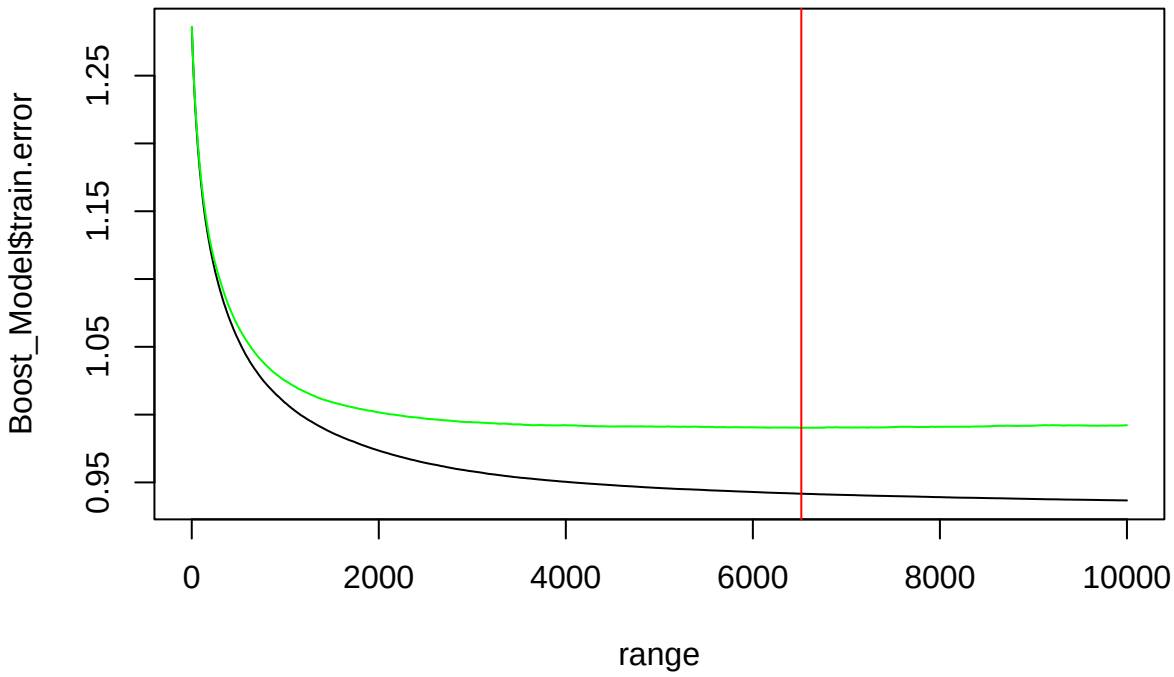
```
## [1] 0.2581756
```

```
# Boosting
set.seed("10302022")
N_Trees = 10000
Boost_Model = gbm(success ~.,
                  data = KS_Train,
                  distribution = "bernoulli",
                  cv.folds = 10,
                  shrinkage = .01,
                  train.fraction = 0.5,
                  n.trees = N_Trees)
Optimal_Trees = which.min(Boost_Model$cv.error)
```

```
# Boosting CV
range = 1:N_Trees
plot(range, Boost_Model$train.error, type = "l")
lines(range, Boost_Model$cv.error, col = "green")
which.min(Boost_Model$train.error)
```

```
## [1] 10000
```

```
abline(v = which.min(Boost_Model$cv.error), col = "red")
```



```
# Boost Performance
Boost_Pred = ifelse(predict(Boost_Model,
                            KS_Test, type = "response", newmfinal = 15)>.5, 1, 0)
```

```
## Using 5103 trees...
```

```
Boost_Table = table(Boost_Pred, KS_Test$success)
Boost_Table_Prop = prop.table(Boost_Table)
Boost_Misc = 1-sum(diag(Boost_Table_Prop))
Boost_Misc
```

```
## [1] 0.2751004
```

```
# Results Table
Results = data.frame(
  Method = c("Logisitic Regression",
             "Lasso Regression",
             "Ridge Regression",
             "Elastic Net",
             "Decision Tree",
             "Bagging",
             "Random Forest",
             "Boosting"),
  Misclass_rate = c(Logistic_Misc,
                    Lasso_Misc,
                    Ridge_Misc,
                    ENet_Misc,
                    Pruned_Tree_Misc,
                    Bag_Misc,
                    RF_Misc,
                    Boost_Misc)
)

Results
```

```
##                 Method Misclass_rate
## 1 Logisitic Regression     0.2882960
## 2     Lasso Regression     0.2900172
## 3     Ridge Regression     0.3141136
## 4          Elastic Net     0.2880092
## 5        Decision Tree     0.2759610
## 6              Bagging     0.2644865
## 7        Random Forest     0.2581756
## 8             Boosting     0.2751004
```

```
write.csv(Results, "Results.csv")
```