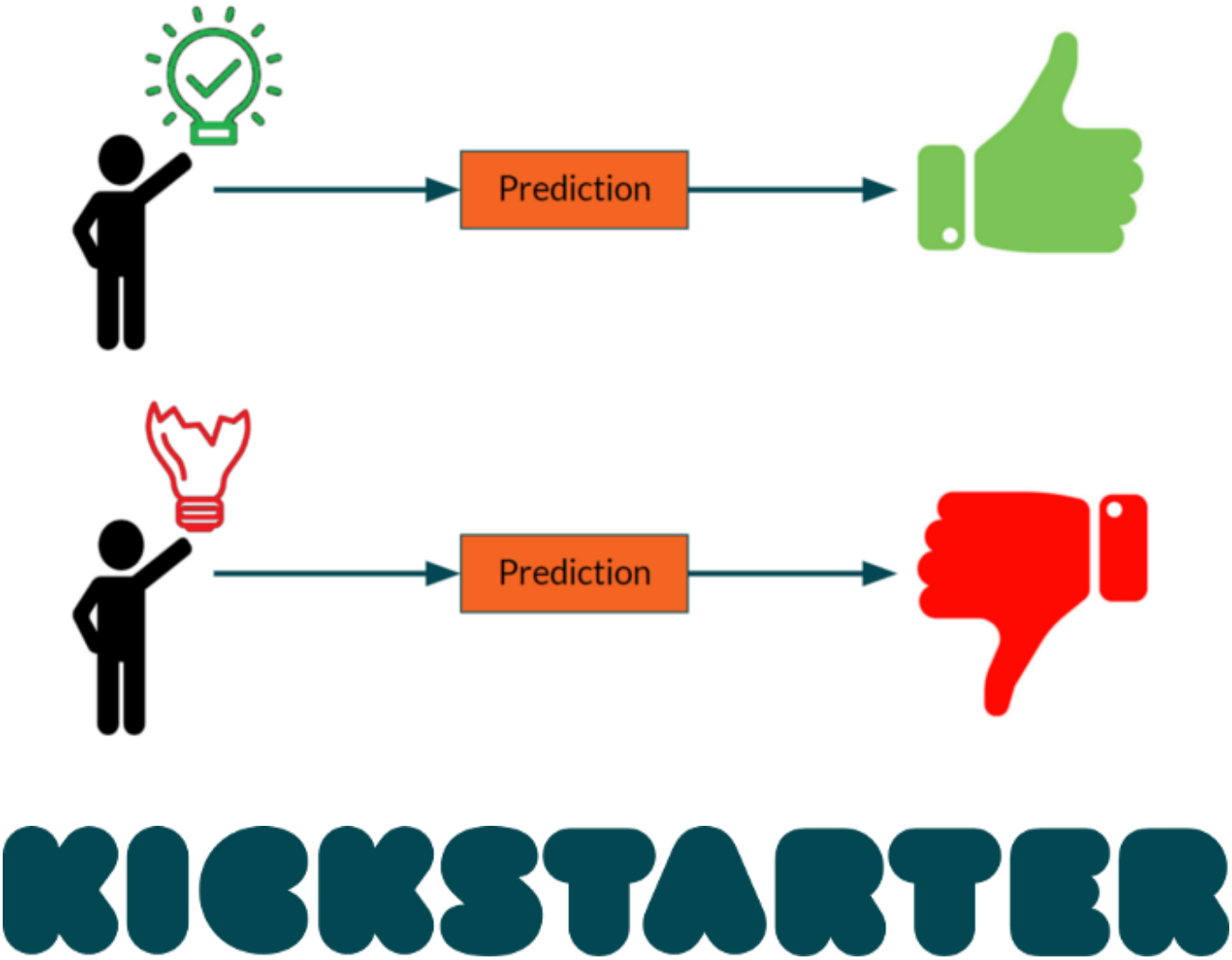
Using machine learning to predict Kickstarter success

Is Kickstarter right for your project? How can you optimise for success?

Laura Lewis Follow
Apr 9, 2019 · 14 min read ★



Project aims

In recent years, the range of funding options for projects created by individuals and small companies has expanded considerably. In addition to savings, bank loans, friends & family funding and other traditional options, crowdfunding has become a popular and readily available alternative. Kickstarter, founded in 2009, is one particularly well-known and popular crowdfunding platform. It has an all-or-nothing funding model, whereby a project is only funded if it meets its goal amount; otherwise no money is given by backers to a project.

A huge variety of factors contribute to the success or failure of a project — in general, and also on Kickstarter. Some of these are able to be quantified or categorised, which allows for the construction of a model to attempt to predict whether a project will succeed or not. The aim of this project is to construct such a model and also to analyse Kickstarter project data more generally, in order to help potential project creators assess whether or not Kickstarter is a good funding option for them, and what their chances of success are.

Data source

The dataset used in this project was downloaded in .csv format from a webscrape conducted by a webscraping site called Web Robots. The dataset contains data on all projects hosted on Kickstarter between the company's launch in April 2009 until the date of the webscrape on 14 March 2019. The dataset contains 209,222 projects, although some of these are duplicates.

Cleaning and pre-processing

A fair amount of cleaning was required to get the dataset into a format suitable for applying machine learning models. If you just can't get enough of `df.isna()` and `df.drop()` you can check out the full Jupyter notebook code in my GitHub repository.

After duplicates and irrelevant rows were dropped (e.g. projects which were cancelled mid-campaign, or which were still live), I was left with a decent sized dataset of 168,979 projects.

The columns which were kept or calculated were:

- The project goal (in USD)
- Campaign length — number of days from launch to deadline
- Number of days from page creation to project launch
- Blurb word length
- Name word length
- Whether the project was highlighted as a staff pick (one-hot encoded)
- Category (one-hot encoded)
- Country (one-hot encoded)
- Month a project was launched in (one-hot encoded)
- Month of a project's deadline (one-hot encoded)
- Day of the week a project was launched on (one-hot encoded)
- Day of the week of a project's deadline (one-hot encoded)
- Two-hour time window a project was launched in (one-hot encoded)
- Two-hour time window of a project's deadline (one-hot encoded)

Some features were initially retained for exploratory data analysis (EDA) purposes, but were then dropped in order to use machine learning models. These included features that are related to outcomes (e.g. the amount pledged and the number of backers) rather than related to the properties of the project itself (e.g. category, goal, length of campaign).

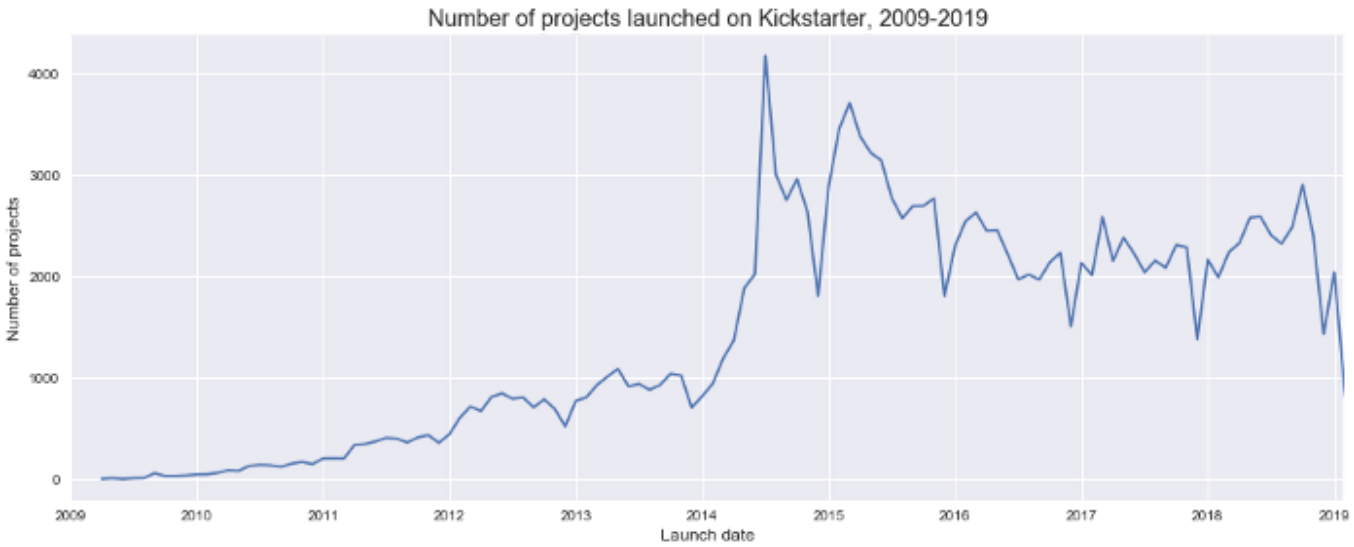
Exploratory data analysis

Now for the colourful part.

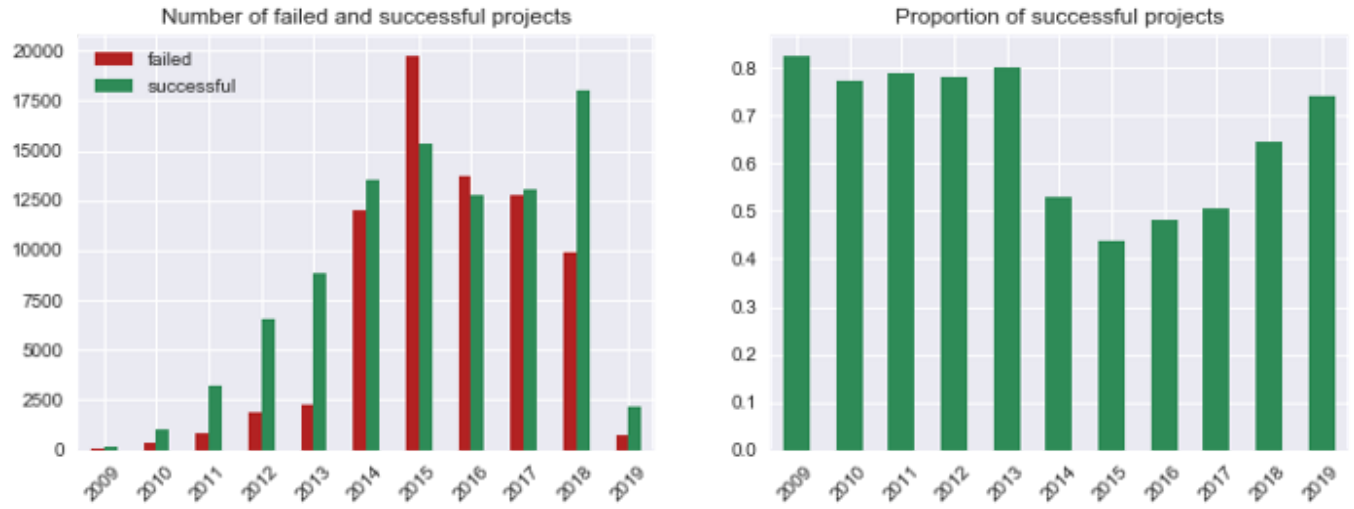
Kickstarter has grown massively since its launch in 2009, particularly during 2014 when expansion really started to ramp up. The proportion of projects that succeed decreased considerably at this point, however, as the site was flooded with a much larger number

of projects. The success rate has been on the increase in recent years though — so there's still hope.

> ***Overall, 56% of completed projects (i.e. those that have finished and weren't cancelled or suspended) were successful.***



Changes over time in the number of projects launched on Kickstarter



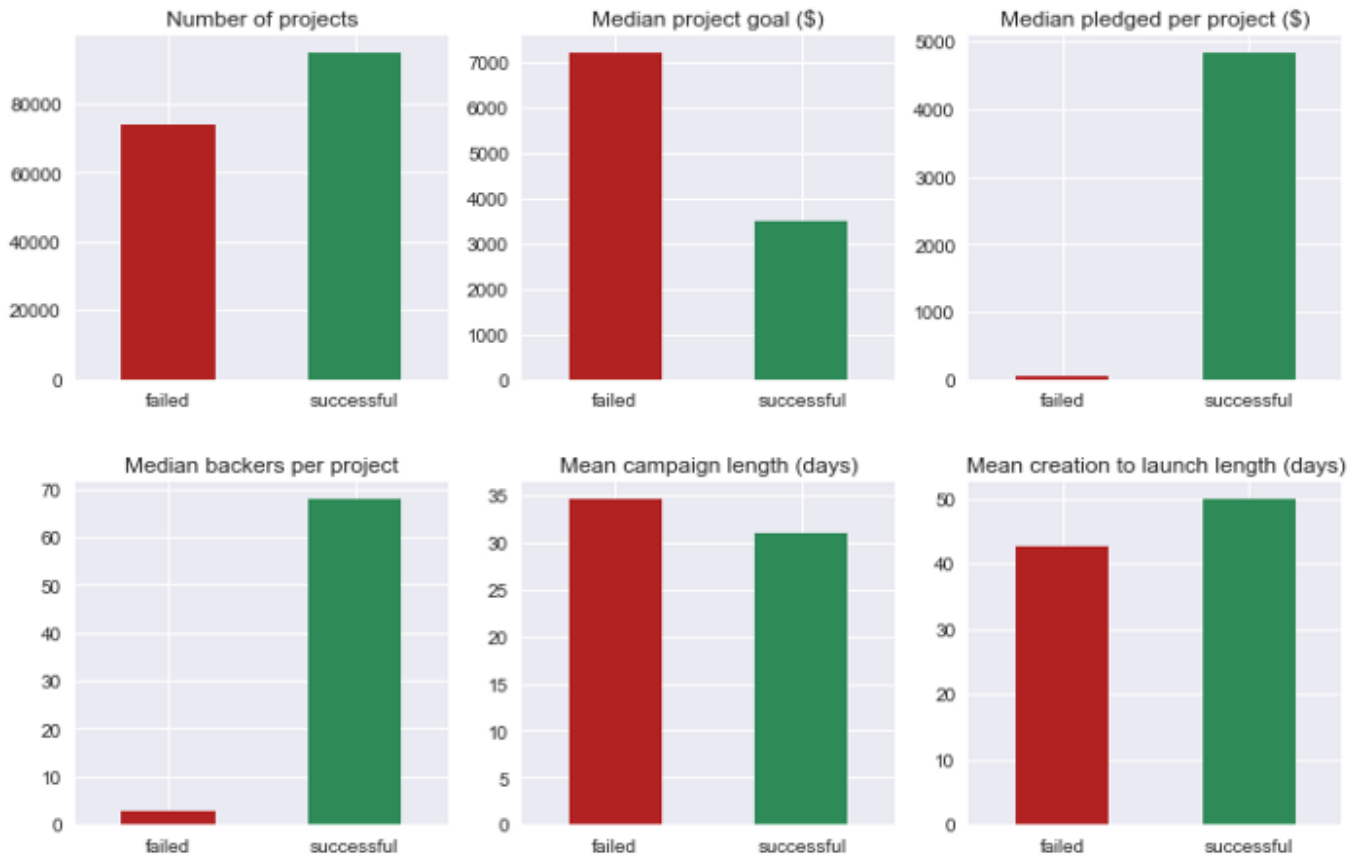
Changes over time in project successes and failures

The graphs below show the differences in some of the features between successful and failed projects. The key takeaways from this are:

- Unsurprisingly, **successful projects tend to have smaller (and therefore more realistic) goals** — the median amount sought by successful projects is about half

that of failed projects (medians are used due to high positive skew of funding and goal amounts).

- The differences in the median amount pledged per project are more surprising. The median amount pledged per successful project is notably higher than the median amount requested, suggesting that **projects that meet their goal tend to go on to gain even more funding, and become 'over-funded'**.
- On a related note, the difference between failed and successful companies is much larger in terms of amount pledged and the number of backers, compared to goal amount. Probably **once potential backers see that a project looks like it will be successful, they are much more likely to jump on the bandwagon** and fund it.
- Successful projects have slightly **shorter campaign lengths**, but take slightly longer to launch (measured from the time the project was first created on the site).
- Roughly 20% of successful projects were highlighted on the site as staff picks. It does not seem unreasonable to suggest a causative relationship here, i.e. that **projects that are chosen as staff picks are much more likely to go on to be successful**, and that only a few staff picks go on to fail.



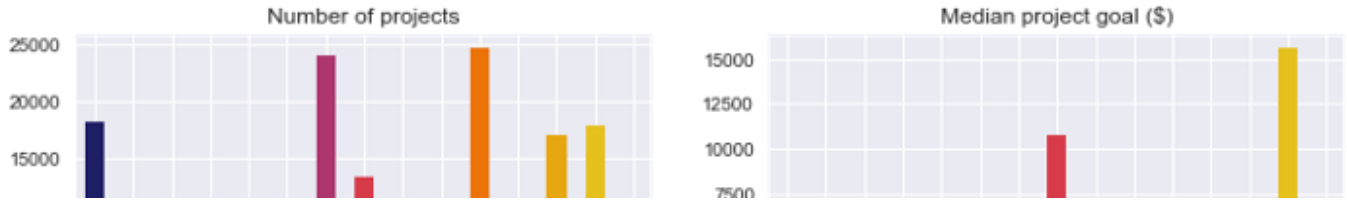
Comparison of features between successful and failed projects

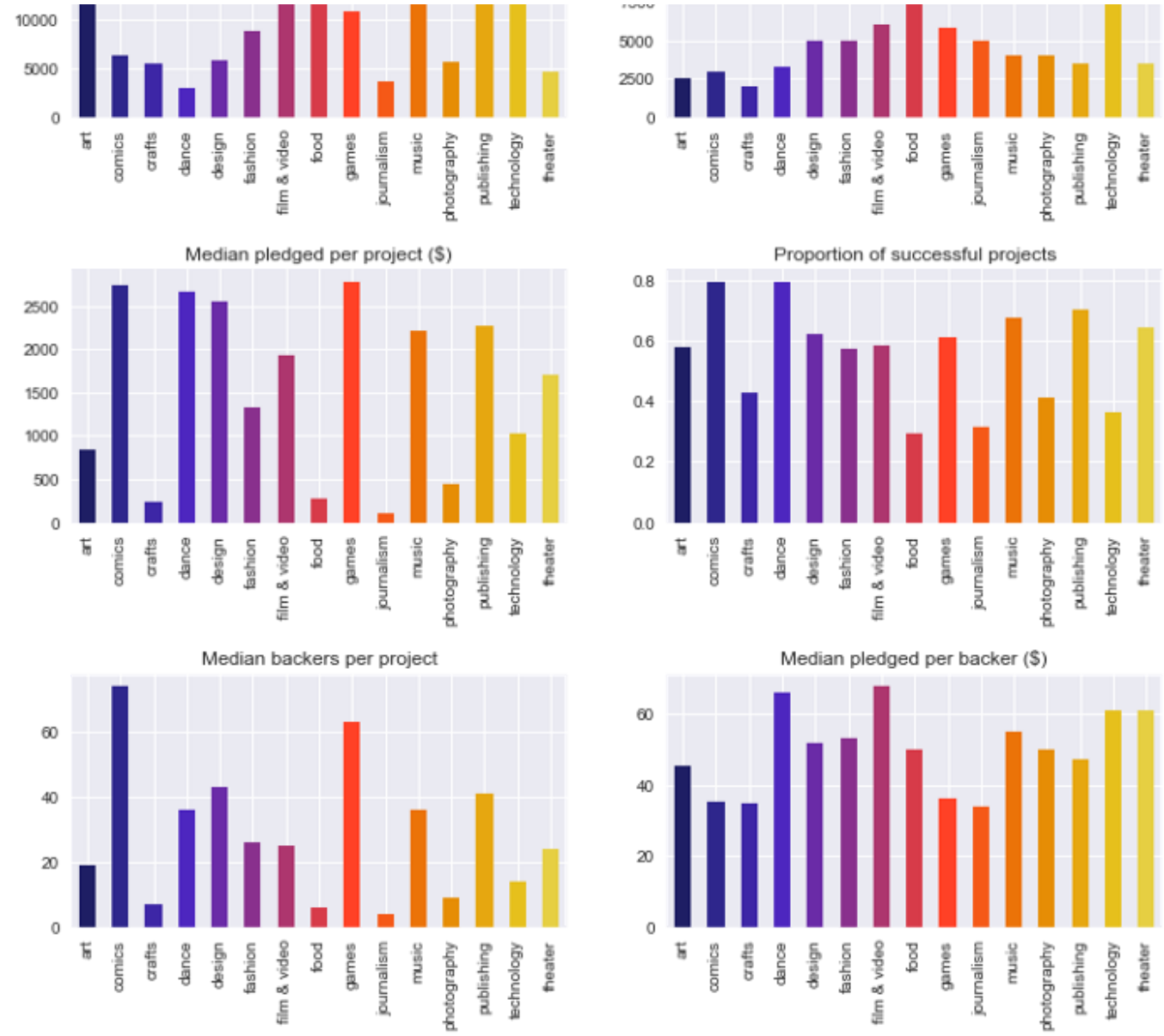
Various other features were explored, in terms of project number, goal and funding amounts, backers and success rates. For example, the graphs below show the differences between different project categories (the code is also provided). The key takeaways from this are:

- The best project types to launch on Kickstarter are **comics** (on the grounds of success rate, number of backers and amount pledged), **dance** (success rate and amount pledged) and **games** (amount pledged and number of backers). This is probably at least partly due to their relatively small funding goals — as noted above, projects with smaller goals tend to be more successful.
- Although **comics** and **games** tend to attract the most backers, each backer tends to pledge relatively little. **Dance** and **film & video** tend to attract the most generous backers.
- **Technology** projects have the highest median goal size by far. However, they are towards the bottom of the leaderboard in terms of the median amount actually pledged.
- The worst performing categories are **food**, **journalism** and **technology.**

```
# Code used to create the graphs below

# Importing the required libraries
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib import cm
import numpy as np
```

```
# Creating a dataframe grouped by category with columns for failed
and successful
cat_df =
pd.get_dummies(df.set_index('category').state).groupby('category').s
um()

# Plotting
fig, ((ax1, ax2), (ax3, ax4), (ax5, ax6)) = plt.subplots(3, 2,
figsize=(12,12))

color = cm.CMRmap(np.linspace(0.1,0.8,df.category.nunique()))

df.groupby('category').category.count().plot(kind='bar', ax=ax1,
color=color)
ax1.set_title('Number of projects')
ax1.set_xlabel('')

df.groupby('category').usd_goal.median().plot(kind='bar', ax=ax2,
color=color)
ax2.set_title('Median project goal ($)')
ax2.set_xlabel('')

df.groupby('category').usd_pledged.median().plot(kind='bar', ax=ax3,
color=color)
ax3.set_title('Median pledged per project ($)')
ax3.set_xlabel('')

cat_df.div(cat_df.sum(axis=1), axis=0).successful.plot(kind='bar',
ax=ax4, color=color) # Normalizes counts across rows
ax4.set_title('Proportion of successful projects')
ax4.set_xlabel('')

df.groupby('category').backers_count.median().plot(kind='bar',
ax=ax5, color=color)
ax5.set_title('Median backers per project')
ax5.set_xlabel('')

df.groupby('category').pledge_per_backer.median().plot(kind='bar',
ax=ax6, color=color)
ax6.set_title('Median pledged per backer ($)')
ax6.set_xlabel('')

fig.subplots_adjust(hspace=0.6)
plt.show()
```





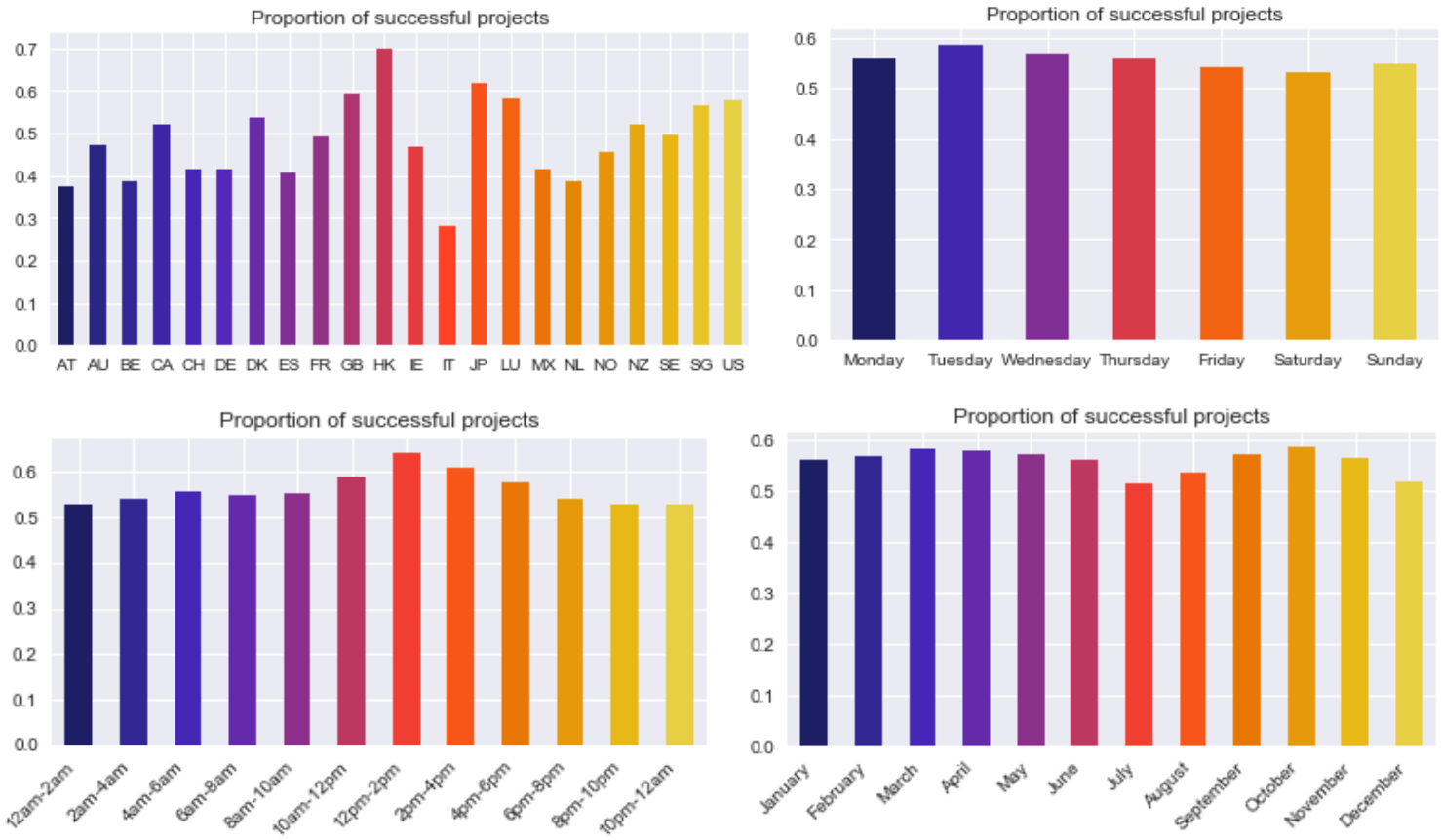
Exploring the 'category' feature

In the interests of space and retinas, only the 'success proportion' graphs will be shown below for additional features. Again, for more detail, feel free to check out my GitHub repository. The key takeaways from this are:

- **Hong Kong is home to a greater proportion of successful projects** (they also have a higher median number of backers and amount of funding).
- **Tuesday is the best day to launch a project**, and **weekends are the worst** (the same pattern holds for the amount raised and the number of backers).
- **12pm to 2pm UTC** is the best time to launch a project — it also has the greatest median number of backers and amount of funding. **6pm to 4am UTC** is the worst

time to launch.

- **October is the best month to launch a project** — it also has the greatest median number of backers and amount of funding. **July and December are the worst months.**



Top left: success rates by country. Top right: success rates by the day of the week on which projects were launched. Bottom left: success rates by the time of day at which projects were launched (in UTC/GMT). Bottom right: success rates by the month in which projects were launched.

Preparing the data for machine learning

The ultimate goal of this project was to create a model that could predict, with a good level of accuracy, whether a project was likely to succeed or fail.

In order to prepare the data for machine learning, the following steps were taken (code below):

1. One-hot encoding categorical variables.

2. Separating the data into the dependent target variable ‘y’ (in this case ‘state’, i.e. project success or failure) and the independent features ‘X’.

3. Transforming the features in X so that they are all on the same scale. For this project, StandardScaler from Scikit-learn was used to transform each feature to a mean of 0 and a standard deviation of 1.

4. The data was separated into a training and test set, for robust evaluation of the models.

```
# Importing the required libraries
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split

# 1) Creating dummy variables
df_transformed = pd.get_dummies(df_transformed)

# 2) Separating into X and y
X_unscaled = df_transformed.drop('state', axis=1)
y = df_transformed.state

# 3) Transforming the data
scaler = StandardScaler()
X = pd.DataFrame(scaler.fit_transform(X_unscaled),
columns=list(X_unscaled.columns))

# 4) Splitting into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X,y,
test_size=0.3, random_state=123)
```

It is good practice to choose an evaluation method before running machine learning models — not after. The weighted average F1 score was chosen. The F1 score calculates the harmonic mean between precision and recall, and is a suitable measure because there is no preference for false positives or false negatives in this case (both are equally bad). The weighted average will be used because the classes are of slightly different sizes, and we want to be able to predict both successes and failures.

> It is good practice to choose an evaluation method before running machine learning models — not after.

Model 1: vanilla logistic regression

Logistic regression can be used as a binary classifier in order to predict which of two categories a data point falls in to.

To create a baseline model to improve upon, a logistic regression model was fitted to the data, with default parameters.

```
# Importing the required libraries
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report

# Fitting a logistic regression model with default parameters
logreg = LogisticRegression()
logreg.fit(X_train,y_train)

# Making predictions
y_hat_train = logreg.predict(X_train)
y_hat_test = logreg.predict(X_test)

# Logistic regression scores
print("Logistic regression score for training set:",
round(logreg.score(X_train, y_train),5))
print("Logistic regression score for test set:",
round(logreg.score(X_test, y_test),5))
print("\nClassification report:")
print(classification_report(y_test, y_hat_test))
```

```
Logistic regression score for training set: 0.71157
Logistic regression score for test set: 0.70878

Classification report:
              precision    recall  f1-score   support

           0       0.70      0.59      0.64     22210
           1       0.71      0.80      0.76     28484

   micro avg       0.71      0.71      0.71     50694
```

micro avg	0.71	0.71	0.71	50694
macro avg	0.71	0.70	0.70	50694
weighted avg	0.71	0.71	0.70	50694

Results of the vanilla linear regression model

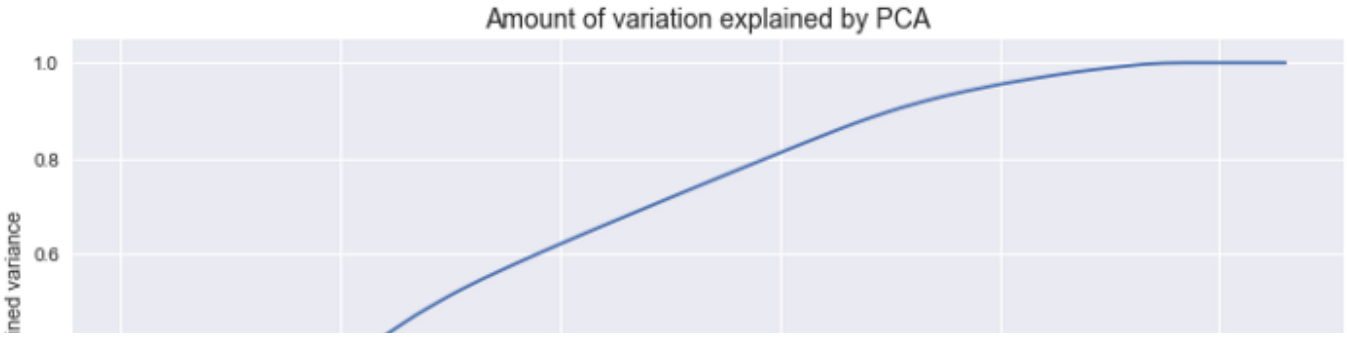
Not bad. The model has a weighted average F1 score of 0.70. The aim is now to improve upon this score.

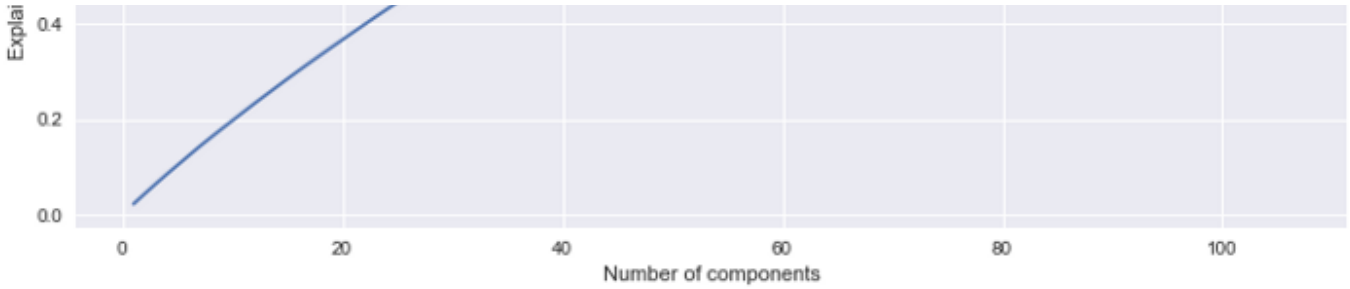
Principal Component Analysis

There were a large number of features (106) in the dataset used for the initial logistic regression model. PCA (Principal Component Analysis) was used to reduce this into a smaller number of components which still explain as much variation in the data as possible. This can help improve model fitting and accuracy.

The graph below (produced by the code below) shows that there was no obvious cut-off for the number of components to use in PCA.

```
# Importing the required libraries
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA

# Fitting PCA
pca = PCA()
pca.fit_transform(X)
explained_var = np.cumsum(pca.explained_variance_ratio_)

# Plotting the amount of variation explained by PCA with different
numbers of components
plt.plot(list(range(1, len(explained_var)+1)), explained_var)
plt.title('Amount of variation explained by PCA', fontsize=14)
plt.xlabel('Number of components')
plt.ylabel('Explained variance');
```





Plotting the amount of variation in the data explained by PCA using various numbers of components

The following results were found:

```
Number of components explaining 80% of variance: 58
Number of components explaining 90% of variance: 70
Number of components explaining 99% of variance: 90
```

To choose the number of components to use in the machine learning models, each of these values was plugged into a pipeline for a logistic regression model using the default parameters:

```
# Running a for loop to test different values of n_components
n_comps = [58,70,90]
for n in n_comps:
 pipe = Pipeline([('pca', PCA(n_components=n)), ('clf',
LogisticRegression())])
 pipe.fit(X_train, y_train)
 print("\nNumber of components:", n)
 print("Score:", round(pipe.score(X_test, y_test),5))
```

The results showed that the score is highest for 90 components, although the difference is small (c. 3% improvement from 58 components):

```
Number of components: 58
Score: 0.67831

Number of components: 70
Score: 0.6858

Number of components: 90
Score: 0.70799
```

Model 2: logistic regression with PCA and parameter optimisation

The logistic regression model can potentially be further improved by optimising its parameters. GridSearchCV was used to test multiple different regularisation parameters (values of C), penalties (l1 or l2) and models with and without an intercept.

```
# Importing the required libraries
from sklearn.model_selection import GridSearchCV

# Timing how long the model takes to run
logreg_start = time.time()

# Building the pipeline
pipe_logreg = Pipeline([('pca', PCA(n_components=90)),
                    ('clf', LogisticRegression())])

# Creating the parameters to test
params_logreg = [
    {'clf__penalty': ['l1', 'l2'],
     'clf__fit_intercept': [True, False],
        'clf__C': [0.001, 0.01, 1, 10]
    }
]

# Using GridSearchCV to test multiple different parameters
grid_logreg = GridSearchCV(estimator=pipe_logreg,
                  param_grid=params_logreg,
                  cv=5)

grid_logreg.fit(X_train, y_train)

logreg_end = time.time()

logreg_best_score = grid_logreg.best_score_
logreg_best_params = grid_logreg.best_params_

# Printing the results
print(f"Time taken to run: {round((logreg_end - logreg_start)/60,1)}
minutes")
print("Best accuracy:", round(logreg_best_score,2))
print("Best parameters:", logreg_best_params)
```

Results:

```
Time taken to run: 48.56 minutes
Best accuracy: 0.71
Best parameters: {'clf__C': 10, 'clf__fit_intercept': True,
'clf__penalty': 'l2'}
```

A classification report and a confusion matrix were then produced for the logistic regression model using the best parameters (according to the accuracy score).

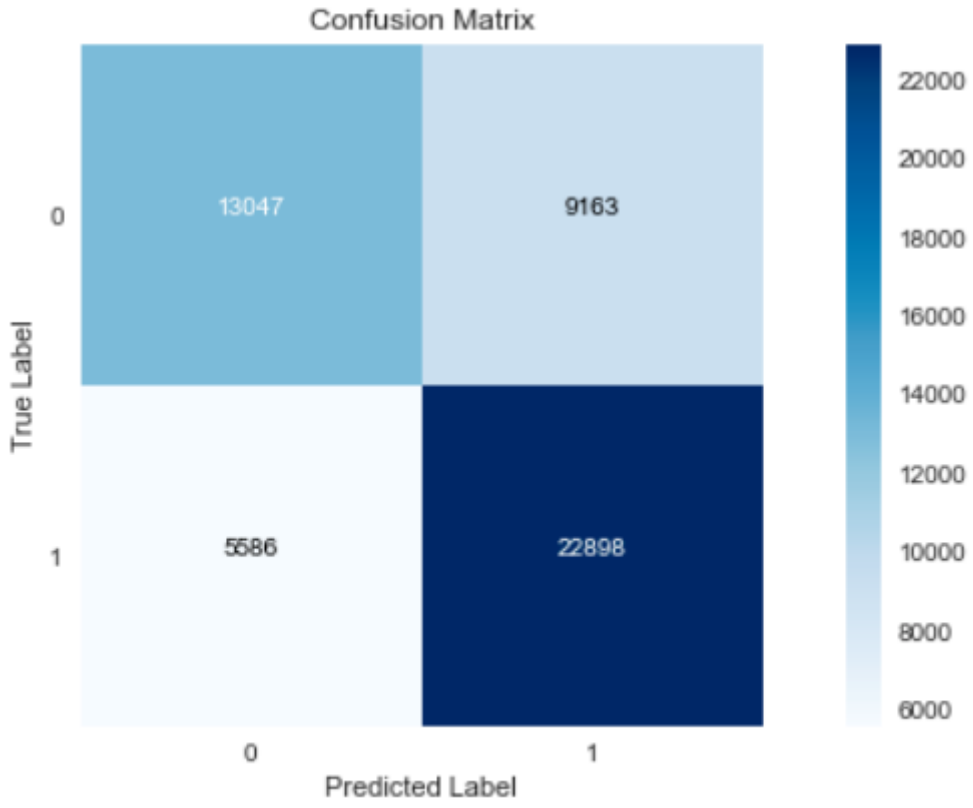
The confusion matrix was produced using the following function:

```
def plot_cf(y_true, y_pred, class_names=None, model_name=None):
    """Plots a confusion matrix"""
    cf = confusion_matrix(y_true, y_pred)
    plt.imshow(cf, cmap=plt.cm.Blues)
    plt.grid(b=None)
    if model_name:
        plt.title("Confusion Matrix: {}".format(model_name))
    else:
        plt.title("Confusion Matrix")
    plt.ylabel('True Label')
    plt.xlabel('Predicted Label')

    class_names = set(y_true)
    tick_marks = np.arange(len(class_names))
    if class_names:
        plt.xticks(tick_marks, class_names)
        plt.yticks(tick_marks, class_names)

    thresh = cf.max() / 2.

    for i, j in itertools.product(range(cf.shape[0]),
range(cf.shape[1])):
        plt.text(j, i, cf[i, j], horizontalalignment='center',
color='white' if cf[i, j] > thresh else 'black')

plt.colorbar()
```

The full results for the best logistic regression model are below:

```
Logistic regression score for training set: 0.71106
Logistic regression score for test set: 0.70906

Classification report:
              precision    recall  f1-score   support
```

0	0.70	0.59	0.64	22210
1	0.71	0.80	0.76	28484
micro avg	0.71	0.71	0.71	50694
macro avg	0.71	0.70	0.70	50694
weighted avg	0.71	0.71	0.70	50694



Results of the best logistic regression model

After hyperparameter tuning, the model's accuracy score is the same as the logistic regression model using default parameters (0.70 weighted average F1 score). Disappointing.

Model 3: Random Forests

Next, a Random Forest classifier was used. The Random Forest algorithm is a supervised learning algorithm that can be used for classification. It works by building multiple different decision trees to predict which category a data point belongs to.

Again, GridSearchCV was used to test multiple different hyperparameters, in order to optimise the model.

```
# Importing the required libraries
from sklearn.ensemble import RandomForestClassifier

# Using GridSearchCV to test multiple different parameters
rf_start = time.time()

pipe_rf = Pipeline([('pca', PCA(n_components=90)),
                    ('clf', RandomForestClassifier())])

params_rf = [
  {'clf__n_estimators': [100],
   'clf__max_depth': [20, 30, 40],
   'clf__min_samples_split':[0.001, 0.01]
  }
]

grid_rf = GridSearchCV(estimator=pipe_rf,
                  param_grid=params_rf,
                  cv=5)

grid_rf.fit(X_train, y_train)

rf_end = time.time()

rf_best_score = grid_rf.best_score_
rf_best_params = grid_rf.best_params_

print(f"Time taken to run: {round((rf_end - rf_start)/60,1)}
minutes")
print("Best accuracy:", round(rf_best_score,2))
print("Best parameters:", rf_best_params)
```

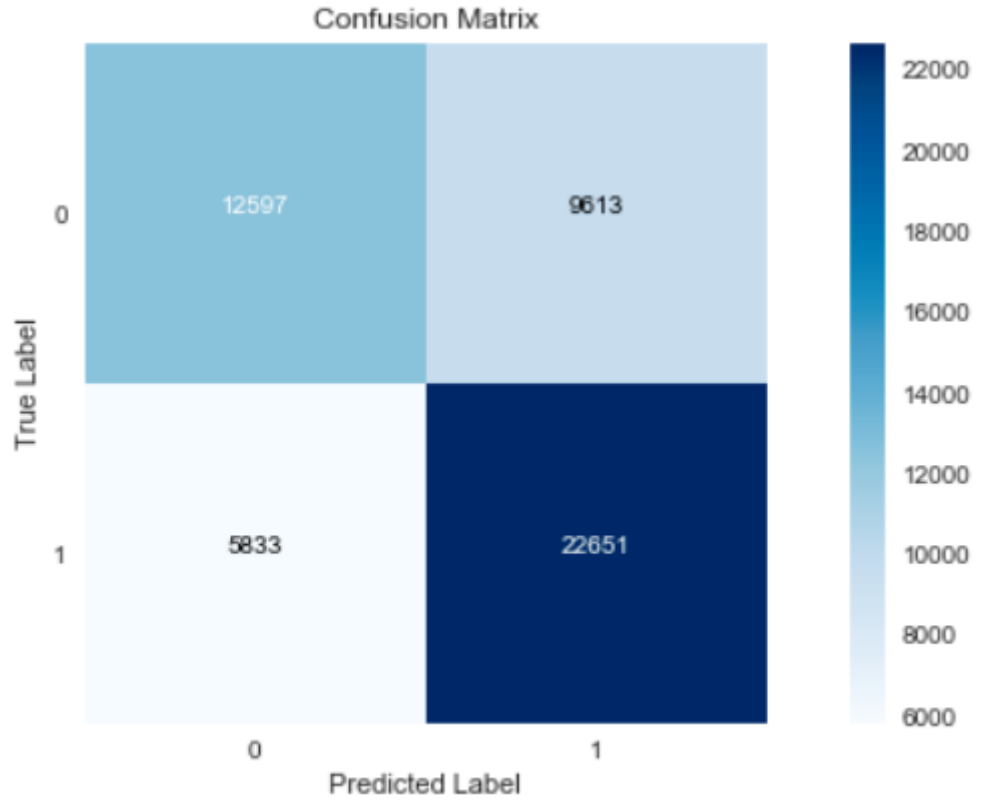
Results:

```
Time taken to run: 72.2 minutes
Best accuracy: 0.7
Best parameters: {'clf__max_depth': 30, 'clf__min_samples_split':
0.001, 'clf__n_estimators': 100}
```

The full results for the best Random Forest model are below:

```
Random Forest score for training set: 0.7835
Random Forest score for test set: 0.69531

Classification report:
```

	precision	recall	f1-score	support
0	0.68	0.57	0.62	22210
1	0.70	0.80	0.75	28484
micro avg	0.70	0.70	0.70	50694
macro avg	0.69	0.68	0.68	50694
weighted avg	0.69	0.70	0.69	50694



Results of the best Random Forest model

After hyperparameter tuning, the model's weighted average F1 score increased from 0.65 for a model with default settings to 0.69. This is similar to, although slightly worse than, the logistic regression model. Also, the difference between the score for the training set and the test set suggests there might be some over-fitting. There may well be more scope for hyperparameter tuning here to further improve the model, but time precluded it.

Model 4: XGBoost

Ah, the darling of the Kaggle world. XGBoost is very 'in' right now. This is a form of gradient boosting algorithm. Similar to Random Forests, it is an ensemble method that

produces multiple decision trees to improve classification of data points, but it uses gradient descent to improve the performance of the model for the data points that are particularly difficult to classify.

Back to good ol' GridSearchCV for hyperparameter testing:

```
# Importing the required libraries
from sklearn.model_selection import GridSearchCV

# Using GridSearchCV to test multiple different parameters
xgb_start = time.time()

pipe_xgb = Pipeline([('pca', PCA(n_components=90)),
                     ('clf', xgb.XGBClassifier())])

params_xgb = [
  {'clf__n_estimators': [100],
   'clf__max_depth': [25, 35],
   'clf__learning_rate': [0.01, 0.1],
   'clf__subsample': [0.7, 1],
   'clf__min_child_weight': [20, 100]
  }
]

grid_xgb = GridSearchCV(estimator=pipe_xgb,
                  param_grid=params_xgb,
                  cv=5)

grid_xgb.fit(X_train, y_train)

xgb_end = time.time()

xgb_best_score = grid_xgb.best_score_
xgb_best_params = grid_xgb.best_params_

print(f"Time taken to run: {round((xgb_end - xgb_start)/60,1)}
minutes")
print("Best accuracy:", round(xgb_best_score,2))
print("Best parameters:", xgb_best_params)
```

Results:

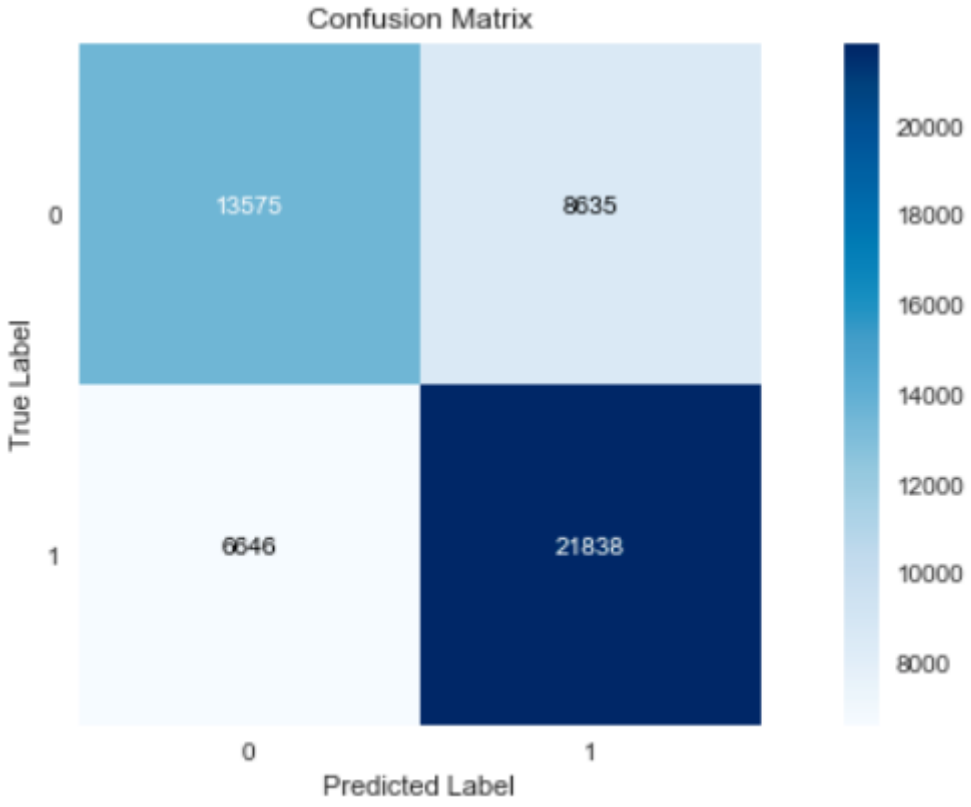
```
Time taken to run: 865.4 minutes
Best accuracy: 0.7
Best parameters: {'clf__learning_rate': 0.1, 'clf__max_depth': 35,
```

```
'clf__min_child_weight': 100, 'clf__n_estimators': 100,
'clf__subsample': 0.7}
```

Yikes. 14 and a half hours, and it was still only able to achieve the same accuracy as the initial regression model (this was also only a 0.01 increase in accuracy from an XGBoost model that was run with default parameters).

The full results for the best XGBoost model are below:

```
XGBoost score for training set: 0.7771
XGBoost score for test set: 0.69856

Classification report:
              precision    recall  f1-score   support

           0       0.67      0.61      0.64     22210
           1       0.72      0.77      0.74     28484

   micro avg       0.70      0.70      0.70     50694
   macro avg       0.69      0.69      0.69     50694
weighted avg       0.70      0.70      0.70     50694
```



Results of the best XGBoost model

As with the Random Forest model, the difference between the accuracy score for the training set and the test set suggests there might be some over-fitting. Again, there may well be more scope for hyperparameter tuning here to further improve the model — but I didn't have another 14 and a half hours to spare.

Model evaluation

Each model was able to achieve an accuracy of about **70%**, after parameter tuning. Although it was relatively easy to reach roughly this level of accuracy, parameter tuning was only able to increase accuracy levels by a small amount. Possibly the reasonably large amount of data for each of only two categories meant that there was enough data for even a relatively simple model (e.g. logistic regression with default settings) to achieve a good level of validation accuracy.

The best Random Forest and XGBoost models created still showed some degree of over-fitting. Further parameter tuning would be required to attempt to reduce this.

The final chosen model is the tuned **logistic regression model**. This is because, although each model was able to achieve a similar level of accuracy for the test set, this is the only model that did not exhibit overfitting.

Interestingly, **each model performed worse at predicting failures compared to successes**, with a lower true negative rate than true positive rate. I.e. it classified quite a few failed projects as successes, but relatively few successful projects as failures. Possibly the factors that might cause a project to fail are more likely to be beyond the scope of the data, e.g. poor marketing, insufficient updates, or not replying to messages from potential backers.

The false positive and false negative rates mean that, if the data about a new project is fed through the model to make a prediction about its success or failure:

- if the project is going to end up being a success, the model will correctly predict this as a success about 80% of the time
- if the project is going to end up being a failure, the model will only correctly predict this as a failure about 60% of the time (and the rest of the time will incorrectly

predict it as a success)

Recommendations for project creators considering Kickstarter

Some of the factors that had a **positive effect** on success rate and/or the amount of money received are:

Most important:

- Smaller project goals
- Being chosen as a staff pick (a measure of quality)
- Comics, dance and games projects
- Projects from Hong Kong

Less important:

- Shorter campaigns
- Taking longer between creation and launch
- Film & video and music projects (popular categories on the site, and fairly successful)
- Launching on a Tuesday (although this is also the most common day to launch a project, so beware the competition)
- Launching in October
- Launching between 12pm and 2pm UTC (this is of course related to the country a project is launched from, but remember that backers can come from all over the world)

Factors which had a **negative effect** on success rate and/or the amount of money received are:

Most negative:

- Large goals

- Food and journalism projects
- Projects from Italy

Less negative:

- Longer campaigns
- Launching on a weekend
- Launching in July or December
- Launching between 6pm and 4am UTC

Overall, Kickstarter is well suited to small, high-quality projects, particularly comics, dance and games. It is less suited to larger projects, particularly food (e.g. restaurants) and journalism projects.

. . .

Thanks for reading this far! If you have any thoughts, comments or suggestions, please add them below.

Machine Learning Data Science Kickstarter Crowdfunding Startup