Forecasting the success of Kickstarter Projects

Colin Howlett

09/05/2020

Introduction

The purpose of this project was to investigate the potential for forecasting the success or otherwise of projects on Kickstarter.[1]

[1] https://www.kickstarter.com/

Kickstarter is a website, started in 2009, that provides facilities for people to raise funds for a project. For each project, it is possible to enter a description, or blurb for the project, and set a monetary goal of funds to be raised, as well as a deadline for when the goal should be acheived. Users of the website can decide to become backers of a project by pledging to donate money to it, in exchange for a reward, in the case where the project meets it's money-raising goal.

The purpose of this investigation, was to see how well, given the information available at the start of a project's journey on Kickstarter, the success of the project in raising the requested funds could be forecast.

Data was obtained from a data set hosted on Kaggle.[2]. The data set contains data for over 400,000 projects over 10 years.

[2] https://www.kaggle.com/toshimelonhead/400000-kickstarter-projects

Method

As set out above, the data was retreived from Kaggle and split into three files for uploading to Github. The first step then is to rebuild the dataset from those three compressed files.

```
if (!require(tidyverse)) install.packages('tidyverse')
if (!require(caret)) install.packages('caret')
if (!require(sentimentr)) install.packages('sentimentr')
library(sentimentr)
library(tidyverse)
library(caret)

ks1 <- read_csv(unzip("kickstarter1.csv.zip"))
ks2 <- read_csv(unzip("kickstarter2.csv.zip"))
ks3 <- read_csv(unzip("kickstarter3.csv.zip"))
```

```
ks <- rbind(ks1, ks2)
ks <- rbind(ks, ks3)

rm(ks1, ks2, ks3)
```

Once available in R as a single dataframe, the next step is to set aside a portion of the data for final assessment of the performance of the model:

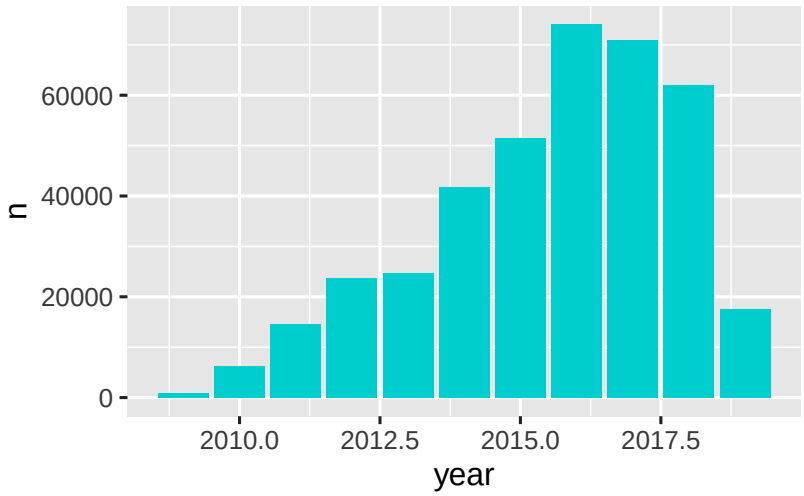
```
set.seed(1, sample.kind="Rounding")

test_index <- createDataPartition(y = ks$binary_state,
                                  times = 1,
                                  p = 0.1,
                                  list = FALSE)
kstrain <- ks[-test_index,]
temp <- ks[test_index,]

validation <- temp %>%
  semi_join(kstrain, by = "category_name") %>%
  semi_join(kstrain, by = 'location_country') %>%
  semi_join(kstrain, by = 'blurb_length')

removed <- anti_join(temp, validation)

kstrain <- rbind(kstrain, removed)
```

Once the data has been imported and a validation data set has been set aside, it is possible to examine the nature of the data available. In the training data `kstrain` there are 387843 Kickstarter projects'. The first project was in 2009 and the last in 2019.

```
num_rows <- nrow(kstrain)

earliest_year <- min(unique(kstrain$year))
latest_year <- max(unique(kstrain$year))
```

The distribution of projects by year looks like this:

```
kstrain %>% group_by(year) %>% summarize(n=n()) %>%
  ggplot(aes(year,n, fill=I("cyan3"))) +
  geom_bar(stat="identity")
```



The variable to predict is the 'binary_state' - whether the project was successful in raising its crowdfunding, or if it failed.

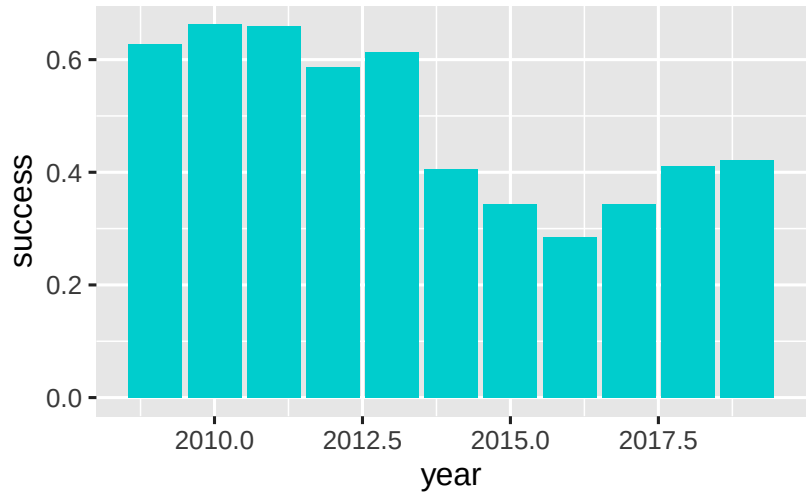
```
possible_outcomes <- unique(kstrain$binary_state)
```

Examining the data, it can be seen that 40.3% were successful.

```
success <- mean(kstrain$binary_state == "successful")
```

It can also be clearly seen that the success percentage varies by year.

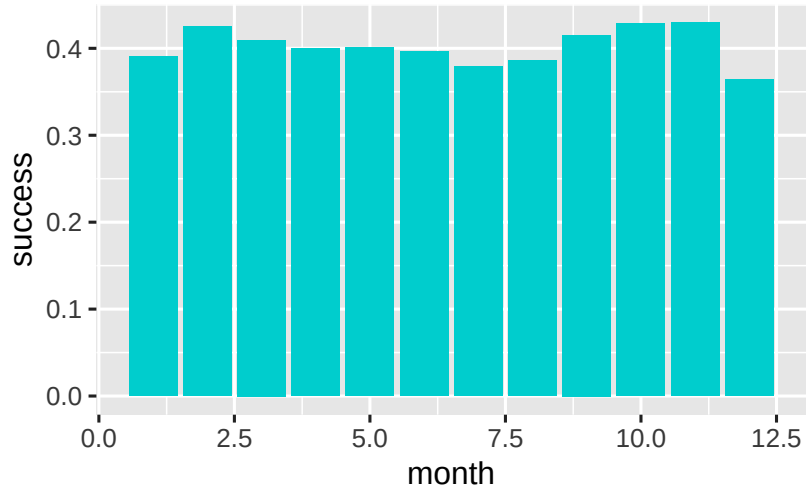
```
kstrain %>% group_by(year) %>%
  summarize(success=mean(binary_state == "successful")) %>%
  ggplot(aes(year, success, fill=I("cyan3"))) +
  geom_bar(stat="identity")
```



It would appear that there is a clear effect by year.

Examining the data by month shows that there is also a monthly effect.

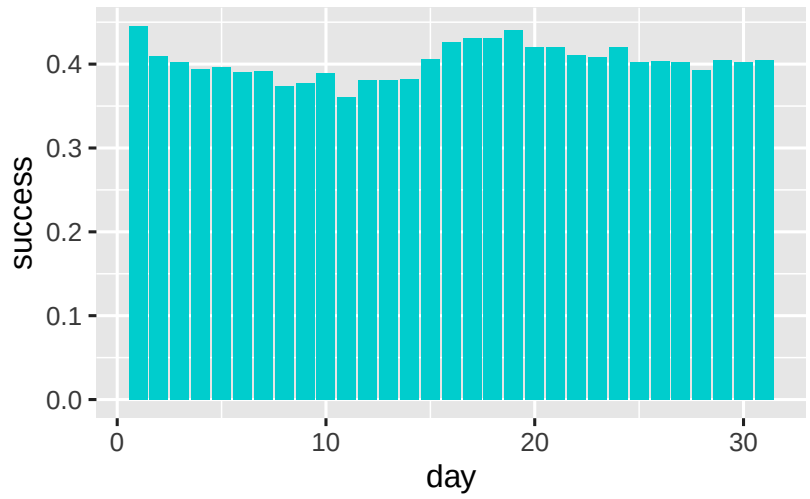
```
kstrain %>% group_by(month) %>%
  summarize(success=mean(binary_state == "successful")) %>%
  ggplot(aes(month, success, fill=I("cyan3"))) +
  geom_bar(stat="identity")
```



There seems to be much less variation, although a launch in September, October or November seems to give a small boost in success.

Information about the day of the month of the launch can also be examined:

```
kstrain %>% group_by(day) %>%
  summarize(success=mean(binary_state == "successful")) %>%
  ggplot(aes(day, success, fill=I("cyan3"))) +
  geom_bar(stat="identity")
```



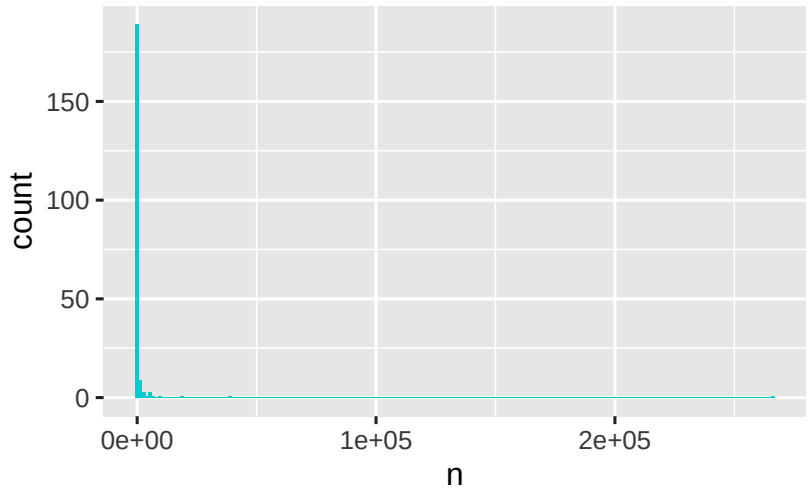
There seems to be a clear boost for projects launched on the 1st of the month, and for those launched in the middle of the month.

There is also data available in the data set on the location of each project. It can be seen that there were projects in nearly every country on earth. In total 208 countries were represented.

```
countries <- length(unique(kstrain$location_country))
```

Examining the distribution of projects by country shows that the overwhelming majority of projects are concentrated into just a few countries.

```
kstrain %>% group_by(location_country) %>%
  summarize(n=n()) %>%
  ggplot(aes(n, fill=I("cyan3"))) +
  geom_histogram(bins=200)
```



In fact it can be seen that only 19 countries have more than 1000 projects, and the great majority are in the US:

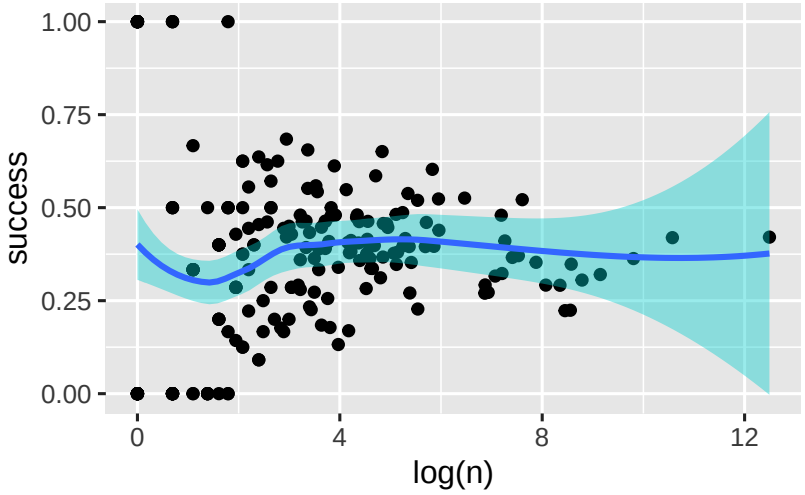
```
knitr::kable(
  kstrain %>% group_by(location_country) %>%
    summarize(n=n(),
              success=mean(binary_state=="successful")) %>%
    filter(n >= 1000) %>% arrange(desc(n))
)
```

location_country	n	success
US	265969	0.4207596
GB	39076	0.4193623
CA	18097	0.3632094
AU	9398	0.3202809
DE	6546	0.3053773
FR	5290	0.3480151
IT	5200	0.2242308
MX	4678	0.2231723
ES	4235	0.2913813
NL	3201	0.2917838
SE	2640	0.3530303
HK	2010	0.5213930
NZ	1860	0.3709677
DK	1652	0.3662228
SG	1425	0.4105263
CH	1347	0.3229399
JP	1328	0.4796687
IE	1177	0.3160578

location_country	n	success
BE	1025	0.2721951

Plotting the mean success in a country against a log transformation of the number of projects a country has, it can be seen that there is a small effect due to the country particularly for countries with a moderate number of projects, however, this may possibly be noise due to there being so few projects.

```
kstrain %>% group_by(location_country) %>%
  summarize(n=n(),
            success=mean(binary_state=="successful")) %>%
  ggplot(aes(log(n), success, fill=I("cyan3"))) +
  geom_point() + geom_smooth()
```

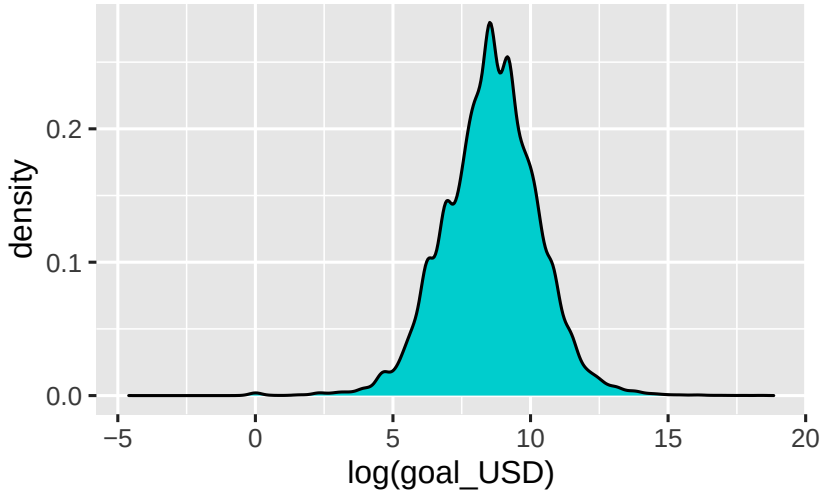


Examining the effect of the monetary goal set for each project, reveals that there are there are 74674 distinct goal values in USD

```
goal_values <- length(unique(kstrain$goal_USD))
```

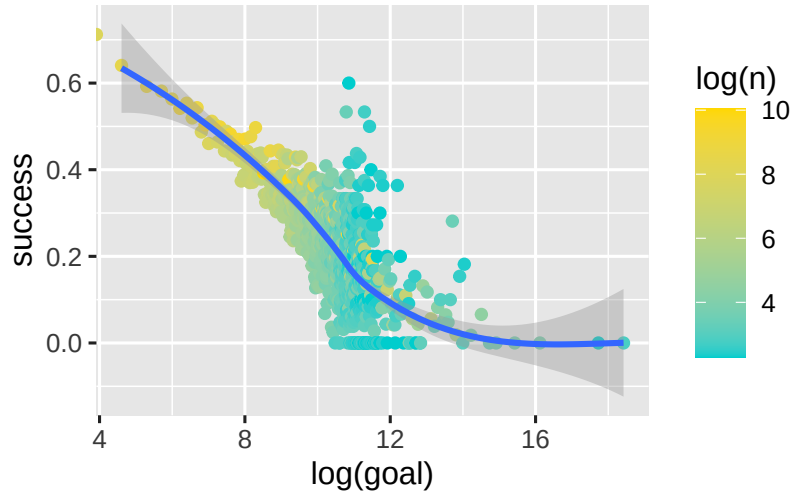
Examining the distribution of a log transformation of the goal, reveals a roughly normal distribution.

```
kstrain %>% ggplot(aes(log(goal_USD), fill=I("cyan3"))) +
  geom_density(adjust=2)
```



Examining the implications of the goal on the success rate reveals that the goal amount seems to have a strong influence on the success or otherwise of the project. This is perhaps intuitive, as a smaller goal is easier to raise than a larger one.

```
kstrain %>% group_by(goal=round(goal_USD, -2)) %>%
  summarize(n = n(),
            success=mean(binary_state=="successful")) %>%
  filter(n >= 10) %>%
  ggplot(aes(log(goal), success, color=log(n))) +
  geom_point() +
  scale_color_gradient(low="cyan3", high="gold1") +
  geom_smooth()
```

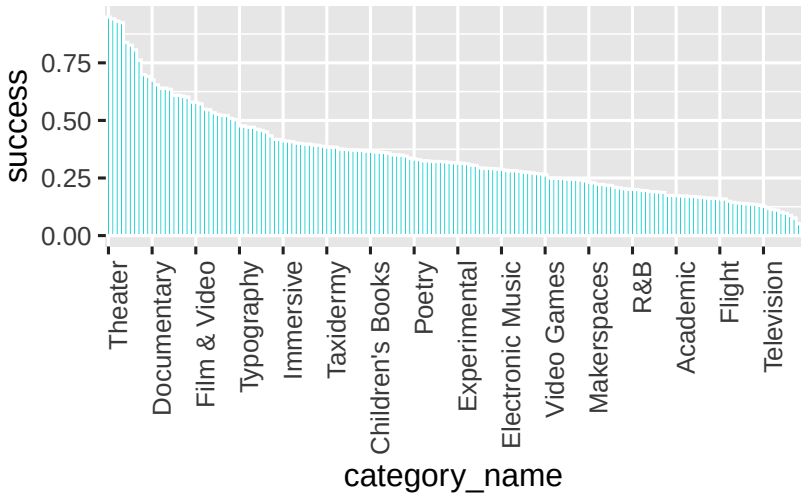


Turning to examine the effect of the category on the success of a project, it can be seen that there are 159 distinct categories, but they are not at all equal in their rate of success.

```
num_categories <- length(unique(kstrain$category_name))
```

Plotting success rate against category, it can be seen that category is very predictive of success.

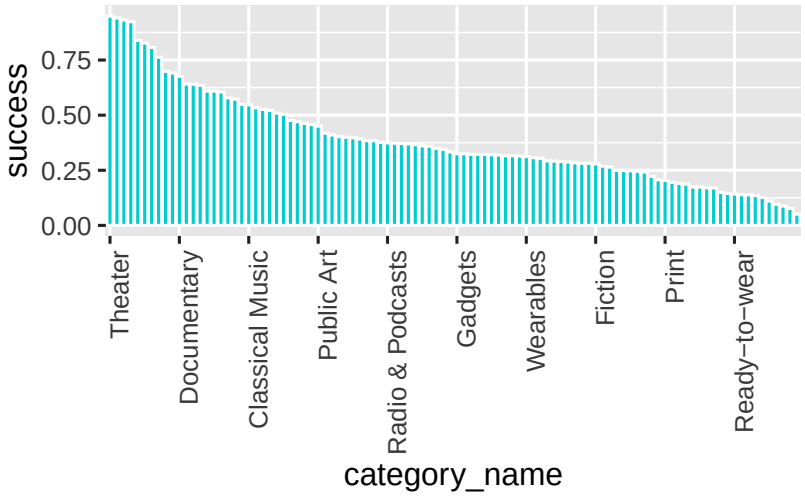
```
every_nth = function(n) {
  return(function(x) {x[c(TRUE, rep(FALSE, n - 1))]})}

kstrain %>% group_by(category_name) %>%
  summarize(success=mean(binary_state=="successful")) %>%
  arrange(desc(success))  %>%
  mutate(category_name=factor(category_name,
                              levels=category_name)) %>%
  ggplot(aes(category_name, success,
             fill=I("cyan3"), color=I("white"))) +
  geom_bar(stat="identity") +
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  scale_x_discrete(breaks = every_nth(n = 10))
```



This remains true, even excluding those categories with fewer than 1000 projects, to avoid a skewing effect due to low numbers of projects.

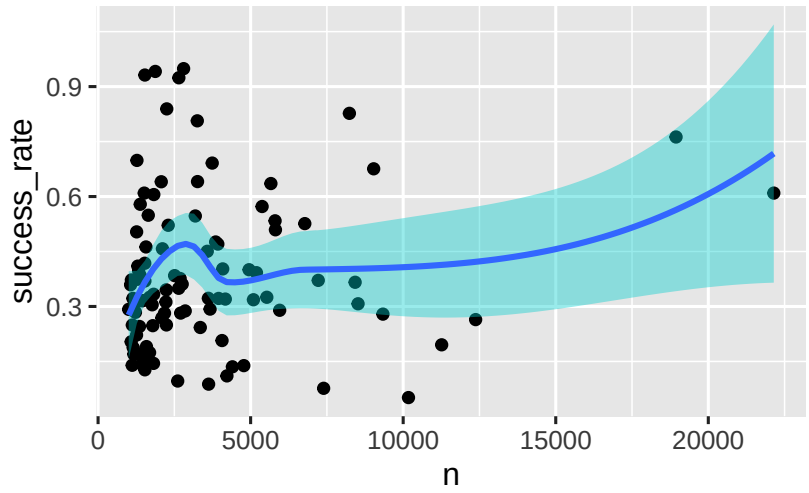
```
kstrain %>% group_by(category_name) %>%
  summarize(success=mean(binary_state=="successful"),
            n=n()) %>%
```

```
  filter(n>=1000) %>%
  arrange(desc(success))  %>%
  mutate(category_name=factor(category_name,
                              levels=category_name)) %>%
  ggplot(aes(category_name, success,
             fill=I("cyan3"), color=I("white"))) +
  geom_bar(stat="identity") +
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  scale_x_discrete(breaks = every_nth(n = 10))
```



A very strong effect can still be seen, and in fact, filtering out categories with fewer than 1000 projects and just plotting the number of projects against success_rate still shows an effect.

```
kstrain %>% group_by(category_name) %>%
  summarize(success_rate=mean(binary_state=="successful"),
            n=n()) %>% filter(n>=1000) %>%
  ggplot(aes(n, success_rate, fill=I("cyan3"))) +
  geom_point() + geom_smooth()
```



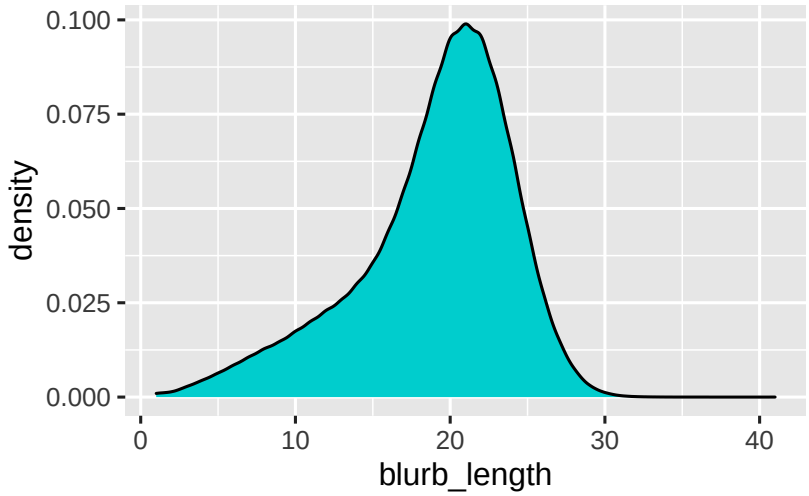
The data set also contains information about the blurb posted for each project - both the blurb itself and the derived value of it's length.

There are only 37 unique values for blurb length.

```
blurb_lengths <- length(unique(kstrain$blurb_length))
```

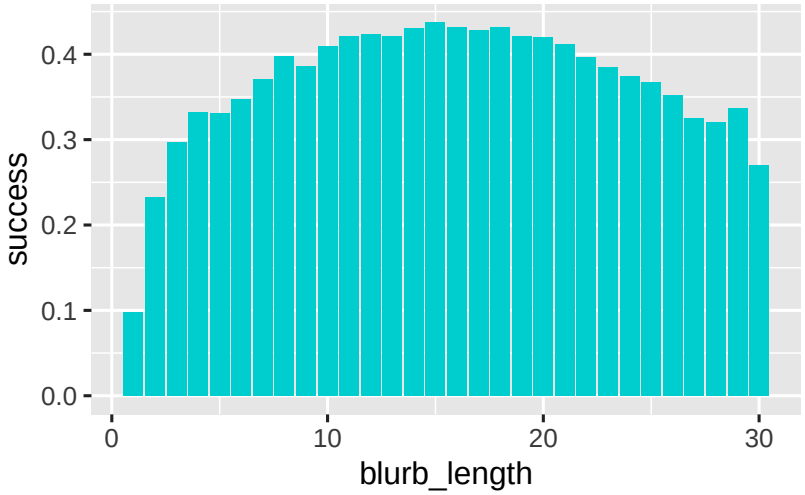
Plotting blurb length against mean success rate, it can be seen that the distribution is a kind of skewed normal distribution.

```
kstrain %>% ggplot(aes(blurb_length, fill=I("cyan3"))) +
  geom_density(adjust=1.8)
```



Filtering out rare long blurbs reveals a humped distribution, suggesting some impact from blurb length on success.

```
kstrain %>% group_by(blurb_length) %>%
  filter(blurb_length <= 30) %>%
  summarize(success=mean(binary_state=="successful")) %>%
  ggplot(aes(blurb_length, success, fill=I("cyan3"))) +
  geom_bar(stat="identity")
```



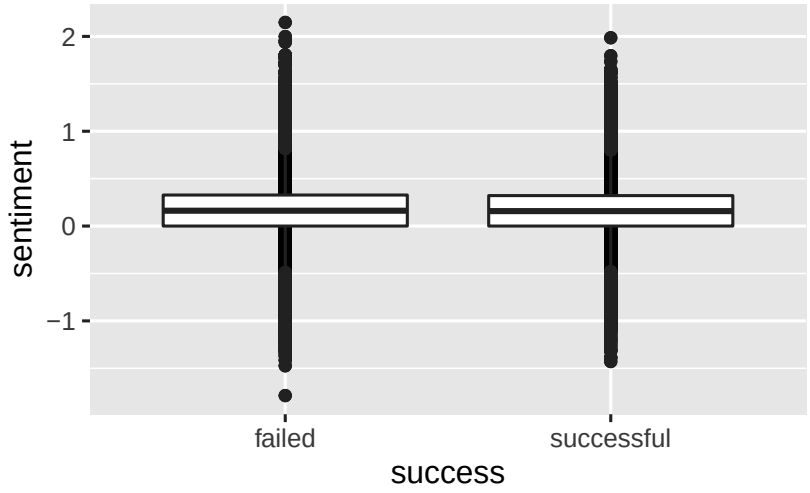
In order to understand the impact of the blurb itself on the success rate of the project, a sentiment analysis was attempted, using the 'sentimentr' package.[3] Sentimentr can analyse each sentence or group of sentences and produce a single positive or negative value summarizing the positive or negative sentiments expressed.

[3] https://cran.r-project.org/web/packages/sentimentr/readme/README.html

```
blurb_sentences <- get_sentences(kstrain$blurb)
blurb_sentiments <- sentiment_by(blurb_sentences)
```

A boxplot of the sentiments of each possible outcome - successful or failed - shows very little difference.

```
data.frame(success=kstrain$binary_state,
           sentiment=blurb_sentiments$ave_sentiment) %>%
  ggplot(aes(success, sentiment)) +
  geom_point() + geom_boxplot()
```



Summarizing the data in tabular form shows a small bias - surprisingly the failed projects have slightly more positive blurbs than the successful ones. This does not seem to be particularly predictive.

```
knitr::kable(
  data.frame(success=kstrain$binary_state,
             sentiment=blurb_sentiments$ave_sentiment) %>%
    group_by(success) %>%
    summarize(mean(sentiment))
)
```

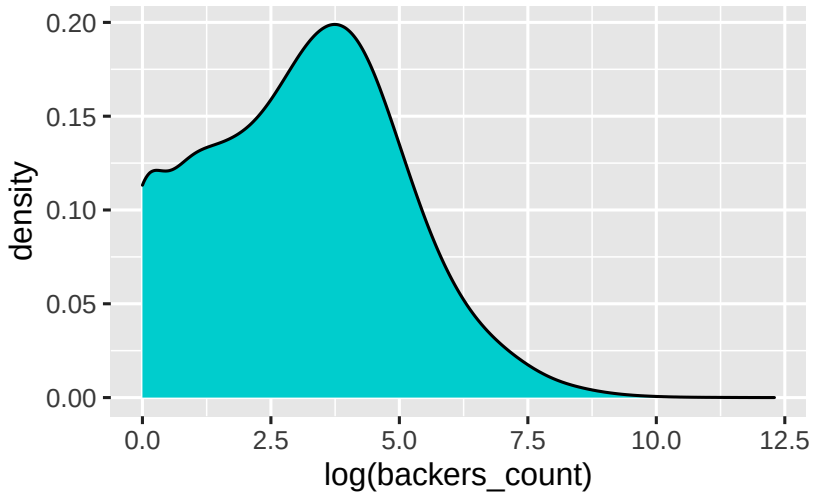
success	mean(sentiment)
failed	0.178743
successful	0.169556

Examining the number of backers for each project, it can be seen that there are 4649 unique values.

```
poss_backer_val_num <- length(unique(kstrain$backers_count))
```

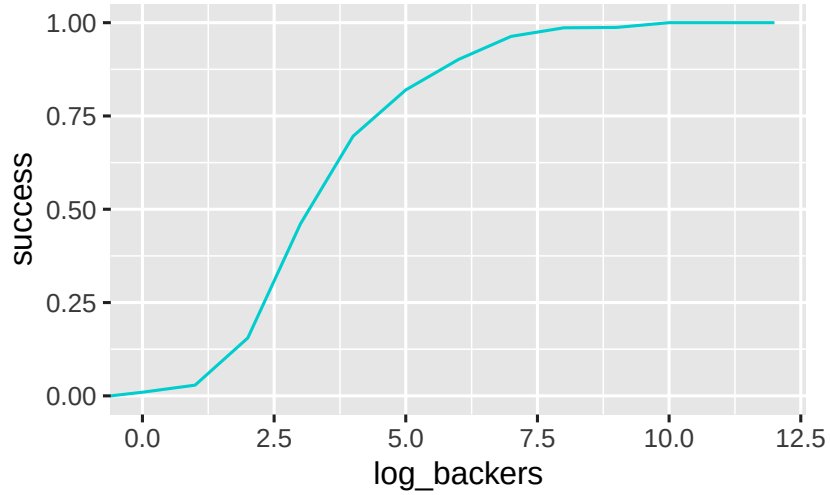
Plotting the density of a log transformation of those values shows a clear hump in the distribution.

```
kstrain %>% ggplot(aes(log(backers_count),
                       fill=I("cyan3"))) +
  geom_density(adjust=3)
```



There is also a clear correlation between the number of backers and the mean success - this is not surprising, since it is backers who provide the money.

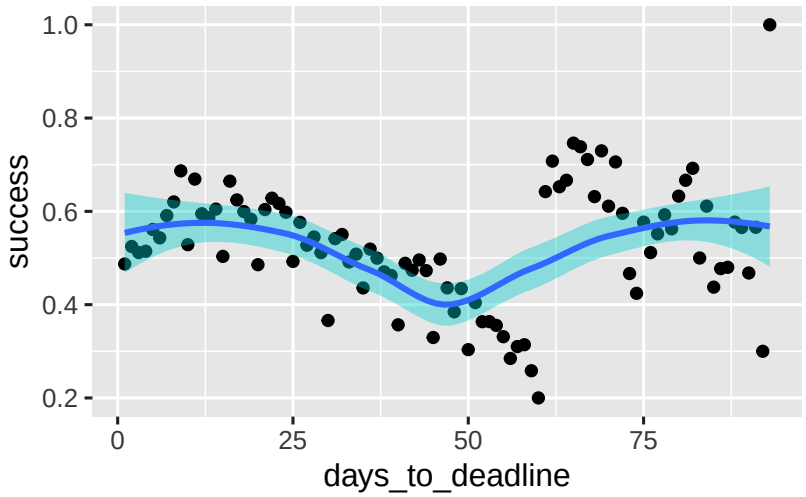
```
kstrain %>%
  group_by(log_backers=round(log(backers_count))) %>%
  summarize(success=mean(binary_state=="successful")) %>%
  ggplot(aes(log_backers, success, color=I("cyan3"))) +
  geom_line()
```



Turning to examine the days to the deadline for each project reveals only 93 different values, so it can be treated as a categorical variable.

Plotting the mean success against days to the deadline shows two distinct groups.

```
kstrain %>% group_by(days_to_deadline) %>%
  summarize(success=mean(binary_state=="successful")) %>%
  ggplot(aes(days_to_deadline, success,
             fill=I("cyan3"))) +
  geom_point() +
  geom_smooth()
```



There clearly seems to be an effect - it seems some projects race out of the gates, and some take longer to get going.

The eventual success rate does not seem that different.

On Kickstarter it is possible for Kickstarter staff to select particular projects to be highlighted as a 'Staff Pick'. This is a binary variable, so we can tabulate how it correlates with success.

```
knitr::kable(
  kstrain %>% group_by(staff_pick) %>%
    summarize(success=mean(binary_state=="successful"))
)
```

staff_pick	success
FALSE	0.3618060
TRUE	0.7558151

76% of the time, a staff pick was successful, while a project that was not a staff pick was only successful 36% of the time. It is not clear whether this is

merely correlation or causation.

The same analysis can be performed for so-called spotlight projects.

```
knitr::kable(
  kstrain %>% group_by(spotlight) %>%
    summarize(success=mean(binary_state=="successful"))
)
```

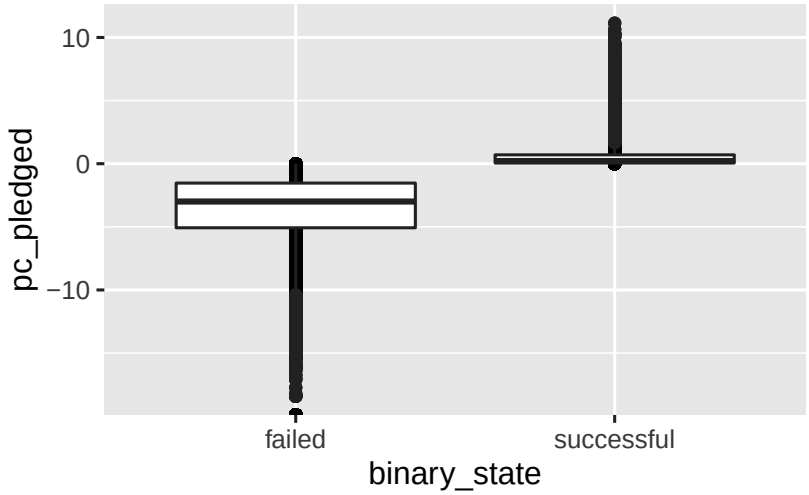
spotlight	success
FALSE	0.0981362
TRUE	0.9999848

Here it can be seen that spotlight projects were essentially always successful, while non-spotlight projects were only successful around 10% of the time.

This is because 'Spotlight' is a facility made available to projects by Kickstarter after they have been successful.

Finally, a boxplot of the amount pledged as a percent of the goal for both successful and failed projects reveals what would be expected:

```
kstrain %>% mutate(pc_pledged=log(usd_pledged/goal_USD)) %>%
  ggplot(aes(binary_state, pc_pledged)) +
  geom_point() + geom_boxplot()
```



Clearly it can be seen that for successful projects their percent pledged is

above the target, whereas for unsuccessful ones this is below the target.

Indeed this is the definition of success.

Data Cleaning

Having completed the initial investigation, the data needed to be cleaned. Only some columns are valuable for prediction.

```
kstrain_clean <- kstrain %>%
  select(year, month, day, location_country,
         category_name, goal_USD, blurb_length,
         days_to_deadline, backers_count,
         staff_pick, binary_state, usd_pledged)
validation_clean <- validation %>%
  select(year, month, day, location_country,
         category_name, goal_USD, blurb_length,
         days_to_deadline, backers_count,
         staff_pick, binary_state, usd_pledged)
```

It is helpful to change the binary state outcome variable to a logical value, take a rounded log transformation of the goal[4] and change the location_country and category_name vectors to factors.

[4] A log transformation centres the values with predictive value, while rounding reduces the number of possible values, making regularization simpler.

```
kstrain_clean <- kstrain_clean %>%
  mutate(location_country=factor(location_country),
         category_name=factor(category_name),
         binary_state=binary_state=="successful",
         log_goal=round(log(goal_USD)))
validation_clean <- validation_clean %>%
  mutate(location_country=factor(location_country),
         category_name=factor(category_name),
         binary_state=binary_state=="successful",
         log_goal=round(log(goal_USD)))
```

The first 7 columns are effectively inputs, knowable at the start of the process, while the last 4 are outputs - unknowable at the outset of the project.

The kstrain_clean data set was further split into a training and test set, used for optimisation during the training process.

```
set.seed(1, sample.kind="Rounding")
test_index <- createDataPartition(
  y = kstrain_clean$binary_state,
  times = 1, p = 0.2, list = FALSE)
train_set <- kstrain_clean[-test_index,]
temp <- kstrain_clean[test_index,]

test_set <- temp %>%
  semi_join(train_set, by = "category_name") %>%
  semi_join(train_set, by = 'location_country') %>%
  semi_join(train_set, by = 'blurb_length')

removed <- anti_join(temp, test_set)

train_set <- rbind(train_set, removed)
```

The accuracy metric used for regularization was RMSE.

```
RMSE <- function(true_ratings, predicted_ratings){
  sqrt(mean((true_ratings - predicted_ratings)^2))
}
```

The most rudimentary prediction is simply the average outcome - 40.3% of projects were successful.

```
mu <- mean(train_set$binary_state)
```

The data was regularized, optimizing for lambda - a weighting factor designed to reduce the weight given to noisy biases from sample groups with only a few members.

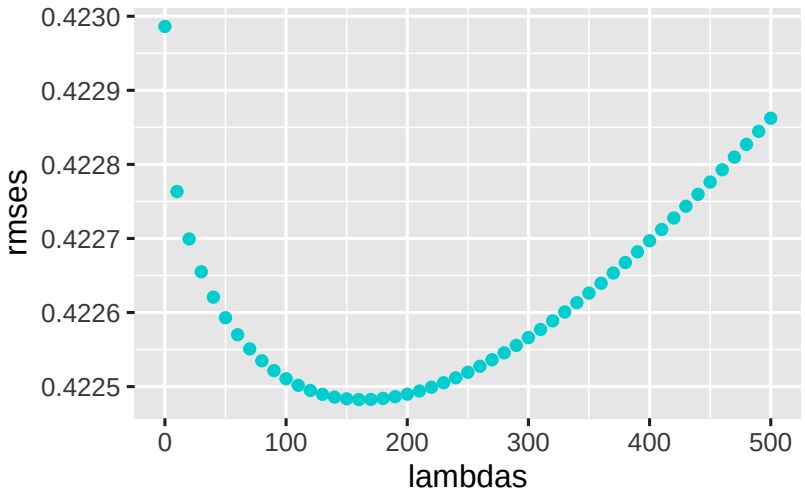
```
lambdas <- seq(0, 500, 10)

rmses <- sapply(lambdas, function(l){

  year_avgs <- train_set %>%
    group_by(year) %>%
    summarize(b_y = sum(binary_state - mu)/(n()+l))

  month_avgs <- train_set %>%
    left_join(year_avgs, by='year') %>%
    group_by(month) %>%
    summarize(b_m = sum(binary_state - mu - b_y)/(n()+l))
```

```
day_avgs <- train_set %>%
  left_join(year_avgs, by='year') %>%
  left_join(month_avgs, by='month') %>%
  group_by(day) %>%
  summarize(b_d = sum(binary_state - mu - b_y -
                        b_m)/(n()+1))

country_avgs <- train_set %>%
  left_join(year_avgs, by='year') %>%
  left_join(month_avgs, by='month') %>%
  left_join(day_avgs, by='day') %>%
  group_by(location_country) %>%
  summarize(b_co = sum(binary_state - mu - b_y -
                         b_m - b_d)/(n()+1))

category_avgs <- train_set %>%
  left_join(year_avgs, by='year') %>%
  left_join(month_avgs, by='month') %>%
  left_join(day_avgs, by='day') %>%
  left_join(country_avgs, by='location_country') %>%
  group_by(category_name) %>%
  summarize(b_cat = sum(binary_state - mu - b_y -
                          b_m - b_d - b_co)/(n()+1))

goal_avgs <- train_set %>%
  left_join(year_avgs, by='year') %>%
  left_join(month_avgs, by='month') %>%
  left_join(day_avgs, by='day') %>%
  left_join(country_avgs, by='location_country') %>%
  left_join(category_avgs, by='category_name') %>%
  group_by(log_goal) %>%
  summarize(b_g = sum(binary_state - mu - b_y -
                        b_m - b_d - b_co -
                        b_cat)/(n()+1))

blurb_avgs <- train_set %>%
  left_join(year_avgs, by='year') %>%
  left_join(month_avgs, by='month') %>%
  left_join(day_avgs, by='day') %>%
  left_join(country_avgs, by='location_country') %>%
  left_join(category_avgs, by='category_name') %>%
  left_join(goal_avgs, by='log_goal') %>%
  group_by(blurb_length) %>%
```

```
    summarize(b_bl = sum(binary_state - mu - b_y -
                         b_m - b_d - b_co -
                         b_cat - b_g)/(n()+1))

  deadline_avgs <- train_set %>%
    left_join(year_avgs, by='year') %>%
    left_join(month_avgs, by='month') %>%
    left_join(day_avgs, by='day') %>%
    left_join(country_avgs, by='location_country') %>%
    left_join(category_avgs, by='category_name') %>%
    left_join(goal_avgs, by='log_goal') %>%
    left_join(blurb_avgs, by='blurb_length') %>%
    group_by(days_to_deadline) %>%
    summarize(b_dd = sum(binary_state - mu - b_y -
                         b_m - b_d - b_co -
                         b_cat - b_g - b_bl)/(n()+1))

  predicted_ratings <- test_set %>%
    left_join(year_avgs, by='year') %>%
    left_join(month_avgs, by='month') %>%
    left_join(day_avgs, by='day') %>%
    left_join(country_avgs, by='location_country') %>%
    left_join(category_avgs, by='category_name') %>%
    left_join(goal_avgs, by='log_goal') %>%
    left_join(blurb_avgs, by='blurb_length') %>%
    left_join(deadline_avgs, by='days_to_deadline') %>%
    mutate(pred = mu + b_y +
             b_m + b_d + b_co +
             b_cat +  b_g + b_bl + b_dd) %>%
    pull(pred)

  return(RMSE(predicted_ratings, test_set$binary_state))
})
```

Plotting the RMSE for each lambda shows where the minimum RMSE is to be found:

```
qplot(lambdas, rmses, color=I("cyan3"))
```



The lambda value producing the lowest RMSE was 160.

```
optimum_lambda <- lambdas[which.min(rmses)]
```

The RMSE returned with a lambda of 160 on the test set is 0.4224825.

```
optimum_rmse <- min(rmses)
```

Training and test sets with the biases added as columns were created using biases calculated with a lambda of 160.

```
l <- 160

year_avgs <- train_set %>%
  group_by(year) %>%
  summarize(b_y = sum(binary_state - mu)/(n()+l))

month_avgs <- train_set %>%
  left_join(year_avgs, by='year') %>%
  group_by(month) %>%
  summarize(b_m = sum(binary_state - mu - b_y)/(n()+l))

day_avgs <- train_set %>%
  left_join(year_avgs, by='year') %>%
  left_join(month_avgs, by='month') %>%
  group_by(day) %>%
  summarize(b_d = sum(binary_state - mu - b_y -
                        b_m)/(n()+l))
```

```
country_avgs <- train_set %>%
  left_join(year_avgs, by='year') %>%
  left_join(month_avgs, by='month') %>%
  left_join(day_avgs, by='day') %>%
  group_by(location_country) %>%
  summarize(b_co = sum(binary_state - mu - b_y -
                         b_m - b_d)/(n()+1))

category_avgs <- train_set %>%
  left_join(year_avgs, by='year') %>%
  left_join(month_avgs, by='month') %>%
  left_join(day_avgs, by='day') %>%
  left_join(country_avgs, by='location_country') %>%
  group_by(category_name) %>%
  summarize(b_cat = sum(binary_state - mu - b_y -
                          b_m - b_d - b_co)/(n()+1))

goal_avgs <- train_set %>%
  left_join(year_avgs, by='year') %>%
  left_join(month_avgs, by='month') %>%
  left_join(day_avgs, by='day') %>%
  left_join(country_avgs, by='location_country') %>%
  left_join(category_avgs, by='category_name') %>%
  group_by(log_goal) %>%
  summarize(b_g = sum(binary_state -
                        mu - b_y - b_m - b_d - b_co -
                        b_cat)/(n()+1))

blurb_avgs <- train_set %>%
  left_join(year_avgs, by='year') %>%
  left_join(month_avgs, by='month') %>%
  left_join(day_avgs, by='day') %>%
  left_join(country_avgs, by='location_country') %>%
  left_join(category_avgs, by='category_name') %>%
  left_join(goal_avgs, by='log_goal') %>%
  group_by(blurb_length) %>%
  summarize(b_bl = sum(binary_state - mu - b_y -
                         b_m - b_d - b_co -
                         b_cat - b_g)/(n()+1))

deadline_avgs <- train_set %>%
  left_join(year_avgs, by='year') %>%
  left_join(month_avgs, by='month') %>%
```

```
  left_join(day_avgs, by='day') %>%
  left_join(country_avgs, by='location_country') %>%
  left_join(category_avgs, by='category_name') %>%
  left_join(goal_avgs, by='log_goal') %>%
  left_join(blurb_avgs, by='blurb_length') %>%
  group_by(days_to_deadline) %>%
  summarize(b_dd = sum(binary_state - mu - b_y -
                         b_m - b_d - b_co -
                         b_cat - b_g - b_bl)/(n()+1))

train_bias <- train_set %>%
  left_join(year_avgs, by='year') %>%
  left_join(month_avgs, by='month') %>%
  left_join(day_avgs, by='day') %>%
  left_join(country_avgs, by='location_country') %>%
  left_join(category_avgs, by='category_name') %>%
  left_join(goal_avgs, by='log_goal') %>%
  left_join(blurb_avgs, by='blurb_length') %>%
  left_join(deadline_avgs, by='days_to_deadline')

test_bias <- test_set %>%
  left_join(year_avgs, by='year') %>%
  left_join(month_avgs, by='month') %>%
  left_join(day_avgs, by='day') %>%
  left_join(country_avgs, by='location_country') %>%
  left_join(category_avgs, by='category_name') %>%
  left_join(goal_avgs, by='log_goal') %>%
  left_join(blurb_avgs, by='blurb_length') %>%
  left_join(deadline_avgs, by='days_to_deadline')
```

The process of applying the selected algorithms to the training was undertaken after the training data with biases had been created.

The selection process of model types to be trained on the data was governed by a number of considerations. Firstly a linear regression model was selected as a baseline, with a number of other simpler approaches selected as comparators. In addition, the intention was to select models of varying types - a knn model, a random forest model and a neural net model, to test their effectiveness on the data. Finally it was intended to try an ensemble model, averaging the predictions of a selection of the other models. The particular algorithms selected were guided by a benchmark analysis of the time performance of various implementations within R. [5]

[5] Pafka, Szilard, benchm-ml, GitHub 2019, https://github.com/szilard/benchm-ml

The first model to be trained was a linear regression model:

```
fitlm <- lm(binary_state ~ b_y + b_m + b_d + b_co +
              b_cat + b_g + b_bl + b_dd, data=train_bias)
y_hatlm <- predict(fitlm, test_bias)
predictionslm <- ifelse(y_hatlm > 0.5, TRUE, FALSE)
cm_lm <- confusionMatrix(factor(predictions),
                         factor(test_bias$binary_state))
```

This produced an overall accuracy on the test set of 0.7333677

This was followed by training a generalized linear model:

```
fitglm <- glm(binary_state ~ b_y + b_m + b_d + b_co +
                b_cat + b_g + b_bl + b_dd, data=train_bias,
              family="binomial")
y_hatglm <- predict(fitglm, newdata=test_bias,
                    type="response")
predictionsglm <- ifelse(y_hatglm > 0.5, TRUE, FALSE)
cm_glm <- confusionMatrix(factor(predictionsglm),
                          factor(test_bias$binary_state))
```

This produced a slightly lower overall accuracy of 0.7332646

The next model to be attempted was a K-nearest neighbours or knn model:

```
train_bias <- train_bias %>%
  mutate(binary_state=factor(binary_state))
test_bias <- test_bias %>%
  mutate(binary_state=factor(binary_state))

fitknn <- train(binary_state ~ b_y + b_m + b_d + b_co +
                  b_cat + b_g + b_bl + b_dd,
                method="knn",
                data=train_bias,
                tuneGrid=data.frame(k=seq(3,30,3)))
y_hatknn <- predict(fitknn, newdata=test_bias,
                    type="raw")
predictionsknn <- ifelse(y_hatknn > 0.5, TRUE, FALSE)
cm_knn <- confusionMatrix(factor(predictionsknn),
                          test_bias$binary_state)
```

Unfortunately this did not complete after nearly 18 hours of processing.

The next model attempted was QDA:

```
fitqda <- train(binary_state ~ b_y + b_m + b_d + b_co +
                  b_cat + b_g + b_bl + b_dd,
                method="qda", data=train_bias)
predictionsqda <- predict(fitqda, newdata=test_bias)
cm_qda <- confusionMatrix(predictionsqda,
                          test_bias$binary_state)
```

This again produced a lower accuracy than the linear regression model of 0.7204.

The next model attempted was a random forest model using the randomForest library:

```
fitrf <- train(binary_state ~ b_y + b_m + b_d + b_co +
                  b_cat + b_g + b_bl + b_dd,
               method="rf", data=train_bias)
predictionsrf <- predict(fitrf, newdata=test_bias)
cm_rf <- confusionMatrix(predictionsrf,
                         test_bias$binary_state)
```

Unfortunately after several hours of training this model also did not complete due to the computational requirements.

Due to the long training times encountered, a decision was made to attempt to train models with a subset of the training data.

First a training set with just 1% of the training data was created.

```
set.seed(1, sample.kind="Rounding")
train_index001 <- createDataPartition(
  y = train_bias$binary_state,
  times = 1, p = 0.01,
  list = FALSE)
train_bias001 <- train_bias[train_index001,]
train_bias001 <- train_bias001 %>%
  mutate(binary_state=factor(binary_state)) %>%
  select(b_y, b_m, b_d, b_co, b_cat, b_g, b_bl,
         b_dd, binary_state)
```

Then a training set with 10% of the training data was made.

```
set.seed(1, sample.kind="Rounding")
train_index01 <- createDataPartition(
```

```
  y = train_bias$binary_state, times = 1,
  p = 0.1,
  list = FALSE)
train_bias01 <- train_bias[train_index01,]

train_bias01 <- train_bias01 %>%
  mutate(binary_state=factor(binary_state)) %>%
  select(b_y, b_m, b_d, b_co, b_cat, b_g, b_bl,
         b_dd, binary_state)
```

Finally a training set with 30% of the training data was put in place.

```
set.seed(1, sample.kind="Rounding")
train_index03 <- createDataPartition(
  y = train_bias$binary_state, times = 1,
  p = 0.3,
  list = FALSE)
train_bias03 <- train_bias[train_index03,]

train_bias03 <- train_bias03 %>%
  mutate(binary_state=factor(binary_state)) %>%
  select(b_y, b_m, b_d, b_co, b_cat, b_g, b_bl,
         b_dd, binary_state)
```

To assess the effectiveness of training with a smaller subset of the data, a linear regression model was trained to see the effect on accuracy of using a much smaller training set.

```
fitlm <- lm(binary_state ~ b_y + b_m + b_d + b_co + b_cat +
              b_g + b_bl + b_dd, data=train_bias001)
y_hatlm <- predict(fitlm, test_bias)
predictionslm <- ifelse(y_hatlm > 0.5, TRUE, FALSE)
cm_lm <- confusionMatrix(factor(predictions),
                         factor(test_bias$binary_state))
```

This produced an accuracy of 0.7324 - not too different from that acheived on the full training set.

In light of this an attempt was undertaken to train a random forest on the smallest reduced training data set. On this occasion the Rborist algorithm as used in anticipation that it would be faster to train than the randomForest library.

```
fitrbo <- train(binary_state ~ b_y + b_m + b_d + b_co +
                  b_cat + b_g + b_bl + b_dd,
                method="Rborist", data=train_bias001)
predictionsrbo <- predict(fitrbo, newdata=test_bias)
cm_rbo <- confusionMatrix(predictionsrbo,
                          factor(test_bias$binary_state))
```

Unfortunately, this model only managed an an accuracy of 0.6 - not very good.

```
fitrbo <- train(binary_state ~ b_y + b_m + b_d + b_co +
                  b_cat + b_g + b_bl + b_dd,
                method="Rborist", data=train_bias01)
predictionsrbo <- predict(fitrbo, newdata=test_bias)
cm_rbo <- confusionMatrix(predictionsrbo,
                          factor(test_bias$binary_state))
```

An attempt with a 10x larger subset of the training data resulted in an even lower accuracy of 0.5968.

A fresh attempt was then made to build a k-nearest neighbours model on the smallest data set.

```
fitknn <- train(binary_state ~ b_y + b_m + b_d + b_co +
                  b_cat + b_g + b_bl + b_dd,
                method="knn",
                data=train_bias001,
                tuneGrid=data.frame(k=7))
predictionsknn <- predict(fitknn, newdata=test_bias,
                          type="raw")
cm_knn <- confusionMatrix(predictionsknn,
                          factor(test_bias$binary_state))
```

This delivered an accuracy of 0.7025. This was promising enough that an attempt was then made to tune the k parameter.

```
fitknn <- train(binary_state ~ b_y + b_m + b_d + b_co +
                  b_cat + b_g + b_bl + b_dd,
                method="knn",
                data=train_bias001,
                tuneGrid=data.frame(k=seq(30,100,10)))
```

It was found that the best k value was 50.

```
predictionsknn <- predict(fitknn, newdata=test_bias,
                          type="raw")
cm_knn <- confusionMatrix(predictionsknn,
                          factor(test_bias$binary_state))
```

This newly tuned model delivered a prediction accuracy of 0.7258 - still not as good as a linear model, but close. The same model was then trained on progressively larger subsets of the training data.

```
fitknn <- train(binary_state ~ b_y + b_m + b_d + b_co +
                  b_cat + b_g + b_bl + b_dd,
                method="knn",
                data=train_bias01,
                tuneGrid=data.frame(k=50))

predictionsknn <- predict(fitknn, newdata=test_bias,
                          type="raw")
cm_knn <- confusionMatrix(predictionsknn,
                          factor(test_bias$binary_state))
```

Training on 10% of the training data delivered a prediction accuracy of 0.7301 - still not as good as a linear model, but again an improvement.

Training the same model on 30% of the training data took considerably longer and only deliverd a small incremental improvement:

```
fitknn <- train(binary_state ~ b_y + b_m + b_d + b_co +
                  b_cat + b_g + b_bl + b_dd,
                method="knn",
                data=train_bias03,
                tuneGrid=data.frame(k=50))

predictionsknn <- predict(fitknn, newdata=test_bias,
                          type="raw")
cm_knn <- confusionMatrix(predictionsknn,
                          factor(test_bias$binary_state))
```

A prediction accuracy of 0.7317.

Another attempt was made at training a random forest algorithm on a subset of the training data, this time using the ‘ranger’ model. Again, this was selected in anticipation of high training speed.

```
fitra <- train(binary_state ~ b_y + b_m + b_d + b_co +
                 b_cat + b_g + b_bl + b_dd,
               method="ranger", data=train_bias01,
               num.trees=50)
predictionsra <- predict(fitra, newdata=test_bias)
cm_ra <- confusionMatrix(predictionsra,
                         factor(test_bias$binary_state))
```

This model delivered an accuracy of 0.7315 pretty quickly. Again, an attempt was made to tune the model, by increasing the number of trees in the random forest, first to 200:

```
fitra <- train(binary_state ~ b_y + b_m + b_d + b_co +
                 b_cat + b_g + b_bl + b_dd,
               method="ranger", data=train_bias01,
               num.trees=200)
predictionsra <- predict(fitra, newdata=test_bias)
cm_ra <- confusionMatrix(predictionsra,
                         factor(test_bias$binary_state))
```

This delivered an accuracy of 0.7359 - better than the linear model! Increasing the tree count again further improved the result:

```
fitra <- train(binary_state ~ b_y + b_m + b_d + b_co +
                 b_cat + b_g + b_bl + b_dd,
               method="ranger", data=train_bias01,
               num.trees=500)
predictionsra <- predict(fitra, newdata=test_bias)
cm_ra <- confusionMatrix(predictionsra,
                         factor(test_bias$binary_state))
```

500 trees delivered an accuracy of 0.7365. An attempt was then made to train this model on a larger subset of the training data - 30%:

```
fitra <- train(binary_state ~ b_y + b_m + b_d + b_co +
                 b_cat + b_g + b_bl + b_dd,
               method="ranger", data=train_bias03,
               num.trees=500)
predictionsra <- predict(fitra, newdata=test_bias)
cm_ra <- confusionMatrix(predictionsra,
                         factor(test_bias$binary_state))
```

This delivered an accuracy of 0.7437 That's a significant improvement.

Next, attention was turned to training a neural net, using the nnet model. This model was selected as it has relatively few tuning parameters.

Even on a very small data set, the model delivered promising results: an accuracy of 0.7322

```
fitnn <- train(binary_state ~ b_y + b_m + b_d + b_co +
                 b_cat + b_g + b_bl + b_dd,
               method="nnet", data=train_bias01)
predictionsnn <- predict(fitnn, newdata=test_bias)
cm_nn <- confusionMatrix(predictionsnn,
                         factor(test_bias$binary_state))
```

Training on 10% of the training data delivered an accuracy of 0.7347 - once again, better than a linear model.

```
fitnn <- train(binary_state ~ b_y + b_m + b_d + b_co +
                 b_cat + b_g + b_bl + b_dd,
               method="nnet", data=train_bias03)
predictionsnn <- predict(fitnn, newdata=test_bias)
cm_nn <- confusionMatrix(predictionsnn,
                         factor(test_bias$binary_state))
```

Training on 30% of the training set however, only delivered an accuracy of 0.7339 - still better than a linear model, but slightly reduced in accuracy from training on the smaller set. This may be an indication of overtraining.

Finally, an attempt was made to build an ensemble prediction, by averaging the predictions of the linear model, the knn model, the ranger model and the neural net.

```
ensemble <- (as.numeric(predictionslm) +
               (as.numeric(predictionsknn)-1) +
               (as.numeric(predictionsra)-1) +
               (as.numeric(predictionsnn)-1)) / 4
ensemble <- ifelse(ensemble > 0.5, TRUE, FALSE)
ensemble <- factor(ensemble)
cm_en <- confusionMatrix(ensemble,
                         factor(test_bias$binary_state))
```

This ensemble only delivered an accuracy of 0.7371 - less than the random forest on it's own, but better than any of the other models individually.

Results

The results obtained from testing on the test set can be tabulated so:

Models	Accuracy
Linear Regression	0.7333677
Generalized Linear Model	0.7332646
Quadratic Discriminant Analysis	0.7204000
Linear Regression (1% of training data)	0.7324000
Rborist (1% of training data)	0.6000000
Rborist (10% of training data)	0.5968000
K nearest neighbours (1% of training data, k=7	0.7025000
K nearest neighbours (1% of training data, k=50)	0.7258000
K nearest neighbours (10% of training data, k=50)	0.7301000
K nearest neighbours (30% of training data, k=50)	0.7317000
Ranger (10% of training data, 50 trees)	0.7315000
Ranger (10% of training data, 200 trees)	0.7359000
Ranger (10% of training data, 500 trees)	0.7365000
Ranger (30% of training data, 500 trees)	0.7437000
Neural net (1% of training data)	0.7322000
Neural net (10% of training data)	0.7347000
Neural net (30% of training data)	0.7339000
Ensemble (Linear, KNN, Ranger, Neural net)	0.7371000

Ranger was the top performing model, and GLM, QDA, Rborist and KNN were unable to outperform linear regression.

So to finally validate the Ranger model, it's accuracy needed to be assessed on the validation set.

First the regularized bias terms need to be added to the validation data:

```
validation_bias <- validation_clean %>%
  left_join(year_avgs, by='year') %>%
  left_join(month_avgs, by='month') %>%
  left_join(day_avgs, by='day') %>%
  left_join(country_avgs, by='location_country') %>%
  left_join(category_avgs, by='category_name') %>%
  left_join(goal_avgs, by='log_goal') %>%
  left_join(blurb_avgs, by='blurb_length') %>%
  left_join(deadline_avgs, by='days_to_deadline')
```

```
predictionsra_v <- predict(fitra, newdata=validation_bias)
cm_ra_v <- confusionMatrix(
    predictionsra_v,
    factor(validation_bias$binary_state)
  )
```

Testing against this validation data delivers a final accuracy of **0.7427**.

Conclusion

In this paper we discussed the application of various machine learning models to the task of predicting the outcome of Kickstarter projects over 10 years, from 2009 to 2019. The predictive quality of a variety of variables was examined, and models ranging from linear regression, through knn, random forest and neural networks were trained on the data. Using data only knowable at the outset of each project a prediction accuracy of 74.3% was acheived. While not that impressive in machine learning terms, it is nevertheless remarkable that before any possible backers have seen a project, it's likelihood of success is already set and discernible from data supplied when it is first posted to the Kickstarter website.

Due to the challenges of dealing with a large volume of data and assessing a wide variety of algorithms, relatively little attention was given to tuning each algorithm precisely. It therefore seems likely that improved results could be obtained by more careful tuning.

It may also be the case, that forecasting other variables such as the number of backers or whether the project will be picked as a Staff Pick may yield even stronger results. Finally, it is noted that even a week after a project's launch much more would be known about the project's popularity, so it may be that a model built to give forecasts throughout the lifecycle of a project could deliver much more accurate forecasts even shortly after a project has been launched. All of these possible approaches would seem to be fruitful avenues of future research.