ECON 573 - Final Proejct

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

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

```r
library(lubridate)
```

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

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

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

```
## Loading required package: lattice

##
## Attaching package: 'lattice'

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

```r
library(glmnet)
```

```
## Loading required package: Matrix

## Loaded glmnet 4.1-4
```

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

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

```r
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
```

```r
library(gbm)
```

```
## Loaded gbm 2.1.8.1
```

```r
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
```

```r
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
# Sampling of Test / Train
seed = 5
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
## -5.9895  -0.8353  -0.4062   0.9485   6.4823
##
## Coefficients:
##                              Estimate Std. Error z value Pr(>|z|)
## (Intercept)                 8.866e-01  5.140e-01   1.725 0.084535 .
## goal_adj                   -9.925e-06  6.068e-07 -16.357  < 2e-16 ***
## countryAT                  -8.993e-01  4.580e-01  -1.963 0.049593 *
## countryAU                  -3.809e-01  1.336e-01  -2.852 0.004346 **
## countryBE                  -1.288e+00  6.502e-01  -1.981 0.047613 *
## countryCA                  -2.995e-01  9.832e-02  -3.046 0.002318 **
## countryCH                  -1.003e-01  3.329e-01  -0.301 0.763126
## countryDE                   1.911e-02  1.614e-01   0.118 0.905725
## countryDK                  -1.350e+00  3.985e-01  -3.387 0.000706 ***
## countryES                  -9.735e-01  2.984e-01  -3.262 0.001106 **
## countryFR                  -1.819e-01  1.870e-01  -0.973 0.330669
## countryGB                   1.715e-01  6.348e-02   2.702 0.006891 **
## countryHK                  -1.305e+00  6.562e-01  -1.988 0.046783 *
## countryIE                   2.381e-01  3.262e-01   0.730 0.465586
## countryIT                  -1.236e+00  2.772e-01  -4.460 8.19e-06 ***
## countryMX                  -3.391e+00  7.045e-01  -4.814 1.48e-06 ***
## countryNL                  -1.060e-01  1.770e-01  -0.599 0.549462
## countryNO                  -2.049e+00  5.887e-01  -3.480 0.000502 ***
## countryNZ                  -1.568e-01  2.975e-01  -0.527 0.598043
## countrySE                  -1.564e+00  3.853e-01  -4.059 4.92e-05 ***
## countrySG                  -2.053e-01  6.156e-01  -0.333 0.738786
## categoryAcademic           -1.506e+01  3.646e+02  -0.041 0.967060
## categoryApps               -5.787e-01  1.023e-01  -5.659 1.52e-08 ***
## categoryBlues               1.541e+01  3.795e+02   0.041 0.967610
## categoryExperimental        8.137e-01  1.537e-01   5.295 1.19e-07 ***
## categoryFestivals           8.289e-01  1.339e-01   6.192 5.93e-10 ***
```

```
## categoryFlight              -1.056e+00  1.774e-01  -5.956 2.59e-09 ***
## categoryGadgets             -3.270e-01  8.445e-02  -3.872 0.000108 ***
## categoryHardware            -4.481e-01  7.892e-02  -5.678 1.36e-08 ***
## categoryImmersive            5.904e-01  1.628e-01   3.626 0.000288 ***
## categoryMakerspaces         -1.673e-01  1.957e-01  -0.855 0.392609
## categoryMusical              4.495e-01  1.087e-01   4.136 3.53e-05 ***
## categoryPlaces              -1.523e+01  1.582e+02  -0.096 0.923329
## categoryPlays                6.713e-01  9.661e-02   6.948 3.71e-12 ***
## categoryRobots               4.209e-02  1.364e-01   0.309 0.757602
## categoryShorts               1.514e+01  2.595e+02   0.058 0.953473
## categorySoftware            -1.588e+00  9.376e-02 -16.938  < 2e-16 ***
## categorySound                8.480e-02  1.311e-01   0.647 0.517634
## categorySpaces               4.284e-01  1.997e-01   2.145 0.031916 *
## categoryThrillers           -1.475e+01  3.405e+02  -0.043 0.965443
## categoryWearables           -1.212e-01  1.082e-01  -1.120 0.262661
## categoryWeb                 -1.915e+00  1.006e-01 -19.032  < 2e-16 ***
## categoryWebseries           -1.562e+01  3.545e+02  -0.044 0.964857
## name_len                     7.917e-02  2.141e-02   3.698 0.000217 ***
## name_len_clean               4.518e-02  2.522e-02   1.791 0.073219 .
## blurb_len                   -4.067e-02  7.491e-03  -5.429 5.66e-08 ***
## blurb_len_clean              3.959e-02  1.060e-02   3.734 0.000189 ***
## deadline_weekdayMonday       1.380e-01  8.472e-02   1.628 0.103462
## deadline_weekdaySaturday    -2.104e-02  8.055e-02  -0.261 0.793975
## deadline_weekdaySunday      -7.623e-02  7.952e-02  -0.959 0.337787
## deadline_weekdayThursday    -3.895e-02  7.780e-02  -0.501 0.616596
## deadline_weekdayTuesday      2.062e-01  8.786e-02   2.346 0.018953 *
## deadline_weekdayWednesday    4.892e-02  7.891e-02   0.620 0.535305
## created_at_weekdayMonday     1.700e-02  7.581e-02   0.224 0.822545
## created_at_weekdaySaturday  -1.368e-01  8.756e-02  -1.562 0.118171
## created_at_weekdaySunday    -1.624e-01  8.537e-02  -1.902 0.057184 .
## created_at_weekdayThursday  -4.671e-02  7.774e-02  -0.601 0.547972
## created_at_weekdayTuesday    9.961e-02  7.531e-02   1.323 0.185906
## created_at_weekdayWednesday -1.007e-02  7.699e-02  -0.131 0.895882
## launched_at_weekdayMonday    2.043e-01  8.328e-02   2.453 0.014161 *
## launched_at_weekdaySaturday -1.879e-02  1.121e-01  -0.168 0.866906
## launched_at_weekdaySunday    1.852e-01  1.106e-01   1.675 0.093981 .
## launched_at_weekdayThursday  1.507e-01  8.590e-02   1.754 0.079356 .
## launched_at_weekdayTuesday   4.849e-01  8.106e-02   5.982 2.21e-09 ***
## launched_at_weekdayWednesday 2.668e-01  8.181e-02   3.261 0.001111 **
## deadline_month2              3.026e-01  1.898e-01   1.594 0.110863
## deadline_month3              3.762e-01  2.668e-01   1.410 0.158506
## deadline_month4              3.084e-01  3.342e-01   0.923 0.356077
## deadline_month5              4.891e-01  3.965e-01   1.234 0.217351
## deadline_month6              5.965e-01  4.533e-01   1.316 0.188176
## deadline_month7              5.531e-01  5.088e-01   1.087 0.277046
## deadline_month8              6.477e-01  5.651e-01   1.146 0.251679
## deadline_month9              4.956e-01  6.221e-01   0.797 0.425623
## deadline_month10             5.878e-01  6.812e-01   0.863 0.388239
## deadline_month11             7.277e-01  7.379e-01   0.986 0.324053
## deadline_month12             6.904e-01  7.954e-01   0.868 0.385405
## deadline_yr2010              6.919e-01  1.185e+00   0.584 0.559165
## deadline_yr2011              1.057e+00  1.979e+00   0.534 0.593478
## deadline_yr2012              1.721e+00  2.785e+00   0.618 0.536677
## deadline_yr2013              2.226e+00  3.601e+00   0.618 0.536421
```

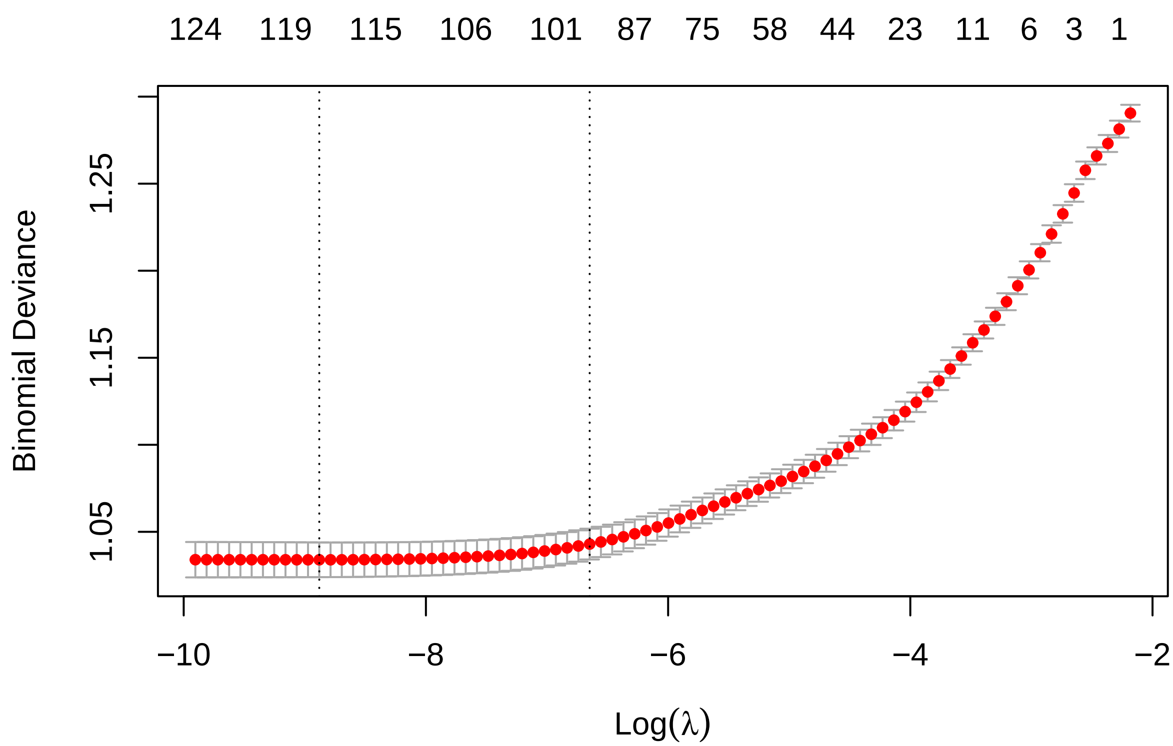
```
## deadline_yr2014              3.092e+00  4.415e+00   0.700 0.483726
## deadline_yr2015              3.633e+00  5.237e+00   0.694 0.487802
## deadline_yr2016              4.562e+00  6.069e+00   0.752 0.452276
## deadline_yr2017              5.126e+00  6.903e+00   0.743 0.457774
## created_at_month2           -3.678e-02  1.307e-01  -0.282 0.778313
## created_at_month3            1.384e-01  1.658e-01   0.835 0.403623
## created_at_month4            9.205e-02  2.114e-01   0.436 0.663176
## created_at_month5            1.179e-01  2.595e-01   0.454 0.649474
## created_at_month6            1.269e-01  3.090e-01   0.411 0.681242
## created_at_month7           -1.244e-01  3.639e-01  -0.342 0.732467
## created_at_month8           -1.641e-01  4.175e-01  -0.393 0.694249
## created_at_month9           -1.336e-01  4.724e-01  -0.283 0.777267
## created_at_month10          -3.565e-01  5.270e-01  -0.676 0.498753
## created_at_month11          -4.455e-01  5.796e-01  -0.769 0.442072
## created_at_month12          -4.597e-01  6.369e-01  -0.722 0.470414
## created_at_yr2010           -1.657e+01  1.455e+03  -0.011 0.990916
## created_at_yr2011           -1.839e+01  1.455e+03  -0.013 0.989916
## created_at_yr2012           -1.920e+01  1.455e+03  -0.013 0.989476
## created_at_yr2013           -1.971e+01  1.455e+03  -0.014 0.989193
## created_at_yr2014           -2.034e+01  1.455e+03  -0.014 0.988848
## created_at_yr2015           -2.079e+01  1.455e+03  -0.014 0.988603
## created_at_yr2016           -2.126e+01  1.455e+03  -0.015 0.988345
## created_at_yr2017           -2.258e+01  1.455e+03  -0.016 0.987623
## launched_at_month2           7.581e-03  1.879e-01   0.040 0.967818
## launched_at_month3          -3.654e-02  2.631e-01  -0.139 0.889529
## launched_at_month4          -2.474e-01  3.352e-01  -0.738 0.460611
## launched_at_month5          -2.572e-01  3.939e-01  -0.653 0.513761
## launched_at_month6          -3.217e-01  4.556e-01  -0.706 0.480190
## launched_at_month7          -3.697e-01  5.144e-01  -0.719 0.472330
## launched_at_month8          -1.298e-01  5.742e-01  -0.226 0.821135
## launched_at_month9          -1.861e-01  6.357e-01  -0.293 0.769684
## launched_at_month10         -1.535e-01  6.960e-01  -0.221 0.825397
## launched_at_month11          2.103e-02  7.547e-01   0.028 0.977771
## launched_at_month12         -1.981e-01  8.100e-01  -0.245 0.806797
## launched_at_yr2010           1.497e+01  1.455e+03   0.010 0.991793
## launched_at_yr2011           1.661e+01  1.455e+03   0.011 0.990896
## launched_at_yr2012           1.684e+01  1.455e+03   0.012 0.990768
## launched_at_yr2013           1.674e+01  1.455e+03   0.012 0.990823
## launched_at_yr2014           1.566e+01  1.455e+03   0.011 0.991416
## launched_at_yr2015           1.557e+01  1.455e+03   0.011 0.991462
## launched_at_yr2016           1.521e+01  1.455e+03   0.010 0.991660
## launched_at_yr2017           1.534e+01  1.455e+03   0.011 0.991590
## create_to_launch_days       -1.248e-03  1.887e-03  -0.662 0.508234
## launch_to_deadline_days     -1.569e-02  3.005e-03  -5.222 1.77e-07 ***
## avg_pledge                   2.223e-03  1.411e-04  15.750  < 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: 18000  on 13944  degrees of freedom
## Residual deviance: 14175  on 13820  degrees of freedom
## AIC: 14425
##
```

```
## Number of Fisher Scoring iterations: 14
```

```r
# Logistic Regression Performance
Logistic_Pred = ifelse(predict(Logistic_Model, KS_Test, type = "response")>.5, 1, 0)
Logistic_Table = table(Logistic_Pred, KS_Test$success)
Logistic_Table_Prop = prop.table(Logistic_Table)
Logistic_Misc = 1-sum(diag(Logistic_Table_Prop))
Logistic_Misc
```

```
## [1] 0.2507172
```

```r
# 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)
```




```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] 120
```

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

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

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

```
##  [1] "countryCH"                "countryDE"
##  [3] "countryNZ"                "countrySG"
##  [5] "deadline_weekdaySaturday" "deadline_month5"
##  [7] "deadline_month8"          "deadline_month12"
##  [9] "deadline_yr2012"          "deadline_yr2013"
## [11] "deadline_yr2014"          "deadline_yr2016"
## [13] "created_at_month7"        "created_at_month8"
## [15] "created_at_yr2010"        "created_at_yr2015"
## [17] "created_at_yr2016"        "launched_at_month2"
## [19] "launched_at_month4"       "launched_at_month8"
## [21] "launched_at_month10"      "launched_at_yr2010"
## [23] "launched_at_yr2015"
```

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

Lasso_Pred = ifelse(predict(Lasso_Model_1se, 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
```

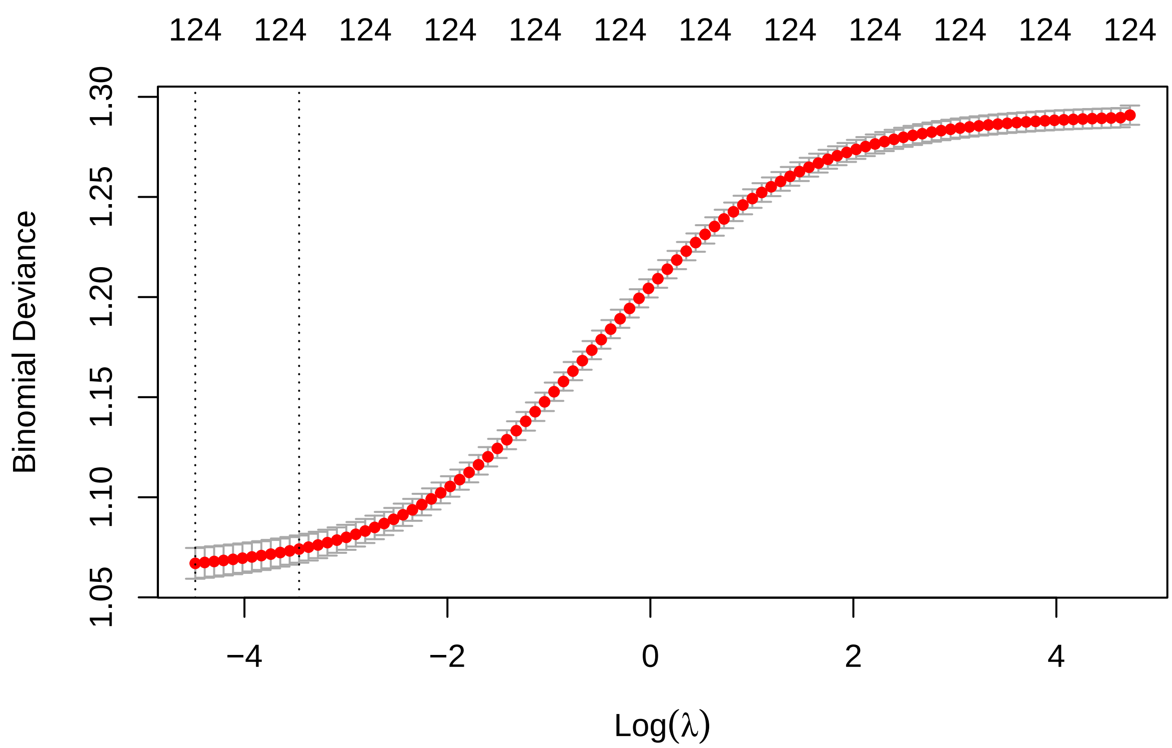
```
## [1] 0.2587493
```

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

```r
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 # Better
```

```
## [1] 0.2498566
```

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




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

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

```r
# Ridge Model Performance - 1se
Ridge_Pred = ifelse(predict(Ridge_Model_1se, x_Test, type = "response")>.5, 1, 0)
Ridge_Table = table(Ridge_Pred, KS_Test$success)
```

```r
Ridge_Table_Prop = prop.table(Ridge_Table)
Ridge_Misc = 1-sum(diag(Ridge_Table_Prop))
Ridge_Misc
```

```
## [1] 0.319564
```

```r
# Ridge Model Performance - Min
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 #Better
```

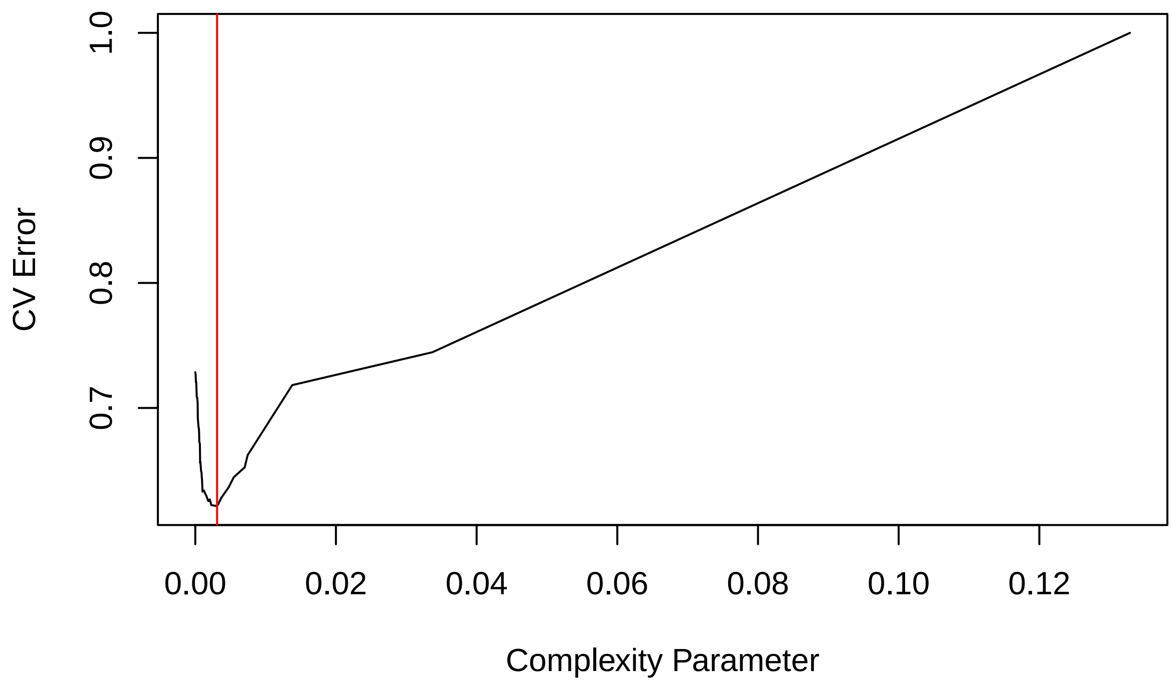
```
## [1] 0.2908778
```

```r
# 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]
```

```r
# 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] 0.2366609
```

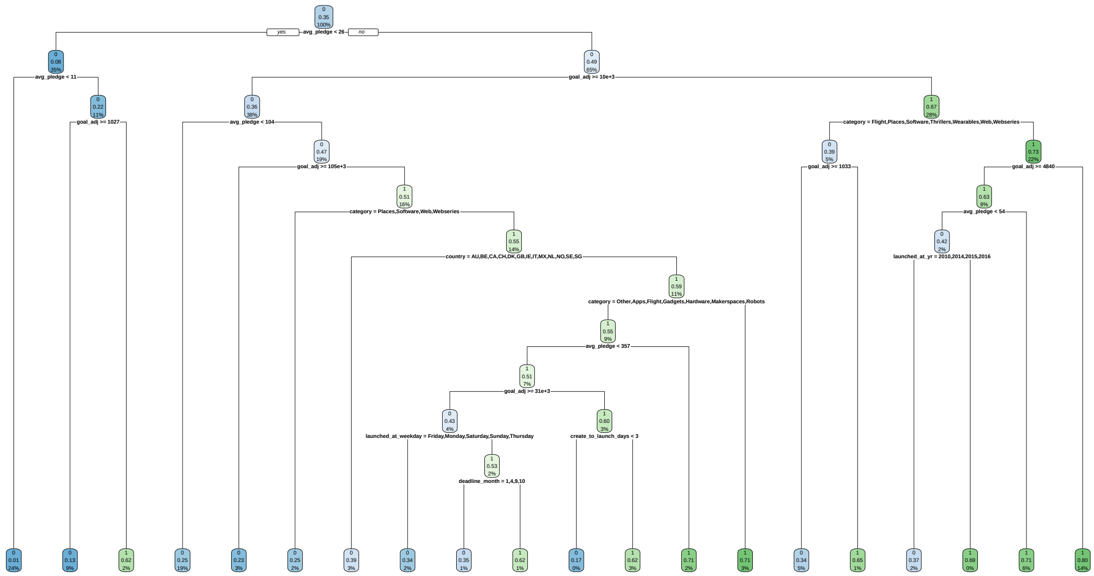
```r
plot(Basic_Tree$cptable[,1], Basic_Tree$cptable[,4],
     type = "l", col = "black", xlab = "Complexity Parameter", ylab = "CV Error")
abline(v = Basic_Tree$cptable[which.min(Basic_Tree$cptable[,4]),1], col = "red")
```




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

```
##              avg_pledge                goal_adj                category
##            1412.4478475             705.9892524             542.8297411
##    create_to_launch_days launch_to_deadline_days                 country
##             164.1906871              66.7502067              40.4889536
##           name_len_clean                name_len             deadline_yr
##              40.1689607              39.8291576              17.0368109
##           launched_at_yr           created_at_yr      launched_at_weekday
##              16.6600065              15.6679512               8.4812528
##           deadline_month       launched_at_month         deadline_weekday
##               7.1423730               5.6489952               3.1398681
##         created_at_month      created_at_weekday         blurb_len_clean
##               2.1322749               0.7218087               0.5687519
##                blurb_len
##               0.4420170
```

```
write.csv(Pruned_Tree$variable.importance, "TreeVI.csv")
rpart.plot(Pruned_Tree)
```



```
# 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] 0.2079748
```

```
# 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:  0.2057
```

```r
# 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")
```

```r
# 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] 0.188755
```

```r
# 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: 19.83%
## Confusion matrix:
##      0    1 class.error
## 0 7619 1491   0.1636663
## 1 1274 3561   0.2634953
```

```r
# Random Forest 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")
```

```r
# 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
```

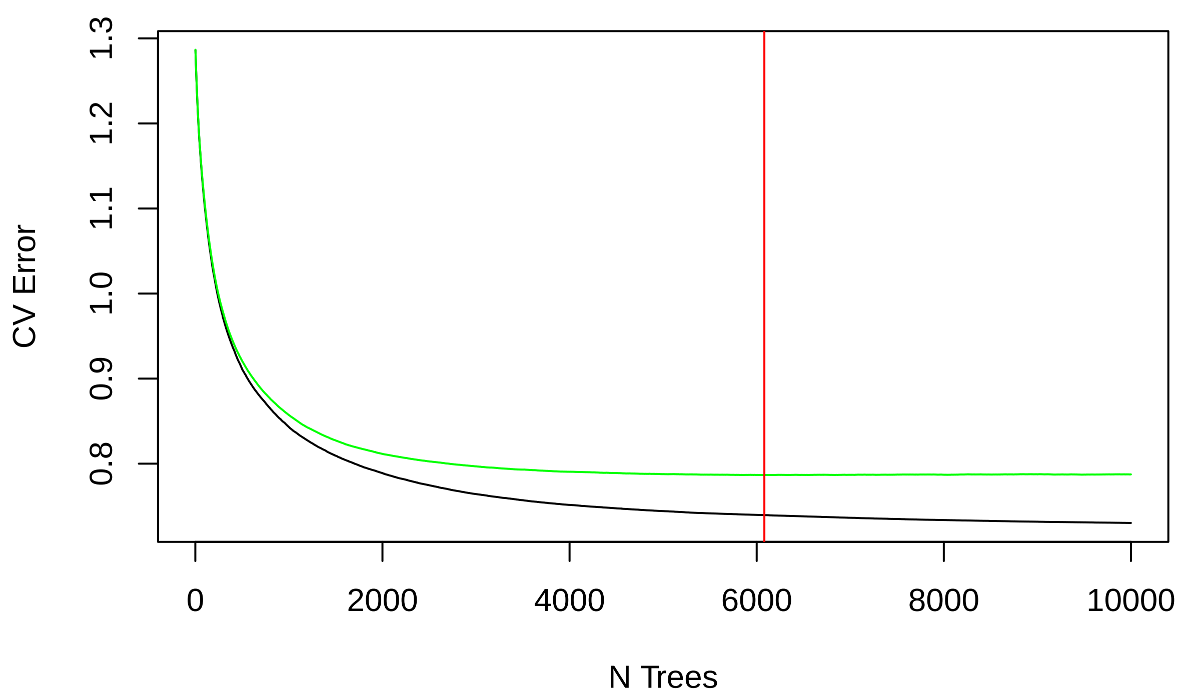
```
## [1] 0.1864601
```

```r
# 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)
```

```r
# Boosting CV
range = 1:N_Trees
plot(range, Boost_Model$train.error, type = "l", xlab = 'N Trees', ylab = 'CV Error')
lines(range, Boost_Model$cv.error, col = "green")
which.min(Boost_Model$cv.error)
```

```
## [1] 6082
```

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



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

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] 0.1858864
```

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

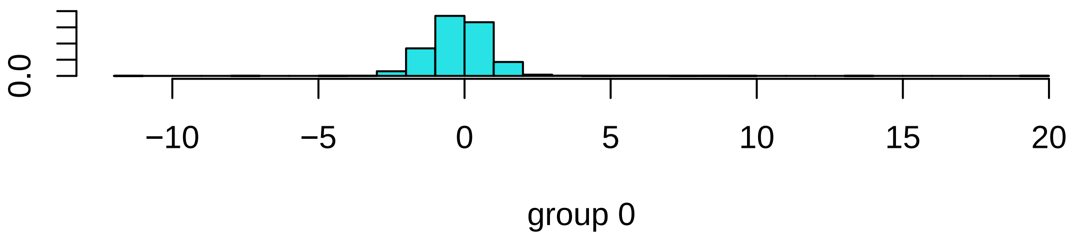
```r
# SVM Linear Performance
SVM_Linear_Pred = predict(SVM_Linear, KS_Test, type = "class")

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
```

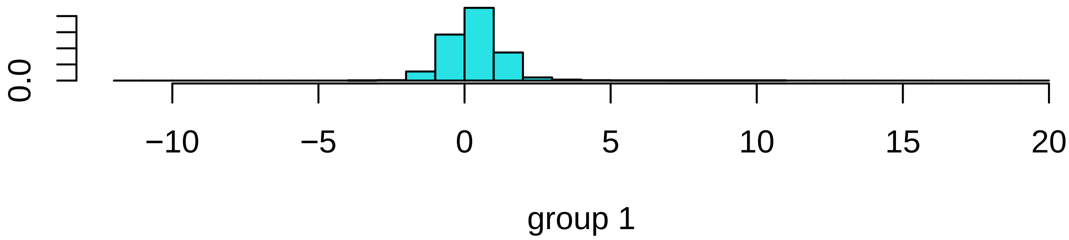
```
## [1] 0.2685026
```

```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] 0.253012
```

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






```
# 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
```

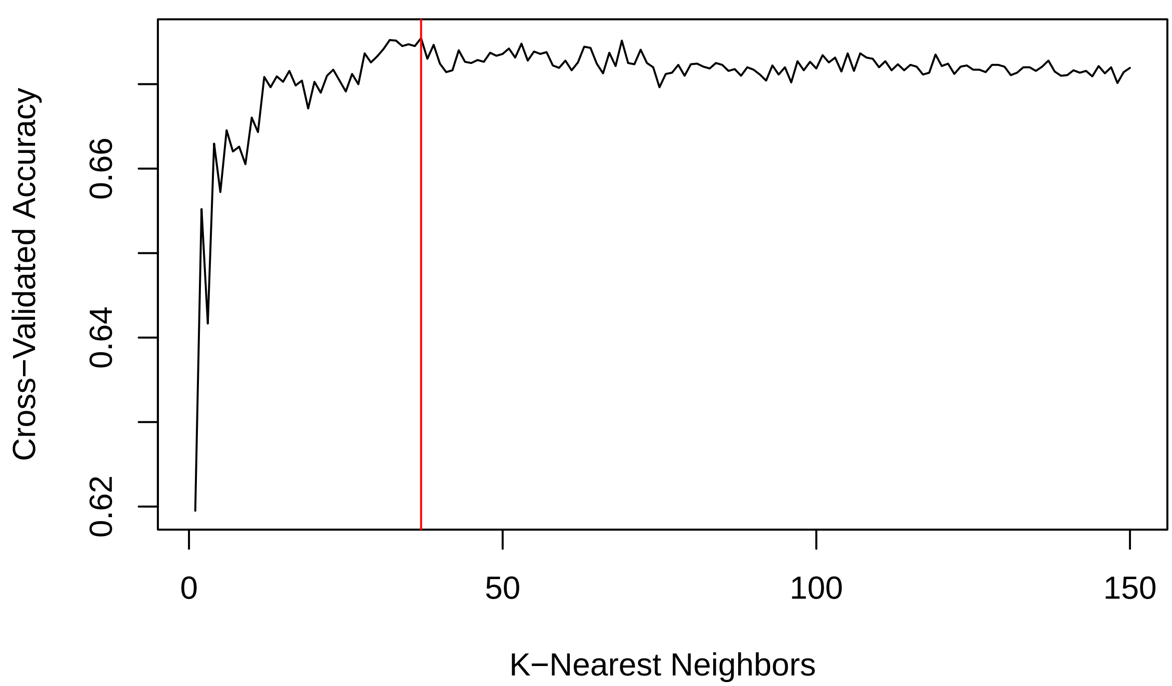
```
## [1] 0.3304647
```

```
# 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] 0.6044177
```

```r
# KNN Train
KS_Scaled = KS_Clean %>%
  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_Train_Scaled = KS_Scaled[Train_Ind,]
KS_Test_Scaled = KS_Scaled[-Train_Ind,]

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")
```



```r
# 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] 0.3164085
```

```r
# 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     0.2507172
## 2      Lasso Regression     0.2498566
## 3      Ridge Regression     0.2908778
## 4         Decision Tree     0.2079748
## 5               Bagging     0.1887550
## 6         Random Forest     0.1864601
## 7              Boosting     0.1858864
## 8            SVM Linear     0.2685026
```

```
## 9            SVM Radial      0.2530120
## 10                  LDA      0.3304647
## 11                  QDA      0.6044177
## 12                  KNN      0.3164085
```

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