Predicting Kickstarter campaign success

David Ika

27/10/2020

1 - Introduction

This Project, Project 2, continues from Project 1 and observes the same dataset.

As mentioned in Project 1, the chosen dataset is titled “Funding Successful Projects on Kickstarter” and can be found on Kaggle here (https://www.kaggle.com/codename007/funding-successful-projects), uploaded by user Lathwal (https://www.kaggle.com/codename007)

The dataset was released by Kickstarter (https://www.kickstarter.com/), a company that connects community investors with start-up projects in an ‘all-or-nothing’ fashion: The user sets a goal for their project, and if it falls short by even $1, zero funding is attained.

Whilst Project 1 explored and visualised the data, Project 2 aims to address the initial business objective set by Kickstarter: to help predict whether a project will be successfully funded. Classification and clustering methods will be used to achieve this.

1.1 - Install packages and load libraries

```
#install.packages()

# processing:
library(ROCR) # AUC analysis of feature variables.
library(rpart) # decision tree.
library(rpart.plot) # decision tree.
library(party) # decision tree.
library(pander) # setting up Pander Options.
library(class) # kNN modelling.
library(grDevices) # clustering: finding convex hull.
library(fpc) # cluster testing (via boot iterations).
library(cluster) # for daisy() function, for gower distance measurement.

#general
library(dplyr) #data cleaning.
library(anytime) #time formats.
library(forcats) #data sorting.
library(scales) #labelling axes.
library(lubridate) #manipulate date/time.
library(stringr) #splitting columns
library(countrycode) #country codes.
library(tidyverse)
library(kableExtra)

#Plotting
library(corrplot)
library(ggplot2)
library(tidyverse)
library(gridExtra)
library(ggthemes)
library(vcd)
library(reshape2)
library(ROCit) # plotting ROC curves.
```

1.2 - Setting a theme to be automatically applied

```
theme_set(theme_minimal()+
              theme(text = element_text(size = 9, colour = "grey20"),
                    axis.text = element_text(size = 10, colour = "grey10"),
                    axis.title = element_text(size=11,face="bold"),
                    plot.title = element_text(size=12,face="bold"),
    panel.grid.major = element_blank(),
    panel.grid.minor = element_blank(),
    panel.border = element_blank(),
    panel.background = element_blank(),
    axis.line = element_line(colour = "grey20",
                      size = 1, linetype = "solid"),
    axis.ticks = element_line(size = 0.5)))
```

2 - Data Preparation

A large portion of this cleaning and transformation process was carried out in Project 1, but I am restating here for clarity, plus performing further transformations as required.

```
ks_base <- read.csv("train.csv")
```

2.1 - Brief observations

```
str(ks_base)
```

- Total of 108,129 projects analysed across 14 variables.
- Shows various data types: character strings, integers, numerical values and Booleans.
- Some chr variables may need to be converted to factors or numeric values.
- Some formats will need to be transformed to be useful.
- Good mix of info: geographic, time-related, author-related, text-based.

```
summary(ks_base)
```

- Huge variance in goal amount: 0 to 10 million.
- To be converted to factor: country, currency, outcome.
- Time-related variables to be transformed: deadline, state_changed_at, created_at, launched_at.

2.1.1 - Variables and context

```
names(ks_base)
```

```
##  [1] "project_id"           "name"                 "desc"
##  [4] "goal"                 "keywords"             "disable_communication"
##  [7] "country"              "currency"             "deadline"
## [10] "state_changed_at"     "created_at"           "launched_at"
## [13] "backers_count"        "final_status"
```

- project_id: unique id of project.
- name: name of project.
- desc: description of project.
- goal: $ amount required for project.
- keywords: words describing project.
- disable_communication: whether project author has opted to disable communication with donors.
- country: country of project's author.
- currency: currency of goal amount.
- deadline: goal must be achieved on or before this date (unix time format).
- state_changed_at: at this time, project status changed to successful or otherwise (1,0). Unix time format.
- created_at: at this time, project was posted to Kickstarter (unix time format).
- launched_at: at this time, project went live on website (unix time format).
- backers_count: number of people who backed the project.
- final_status: whether project was successfully funded (1 = True; 0 = False).

2.1.1.1 - Renaming some variables for clarity

```
names(ks_base)[6] <- "disable_comms"
names(ks_base)[13] <- "backers"
names(ks_base)[14] <- "outcome"
```

2.1.2 - Subsetting & summarising numerical data

```
ks_base_num <- ks_base[,!sapply(ks_base, is.character)]

summary(ks_base_num)
```

```
##       goal           disable_comms   state_changed_at     launched_at
##  Min.   :        0   Mode :logical   Min.   :1.241e+09   Min.   :1.241e+09
##  1st Qu.:     2000   FALSE:107806    1st Qu.:1.347e+09   1st Qu.:1.344e+09
##  Median :     5000   TRUE :323       Median :1.394e+09   Median :1.391e+09
##  Mean   :    36726                   Mean   :1.380e+09   Mean   :1.377e+09
##  3rd Qu.:    13000                   3rd Qu.:1.416e+09   3rd Qu.:1.413e+09
##  Max.   :100000000                   Max.   :1.433e+09   Max.   :1.433e+09
##  NA's   :2                           NA's   :3           NA's   :7
```

- disable_communication: Only 323 out of 108,129 elected to disable this communication. Exclude from analysis (immaterial).
- Time conversions required, as noted.
- NAs have been observed; to be dealt with.

2.2 - Data Cleaning & Transformation

2.2.1 - Checking NAs in each column.

```
(apply(is.na(ks_base), 2, sum))
```

```
##       project_id             name             desc             goal
##                0                6                0                2
##         keywords    disable_comms          country         currency
##                1                0                0                1
##         deadline state_changed_at       created_at      launched_at
##                2                3                0                7
##          backers          outcome
##                0                0
```

2.2.2 - Summing for total NAs

```
sum(apply(is.na(ks_base), 2, sum))
```

```
## [1] 22
```

Thus far, only 22 NAs from entire dataset out of 108,129 obs. Safe to remove without affecting dataset. Assigning non-NA data to ks_base1.

```
ks_base1 <- na.omit(ks_base)
```

2.2.3 - Transform further invalid data into NAs

Some “?” values were identified and so we will convert these, along with blanks and “NA” chr strings, to actual NAs.

```
ks_base1[ks_base1 == "NA"] <- NA
ks_base1[ks_base1 == ""] <- NA
ks_base1[ks_base1 == "?"] <- NA
```

Re-running the check for NAs

```
sum(apply(is.na(ks_base1), 2, sum))
```

```
## [1] 61
```

And again, removing NAs

```
ks_base2 <- na.omit(ks_base1)
```

2.2.4 - General sense checks

With prior context, checking for nonsensical data:

- *goal* should not be negative.
- *state_changed_at* should not be before *created at* nor *launched_at*.
- *deadline* should not be before *created at* nor *launched_at*.

Unless these count for a large portion, we will remove those rows.

```
count(ks_base2[ks_base2[4] < 0, ])
```

```
##   n
## 1 0
```

```
count(ks_base2[ks_base2$deadline < ks_base2$launched_at,])
```

```
##   n
## 1 0
```

```
count(ks_base2[ks_base2$deadline < ks_base2$created_at,])
```

```
##   n
## 1 0
```

```
count(ks_base2[ks_base2$state_changed_at < ks_base2$launched_at,])
```

```
##   n
## 1 0
```

```
count(ks_base2[ks_base2$state_changed_at < ks_base2$created_at,])
```

```
##   n
## 1 0
```

No abnormalities.

2.2.5 - For categorical strings, converting to factors

```
ks_base2$country <- factor(ks_base2$country)
ks_base2$currency <- factor(ks_base2$currency)
ks_base2$outcome <- factor(ks_base2$outcome)
```

2.2.6 - For numeric & continuous strings, converting to numerals.

```
ks_base2$deadline <- as.numeric(ks_base2$deadline)
ks_base2$created_at <- as.numeric(ks_base2$created_at)
ks_base2$backers <- as.numeric(ks_base2$backers)
```

2.2.7 - Cleaning up *country* variable

Converting the country acronyms to long-handed characters, then back into factors.

```
ks_base2$country <- factor(countrycode(ks_base2$country, "iso2c", "country.name"))
```

2.2.8 - Dropping unnecessary variable

Also dropping the project_id variable due to its redundancy, but will use a new variable should we wish to revert.

```
ks_base3 <- select(ks_base2,-1)
```

2.2.9 - Convering unix time format to date

As mentioned, the following variables are in unix time format which will now be converted into a more usable date object. Again, assigning converted columns + dataset to a new variable, should we wish to revert.

- deadline
- state_changed_at
- created_at
- launched_at

```
ks_base4 <- ks_base3
ks_base4[8:11] <- lapply(ks_base4[8:11], anydate)
head(ks_base4[8:11],5)
```

```
##     deadline state_changed_at created_at launched_at
## 1 2014-11-21       2014-11-21 2014-09-26  2014-09-28
## 3 2011-06-19       2011-06-19 2011-05-20  2011-05-20
## 4 2011-05-15       2011-05-15 2011-03-18  2011-04-15
## 7 2011-06-14       2011-06-14 2011-04-22  2011-04-22
## 8 2015-04-14       2015-04-14 2015-03-12  2015-03-18
```

Variables that were in unix time formats now show as yyyy-mm-dd.

2.2.10 - Re-summarise

```
summary(ks_base4)
```

```
##      name               desc                goal             keywords
##  Length:108053      Length:108053      Min.   :        0   Length:108053
##  Class :character   Class :character   1st Qu.:     2000   Class :character
##  Mode  :character   Mode  :character   Median :     5000   Mode  :character
##                                        Mean   :    36739
##                                        3rd Qu.:    13000
##                                        Max.   :100000000
##
##  disable_comms             country         currency        deadline
##  Mode :logical   United States :91974   USD    :91974   Min.   :2009-05-03
##  FALSE:107731    United Kingdom: 8746   GBP    : 8746   1st Qu.:2012-09-04
##  TRUE :322       Canada        : 3734   CAD    : 3734   Median :2014-03-01
##                  Australia     : 1879   AUD    : 1879   Mean   :2013-09-27
##                  Netherlands   :  705   EUR    :  817   3rd Qu.:2014-11-11
##                  New Zealand   :  353   NZD    :  353   Max.   :2015-06-01
##                  (Other)       :  662   (Other):  550
##  state_changed_at       created_at          launched_at
##  Min.   :2009-05-03   Min.   :2009-04-22   Min.   :2009-04-25
##  1st Qu.:2012-09-04   1st Qu.:2012-06-19   1st Qu.:2012-08-02
##  Median :2014-02-28   Median :2013-11-14   Median :2014-01-28
##  Mean   :2013-09-25   Mean   :2013-07-17   Mean   :2013-08-23
##  3rd Qu.:2014-11-10   3rd Qu.:2014-09-02   3rd Qu.:2014-10-09
##  Max.   :2015-06-01   Max.   :2015-05-23   Max.   :2015-05-27
##
##     backers         outcome
##  Min.   :     0.0   0:73514
##  1st Qu.:     2.0   1:34539
##  Median :    17.0
##  Mean   :   123.6
##  3rd Qu.:    65.0
##  Max.   :219382.0
##
```

Overall summary now makes a lot more sense.

2.3 - Selecting variables

2.3.1 - Response variable

From the dataset, it is clear that the response variable will be *outcome*: a binary result of 1 being that the project was successful, and 0 being that it was not successful.

```
head(ks_base4$outcome,5)
```

```
## [1] 1 0 1 1 1
## Levels: 0 1
```

2.3.2 - Feature variables

The feature variables are then chosen from all other variables that remain on the ks_base4 dataframe. We will remove the character-type variables from processing.

```
ks_base5 <- ks_base4[c(3,5:13)]
```

Now removing disable_comms variable due to immateriality, as mentioned.

```
ks_base5 <- ks_base5[c(-2)]
```

We will furthermore create some new variables which correspond to some ways in which we analysed data in Project 1.

- Time between creating the project and launching it:

```
ks_base5$launched_created <- as.numeric(ks_base5$launched_at - ks_base5$created_at)
```

- Pre-stated length of capmaign, from launch date to given deadline:

```
ks_base5$prestated_duration <- as.numeric(ks_base5$deadline - ks_base5$launched_at)
```

- Actual length of campaign, from launch date to outcome date (shown in days):

```
ks_base5$actual_duration <- as.numeric(ks_base5$state_changed_at - ks_base5$launched_at)
```

- Note that the state_changed_at variable (and hence actual_duration) coincides with the time that outcome occurs. That is, when the state of the project changes is when the project becomes successful or fails. We must therefore be careful when commenting on it, but we can also keep it in to validate our expectations that it should not be a well-performing predictor.

Since we have now calculated the new time-series variables, we will now also convert the date formats back to unix time format, to allow for proper numerical processing later on.

```
ks_base5$deadline <- as.numeric(as.POSIXct(ks_base5$deadline, origin="1970-01-01"))
ks_base5$state_changed_at <- as.numeric(as.POSIXct(ks_base5$state_changed_at, origin="1970-01
-01"))
ks_base5$created_at <- as.numeric(as.POSIXct(ks_base5$created_at, origin="1970-01-01"))
ks_base5$launched_at <- as.numeric(as.POSIXct(ks_base5$launched_at, origin="1970-01-01"))
```

2.4 - Splitting the dataset

Here, we will balance the ks_base4 dataset (the cleaned and transformed dataset) before splitting it into ks_train and ks_test.

Checking balance of successful versus unsuccessful outcomes:

```
summary(ks_base5$outcome == 0)
```

```
##    Mode   FALSE    TRUE
## logical   34539   73514
```

Ratio of successful:unsuccessful shows approximately 3.5:7.5

Reducing the size of ks_base via random removal of half of the unsuccessful outcomes. This will improve balance of the outcome variable which will aid in analysis and assist with computer processing limitations later on.

```
set.seed(31421)

ks_base5 <- ks_base5[-sample(which(ks_base5[, "outcome"]==0), .5*sum(ks_base5[,"outcome"]==0
)),]

# new total length:
dim(ks_base5)[1]
```

```
## [1] 71296
```

```
# improved balance:
summary(ks_base5$outcome == 0)
```

```
##    Mode   FALSE    TRUE
## logical   34539   36757
```

Creating the sample group column:

```
set.seed(145679)

ks_base5$sampling <- runif(dim(ks_base5)[1])
```

Splitting the data into train and test datasets (roughly 90/10 split).

```
ks_train_all <- subset(ks_base5, ks_base5$sampling <= 0.91)

ks_test <- subset(ks_base5, ks_base5$sampling > 0.91)

dim(ks_train_all)[1]
```

```
## [1] 64867
```

```
dim(ks_test)[1]
```

```
## [1] 6429
```

Now we further split *ks_train_all* into *ks_train* and *ks_cal*, again using the sampling column and again with a 90/10 split.

```
set.seed(28978)

useForCal <- rbinom(n=dim(ks_train_all)[[1]],size=1,prob=0.91) > 0

ks_train <- subset(ks_train_all,useForCal)
ks_cal <- subset(ks_train_all,!useForCal)

dim(ks_train)[1]
```

```
## [1] 58986
```

```
dim(ks_cal)[1]
```

```
## [1] 5881
```

3 - Classification

We will develop and process the model with ks_train, reprocess it with ks_cal if needed, before testing it with ks_test.

Isolating feature variables

```
ks_vars <- setdiff(colnames(ks_base5),list('sampling','outcome'))

ks_vars
```

```
##  [1] "goal"               "country"            "currency"
##  [4] "deadline"           "state_changed_at"   "created_at"
##  [7] "launched_at"        "backers"            "launched_created"
## [10] "prestated_duration" "actual_duration"
```

3.1 - Null & Saturated models

Producing these models will give us a better idea of the data at hand, and give us the lower and upper performance bounds, respectively.

3.1.1 - Null model

The null model is simply looking at the probability of a successful outcome, assuming all feature variables have zero effect.

```
null_success <- sum(ks_train$outcome==1)/length(ks_train$outcome)
null_success
```

```
## [1] 0.4838436
```

We consider the null model the lower bound: if we cannot utilise the feature variables to outperform this null model prediction, then we should not proceed.

3.1.2 - Saturated model via logistic regression

Stating the formula and model:

```
ks_formula <- as.formula(paste('outcome==1',
                               paste(ks_vars,collapse=' + '),
                               sep=' ~ '))

ks_model <- glm(ks_formula,family=binomial(link='logit'), data=ks_train)
```

For the train and test data sets, applying the model and showing as a new column, “pred”. Then producing a sample showing examples of predicted outcomes.

```
ks_train$pred <- predict(ks_model,newdata=ks_train, type='response')

sample <- ks_train[c(10,250,3000,4521),c('outcome','pred')]
kable(sample)
```

	outcome	pred
20	0	0.0000000
497	0	0.0042534
55880		0.0934541
83860		0.0000000

We are looking at all variables and setting the threshold to 0.5. So, we are stating that the prediction is identifying successful outcomes as > 0.5, and unsuccessful outcomes <= 0.5:

```
cM <- table(truth = ks_train$outcome, prediction = ks_train$pred > 0.5)
kable(cM)
```

	FALSE	TRUE
0	26417	4029
1	4638	23902

The confusion matrix indicates somewhat accurate results, since the true negatives and true positives are far greater than the false negatives and false positives.

3.1.2.1 - Accuracy

Accuracy is the most widely-known measure of classifier performance. We can define it as the fraction of the time that the classifier is correct. Calculated as follows.

We note that we can only use this because we have balanced classes.

```
Accuracy = (cM[1,1]+cM[2,2])/sum(cM)
Accuracy
```

```
## [1] 0.8530668
```

Very high accuracy on the training data; to be expected.

3.1.2.2 - Precision and recall

Precision is the fraction of items that the classifier flags as being in the class, when they are actually in the class (how often a positive indication turns out to be correct).

Recall is the fraction of items in the class that are detected by the classifier.

The F1 score is then a harmonic mean of precision and recall.

```
precision = cM[2,2]/(cM[2,2]+cM[1,2])

recall = cM[2,2]/(cM[2,2]+cM[2,1])

F1 = 2*precision*recall / (precision + recall)

F1
```

```
## [1] 0.846523
```

The high F1 score here indicates high levels of precision and recall from the log regression model.

3.2 - Univariate analysis

3.3 - Processing feature variables

First, we split the ks_train data between categorical and numerical data.

```
cat_vars <- ks_vars[sapply(ks_train[,ks_vars],class) %in% c('factor','character')]

numeric_vars <- ks_vars[sapply(ks_train[,ks_vars],class) %in% c('numeric','integer')]

print(cat_vars)
```

```
## [1] "country"  "currency"
```

```
print(numeric_vars)
```

```
## [1] "goal"               "deadline"           "state_changed_at"
## [4] "created_at"         "launched_at"        "backers"
## [7] "launched_created"   "prestated_duration" "actual_duration"
```

Writing a function to process the variables efficiently:

```
mkPredC <- function(outCol,varCol,appCol) {
  pPos <- sum(outCol=='1')/length(outCol) # being the Null model
  naTab <- table(as.factor(outCol[is.na(varCol)]))
  pPosWna <- (naTab/sum(naTab))[1]
  vTab <- table(as.factor(outCol),varCol)
  pPosWv <- (vTab[1,]+1.0e-3*pPos)/(colSums(vTab)+1.0e-3)
  pred <- pPosWv[appCol]
  pred[is.na(appCol)] <- pPosWna
  pred[is.na(pred)] <- pPos
  pred
}
```

3.3.1 - Categorical variables

Applying the functions for single-variable predictions of categorical variables:

```
for(v in cat_vars) {
  pi <- paste('pred',v,sep='')
  ks_train[,pi] <- mkPredC(ks_train[,'outcome'], ks_train[,v], ks_train[,v])
  ks_cal[,pi] <- mkPredC(ks_train[,'outcome'], ks_train[,v],ks_cal[,v])
  ks_test[,pi] <- mkPredC(ks_train[,'outcome'], ks_train[,v], ks_test[,v])
}
```

Now we can evaluate the area under the ROC curve ('AUC'):

```
calcAUC <- function(predcol,outcol) {
  perf <- performance(prediction(predcol,outcol==1),'auc')
  as.numeric(perf@y.values)
}
```

Note we set the threshold here to 0, to observe all values for each variable:

```
for(v in cat_vars) {
  pi <- paste('pred',v,sep='')
  aucTrain <- calcAUC(ks_train[,pi],ks_train[,'outcome'])
  if(aucTrain>=0) {
    aucCal <- calcAUC(ks_cal[,pi],ks_cal[,'outcome'])
    print(sprintf(
      "%s, trainAUC: %4.3f calibrationAUC: %4.3f",
      pi, aucTrain, aucCal))
  }
}
```

```
## [1] "predcountry, trainAUC: 0.474 calibrationAUC: 0.471"
## [1] "predcurrency, trainAUC: 0.474 calibrationAUC: 0.471"
```

Not ideal as the AUCs for both variables are < 0.5. A higher AUC score is desirable, as it indicates better recollection and specificity.

3.3.2 - Numerical variables

```
for(v in numeric_vars) {
  pi <- paste('pred',v,sep='')
  ks_train[,pi] <- mkPredC(ks_train[,'outcome'], ks_train[,v], ks_train[,v])
  ks_cal[,pi] <- mkPredC(ks_train[,'outcome'], ks_train[,v],ks_cal[,v])
  ks_test[,pi] <- mkPredC(ks_train[,'outcome'], ks_train[,v], ks_test[,v])
}
```

We use the same calcAUC function as previous, and now calculate AUCs for the numerical variables:

```
for(v in numeric_vars) {
  pi <- paste('pred',v,sep='')
  aucTrain <- calcAUC(ks_train[,pi],ks_train[,'outcome'])
  if(aucTrain>=0) {
    aucCal <- calcAUC(ks_cal[,pi],ks_cal[,'outcome'])
    print(sprintf(
      "%s, trainAUC: %4.3f calibrationAUC: %4.3f",
      pi, aucTrain, aucCal))
  }
}
```

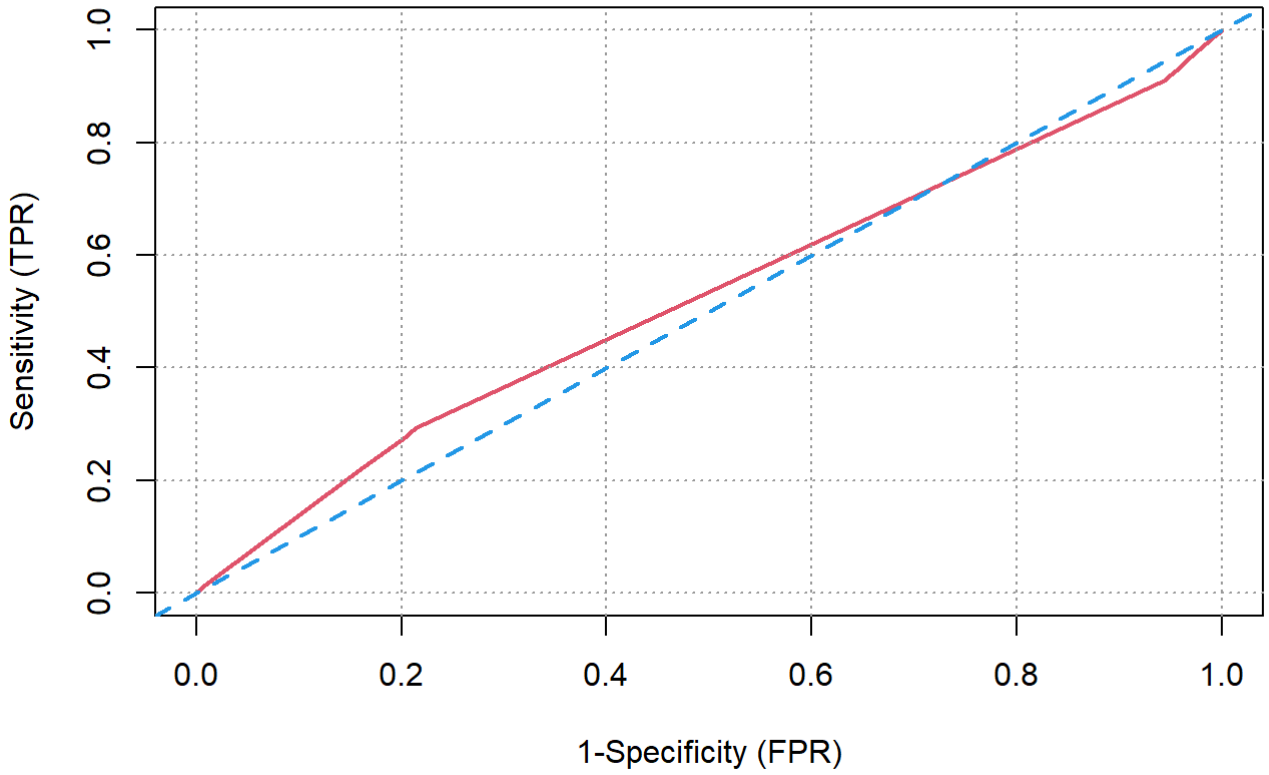
```
## [1] "predgoal, trainAUC: 0.524 calibrationAUC: 0.531"
## [1] "preddeadline, trainAUC: 0.500 calibrationAUC: 0.500"
## [1] "predstate_changed_at, trainAUC: 0.500 calibrationAUC: 0.500"
## [1] "predcreated_at, trainAUC: 0.500 calibrationAUC: 0.500"
## [1] "predlaunched_at, trainAUC: 0.500 calibrationAUC: 0.500"
## [1] "predbackers, trainAUC: 0.498 calibrationAUC: 0.493"
## [1] "predlaunched_created, trainAUC: 0.500 calibrationAUC: 0.499"
## [1] "predprestated_duration, trainAUC: 0.408 calibrationAUC: 0.421"
## [1] "predactual_duration, trainAUC: 0.497 calibrationAUC: 0.487"
```

The numerical variables have performed better than the categorical variables, but AUCs are still not as high as we might like. The best-performing single-variable is *predgoal* due to having the highest combined AUC.

3.4 - Plotting

3.4.1 - Prediction with *predgoal* against the null model

```
library(ROCit)

plot_roc <- function(predcol, outcol){
  ROCit_obj <- rocit(score=predcol,class=outcol=='1')
  plot(ROCit_obj, col = c(2,4),
       legend = FALSE,YIndex = FALSE, values = FALSE)
}

plot_roc(ks_train$predgoal, ks_train[['outcome']])
```

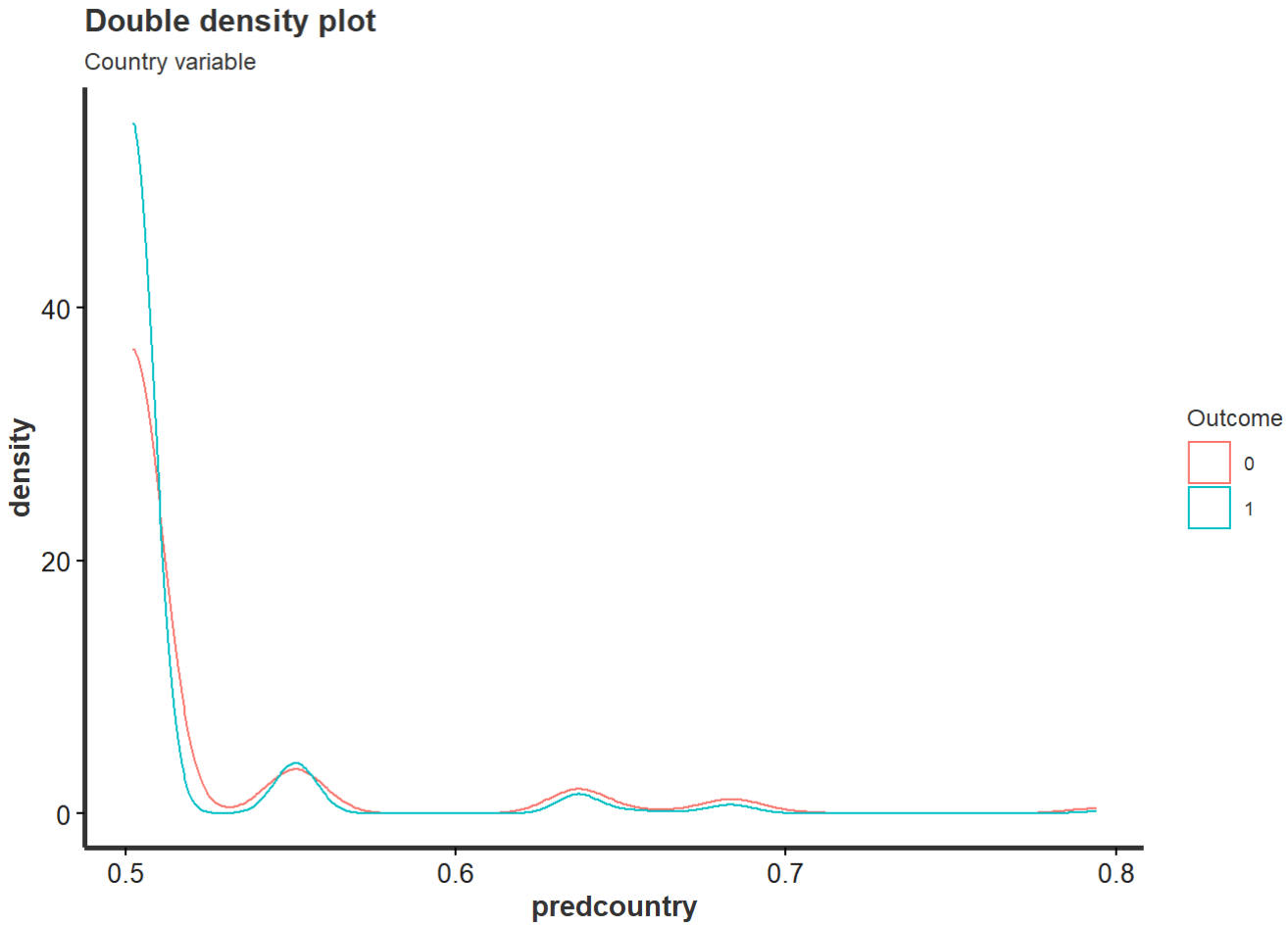


Slight improvement from the null model to the best-performing single-variate model (predgoal). A greater positive difference would have been desirable.

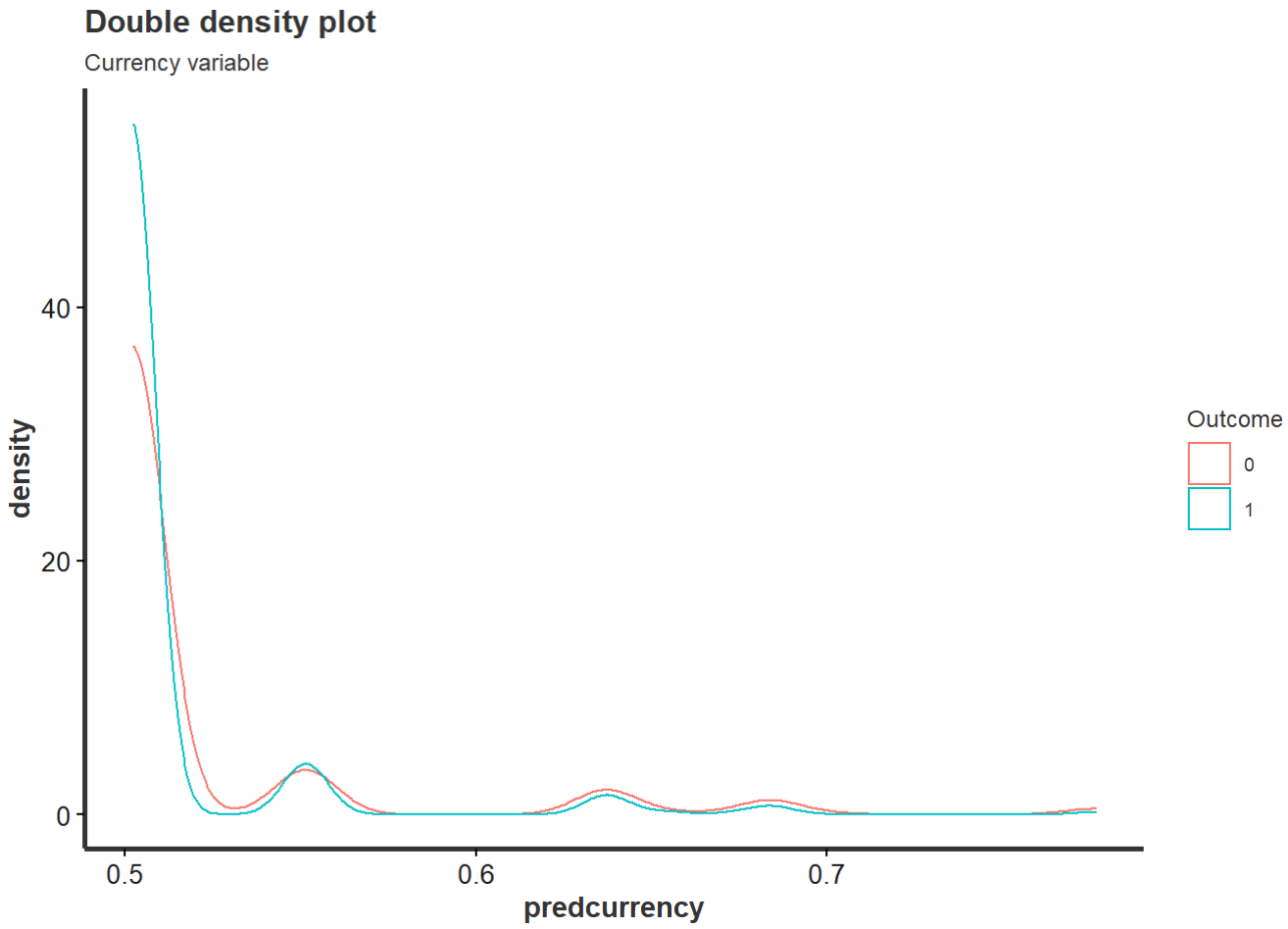
3.4.2 - Double density plots of variables

Below, we plot each single variable in a double-density plot, comparing determination of success and failure. Greater difference between the lines indicating a better-performing variable. Removed variable plots that were non-existant due to AUCs = 0.5.

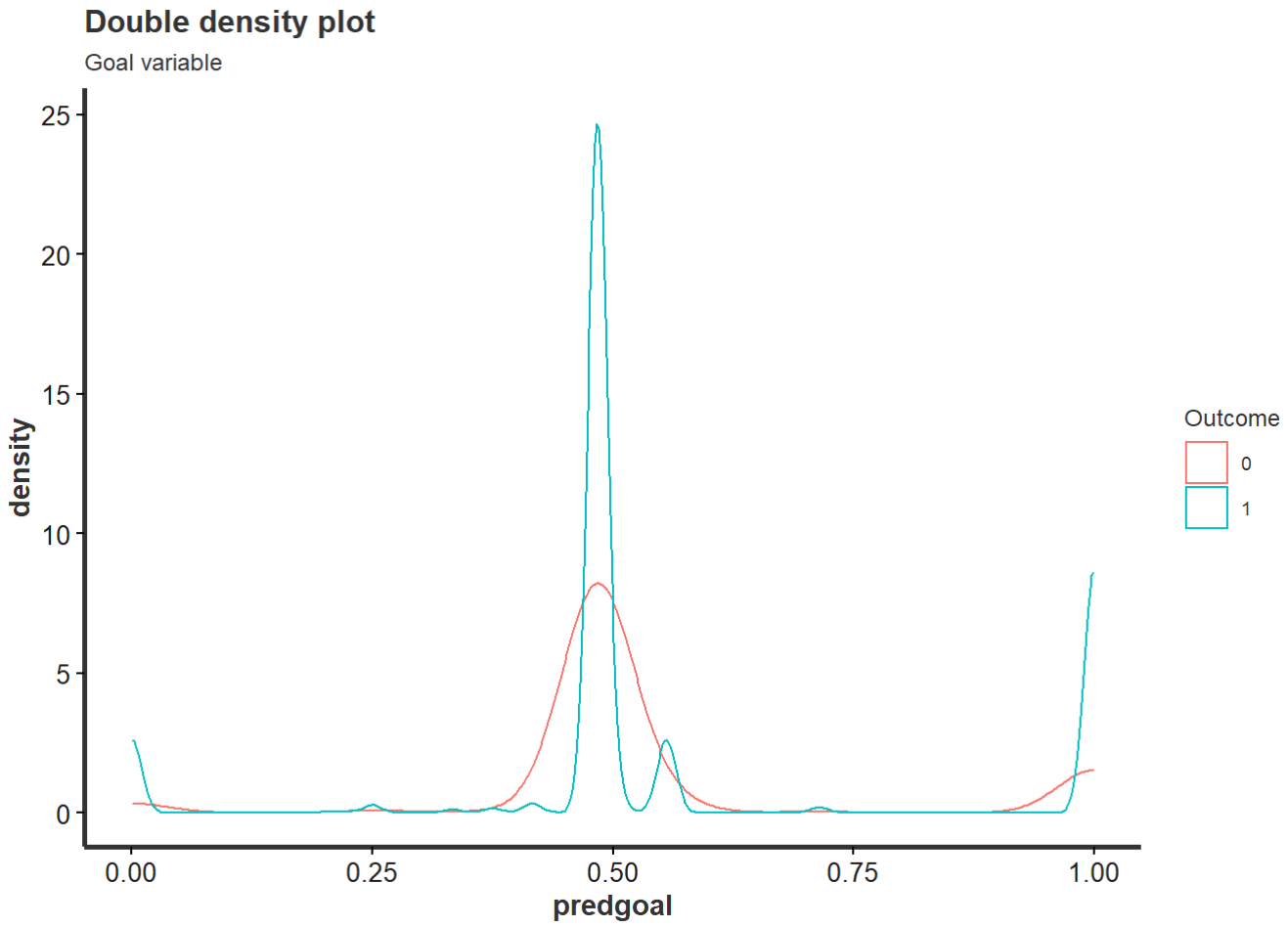
```
ggplot(data=ks_cal) +
  geom_density(aes(x=predcountry,color=as.factor(outcome))) +
  labs(title = "Double density plot",
          subtitle = "Country variable",
          color = "Outcome")
```



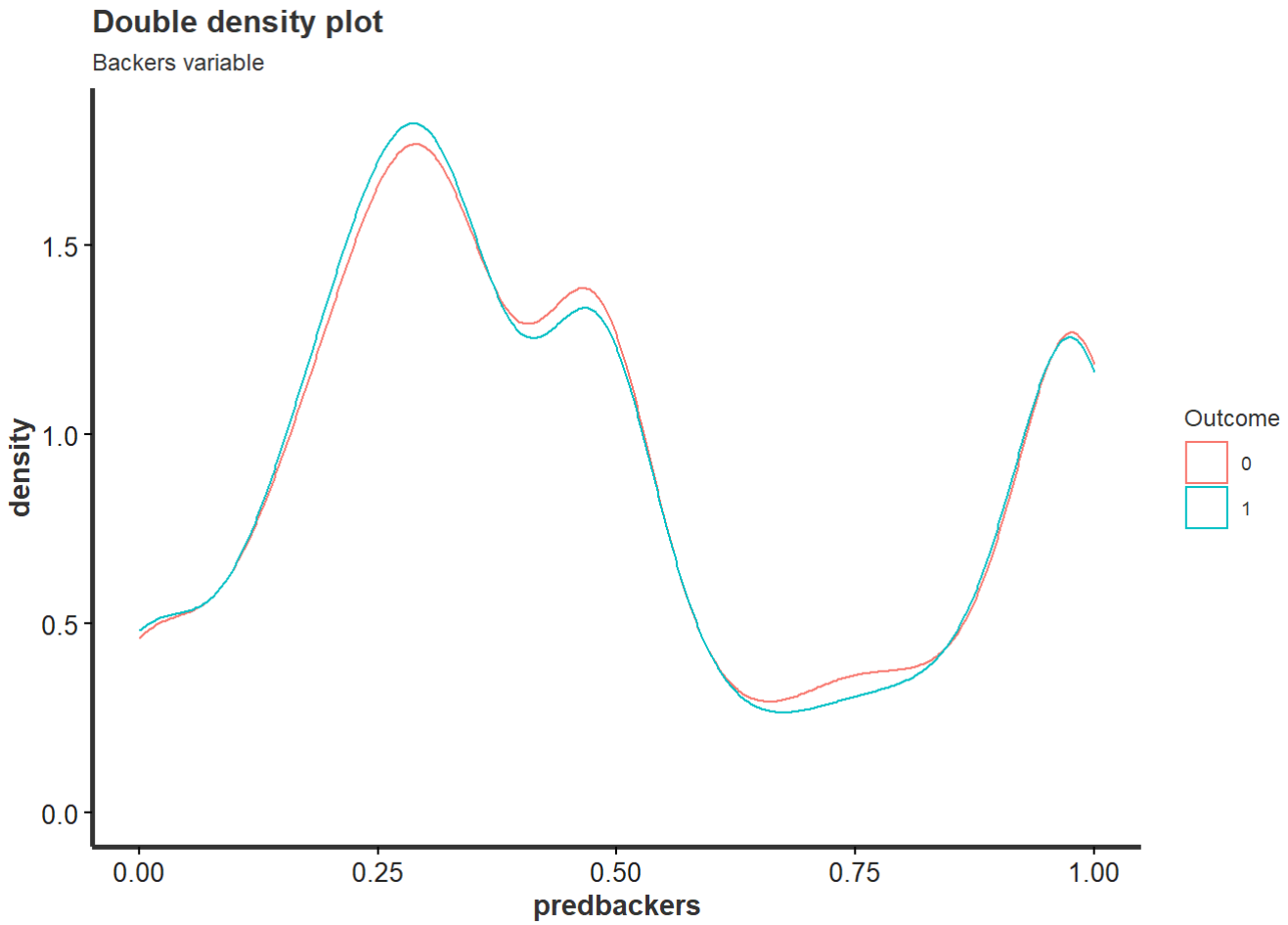

```
ggplot(data=ks_cal) +
  geom_density(aes(x=predcurrency,color=as.factor(outcome))) +
  labs(title = "Double density plot",
          subtitle = "Currency variable",
          color = "Outcome")
```



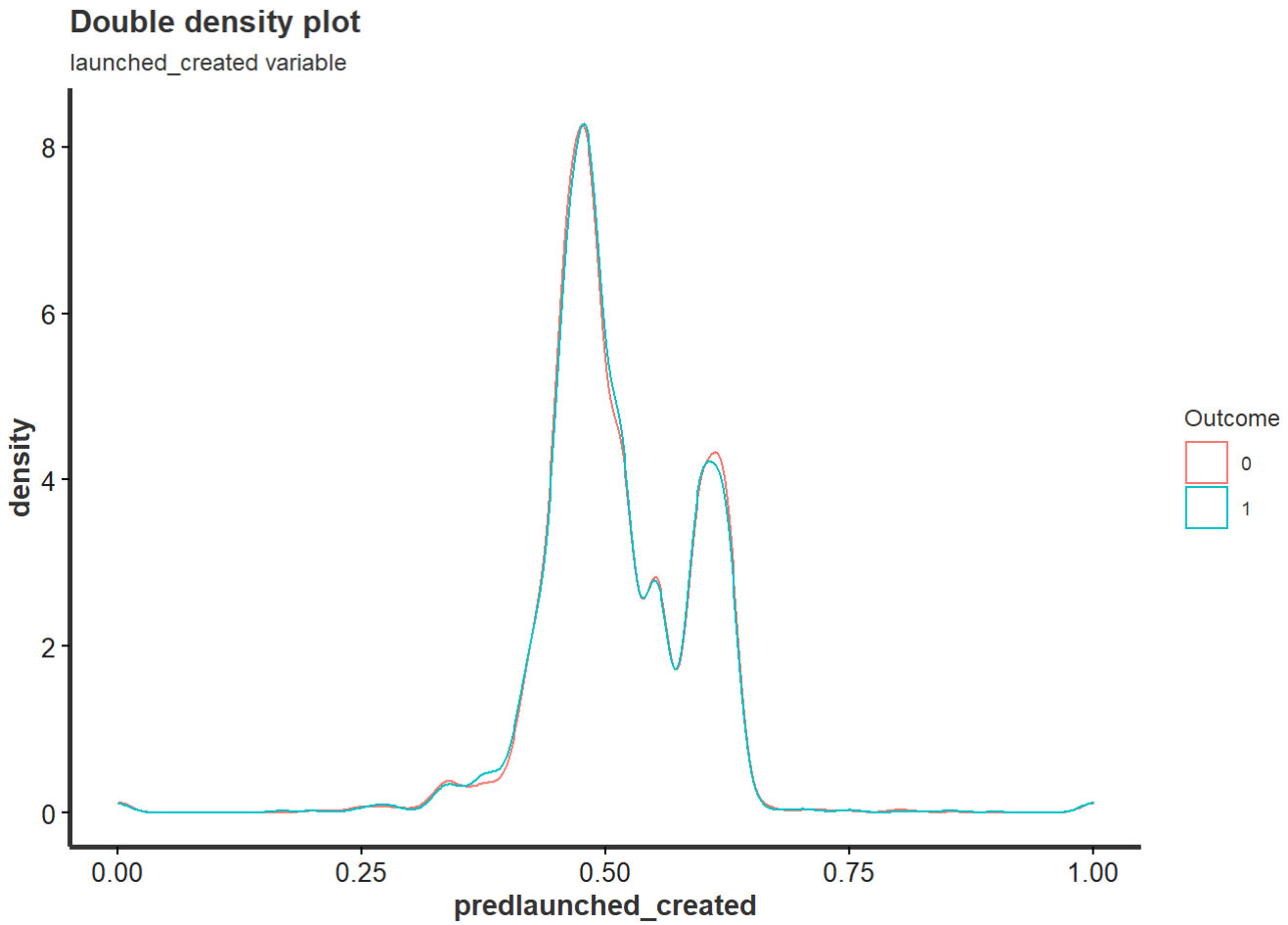

```
ggplot(data=ks_cal) +
  geom_density(aes(x=predgoal,color=as.factor(outcome))) +
  labs(title = "Double density plot",
          subtitle = "Goal variable",
          color = "Outcome")
```



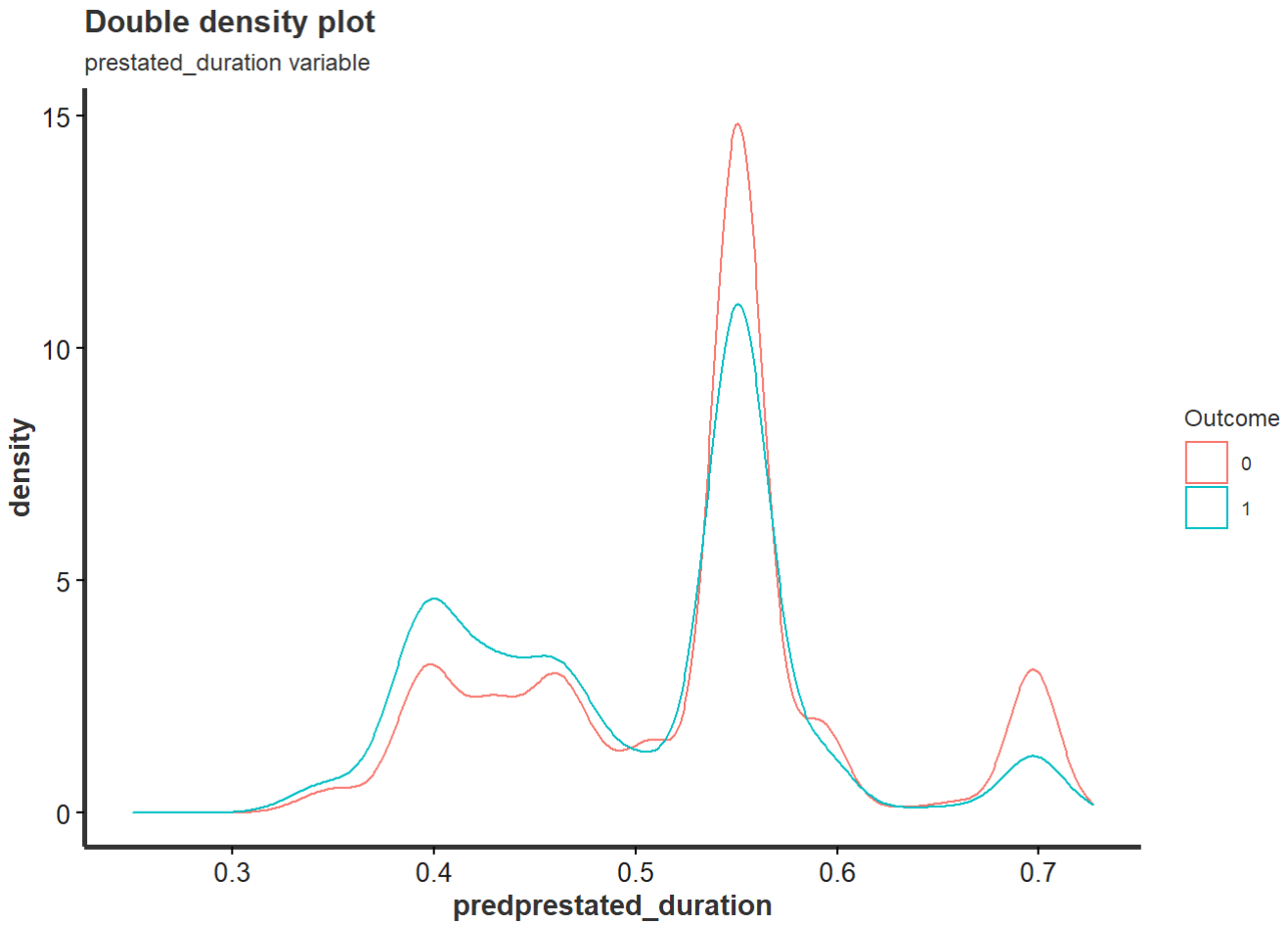

```
ggplot(data=ks_cal) +
  geom_density(aes(x=predbackers,color=as.factor(outcome))) +
  labs(title = "Double density plot",
          subtitle = "Backers variable",
          color = "Outcome")
```



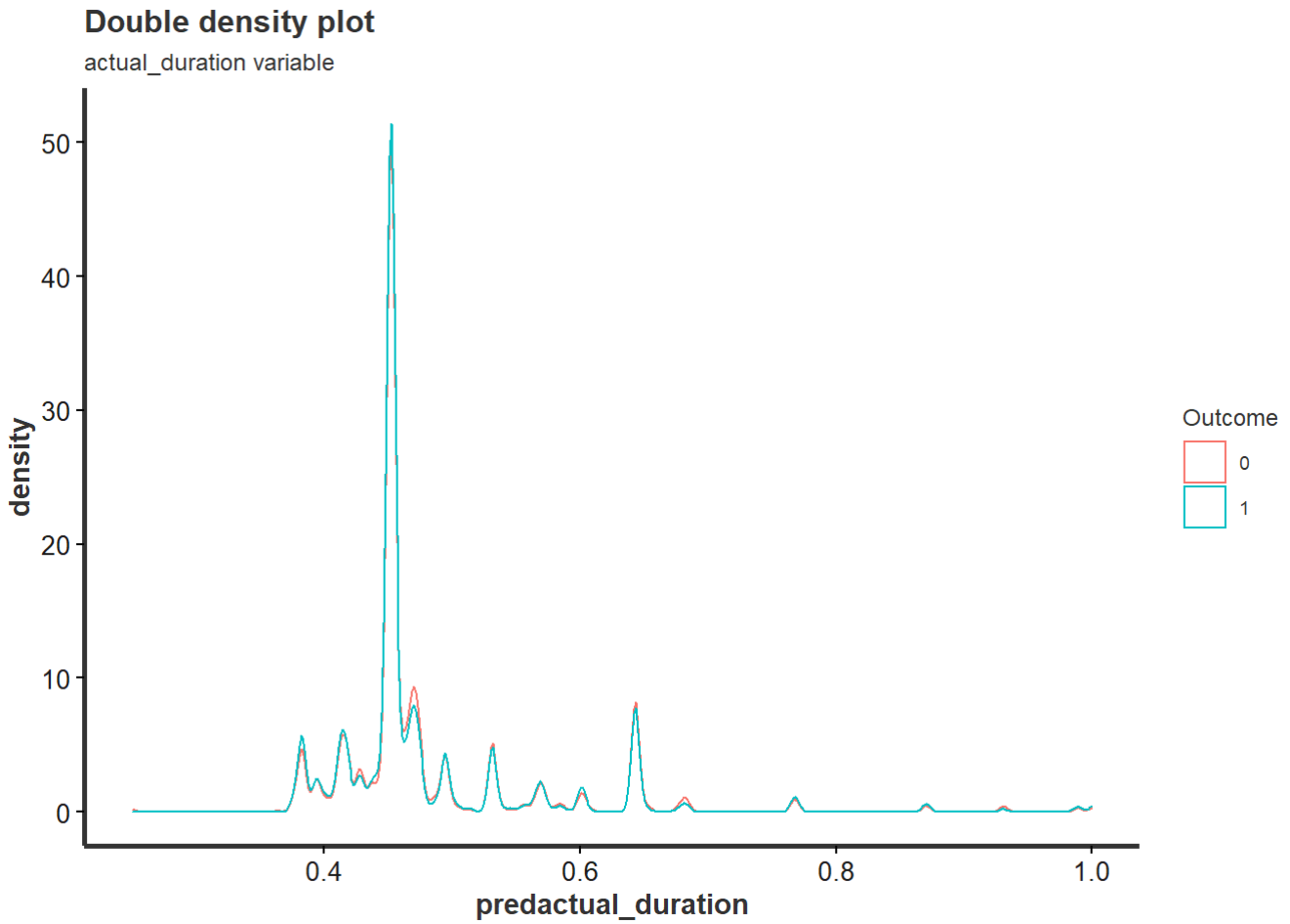

```
ggplot(data=ks_cal) +
  geom_density(aes(x=predlaunched_created,color=as.factor(outcome))) +
  labs(title = "Double density plot",
          subtitle = "launched_created variable",
          color = "Outcome")
```




```
ggplot(data=ks_cal) +
  geom_density(aes(x=predprestated_duration,color=as.factor(outcome))) +
  labs(title = "Double density plot",
          subtitle = "prestated_duration variable",
          color = "Outcome")
```




```
ggplot(data=ks_cal) +
  geom_density(aes(x=predactual_duration,color=as.factor(outcome))) +
  labs(title = "Double density plot",
          subtitle = "actual_duration variable",
          color = "Outcome")
```



The double density plots show *predgoal* to be the best-performer, followed by *predprestated_duration.*

Notice how the *predactual_duration* variable has virtually no distance between success and failure, indicating a poor-performing variable, which makes sense going back to previous commentary: it contains a trait (state_changed_at) that is defined at the same time as the outcome.

3.5 - All results in a dataframe

```
all.vars <- c(cat_vars, numeric_vars)
models.auc <- data.frame(model.type = 'univariate',
                         model.name = all.vars,
                         train.auc = sapply(all.vars, function(v){pi <- paste('pred',v,sep=''
); calcAUC(ks_train[,pi], ks_train[,'outcome'])}),
                         cal.auc = sapply(all.vars, function(v){pi <- paste('pred',v,sep='');
calcAUC(ks_cal[,pi],ks_cal[,'outcome'])}))

kable(models.auc[order(-models.auc$cal.auc), ])
```

	model.type	model.name	train.auc	cal.auc
goal	univariate	goal	0.5238899	0.5314971
deadline	univariate	deadline	0.5000000	0.5000000
state_changed_at	univariate	state_changed_at	0.5000000	0.5000000
created_at	univariate	created_at	0.5000000	0.5000000
launched_at	univariate	launched_at	0.5000000	0.5000000
launched_created	univariate	launched_created	0.4997789	0.4988725
backers	univariate	backers	0.4982524	0.4933403
actual_duration	univariate	actual_duration	0.4969359	0.4870778
currency	univariate	currency	0.4744382	0.4712247

	model.type	model.name	train.auc	cal.auc
country	univariate	country	0.4744356	0.4712207
prestated_duration	univariate	prestated_duration	0.4077166	0.4206504

3.6 - Feature variable evaluation

3.6.1 - Log-likelihood method

First, we compute the log likelihood of the null model, giving us a reference point.

```
# Define a convenience function to compute log likelihood.

logLikelihood <- function(outCol,predCol) {
  sum(ifelse(outCol=='1',log(predCol),log(1-predCol)))
}
# Compute the base rate of a NULL model

baseRateCheck <- logLikelihood(
  ks_cal[,'outcome'], sum(ks_cal[,'outcome']=='1')/length(ks_cal[,'outcome'])
)

baseRateCheck
```

```
## [1] -4074.984
```

Now, we pick feature variables based on deviance improvement from the null model. A greater deviance improvement (or a smaller deviance deterioration), the better.

- Running for categorical variables:

```
selVars <- c()
minStep <- -200

for(v in cat_vars) {
  pi <- paste('pred',v,sep='')
  liCheck <- 2*((logLikelihood(ks_cal[,'outcome'],ks_cal[,pi]) - baseRateCheck))

if(liCheck>minStep) {
  print(sprintf("%s, calibrationScore: %g",pi,liCheck))
  selVars <- c(selVars,pi)
  }
}
```

```
## [1] "predcountry, calibrationScore: -167.727"
## [1] "predcurrency, calibrationScore: -166.27"
```

- Running for numerical variables:

```
for(v in numeric_vars) {
  pi <- paste('pred',v,sep='')
  liCheck <- 2*((logLikelihood(ks_cal[,'outcome'],ks_cal[,pi]) - baseRateCheck))

if(liCheck>minStep) {
  print(sprintf("%s, calibrationScore: %g",pi,liCheck))
  selVars <- c(selVars,pi)
  }
}
```

```
## [1] "preddeadline, calibrationScore: -0.633896"
## [1] "predstate_changed_at, calibrationScore: -0.633896"
## [1] "predcreated_at, calibrationScore: -0.633896"
## [1] "predlaunched_at, calibrationScore: -0.633896"
```

Note how we have set the minStep to -200, so we are only showing variables with a deviance deterioration of less than 200.We are choosing variables with the least deterioration:

```
print(selVars)
```

```
## [1] "predcountry"          "predcurrency"         "preddeadline"
## [4] "predstate_changed_at" "predcreated_at"       "predlaunched_at"
```

3.7 - Multivariate models

3.8 - k-Nearest Neighbour analysis

Another multivariate analysis method is k-Nearest Neighbour, predicting properties of data based on other data that is most similar.

```
# function for kNN processing
nK <- 51

#forced to use a further reduced ks_train due to computer limitations.
ks_train_reduced <- subset(ks_train, ks_train$sampling > 0.9)

#ks_cal_reduced <- subset(ks_cal, ks_cal$sampling > 0.9)

#ks_test_reduced <- subset(ks_test, ks_test$sampling > 0.9)

knnTrain <- ks_train_reduced[,selVars]
knnCl <- ks_train_reduced[,'outcome']==1

knnPred <- function(df) {
  knnDecision <- knn(knnTrain,df,knnCl,k=nK,prob=T,use.all=T)
  ifelse(knnDecision==TRUE,
         attributes(knnDecision)$prob,
         1-(attributes(knnDecision)$prob))
}

print(calcAUC(knnPred(ks_train_reduced[,selVars]),ks_train_reduced[,'outcome']))
```

```
## [1] 0.5188841
```

```
print(calcAUC(knnPred(ks_cal[,selVars]),ks_cal[,'outcome']))
```

```
## [1] 0.5125749
```

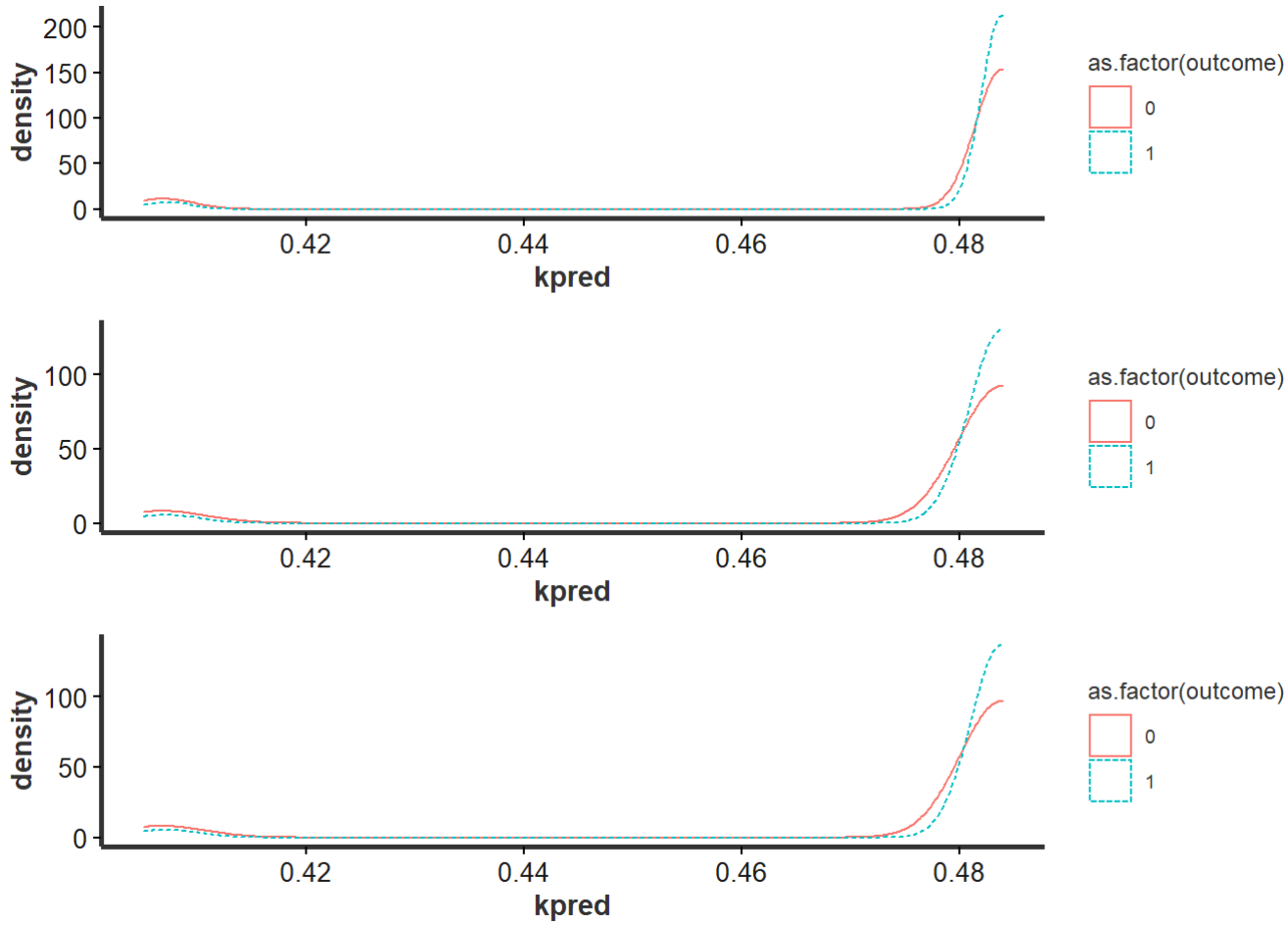
```
print(calcAUC(knnPred(ks_test[,selVars]),ks_test[,'outcome']))
```

```
## [1] 0.5134791
```

The higher AUC values, compared to those produced by univariate modelling, indicate that the kNN model is the better performer. The similar AUC values between the train, calibration and test datasets imply that there is also minimal overfitting - a positive trait.

3.8.1 - Visualising kNN

```
ks_train$kpred <- knnPred(ks_train[,selVars])
ks_cal$kpred <- knnPred(ks_cal[,selVars])
ks_test$kpred <- knnPred(ks_test[,selVars])

co1 <- ggplot(data=ks_train) +
  geom_density(aes(x=kpred,
                   color=as.factor(outcome),
                   linetype=as.factor(outcome)))

co2 <- ggplot(data=ks_cal) +
  geom_density(aes(x=kpred,
                   color=as.factor(outcome),
                   linetype=as.factor(outcome)))

co3 <- ggplot(data=ks_test) +
  geom_density(aes(x=kpred,
                   color=as.factor(outcome),
                   linetype=as.factor(outcome)))

grid.arrange(co1, co2, co3, nrow = 3)
```



Graphs show similar performance between the 3 sets, as expected.

3.8.2 - Logistic regression on selected variables

```
f <- paste('outcome','==1 ~ ',paste(selVars,collapse=' + '),sep='')
gmodel <- glm(as.formula(f),data=ks_train,
              family=binomial(link='logit'))

print(calcAUC(predict(gmodel,newdata=ks_train),ks_train[,'outcome']))
```

```
## [1] 0.5255641
```

```
print(calcAUC(predict(gmodel,newdata=ks_test),ks_test[,'outcome']))
```

```
## [1] 0.5266839
```

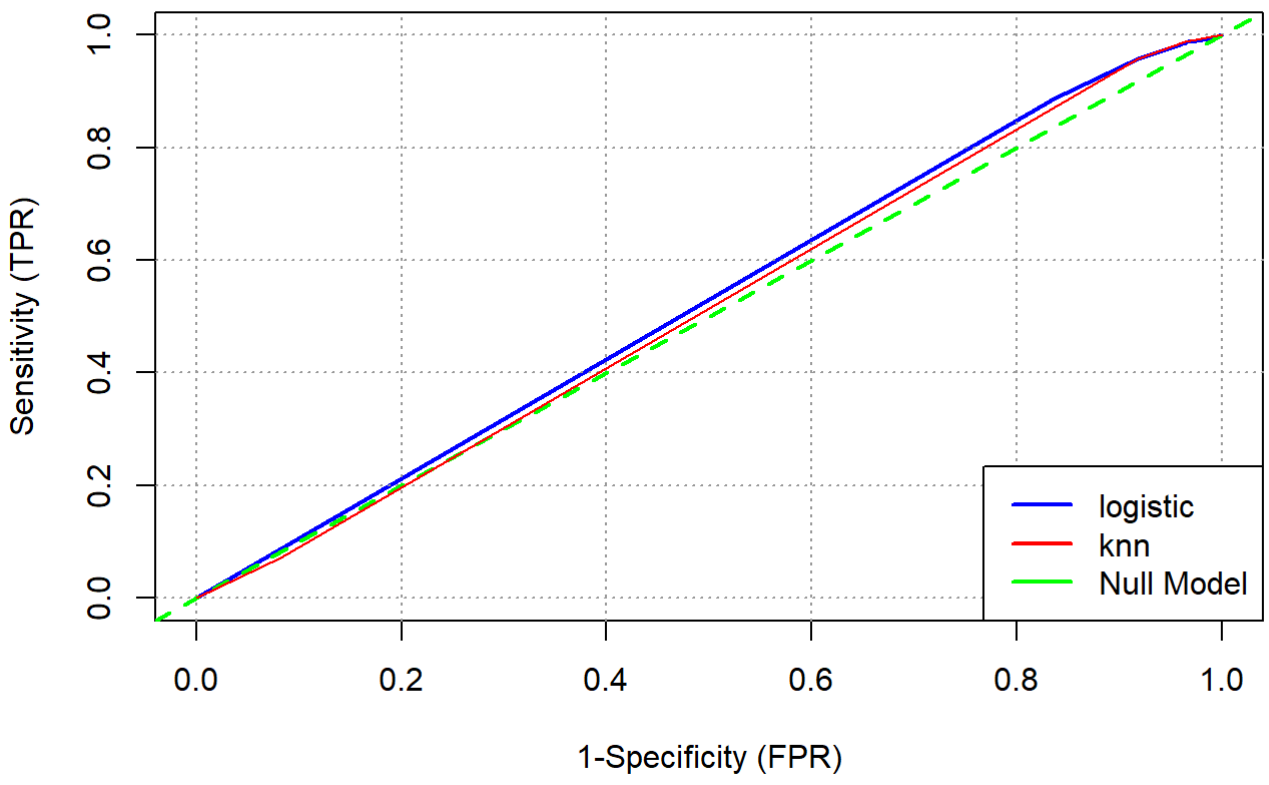
```
print(calcAUC(predict(gmodel,newdata=ks_cal),ks_cal[,'outcome']))
```

```
## [1] 0.5287761
```

AUCs indicate similar performance to kNN modelling.

3.8.3 - Plotting logistic regression model against kNN model (selected variables only):

```
plot_roc <- function(predcol1, outcol1, predcol2, outcol2){
  roc_1 <- rocit(score=predcol1,class=outcol1==1)
  roc_2 <- rocit(score=predcol2,class=outcol2==1)
  plot(roc_1, col = c("blue","green"),
       lwd = 3,
       legend = FALSE,
       YIndex = FALSE,
       values = TRUE)

lines(roc_2$TPR ~ roc_2$FPR,
      lwd = 1,
      col = c("red","green"))

legend("bottomright",
       col = c("blue","red", "green"),
       c("logistic", "knn", "Null Model"),
       lwd = 2)
}

pred_gmodel_roc <- predict(gmodel,newdata=ks_test)

pred_knn_roc <- knnPred(ks_test[,selVars])

plot_roc(pred_gmodel_roc, ks_test[['outcome']],
         pred_knn_roc, ks_test[['outcome']])
```



Comparing to the previous univariate graph of the best-performer, the logistic regression and kNN models perform better.

3.9 - Decision tree modelling (all variables)

First, we set helper functions to evaluate and display results of models, which will be used for each model later on.

Core processing:

```
performanceMeasures <- function(pred, truth, name = "model") {
  dev.norm <- -2 * logLikelihood(truth, pred)/length(pred)
  ctable <- table(truth =='1', pred == (pred > -1000))
  accuracy <- sum(diag(ctable)) / sum(ctable)
  precision <- ctable[1, 1] / sum(ctable[, 1])
  recall <- ctable[1, 1] / sum(ctable[1, ])
  f1 <- 2 * precision * recall / (precision + recall)
  data.frame(model = name, precision = precision, recall = recall, f1 = f1, dev.norm = dev.no
rm)
}
```

Improving aesthetics of evaluations:

```
panderOpt <- function(){
  panderOptions("plain.ascii", TRUE)
  panderOptions("keep.trailing.zeros", TRUE)
  panderOptions("table.style", "simple")
}

pretty_perf_table <- function(model,training,test){
  # Option setting for Pander
  panderOpt()
  perf_justify <- "lrrrr"
  # comparing performance on training vs. test
  pred_train <-predict(model,newdata=training)
  truth_train <- training[,'outcome']
  pred_test<-predict(model,newdata=test)
  truth_test <- test[,'outcome']

  trainperf_tree <- performanceMeasures(pred_train,truth_train,"training")
  testperf_tree <- performanceMeasures(pred_test,truth_test, "test")

  perftable <- rbind(trainperf_tree, testperf_tree)

  pandoc.table(perftable, justify = perf_justify)
}
```

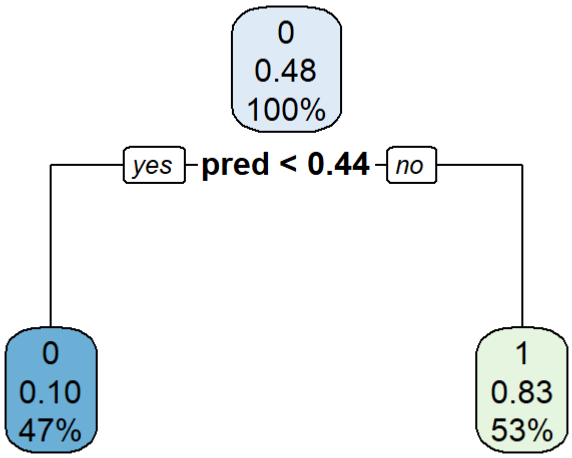
For plotting:

```
library(ROCit)
plot_roc <- function(predcol1, outcol1, predcol2, outcol2){
  roc_1 <- rocit(score=predcol1,class=outcol1==1)
  roc_2 <- rocit(score=predcol2,class=outcol2==1)
  plot(roc_1, col = c("blue","green"), lwd = 3,
       legend = FALSE,YIndex = FALSE, values = TRUE)
  lines(roc_2$TPR ~ roc_2$FPR, lwd = 1, col = c("red","green"))
  legend("bottomright", col = c("blue","red", "green"), c("Test Data", "Training Data", "Null
Model"), lwd = 2)
}
```

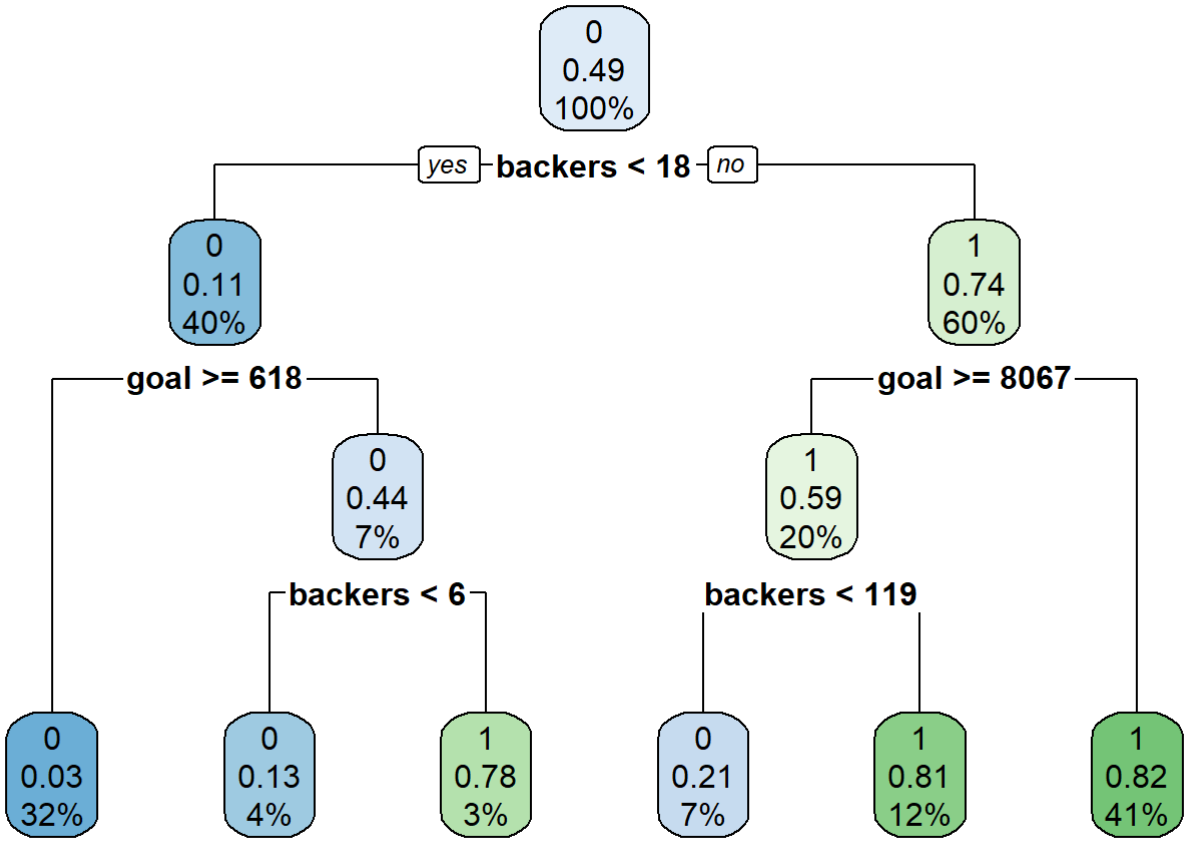
3.9.1 - Basic decision tree:

The decision tree model uses piece-wise constants - it splits the data into pieces and then uses a simple memorised constant on each piece. Setting the order of each piece of the data is important.

```
dt_1 <- rpart(formula = outcome ~ ., data = ks_train)

rpart.plot(dt_1)
```




```
dt_2 <- rpart(formula = outcome ~ ., data = ks_cal)

rpart.plot(dt_2)
```



Processing the basic rpart decision tree for ks_train and ks_cal produces lacklustre information.

3.9.2 - Re-processing for all variables, predVariables, and selected variables (selVars)

For each process of using all (i) variables, (ii) predicted variables and (iii) selected variables, we are processing the AUCs, then showing the analysis in tabular form with precision, recall, f1 and deviance from the norm, and then plotting and printing each tree.

3.9.2.1 - (i) All variables:

3.9.2.1.1 - Summarising all variables

```
fV <- paste('outcome',' == 1 ~ ', paste(c(cat_vars,numeric_vars), collapse=' + '), sep='')

tmodel_all <- rpart(fV, data=ks_train)

print(calcAUC(predict(tmodel_all,newdata=ks_train), ks_train[,'outcome']))
```

```
## [1] 0.9093684
```

```
print(calcAUC(predict(tmodel_all,newdata=ks_test), ks_test[,'outcome']))
```

```
## [1] 0.913974
```

```
print(calcAUC(predict(tmodel_all,newdata=ks_cal), ks_cal[,'outcome']))
```

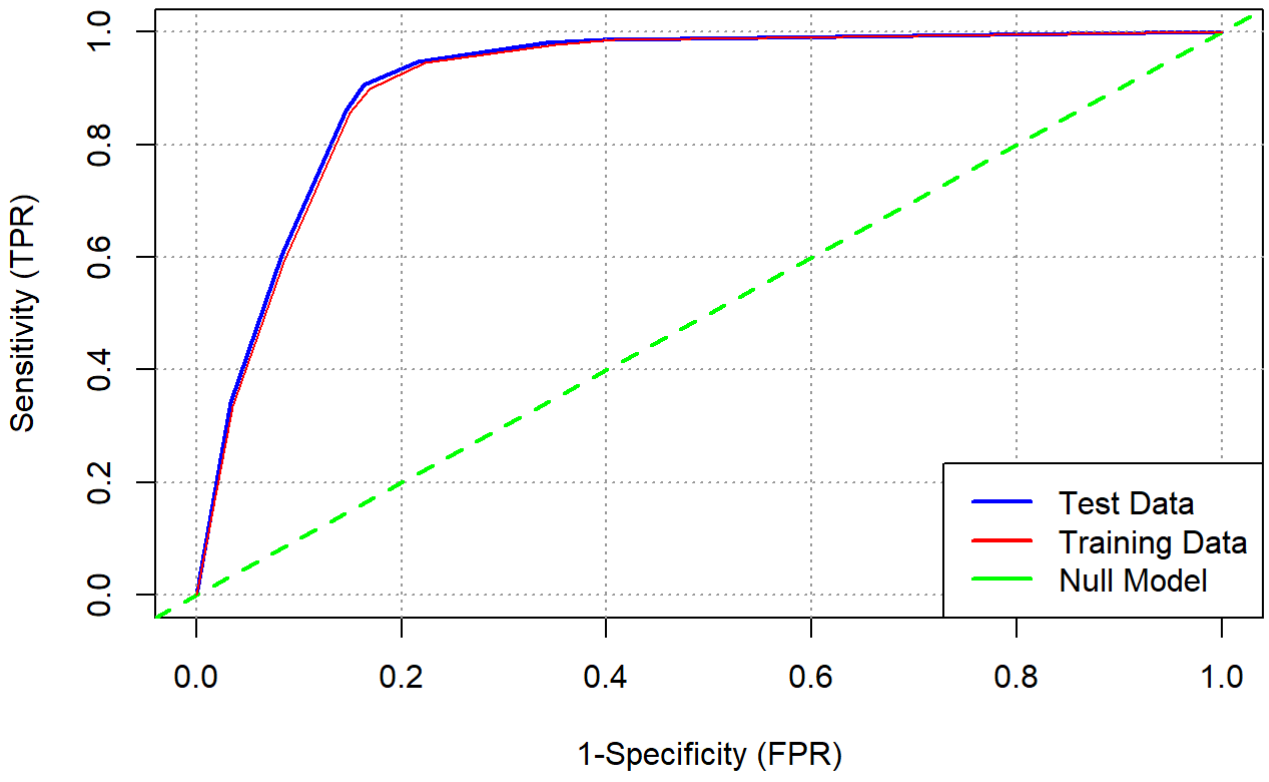
```
## [1] 0.9029082
```

```
pretty_perf_table(tmodel_all, ks_train, ks_test)
```

```
##
##
## model        precision   recall       f1   dev.norm
## ---------- ----------- -------- -------- ----------
## training        0.5162        1   0.6809     0.6980
## test            0.5142        1   0.6792     0.6769
```

3.9.2.1.2 - Plotting tmodel_all

```
pred_test_roc <-predict(tmodel_all,newdata=ks_test)
pred_train_roc<-predict(tmodel_all,newdata=ks_train)

plot_roc(pred_test_roc, ks_test[['outcome']], pred_train_roc, ks_train[['outcome']])
```



We see that the decision tree analysis has performed vastly better in both the test and training datasets compared to the null model. Again, no signs of overfitting as the test and training data perform similarly.

3.9.2.1.3 - Printing tmodel_all

```
print(tmodel_all)
```

```
## n= 58986
##
## node), split, n, deviance, yval
##       * denotes terminal node
##
##  1) root 58986 14731.1000 0.48384360
##    2) backers< 14.5 22259  1700.4100 0.08333708
##      4) goal>=696.5 18466   399.9411 0.02214881 *
##      5) goal< 696.5 3793   894.7435 0.38122860
##       10) backers< 4.5 1971   198.5429 0.11364790 *
##       11) backers>=4.5 1822   402.4149 0.67069150 *
##    3) backers>=14.5 36727  7296.2880 0.72657720
##      6) backers< 50.5 14323  3457.8980 0.59261330
##       12) goal>=3162.5 4799   757.7012 0.19649930 *
##       13) goal< 3162.5 9524  1567.7820 0.79220920 *
##      7) backers>=50.5 22404  3417.0140 0.81222100
##       14) goal>=6070 11771  2297.2410 0.73417720
##         28) backers< 110.5 2923   721.3794 0.44338010 *
##         29) backers>=110.5 8848  1247.0270 0.83024410 *
##       15) goal< 6070 10633   968.7097 0.89861750 *
```

3.9.2.2 - (ii) Predicted variables

3.9.2.2.1 - Summarising predicted variables

```
tVars <- paste('pred',c(cat_vars, numeric_vars),sep='')
fV2 <- paste('outcome',' ==1 ~ ', paste(tVars,collapse=' + '),sep='')

tmodel_pred <- rpart(fV2,data=ks_train)

print(calcAUC(predict(tmodel_pred,newdata=ks_train), ks_train[,'outcome']))
```

```
## [1] 0.5692044
```

```
print(calcAUC(predict(tmodel_pred,newdata=ks_test), ks_test[,'outcome']))
```

```
## [1] 0.5707428
```

```
print(calcAUC(predict(tmodel_pred,newdata=ks_cal), ks_cal[,'outcome']))
```

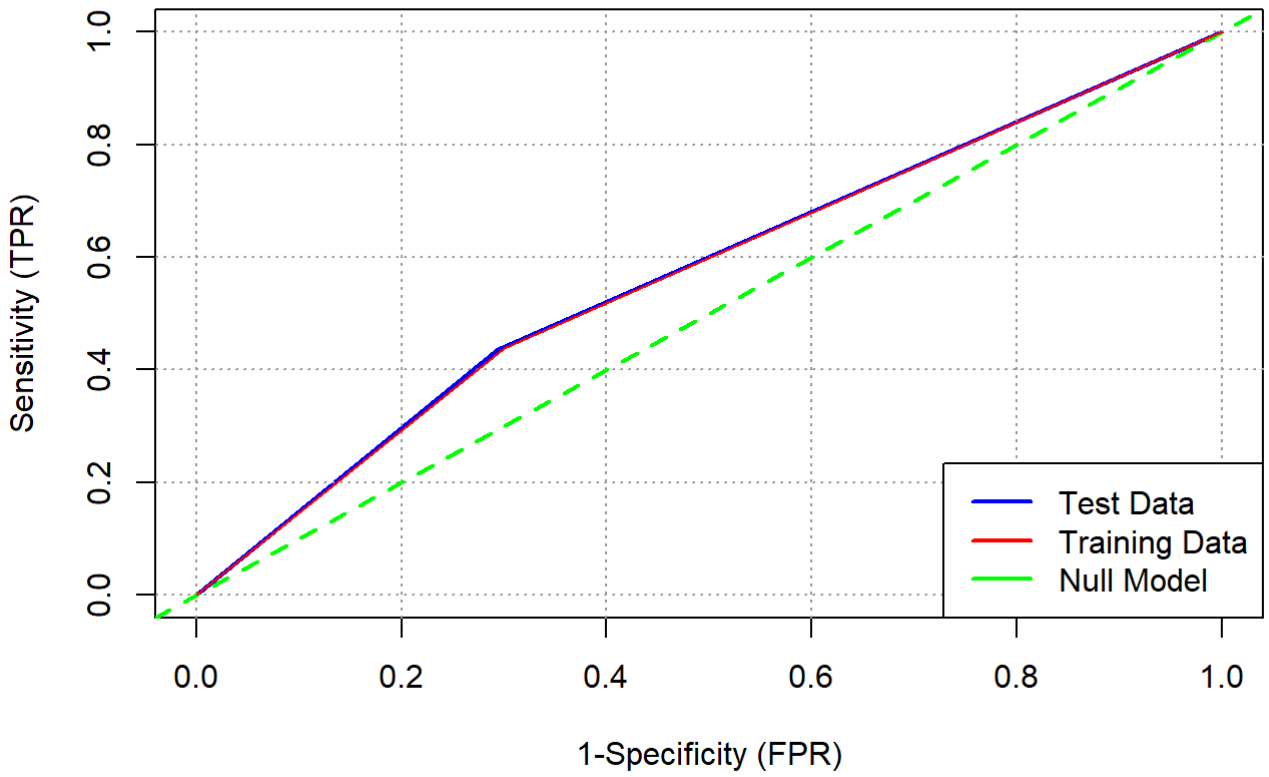
```
## [1] 0.5638406
```

```
pretty_perf_table(tmodel_pred, ks_train, ks_test)
```

```
##
##
## model        precision   recall       f1   dev.norm
## ---------- ----------- -------- -------- ----------
## training        0.5162        1   0.6809      1.365
## test            0.5142        1   0.6792      1.364
```

3.9.2.2.2 - Plotting tmodel_pred

```
pred_test_roc <-predict(tmodel_pred,newdata=ks_test)
pred_train_roc<-predict(tmodel_pred,newdata=ks_train)

plot_roc(pred_test_roc, ks_test[['outcome']], pred_train_roc, ks_train[['outcome']])
```



We see that the decision tree analysis has performed better in both the test and training datasets compared to the null model. Again, no signs of overfitting as the test and training data perform similarly.

3.9.2.2.3 - Printing tmodel_pred:

```
print(tmodel_pred)
```

```
## n= 58986
##
## node), split, n, deviance, yval
##       * denotes terminal node
##
## 1) root 58986 14731.100 0.4838436
##   2) predprestated_duration>=0.4925136 37328  9145.005 0.4292220 *
##   3) predprestated_duration< 0.4925136 21658  5282.783 0.5779850 *
```

3.9.2.3 - (iii) Selected variables:

```
f <- paste('outcome','==1 ~ ', paste(selVars,collapse=' + '),sep='')

tmodel_selected <- rpart(f,data=ks_train, control=rpart.control(cp=0.001,minsplit=1000, minbu
cket=1000,maxdepth=5))

print(calcAUC(predict(tmodel_selected,newdata=ks_train), ks_train[,'outcome']))
```

```
## [1] 0.5186865
```

```
print(calcAUC(predict(tmodel_selected,newdata=ks_test), ks_test[,'outcome']))
```

```
## [1] 0.5198526
```

```
print(calcAUC(predict(tmodel_selected,newdata=ks_cal), ks_cal[,'outcome']))
```

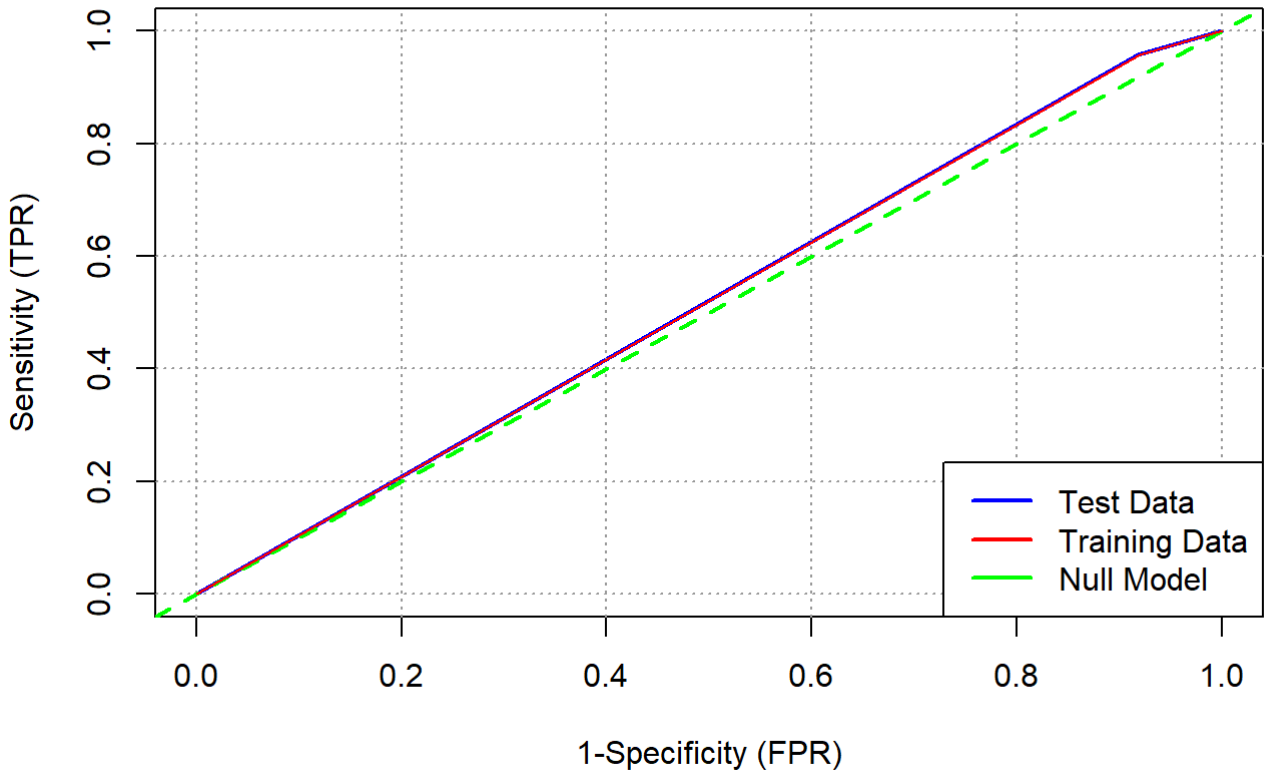
```
## [1] 0.520494
```

```
pretty_perf_table(tmodel_selected, ks_train, ks_test)
```

```
##
##
## model        precision   recall       f1   dev.norm
## ---------- ----------- -------- -------- ----------
## training        0.5162        1   0.6809      1.379
## test            0.5142        1   0.6792      1.379
```

Plotting tmodel_selected:

```
pred_test_roc <-predict(tmodel_selected,newdata=ks_test)
pred_train_roc<-predict(tmodel_selected,newdata=ks_train)

plot_roc(pred_test_roc, ks_test[['outcome']], pred_train_roc, ks_train[['outcome']])
```



We see that the decision tree analysis has performed evenly compared to the null model. Again, no signs of overfitting as the test and training data perform similarly.

Printing tmodel_selected:

```
print(tmodel_selected)
```

```
## n= 58986
##
## node), split, n, deviance, yval
##       * denotes terminal node
##
## 1) root 58986 14731.100 0.4838436
##   2) predcountry>=0.5941073 3649   810.444 0.3329679 *
##   3) predcountry< 0.5941073 55337 13832.120 0.4937926 *
```

3.9.3 - Decision tree conclusion

So we have the best performance using all variables (to be expected), followed by the pred_variables and then the selected variables. We have determined this by seeking higher values of precision, recall (and thus f1), and deviance from the norm, to indicate better performance. By printing the tree we also observe its path and thought process.

4 - Clustering

Clustering allows us to observe whether any of our variables are intrinsically grouped. The aim is to maximise the inter-cluster distance and minimise the intra-cluster distance.

4.1 - Data preparation

Scaling the data and attaining means and standard deviations of numerical values (note that for clustering we are forced to use a further-reduced dataset due to computation limitations):

```
# reducing ks_base5 dataset:
set.seed(234562)
ks_base5_reduced <- subset(ks_base5, ks_base5$sampling > 0.8)

# pre-processing only numerical variables:
vars.to.use <- colnames(ks_base5_reduced[c(1,4:8,10:12)])

pmatrix <- scale(ks_base5_reduced[c(1,4:8,10:12)])

# The mean value: scaled:center
pcenter <- attr(pmatrix, "scaled:center")
# The standard deviation: scaled:scale
pscale <- attr(pmatrix, "scaled:scale")
```

Applying distance function:

```
d <- daisy(pmatrix, metric="gower")
```

We cannot use the Euclidean method as we have some categorical data. The gower method of distance as it considers both quantitative and qualitative data.

4.2 - Forming clusters

We choose k = 8 number of clusters for reasons later shown. Showing and running code, but not showing output due to spatial consumption on RMD file.

4.3 - Plotting clusters

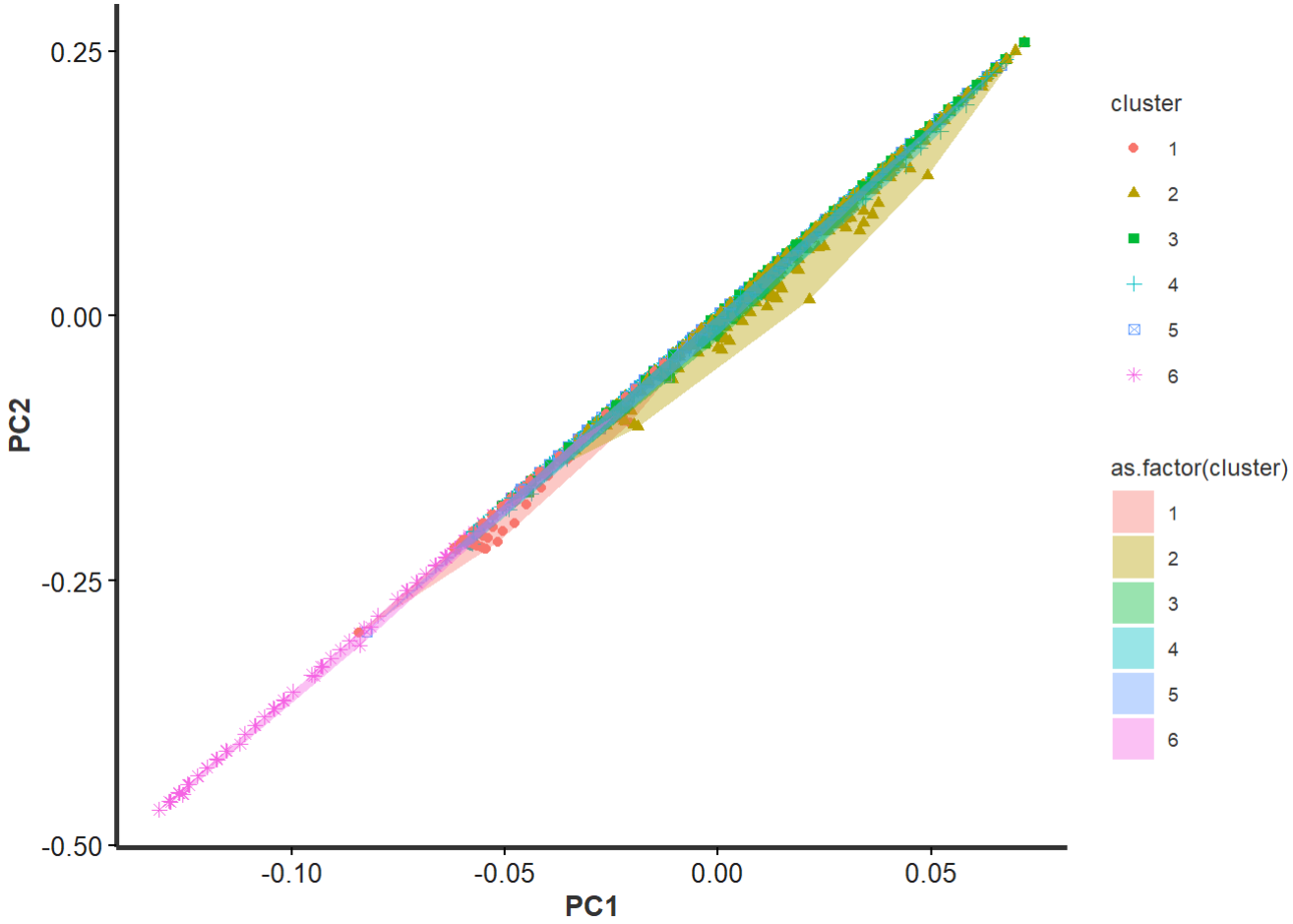
Helper functions for ggplot of clusters:

```
# Calculate the principle components of pmatrix
# Setting k = 6 for ggplot plotting purposes only:
groups <- cutree(pfit, k=6)

princ <- prcomp(pmatrix)
nComp <- 2
project <- as.data.frame(predict(princ, newdata=pmatrix)[,1:nComp])

project_plus_country <- cbind(project,
                      cluster=as.factor(groups),
                      country=ks_base5_reduced$country)
```

Function to find the convex hull of clustered points (country):

```
# finding convex hull

h_country <- do.call( rbind,
                lapply(
                  unique(groups),
                  function(c) {
                    f <- subset(project_plus_country,cluster==c);
                    f[chull(f),]
    }
  )
)
```

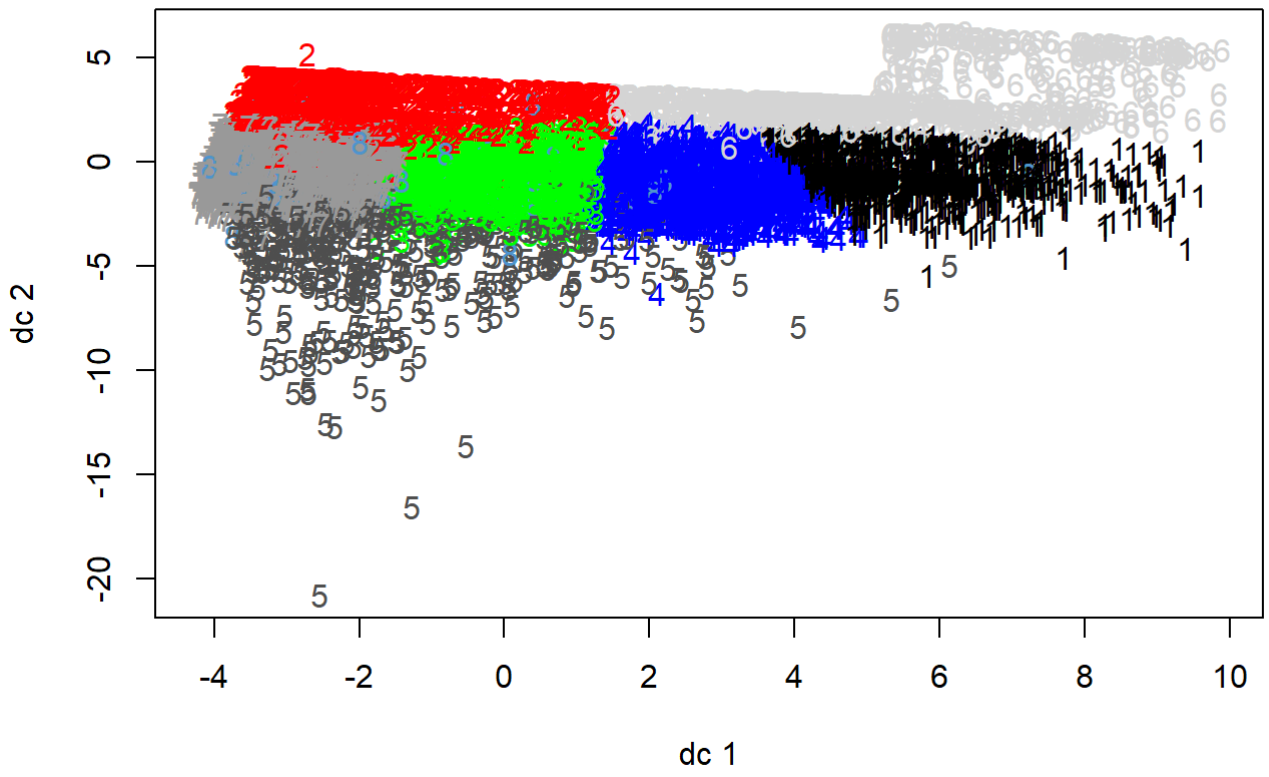
```
# plotting
p1 <- ggplot(project_plus_country, aes(x=PC1, y=PC2)) +
  geom_point(aes(shape=cluster, color=cluster)) +
  geom_polygon(data=h_country,
               aes(group=cluster,
                   fill=as.factor(cluster)),
               alpha=0.4,
               linetype=0)
p1
```



Clustering shows lack of spread.

Alternate view showing each cluster's spread and density, relative to each other:

```
# Kmeans cluster analysis

clus <- kmeans(pmatrix, centers=8)

plotcluster(pmatrix, clus$cluster)
```



4.4 - Assessing cluster effectiveness

We use clusterboot() to assess the stability of clusters, which employs the Jaccard coefficient.

We note that the higher value indicates a higher stability of clustering. Cluster 2, 3 and 6 are the most stable with cluster 3 having a real possibility of clusters. Clusters 2, 4, 5, 6 and 8 have discovered patterns, but with low certainty. Clusters 1 and 7 are unstable.

4.4.1 - Selecting k

Our goal is to select the best-fitting Calinski-Harabasz index.

```
# Function to calculate squared distance between two vectors x and y:
sqr_edist <- function(x, y) {
  sum((x-y)^2)
}

## Fucntion to calculate WSS of a cluster:
wss.cluster <- function(clustermat) {
  c0 <- apply(clustermat, 2, FUN=mean)
  sum(apply(clustermat, 1,
            FUN=function(row){sqr_edist(row,c0)}))
}
```

Function to calculate the intra-cluster distance:

```
wss.total <- function(dmatrix, labels) {
  wsstot <- 0
  k <- length(unique(labels))
  for(i in 1:k){
    wsstot <- wsstot +
      wss.cluster(subset(dmatrix, labels==i))
  }
  wsstot
}
```

Function to calculate the inter-cluster distance:

```
totss <- function(dmatrix) {
  grandmean <- apply(dmatrix, 2, FUN=mean)
  sum(apply(dmatrix, 1,
            FUN=function(row){
              sqr_edist(row, grandmean)
              }
            )
      )
}
```

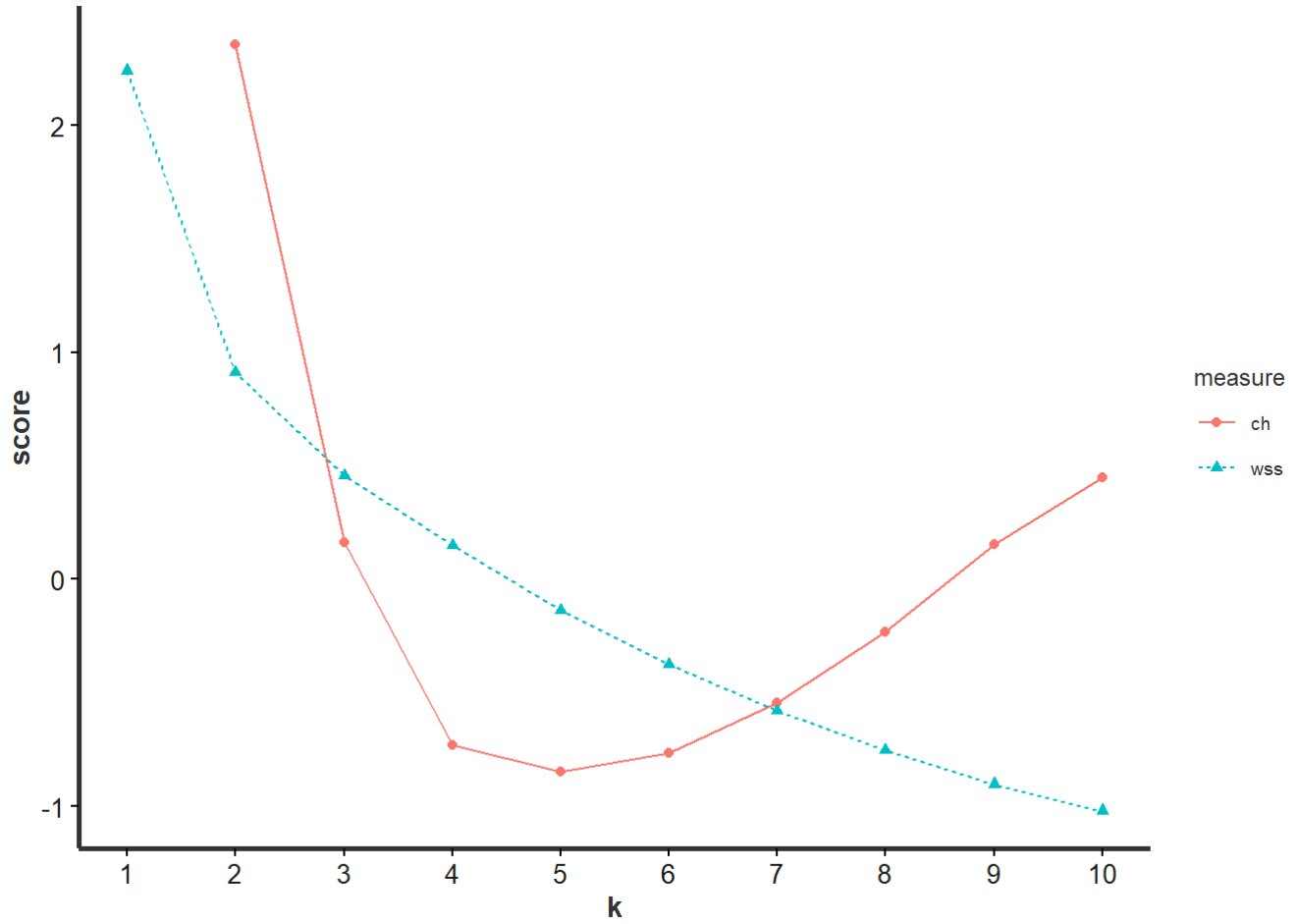
Calculating the index:

```
ch_criterion <- function(dmatrix, kmax, method="kmeans") {
  if(!(method %in% c("kmeans", "hclust"))){
    stop("method must be one of c('kmeans', 'hclust')")
  }

npts <- dim(dmatrix)[1] # number of rows.
totss <- totss(dmatrix)
wss <- numeric(kmax)
crit <- numeric(kmax)
wss[1] <- (npts-1)*sum(apply(dmatrix, 2, var))
for(k in 2:kmax) {
  if(method=="kmeans") {
    clustering <- kmeans(dmatrix, k, nstart=10, iter.max=100)
    wss[k] <- clustering$tot.withinss
    }else { # hclust
      d <- dist(dmatrix, method="euclidean")
      pfit <- hclust(d, method="ward.D2")
      labels <- cutree(pfit, k=k)
      wss[k] <- wss.total(dmatrix, labels)
  }
}
bss <- totss - wss
crit.num <- bss/(0:(kmax-1))
crit.denom <- wss/(npts - 1:kmax)
list(crit = crit.num/crit.denom, wss = wss, totss = totss)
}
```

Plotting the indices:

```
clustcrit <- ch_criterion(pmatrix, 10, method="hclust")
critframe <- data.frame(k=1:10, ch=scale(clustcrit$crit),
                        wss=scale(clustcrit$wss))
critframe <- melt(critframe, id.vars=c("k"),
                  variable.name="measure", value.name="score")
```

```
p <- ggplot(critframe, aes(x=k, y=score, color=measure)) +
  geom_point(aes(shape=measure)) +
  geom_line(aes(linetype=measure)) +
  scale_x_continuous(breaks=1:10, labels=1:10)

p
```



Looking at the graph, we choose 8 as our k-value, because it shows the best increase for the CH index when the WSS index is down-trending. It also still allows a sensible amount of clusters whilst still retaining efficacy. Noticeable 'elbow' structure.

4.5 - kMeans clustering:

Viewing summary statistics, such as sumsquares:

```
kbest.p <- 8
pclusters <- kmeans(pmatrix, kbest.p, nstart=100, iter.max=100)

summary(pclusters)
```

```
##              Length Class  Mode
## cluster      14139  -none- numeric
## centers         72  -none- numeric
## totss            1  -none- numeric
## withinss         8  -none- numeric
## tot.withinss     1  -none- numeric
## betweenss        1  -none- numeric
## size             8  -none- numeric
## iter             1  -none- numeric
## ifault           1  -none- numeric
```

We can also view the centroids of each cluster:

```
pclusters$centers
```

```
##            goal   deadline state_changed_at created_at launched_at     backers
## 1  -0.024111398 -1.4064879       -1.4069832 -1.3585817  -1.3943928 -0.09107736
## 2   0.123831969  0.1603312        0.1620542  0.1280483   0.1604764  9.00493412
## 3  -0.002711363  0.5284099        0.5286906 -0.3222155   0.5277891  0.06323087
## 4 117.059600614  1.1801593        1.1825905  1.2010948   1.1207388 -0.19058515
## 5  -0.003326303  0.8530744        0.8523238  0.8894705   0.8584427 -0.05896518
## 6  -0.018114045 -0.2674559       -0.2665928 -0.2437193  -0.2555451 -0.01327651
## 7   0.025920520  0.6939776        0.6956057  0.6649266   0.6522786 -0.03650231
## 8  -0.020537294 -1.5037122       -1.5035401 -1.5279714  -1.5562776 -0.10522757
##   launched_created prestated_duration actual_duration
## 1      -0.24934418        -0.17121394     -0.14720482
## 2       0.18330088        -0.03940637      0.02171257
## 3       4.66288115        -0.08736533     -0.08740454
## 4      -0.39568329         2.04870457      2.03442719
## 5      -0.13637503        -0.38735187     -0.42310779
## 6      -0.07444050        -0.40415298     -0.35070999
## 7      -0.04393618         1.46598710      1.45551050
## 8      -0.21459474         2.34931254      2.30953147
```

Using kmeansruns() to select k:

```
clustering.ch <- kmeansruns(
  pmatrix, krange=1:10, criterion="ch")
clustering.ch$bestk
```

```
## [1] 2
```

```
clustering.asw <- kmeansruns(
  pmatrix, krange=1:10, criterion="asw")
clustering.asw$bestk
```

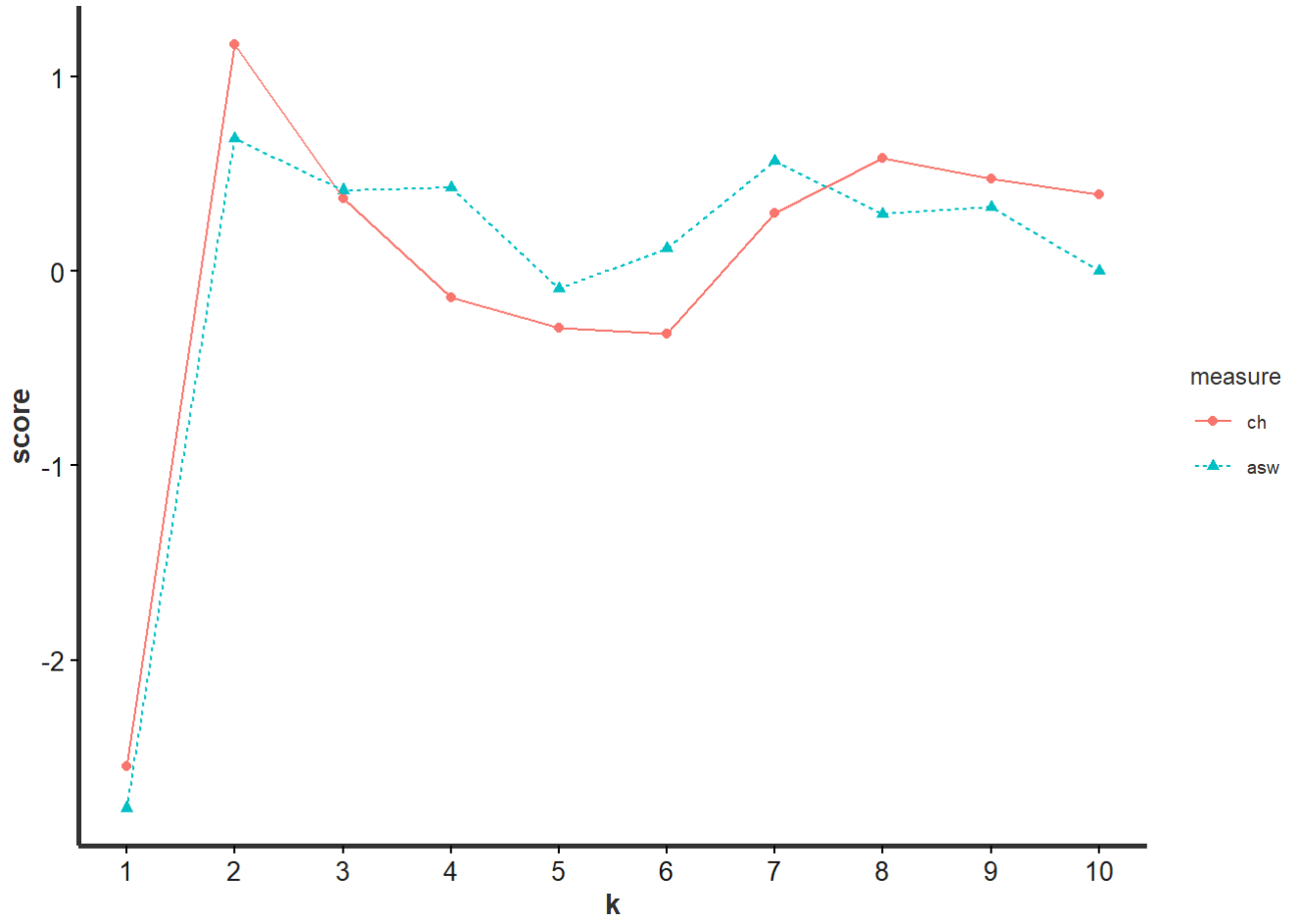
```
## [1] 2
```

```
# Compare the CH values for kmeans() and hclust():
clustering.ch$crit
```

```
##  [1]    0.000 7271.485 5722.975 4716.556 4417.219 4350.645 5571.179 6127.362
##  [9] 5913.246 5762.166
```

Plotting k against CH & ASW:

```
critframe <- data.frame(k=1:10, ch=scale(clustering.ch$crit), asw=scale(clustering.asw$crit))

critframe <- melt(critframe, id.vars=c("k"), variable.name="measure", value.name="score")

p <- ggplot(critframe, aes(x=k, y=score, color=measure)) +
  geom_point(aes(shape=measure)) +
  geom_line(aes(linetype=measure)) +
  scale_x_continuous(breaks=1:10, labels=1:10)

p
```



Again, 8 shows as a suitable k-value as it is uptrending on CH and downtrending on ASW, in a slight 'elbow' fashion.

5 - Conclusion

Project 2 started with preparation of the data via cleaning, transforming, processing sampling and splitting the dataset. We obviously chose the response variable as *outcome* and proceeded to classify the data.

We identified the null model and the saturated model, and showed how they perform, also evaluating the saturated model's precision and recall.

We then moved on to univariate analysis, and processed the feature variables, adding predictors for each one.

We then compared and evaluated their performance via log likelihood and double density plots, and selected the best variables for future use.

From here we began multivariate analysis, performing kNN modelling and decision-tree modelling. We compared the kNN model to the logstic regression model, and noted that kNN did perform slightly better. We noted that the decision-tree model performed vastly beetter than the null model on the training data and test data, and did not show signs of overfitting.

We then moved to clustering with aim of identifying groups and patterns in the dataset. We effectively selected k=8 number of clusters and cross-validated to k-means clustering. We used the gower method of distance measurement as we had a mix of categorical and numerical data. There were signs of clustering but meaning was hard to draw. Clustering was heavily impacted by the large portion of US and USD categories.