ECON 573 - Final Proejct

```
library(readr)
library(stringr)
library(ggplot2)
```

```
## Warning: package 'ggplot2' was built under R version 4.2.2
```

```
library(lubridate)
```

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

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

```
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(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
```

```
library(gbm)
```

```
## Loaded gbm 2.1.8.1
```

```
library(e1071)
library(MASS)
library(Rfast)
```

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

```
## Loading required package: Rcpp
```

```
## Loading required package: RcppZiggurat
```

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

```
library(dplyr)
```

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

```
## The following object is masked from 'package:Rfast':
##
##     nth
```

```
## The following object is masked from 'package:MASS':
##
##     select
```

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

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

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

```
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,
         pledged_usd = pledged*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,
         pledged_adj = pledged * max(CPI) / CPI)
```

```
# 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, backers_count, pledged_adj) %>%
  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),
         avg_pledge = ifelse(backers_count == 0, 0, pledged_adj/backers_count)) %>%
   filter(category != "Comedy", #1
          country != "LU", #2
          ) %>% # Removed for CV issues
  select(success, everything(), -state, -backers_count, -pledged_adj)

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)
}
```

```r
# Sampling of Test / Train
seed = 11012022
set.seed(seed)

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,]
```

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

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

```r
summary(Logistic_Model)
```

```
##
```

```
## Call:
## glm(formula = success ~ ., family = binomial, data = KS_Train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -6.4252  -0.8348  -0.4003   0.9364   4.3537
##
## Coefficients:
##                              Estimate Std. Error z value Pr(>|z|)
## (Intercept)                 7.170e-01  5.378e-01   1.333 0.182470
## goal_adj                   -1.063e-05  6.255e-07 -16.993  < 2e-16 ***
## countryAT                  -1.252e+00  5.344e-01  -2.343 0.019127 *
## countryAU                  -5.449e-01  1.390e-01  -3.920 8.85e-05 ***
## countryBE                  -1.696e+00  8.015e-01  -2.116 0.034316 *
## countryCA                  -3.655e-01  9.887e-02  -3.696 0.000219 ***
## countryCH                  -9.192e-02  3.431e-01  -0.268 0.788756
## countryDE                  -8.331e-02  1.618e-01  -0.515 0.606672
## countryDK                  -1.945e+00  4.319e-01  -4.504 6.66e-06 ***
## countryES                  -8.180e-01  2.880e-01  -2.840 0.004516 **
## countryFR                  -3.760e-01  1.957e-01  -1.922 0.054640 .
## countryGB                   1.952e-01  6.392e-02   3.053 0.002263 **
## countryHK                  -1.952e+00  6.897e-01  -2.830 0.004649 **
## countryIE                   1.900e-01  3.245e-01   0.586 0.558148
## countryIT                  -1.101e+00  2.916e-01  -3.774 0.000161 ***
## countryMX                  -3.206e+00  7.181e-01  -4.464 8.03e-06 ***
## countryNL                  -1.305e-01  1.791e-01  -0.729 0.466112
## countryNO                  -1.967e+00  5.343e-01  -3.681 0.000232 ***
## countryNZ                   5.062e-02  2.980e-01   0.170 0.865116
## countrySE                  -1.904e+00  4.019e-01  -4.736 2.17e-06 ***
## countrySG                  -5.704e-01  6.517e-01  -0.875 0.381442
## categoryAcademic           -1.511e+01  3.674e+02  -0.041 0.967197
## categoryApps               -5.806e-01  1.014e-01  -5.726 1.03e-08 ***
## categoryBlues               1.527e+01  3.500e+02   0.044 0.965210
## categoryExperimental        5.819e-01  1.558e-01   3.735 0.000188 ***
## categoryFestivals           7.582e-01  1.357e-01   5.588 2.29e-08 ***
## categoryFlight             -1.184e+00  1.854e-01  -6.386 1.70e-10 ***
## categoryGadgets            -4.152e-01  8.575e-02  -4.842 1.28e-06 ***
## categoryHardware           -5.085e-01  7.996e-02  -6.360 2.02e-10 ***
## categoryImmersive           5.947e-01  1.752e-01   3.394 0.000688 ***
## categoryMakerspaces        -2.542e-01  2.009e-01  -1.265 0.205746
## categoryMusical             3.517e-01  1.089e-01   3.231 0.001233 **
## categoryPlaces             -1.529e+01  1.582e+02  -0.097 0.923014
## categoryPlays               6.073e-01  9.648e-02   6.294 3.09e-10 ***
## categoryRobots              3.338e-02  1.374e-01   0.243 0.807993
## categoryShorts              1.518e+01  2.389e+02   0.064 0.949351
## categorySoftware           -1.561e+00  9.363e-02 -16.676  < 2e-16 ***
## categorySound              -1.009e-02  1.329e-01  -0.076 0.939489
## categorySpaces              3.761e-01  2.045e-01   1.839 0.065909 .
## categoryThrillers          -1.509e+01  3.320e+02  -0.045 0.963750
## categoryWearables          -1.416e-01  1.099e-01  -1.288 0.197749
## categoryWeb                -2.009e+00  1.027e-01 -19.561  < 2e-16 ***
## categoryWebseries          -1.562e+01  3.242e+02  -0.048 0.961581
## name_len                    6.869e-02  2.180e-02   3.151 0.001628 **
## name_len_clean              5.451e-02  2.561e-02   2.128 0.033300 *
```

```
## blurb_len                    -3.938e-02  7.462e-03  -5.278 1.31e-07 ***
## blurb_len_clean               3.953e-02  1.062e-02   3.722 0.000198 ***
## deadline_weekdayMonday        1.234e-01  8.484e-02   1.455 0.145711
## deadline_weekdaySaturday     -2.827e-04  8.089e-02  -0.003 0.997211
## deadline_weekdaySunday       -1.146e-01  7.983e-02  -1.436 0.151105
## deadline_weekdayThursday     -6.162e-02  7.818e-02  -0.788 0.430607
## deadline_weekdayTuesday       2.120e-01  8.938e-02   2.372 0.017680 *
## deadline_weekdayWednesday     6.098e-02  7.917e-02   0.770 0.441167
## created_at_weekdayMonday      1.452e-01  7.638e-02   1.902 0.057234 .
## created_at_weekdaySaturday   -7.046e-02  8.747e-02  -0.806 0.420494
## created_at_weekdaySunday     -1.923e-02  8.582e-02  -0.224 0.822743
## created_at_weekdayThursday    6.312e-02  7.831e-02   0.806 0.420241
## created_at_weekdayTuesday     1.168e-01  7.612e-02   1.534 0.125005
## created_at_weekdayWednesday   6.425e-02  7.760e-02   0.828 0.407679
## launched_at_weekdayMonday     1.445e-01  8.452e-02   1.710 0.087284 .
## launched_at_weekdaySaturday  -7.248e-02  1.129e-01  -0.642 0.520815
## launched_at_weekdaySunday     1.737e-01  1.110e-01   1.565 0.117623
## launched_at_weekdayThursday   1.700e-01  8.660e-02   1.963 0.049612 *
## launched_at_weekdayTuesday    5.010e-01  8.252e-02   6.071 1.27e-09 ***
## launched_at_weekdayWednesday  1.887e-01  8.260e-02   2.285 0.022333 *
## deadline_month2               3.896e-01  2.011e-01   1.937 0.052759 .
## deadline_month3               3.535e-01  2.772e-01   1.275 0.202201
## deadline_month4               3.275e-01  3.447e-01   0.950 0.342064
## deadline_month5               3.920e-01  4.053e-01   0.967 0.333430
## deadline_month6               4.418e-01  4.625e-01   0.955 0.339443
## deadline_month7               2.512e-01  5.195e-01   0.484 0.628668
## deadline_month8               2.692e-01  5.758e-01   0.468 0.640110
## deadline_month9               7.108e-02  6.333e-01   0.112 0.910640
## deadline_month10              1.535e-01  6.945e-01   0.221 0.825033
## deadline_month11              4.531e-01  7.525e-01   0.602 0.547085
## deadline_month12              5.378e-01  8.106e-01   0.663 0.507059
## deadline_yr2010               2.491e-01  1.183e+00   0.211 0.833168
## deadline_yr2011               9.856e-02  2.009e+00   0.049 0.960878
## deadline_yr2012               4.163e-01  2.831e+00   0.147 0.883107
## deadline_yr2013               8.474e-01  3.659e+00   0.232 0.816835
## deadline_yr2014               1.781e+00  4.490e+00   0.397 0.691676
## deadline_yr2015               2.092e+00  5.325e+00   0.393 0.694491
## deadline_yr2016               2.896e+00  6.170e+00   0.469 0.638785
## deadline_yr2017               3.382e+00  7.018e+00   0.482 0.629840
## created_at_month2            -1.920e-01  1.289e-01  -1.489 0.136525
## created_at_month3            -3.465e-03  1.637e-01  -0.021 0.983107
## created_at_month4            -5.888e-02  2.106e-01  -0.280 0.779782
## created_at_month5            -7.591e-03  2.576e-01  -0.029 0.976487
## created_at_month6            -1.205e-01  3.086e-01  -0.390 0.696194
## created_at_month7            -3.781e-01  3.625e-01  -1.043 0.296822
## created_at_month8            -2.948e-01  4.172e-01  -0.707 0.479741
## created_at_month9            -3.570e-01  4.715e-01  -0.757 0.449007
## created_at_month10           -5.918e-01  5.262e-01  -1.125 0.260731
## created_at_month11           -6.928e-01  5.799e-01  -1.195 0.232238
## created_at_month12           -7.736e-01  6.368e-01  -1.215 0.224442
## created_at_yr2010            -1.668e+01  1.455e+03  -0.011 0.990856
## created_at_yr2011            -1.840e+01  1.455e+03  -0.013 0.989915
## created_at_yr2012            -2.003e+01  1.455e+03  -0.014 0.989017
## created_at_yr2013            -2.093e+01  1.455e+03  -0.014 0.988527
```

```
## created_at_yr2014           -2.177e+01  1.455e+03  -0.015 0.988065
## created_at_yr2015           -2.251e+01  1.455e+03  -0.015 0.987661
## created_at_yr2016           -2.310e+01  1.455e+03  -0.016 0.987337
## created_at_yr2017           -2.420e+01  1.455e+03  -0.017 0.986735
## launched_at_month2           2.151e-01  1.901e-01   1.131 0.257915
## launched_at_month3           2.067e-01  2.662e-01   0.776 0.437554
## launched_at_month4           1.780e-01  3.364e-01   0.529 0.596695
## launched_at_month5           1.221e-01  3.974e-01   0.307 0.758702
## launched_at_month6           2.222e-01  4.618e-01   0.481 0.630374
## launched_at_month7           3.783e-01  5.211e-01   0.726 0.467862
## launched_at_month8           6.404e-01  5.820e-01   1.100 0.271163
## launched_at_month9           5.187e-01  6.446e-01   0.805 0.420962
## launched_at_month10          4.571e-01  7.050e-01   0.648 0.516713
## launched_at_month11          5.103e-01  7.658e-01   0.666 0.505230
## launched_at_month12          2.258e-01  8.208e-01   0.275 0.783289
## launched_at_yr2010           1.568e+01  1.455e+03   0.011 0.991407
## launched_at_yr2011           1.758e+01  1.455e+03   0.012 0.990363
## launched_at_yr2012           1.914e+01  1.455e+03   0.013 0.989509
## launched_at_yr2013           1.938e+01  1.455e+03   0.013 0.989374
## launched_at_yr2014           1.850e+01  1.455e+03   0.013 0.989859
## launched_at_yr2015           1.889e+01  1.455e+03   0.013 0.989646
## launched_at_yr2016           1.880e+01  1.455e+03   0.013 0.989695
## launched_at_yr2017           1.915e+01  1.455e+03   0.013 0.989503
## create_to_launch_days       -1.830e-03  1.887e-03  -0.970 0.332232
## launch_to_deadline_days     -1.550e-02  3.057e-03  -5.072 3.93e-07 ***
## avg_pledge                   2.522e-03  1.495e-04  16.869  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 18011  on 13944  degrees of freedom
## Residual deviance: 14073  on 13820  degrees of freedom
## AIC: 14323
##
## Number of Fisher Scoring iterations: 14
```

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

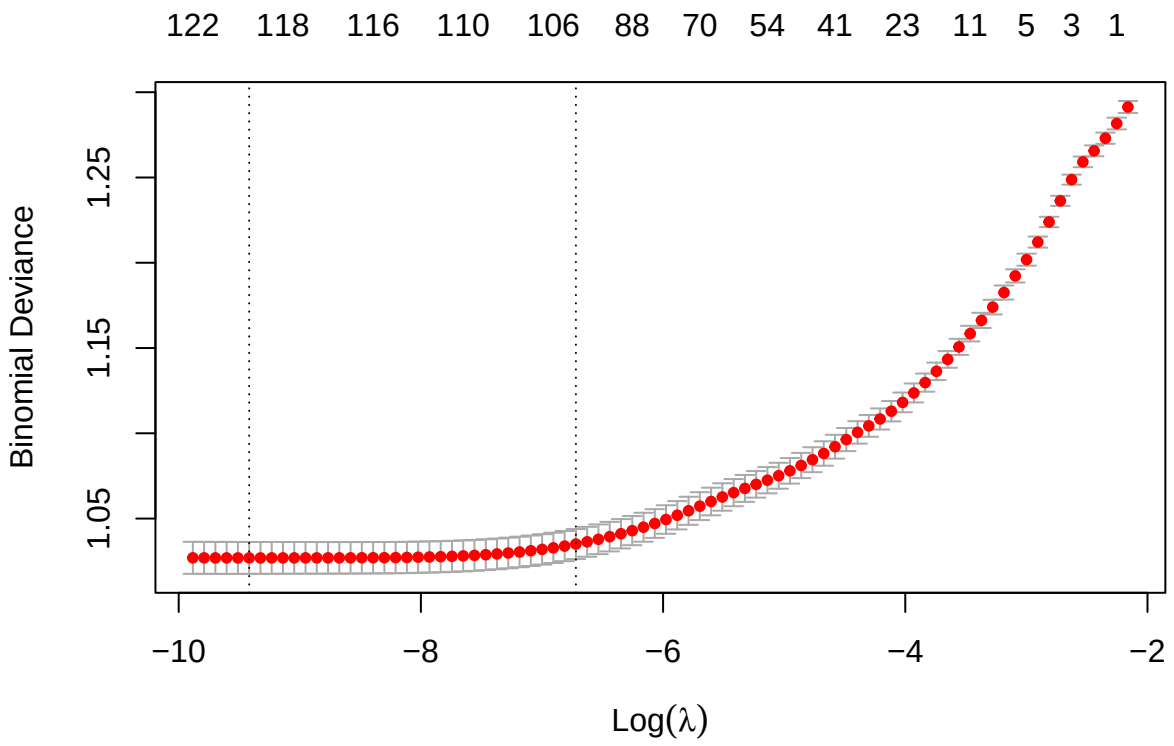
```
## [1] 0.2512729
```

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

```
## [1] 0.2598967
```

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

set.seed(seed)
```

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




```
# 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)
```

```
# 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)
```

```
# 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] 124
```

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

```
## [1] 106
```

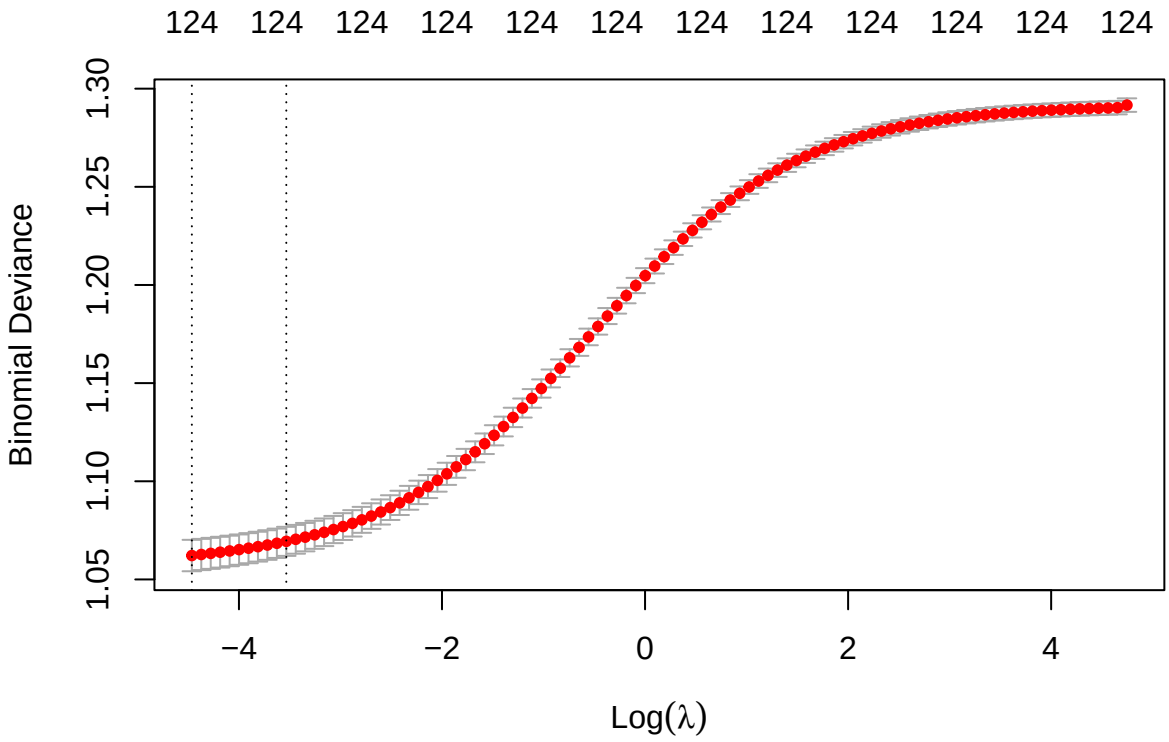
```
Lasso_Diff = Lasso_Comparison[which(
  Lasso_Comparison$lambda_min_coef != Lasso_Comparison$lambda_1se_coef),]
Lasso_Diff$Vars
```

```
##  [1] "countryCH"                "categorySound"
##  [3] "deadline_weekdaySaturday" "deadline_month4"
##  [5] "deadline_month7"          "deadline_month8"
##  [7] "deadline_month11"         "deadline_yr2013"
##  [9] "deadline_yr2014"          "deadline_yr2016"
## [11] "created_at_yr2010"        "created_at_yr2013"
## [13] "created_at_yr2016"        "launched_at_month5"
## [15] "launched_at_month9"       "launched_at_month10"
## [17] "launched_at_yr2010"       "launched_at_yr2011"
## [19] "launched_at_yr2015"       "launched_at_yr2017"
```

```
# 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] 0.2567413
```

```
# Ridge Regression
set.seed(seed)
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] 0.2885829
```

```
# Decision Tree
set.seed(seed)
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
```

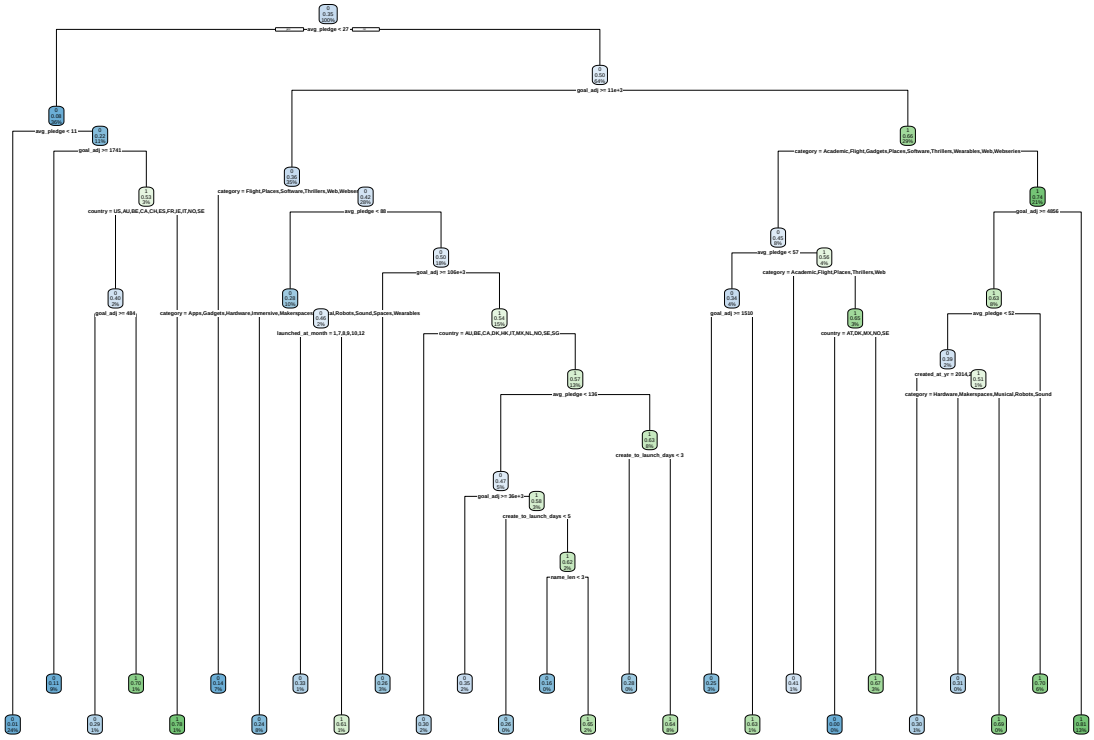
```
## [1] 0.2619048
```

```
# Pruning
Pruned_Tree = prune(Basic_Tree, cp = Tree_Min_Error_Cp)
Pruned_Tree$variable.importance
```

```
##              avg_pledge                goal_adj                category
##           1.475620e+03           6.923726e+02           6.535177e+02
##   create_to_launch_days                 country launch_to_deadline_days
##           2.062763e+02           7.571355e+01           6.981965e+01
##          name_len_clean                name_len           created_at_yr
##           6.966010e+01           5.718525e+01           2.123438e+01
##            deadline_yr          launched_at_yr          deadline_month
##           1.981847e+01           1.733915e+01           1.347290e+01
##       launched_at_month        created_at_month     launched_at_weekday
##           1.037352e+01           8.739700e+00           3.163553e+00
##        deadline_weekday         blurb_len_clean               blurb_len
##           2.112167e+00           1.722117e+00           6.412513e-02
```

```
write.csv(Pruned_Tree$variable.importance, "TreeVI.csv")
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] 0.2165806
```

```
# Bagging
set.seed(seed)
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:  0.2035
```

```
# 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] 0.2062536
```

```
# 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: 19.24%
## Confusion matrix:
##      0    1 class.error
## 0 7617 1484   0.1630590
## 1 1199 3645   0.2475227
```

```
# 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
```

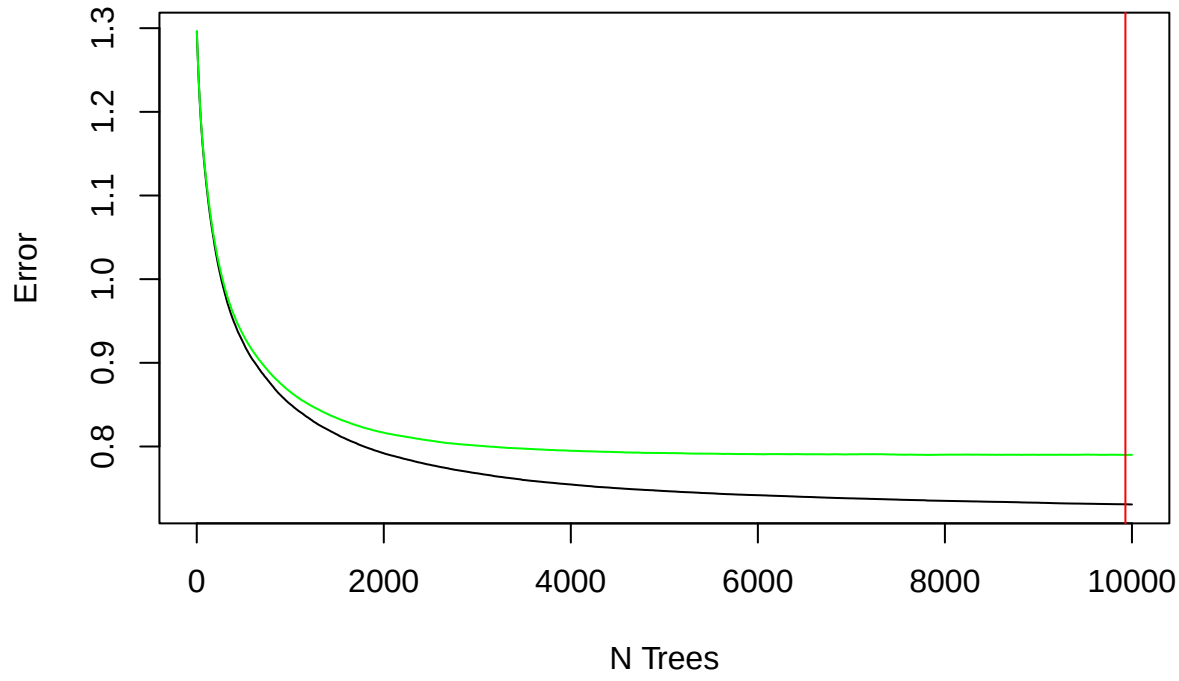
```
## [1] 0.1930579
```

```
# Boosting
set.seed(seed)
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$train.error, type = "l", xlab = 'N Trees', ylab = 'Error')
lines(range, Boost_Model$cv.error, col = "green")
which.min(Boost_Model$train.error)
```

```
## [1] 9999
```

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



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

```
## Using 9294 trees...
```

```r
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] 0.2008032
```

```r
#SVM
SVM_Linear = svm(factor(success) ~ ., data = KS_Train, kernel = "linear", cost = 10)
SVM_Radial = svm(factor(success) ~ ., data = KS_Train, kernel = "radial", cost = 5)
```

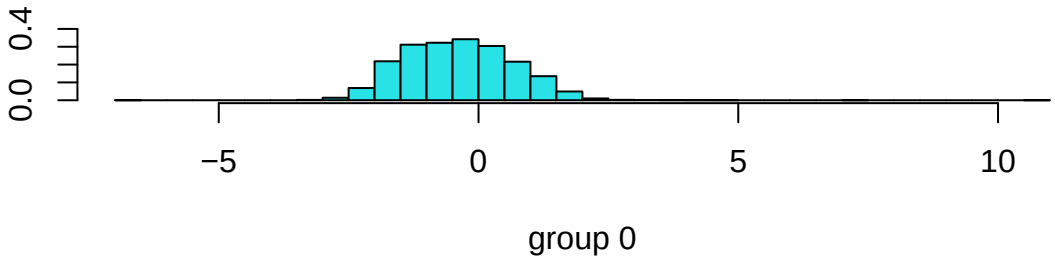
```r
# SVM Linear Performance
SVM_Linear_Pred = predict(SVM_Linear, KS_Test, type = "class")

SVM_Linear_Table = table(SVM_Linear_Pred, KS_Test$success)
SVM_Linear_Table_Prop = prop.table(SVM_Linear_Table)
SVM_Linear_Misc = 1-sum(diag(SVM_Linear_Table_Prop))
SVM_Linear_Misc
```

```
## [1] 0.2659208
```

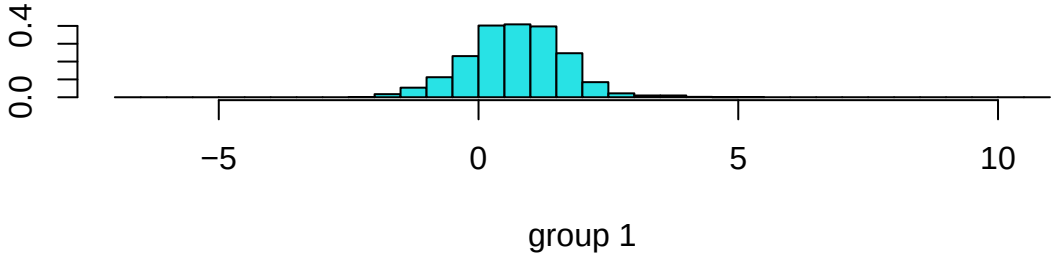
```r
# SVM Radial Performance
SVM_Radial_Pred = predict(SVM_Radial, KS_Test, type = "class")

SVM_Radial_Table = table(SVM_Radial_Pred, KS_Test$success)
SVM_Radial_Table_Prop = prop.table(SVM_Radial_Table)
SVM_Radial_Misc = 1-sum(diag(SVM_Radial_Table_Prop))
SVM_Radial_Misc
```

```
## [1] 0.2664945
```

```r
# LDA
LDA = lda(factor(success) ~., data = KS_Train)
ldahist(data = predict(LDA, KS_Train)$x[,1], g = KS_Train$success)
```



group 0



group 1

```
# LDA Performance
LDA_Pred = predict(LDA, KS_Test)$class

LDA_Table = table(LDA_Pred, KS_Test$success)
LDA_Table_Prop = prop.table(LDA_Table)
LDA_Misc = 1-sum(diag(LDA_Table_Prop))
LDA_Misc
```

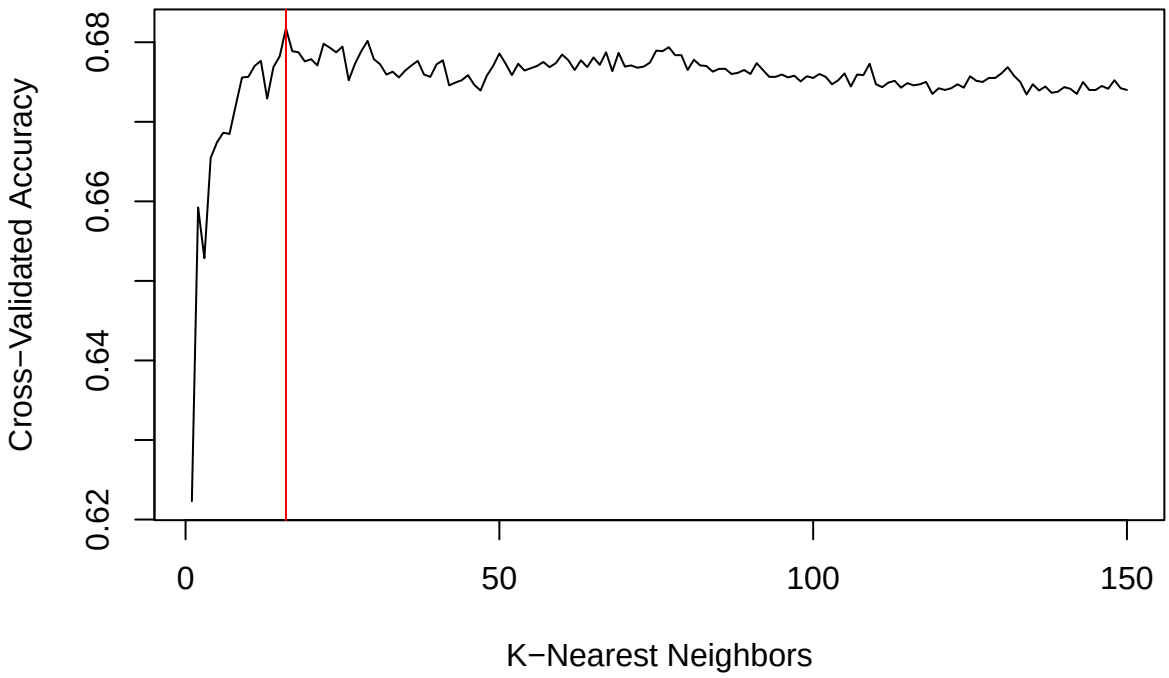
```
## [1] 0.2773953
```

```
# QDA
QDA = qda(factor(success) ~. -country -category -deadline_weekday -created_at_weekday -
            launched_at_weekday -deadline_month -deadline_yr -created_at_month -
            created_at_yr -launched_at_yr -launched_at_month, data = KS_Train)
```

```
# QDA Performance
QDA_Pred = predict(QDA, KS_Test)$class

QDA_Table = table(QDA_Pred, KS_Test$success)
QDA_Table_Prop = prop.table(QDA_Table)
QDA_Misc = 1-sum(diag(QDA_Table_Prop))
QDA_Misc
```

```
## [1] 0.5946644
```

```r
# KNN Train

KS_Train_Scaled = KS_Train %>%
  mutate_if(is.numeric, scale) %>%
  select(-country, -category, -deadline_weekday, -created_at_weekday,
         -launched_at_weekday, -deadline_month, -deadline_yr,
         -created_at_month, -created_at_yr, -launched_at_yr,
         -launched_at_month, -success)

KS_Test_Scaled = KS_Test %>%
  mutate_if(is.numeric, scale) %>%
  select(-country, -category, -deadline_weekday, -created_at_weekday,
         -launched_at_weekday, -deadline_month, -deadline_yr,
         -created_at_month, -created_at_yr, -launched_at_yr,
         -launched_at_month, -success)

k = 1:150
KNN_Train = Rfast::knn.cv(nfolds = 10,
                          seed = seed,
                          y = as.factor(KS_Train$success),
                          x = as.matrix(KS_Train_Scaled),
                          k = k,
                          type = "C",
                          pred.ret = TRUE)

optimal_k = which.max(KNN_Train$crit)
plot(k, KNN_Train$crit, type = "l",
     xlab = "K-Nearest Neighbors", ylab = "Cross-Validated Accuracy")
abline(v = optimal_k, col = "red")
```




```
# KNN
KNN = Rfast::knn(xnew = as.matrix(KS_Test_Scaled),
                 y = as.factor(KS_Train$success),
                 x = as.matrix(KS_Train_Scaled),
                 k = optimal_k)-1

# KNN Performance
KNN_Table = table(KNN, KS_Test$success)
KNN_Table_Prop = prop.table(KNN_Table)
KNN_Misc = 1-sum(diag(KNN_Table_Prop))
KNN_Misc
```

```
## [1] 0.3227194
```

```
# Results Table
Results = data.frame(
  Method = c("Logisitic Regression",
             "Lasso Regression",
             "Ridge Regression",
             "Decision Tree",
             "Bagging",
             "Random Forest",
             "Boosting",
             "SVM Linear",
             "SVM Radial",
```

```
             "LDA",
             "QDA",
             "KNN"),
  Misclass_rate = c(Logistic_Misc,
                    Lasso_Misc,
                    Ridge_Misc,
                    Pruned_Tree_Misc,
                    Bag_Misc,
                    RF_Misc,
                    Boost_Misc,
                    SVM_Linear_Misc,
                    SVM_Radial_Misc,
                    LDA_Misc,
                    QDA_Misc,
                    KNN_Misc)
)

Results
```

```
##                  Method Misclass_rate
## 1  Logisitic Regression     0.2598967
## 2      Lasso Regression     0.2567413
## 3      Ridge Regression     0.2885829
## 4         Decision Tree     0.2165806
## 5               Bagging     0.2062536
## 6         Random Forest     0.1930579
## 7              Boosting     0.2008032
## 8            SVM Linear     0.2659208
## 9            SVM Radial     0.2664945
## 10                  LDA     0.2773953
## 11                  QDA     0.5946644
## 12                  KNN     0.3227194
```

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