

# Predicting Kickstart Project State

March 11, 2018

DePaul University  
CSC 478 Final Project

Sidney Fox - 1524992  
Rebecca Tung - 1448196

HYPERLINK "bookmark://_Toc508746853" .....	Introduction <sup>[OBJ]</sup>
3	
HYPERLINK "bookmark://_Toc508746854" .....	Overview <sup>[OBJ]</sup>
3	
HYPERLINK "bookmark://_Toc508746855" .....	Analysis of Features and Observations <sup>[OBJ]</sup>
4	
HYPERLINK "bookmark://_Toc508746856" .....	Initial Transformations <sup>[OBJ]</sup>
9	
HYPERLINK "bookmark://_Toc508746857" .....	Feature Importance <sup>[OBJ]</sup>
9	
HYPERLINK "bookmark://_Toc508746858" .....	Additional Transformations <sup>[OBJ]</sup>
11	
HYPERLINK "bookmark://_Toc508746859" .....	Model Execution and Analysis of Results <sup>[OBJ]</sup>
11	
HYPERLINK "bookmark://_Toc508746860" .....	Conclusion <sup>[OBJ]</sup>
13	
HYPERLINK "bookmark://_Toc508746861" .....	14

## I. Introduction

The purpose of this document is to outline our approach in predicting the status of different projects presented to the public using the Kickstarter platform. The group leveraged many algorithms and functions available in the sklearn library to operationalize the defined machine learning workflow. This document will provide an overview of the dataset leveraged to complete the machine learning project, as well as an in-depth analysis of the methodologies implemented, the requisite results, and any conclusions, as well as constraints or blockers, the group was able to identify while completing the project.

## II. Overview

The dataset used to complete the group's analysis can be found on the Kaggle website as a competition dataset.<sup>1</sup> There are two datasets listed under the competition page. After a comparison of both datasets, it was determined that both datasets contain the same information in terms of relevant data points. The most recent dataset, entitled "ks-projects-201801.csv", contains a "cleansed" dataset, where all of the misnomers and formatting issues have been removed and / or rectified. After conducting this analysis, the group determined that the only dataset required to complete the project was the ks-projects-201801.csv. All of the analysis presented throughout this document solely references this dataset. After settling on the ks-projects-201801.csv dataset, the group created a GitHub repository and uploaded the .csv file in question to said repository. The group used GitHub to build their analysis leveraging the same base dataset, and complete the necessary objectives independently, within a central repository.

Concerning an overview of the features of the dataset specifically, the table below contains relevant high-level statistics in terms of row count and number of columns below:

Statistic	Value
Number of Rows	378,661
Number of Columns	15

In addition to the number of rows and columns, the feature names and corresponding data types can be found on the next page.

---

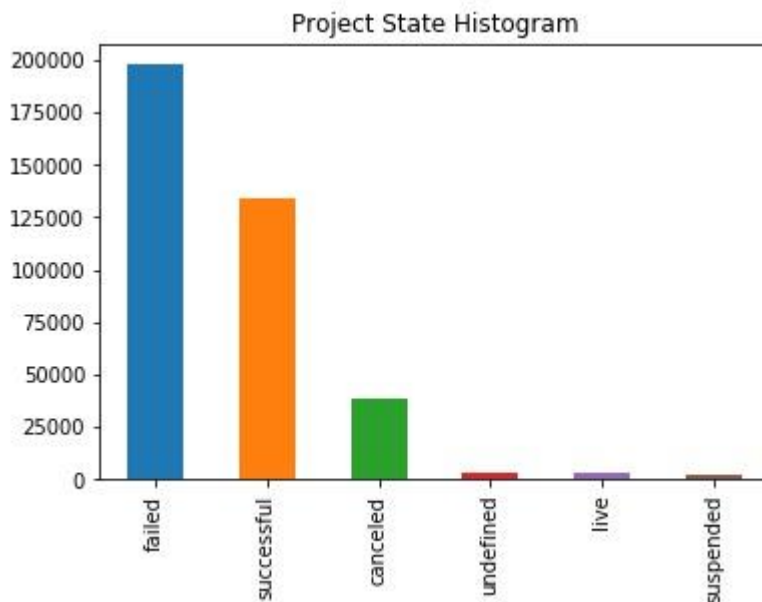
<sup>1</sup> Kemical. "Kickstarter projects." Kaggle.com.  
<https://www.kaggle.com/kemical/kickstarter-projects> (accessed February 13, 2018).

Sequence Number	Column Name	Data Type
1	ID	Numeric
2	name	String
3	category	String
4	main_category	String
5	currency	String
6	deadline	DateTime
7	goal	Numeric
8	launched	DateTime
9	pledged	Numeric
10	state	String
11	backers	Numeric
12	country	String
13	usd pledged	Numeric
14	usd_pledged_real	Numeric
15	usd_goal_real	Numeric

Once the dataset was identified, the group conducted a deeper dive of the data – analyzing features as well as observations.

### III. Analysis of Features and Observations

To gain better insight into the data, the group produced a series of histograms to show distributions by different segments of the data. Please find a histogram, as well as a summary table, of project status below.



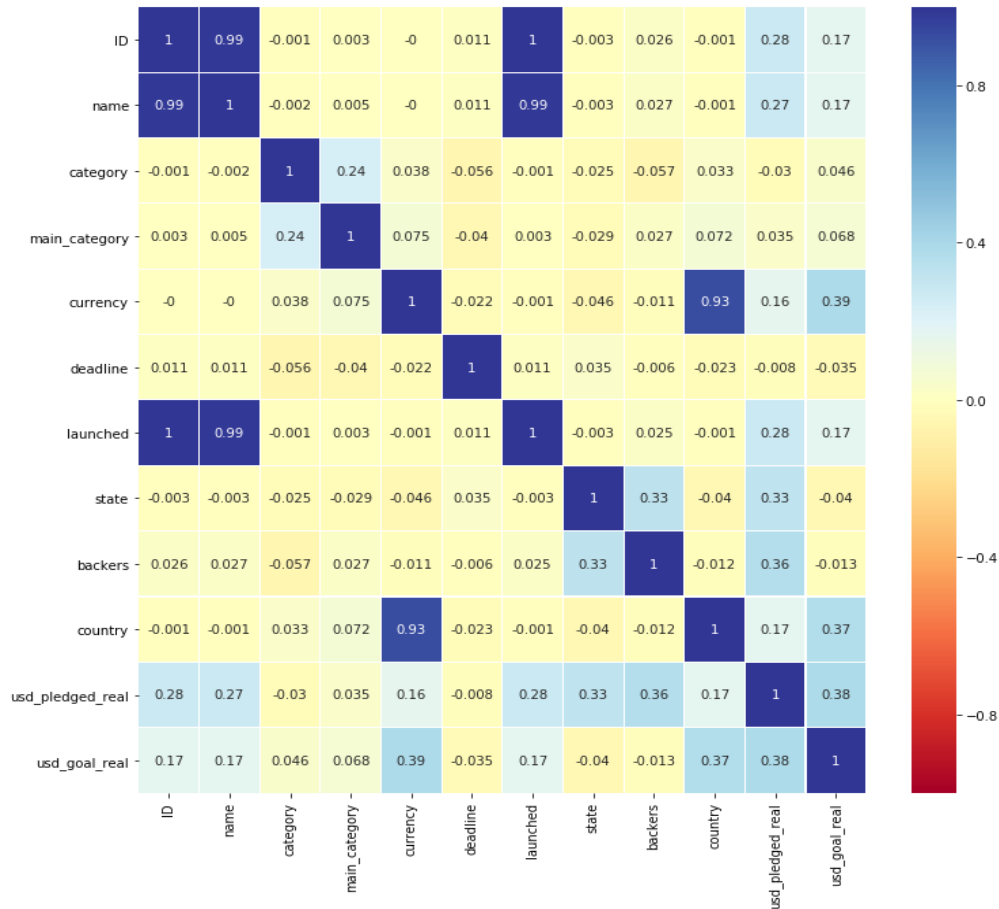
Project State	Observation Count
<b>failed</b>	197,719
<b>successful</b>	133,956
<b>canceled</b>	38,779
<b>undefined</b>	<del>3,562</del>
<b>live</b>	<del>2,799</del>
<b>suspended</b>	1,846

From the information presented in this histogram and corresponding summary table, one can easily determine that most projects fail, with projects succeeding being the second most frequent project state. Aside from projects succeeding, applicable statuses include "canceled", "undefined", "live", and "suspended". The group decided to keep all statuses aside from "undefined" and "live", due to the ambiguity associated with the undefined project state, and the in-flight status of a live project.

Removing the "undefined" and "live" status reduced the total dataset record count to 375,000. Aside from removing observations where the project state was equal to undefined, the group also removed features *goal*, *pledged*, and *usd pledged*. The dataset contains Kickstarter campaigns from a number of different countries, with the stated goal and pledged values in the native currency. The creator of the dataset converted the goal and pledged amounts to United States Dollars (usd). This conversion is represented in the dataset as *usd\_pledged\_real* and *usd\_goal\_real*. Removal of the aforementioned columns reduces the number of meaningful numeric variables to three – *usd\_pledged\_real*, *usd\_goal\_real*, and *backers*. Summary statistics for these variables are shown in the following table.

Statistic	backers	usd_goal_real	usd_pledged_real
count	372300.000000	3.723000e+05	3.723000e+05
mean	106.910040	9.148405e+03	4.572162e+04
std	914.235813	9.170345e+04	1.151326e+06
min	0.000000	0.000000e+00	1.000000e-02
25%	2.000000	3.122000e+01	2.000000e+03
50%	12.000000	6.280000e+02	5.500000e+03
75%	57.000000	4.066000e+03	1.598542e+04

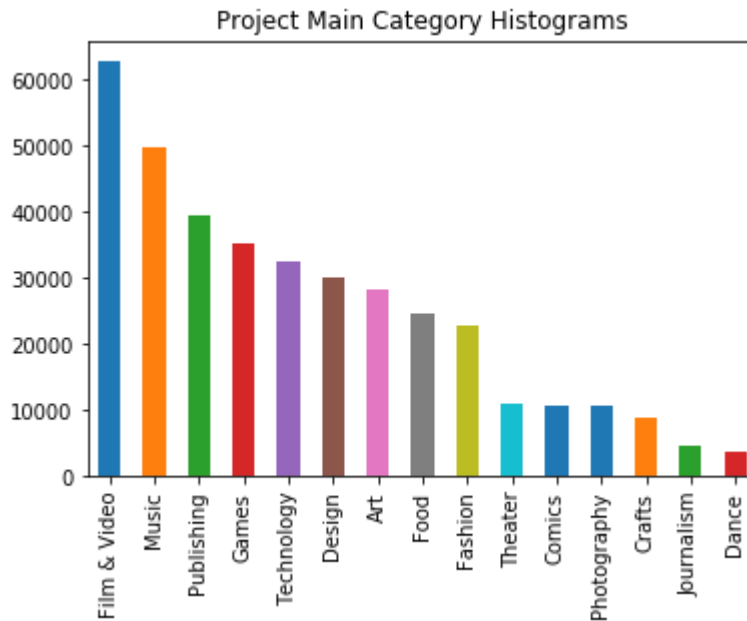
After producing high level summary statics for the numeric variables, the group produced a correlation matrix using the *pearson* method to view any weak or strong relationships within the original data. A heatmap was produced to present the relationships graphically.



The elicited results show strong relationships between *ID* and *name* (correlation = 0.99), *launched* and *name* (correlation = 0.99), as well as *country* and *currency* (correlation = 0.93). There are also marginally strong relationships between *usd\_pledged\_real* and *usd\_pledged\_goal* (correlation = 0.38), *usd\_pledged\_real* and *state* (correlation = 0.33), as well as *backers* and *state* (correlation = 0.33). Based on the operational understanding of the data, this is to be expected. The number of backers positively drive the amount pledged for a given project, and are influential in a project being successful.

Once correlations were identified, the group focused their analysis on the relationship between *main\_category* and *category*. The *main\_category* feature is a parent of the *category* feature, with 170 categories available and fifteen distinct main categories. A breakdown of observations by *main\_category* can be found in the following table, as well as a corresponding histogram.

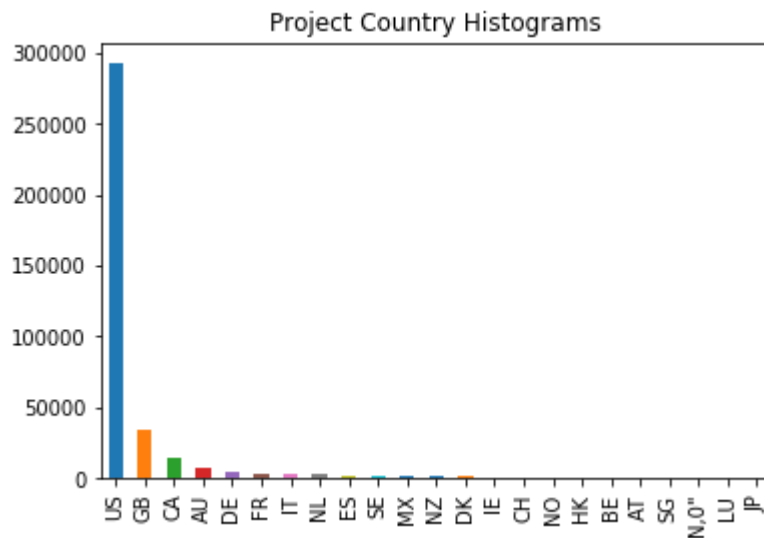
Main Category	Observation Count
Art	28,153
Comics	10,819
Crafts	8,809
Dance	3,767
Design	30,068
Fashion	22,813
Film & Video	62,731
Food	24,602
Games	35,230
Journalism	4,755
Music	49,684
Photography	10,778
Publishing	39,412
Technology	32,566
Theater	10,912



Based on this analysis, the top three categories in terms of observation count are "Film & Video", "Music", and "Publishing".

In addition to reviewing project *status* and *category*, the project group also reviewed the *country* of origin for each Kickstarter project. Please find a breakdown of projects by *country*, as well as a corresponding histogram below.

Country	Observation Count
AT	597
AU	7,839
BE	617
CA	14,756
CH	761
DE	4,171
DK	1,113
ES	2,276
FR	2,939
GB	33,672
HK	618
IE	811
IT	2,878
JP	40
LU	62
MX	1,752
N,0"	235
NL	2,868
NO	708
NZ	1,447
SE	1,757
SG	555
US	292,627



The three most frequent countries in the Kickstarter dataset are the United States (US), Great Britain (GB), and Canada (CA). Completing this analysis brought to the group's attention a misnomer in the data – 234 observations with a country value of "N,0'", which is not a valid country code. These observations were removed from the dataset.



## IV. Initial Transformations

In addition to the analysis completed on the categorical and numeric features found within the Kickstarter dataset, two transformations were completed before algorithm application. The first transformation addressed Kickstarter campaigns without names, or titles. This issue impacted four records, and the group converted the name to the value of "Unknown". The second transformation dealt with invalid data types for the dataset. The Pandas library applied the wrong data type to a few of the features. The impacted features included *deadline*, *launched*, *usd\_goal\_real*, and *usd\_pledged\_real*. The *deadline* and *launched* features were converted to dates, removing time from the observations. The *usd\_goal\_real* and *usd\_pledged\_real* features were converted to integer data types. Transformations were applied to align the working dataset with the data types presented on the project's Kaggle site.

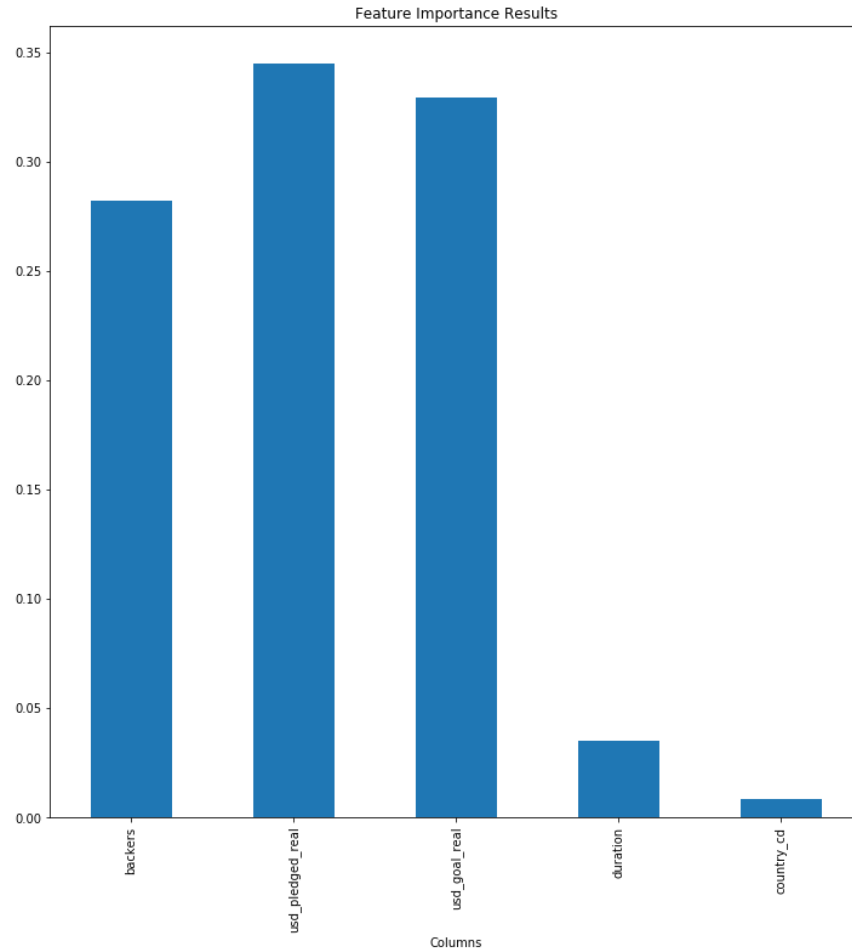
In the end, the group created a calculated column called *duration*, defined as the difference in days between the *deadline* and *launched* dates for a project. The group felt as though *duration* was a better feature to use than taking two arbitrary dates associated with the project. After completing transformations, the group reviewed which features were most important using correction matrix, recursive feature elimination and extra trees classifier.

## V. Feature Importance

The group applied logistic and linear regression independently on the remaining features to identify which features were most important, and to compare and contrast the results of both approaches. *Deadline*, *launched*, ID, and currency were removed from this analysis. It was determined that ID held very little predictive power, if at all. The currency feature was highly correlated with country and may potentially skew the results of the algorithm. The group also converted all non-numeric values to numeric using the LabelEncoder method of the sklearn.preprocessing library. The inputs of the recursive feature importance algorithms must be important. After numeric transformation, the features were split into two subsets – features and target using the training dataset. Every feature that may influence the result of the project state were assigned to the "feature" subset, and the state feature was assigned to the "target" subset. After executing the recursive feature elimination algorithm, leveraging logistic regression and linear regression, the top three features are as follows:

Recursive Feature Elimination Methodology	Top Three Features
Logistic Regression	backers, usd_pledged_real, country_cd
Linear Regression	backers, duration, country_cd

In addition to recursive feature elimination, the group also executed extra trees classifier to identify the associated feature importance of every feature in relation to the project's state. Please find a graph and table of the algorithm results on the following page.



Columns	Feature Importance
backers	0.281865
usd_pledged_real	0.344914
usd_goal_real	0.329439
duration	0.035258
country_cd	0.008524

The top three features according to extra trees classifier are *usd\_pledged\_real*, *usd\_goal\_real*, and *backers*. Of the five features submitted to the feature importance algorithms, the *backers* and *usd\_pledged\_real* features were selected as one of the top three features by each algorithm.

After filtering and transforming the data, as well as analyzing the importance of each feature to the target variable, the group applied machine learning algorithms to the dataset.

## VI. Additional Transformations

Before splitting the dataset into test and train subsets, project *state* was reduced to True and False, where projects with a state "of successful" were set to true and all other projects states were set to False. The rationale behind this transformation was to reduce the complexity of project state and reduce the outcomes from 4 values to 2. Please see the following table for reference.

Project State after Transformation	Project State	Observation Count
True	successful	133,956
False	failed	197,719
	canceled	38,779
	suspended	1,846

In addition to transforming project state, *country* were transformed using the OneHotEncoder method of the sklearn.preprocessing library into 22 binary features; one country per feature. This method is especially useful when feeding categorical data into a sklearn pipeline. The following table shows 5 rows of features after transformation.

backers	usd_pledged_real	usd_goal_real	duration	0	1	2	3	...	17	18	19	20	21
48	4021	183515	30	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0
0	0	43929	25	0.0	0.0	0.0	1.0	...	0.0	0.0	0.0	0.0	0.0
9	604	8000	31	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	1.0
15	277	600	20	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	1.0
3	232	130000	60	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	1.0

## VII. Model Execution and Analysis of Results

Once the necessary transformations were applied, the dataset is splitted to K-Nearest Neighbors (KNN) and logistic regression were applied to the dataset. The data went through many iterations of each test, with different parameters tweaked – mainly the number of neighbors in the KNN algorithm. The best performing model was a KNN algorithm with a leaf size of 30 and the number of neighbors set to five, using the minkowski distance metric. Two summary tables containing the best performing algorithms by micro and overall F1 score, respectively, can be found below.

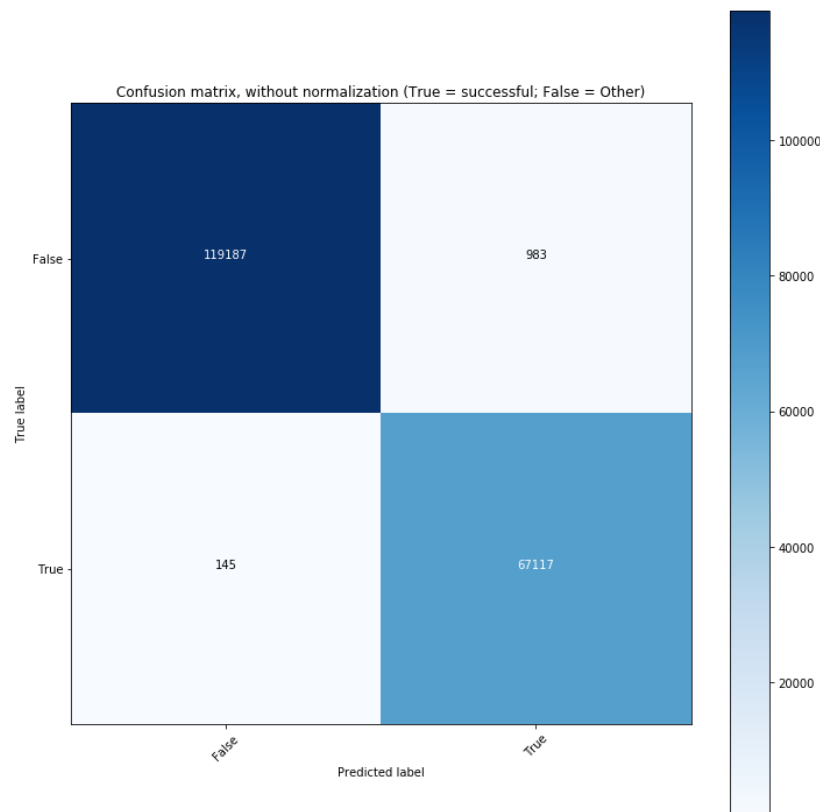
Model Parameters	false	true	micro	Sum of Class F1
<b>N=5-Basic&amp;Country</b>	1.0	1.00	1.0	3.00
<b>N=15-Basic&amp;Country</b>	1.0	0.99	1.0	2.99
<b>N=11-Basic&amp;Country</b>	1.0	1.00	1.0	3.00
<b>N=13-Basic&amp;Country</b>	1.0	0.99	1.0	2.99
<b>N=3-Basic&amp;Country</b>	1.0	1.00	1.0	3.00

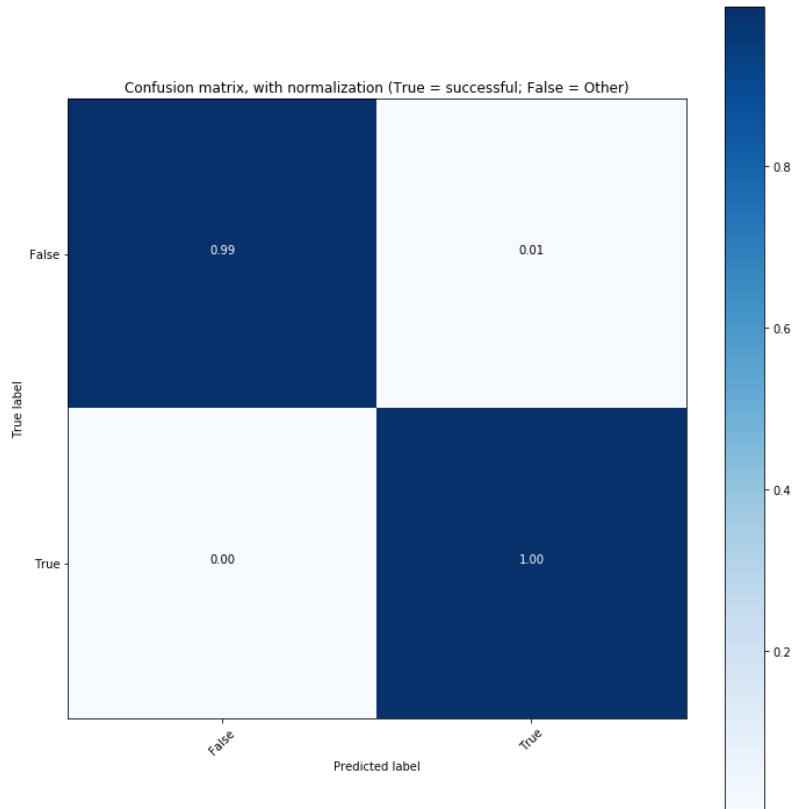
*Best performing models, sorted by micro parameter*

Model Parameters	false	true	micro	Sum of Class F1
<b>N=5-Basic&amp;Country</b>	1.0	1.0	1.0	3.0
<b>N=11-Basic&amp;Country</b>	1.0	1.0	1.0	3.0
<b>N=3-Basic&amp;Country</b>	1.0	1.0	1.0	3.0
<b>N=7-Basic&amp;Country</b>	1.0	1.0	1.0	3.0
<b>N=9-Basic&amp;Country</b>	1.0	1.0	1.0	3.0

*Best performing models, sorted by overall F1 score*

The best performing tops both lists and accurately predicted a project's success, or lack thereof, 99 percent of the time. To further underscore model performance, please find two confusion matrices that show the frequency / percentage of predicting the true label, as well as the frequency / percentage of predicting the false label, on the next page.





To underscore the performance of the model, please find a high-level classification report below.

	precision	recall	f1-score	support
<b>Failed/Cancelled/Suspended</b>	1.00	0.99	1.00	119,240
<b>successful</b>	0.99	1.00	1.00	66,793
<b>avg / total</b>	1.00	1.00	1.00	186,033

## VIII. Conclusion

In conclusion, the project group achieved the stated objective of predicting the success of a kickstart project based on the features provided. The top performing model was KNN with five neighbors, which accurately predicted if a project would succeed or fail 99 percent of the time. In addition to model execution, the project group also applied a series of algorithms to identify the most important features, derived the correlations between each feature in the dataset, and filtered or transformed anomalies in the data. Please find the requisite code used to complete the project within the following Appendix section.

## Appendix

```
# coding: utf-8

# # Kickstarter Projects
# ### CSC 478 Final Project
# #### Contributors:
# * [Rebecca Tung (1448196)](https://github.com/rtungus)
# * [Sidney Fox (1524992)](https://github.com/stfox13)
#
# ## Libraries used through the project:
get_ipython().magic(u'matplotlib inline')
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
import os
import math
import requests
import datetime as dt
import matplotlib as mpl
import io
from pandas import Series, DataFrame
from sklearn.pipeline import Pipeline
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression, LinearRegression
from sklearn import preprocessing
from sklearn.preprocessing import StandardScaler, Imputer
from sklearn.ensemble import ExtraTreesClassifier
from sklearn import preprocessing
from sklearn import svm
```

```

from sklearn.metrics import f1_score, confusion_matrix, accuracy_score, classification_report
import itertools
from sklearn.feature_selection import RFE
from collections import defaultdict

from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"

#Set graph size
mpl.rcParams['figure.figsize'] = (12,12)

np.set_printoptions(suppress=True)

# ## Load raw data as Pandas DataFrame:
url = 'https://raw.githubusercontent.com/stfox13/CSC478FinalProject/master/Data/ks-projects-201801.csv'
kickproj_org= pd.read_csv(url)
len(kickproj_org)

# ## Define Useful Functions
def roundup(x, y):
    return int(math.ceil(float(x) / float(y)) * y)

#Define a fuction to print and plot confusin matrix
def plot_confusion_matrix(cm, classes,
                           normalize=False,
                           title='Confusion matrix',
                           cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.

```

```

"""
if normalize:
    cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
else:
    pass

plt.imshow(cm, interpolation='nearest', cmap=cmap)
plt.title(title)
plt.colorbar()
tick_marks = np.arange(len(classes))
plt.xticks(tick_marks, classes, rotation=45)
plt.yticks(tick_marks, classes)

fmt = '.2f' if normalize else 'd'
thresh = cm.max() / 2.
for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
    plt.text(j, i, format(cm[i, j], fmt),
             horizontalalignment="center",
             color="white" if cm[i, j] > thresh else "black")

plt.tight_layout()
plt.ylabel('True label')
plt.xlabel('Predicted label')

#Define a fuction to calculate and print TP, TN, FP, and FN for each category
def show_statistics(test_y, y_pred, matrix):
    TP = np.diag(matrix)
    FP = np.sum(matrix, axis=0) - TP
    FN = np.sum(matrix, axis=1) - TP
    TN = []
    for i in range(len(matrix)):

```



```

temp = np.delete(matrix, i, 0) # delete ith row
temp = np.delete(temp, i, 1) # delete ith column
TN.append(sum(sum(temp)))
temp_dic = {'TP': TP, 'FP' : FP,
            'TN' : TN, 'FN' : FN}
scoreMatrix = DataFrame.from_dict(temp_dic)
return scoreMatrix

# Define a fuction to print F1 Score for each class and global (micro)
def formatResult(preResult, columnNames):
    resultDF = DataFrame(preResult.values(), columns=columnNames, index=preResult.keys())
    resultDF.loc['sum'] = np.sum(preResult.values(), axis=0)
    resultDF['Sum of Class F1'] = np.append(np.sum(preResult.values(), axis=1), np.NaN)
    return resultDF

#####KNN#####
#Define a function to run KNeighborsClassifier with different n_neighbors and store f1 score
def runKNN(trainX, trainY, testX, testY, number, f1_only = False, trainSetName = "", dic_result_knn = {}):
    i = 3
    cls = KNeighborsClassifier(n_neighbors=i)
    while i <= number:
        #print i
        cls = KNeighborsClassifier(n_neighbors=i)
        cls.fit(trainX, trainY)
        predY = cls.predict(testX)
        result = f1_score(testY, predY, average=None).round(2)
        result = np.append(result, f1_score(testY, predY, average='micro').round(2))

        #print results
        dic_result_knn['N=' + str(i) + '-' + trainSetName] = result
        #print "n_neighbors = " + str(i) + " : " + result

```

```

        i = i + 2

    return dic_result_knn

#####LogisticRegression#####
# Define a function to run LogisticRegression with different class_weight settings and store f1 score
def runLogistic(trainX, trainY, testX, testY, f1_only = False, trainSetName = "", dic_result_log = {}):
    cls = LogisticRegression()
    cls.fit(trainX, trainY)
    predY = cls.predict(testX)
    result = f1_score(testY, predY, average=None).round(2)
    result = np.append(result, f1_score(testY, predY, average='micro').round(2))
    #print results
    dic_result_log['CWeight = None - ' + trainSetName] = result

    cls = LogisticRegression(class_weight='balanced')
    cls.fit(trainX, trainY)
    predY = cls.predict(testX)
    result = f1_score(testY, predY, average=None).round(2)
    result = np.append(result, f1_score(testY, predY, average='micro').round(2))
    #print results
    dic_result_log['CWeight = balanced - ' + trainSetName] = result
    return dic_result_log

#####SVM#####
# Define a function to run SVM with different kernel settings and store f1 score
def runSVM(trainX, trainY, testX, testY, f1_only = False, trainSetName = "", dic_result_log = {}):

    C = 1.0 # SVM regularization parameter
    svc = svm.SVC(kernel='linear', C=C, decision_function_shape='ovr').fit(trainX, trainY)
    predY = svc.predict(X_plot)
    result = f1_score(testY, predY, average=None).round(2)

```

```

result = np.append(result, f1_score(testY, predY, average='micro').round(2))
print results
dic_result_log['SVCKernel = linear - ' + trainSetName] = result

svc = svm.SVC(kernel='rbf', C=C, decision_function_shape='ovr').fit(trainX, trainY)
predY = svc.predict(X_plot)
result = f1_score(testY, predY, average=None).round(2)
result = np.append(result, f1_score(testY, predY, average='micro').round(2))
print results
dic_result_log['SVCKernel = rbf - ' + trainSetName] = result

return dic_result_log

# ## Check the Y data:
#Plot histogram
kickproj_org['state'].value_counts().plot(kind='bar', title='Project State Histograms')

# ### Drop projects when the state is equal to "undefined":
# Remove state = 'undefined'
kickproj = kickproj_org[(kickproj_org['state'] != 'undefined') & (kickproj_org['state'] != 'live')]
len(kickproj)
kickproj['state'].value_counts().plot(kind='bar', title='Project State Histograms')
kickproj.head(5)

# ##### Since we have the goal and pledge amounts converted to US dollars (usd), we will drop the
original goal and pledged columns:
kickproj = kickproj.drop(['goal', 'pledged', 'usd pledged'], axis=1)
len(kickproj)
kickproj.head(5)

```

```

# ## Check the X data:
kickproj.describe()
kickproj.corr()

categoryDF = kickproj.groupby(['category']).size().reset_index(name='counts')
len(categoryDF)
categoryDF.head(5)

kickproj.groupby(['main_category']).size().reset_index(name='counts')
kickproj['main_category'].value_counts().plot(kind='bar', title='Project Main Category Histograms')

cateDF = kickproj.groupby(['main_category', 'category']).size().reset_index(name='counts')
len(cateDF)
cateDF.head(40)

kickproj.groupby(['country']).size().reset_index(name='counts')
kickproj['country'].value_counts().plot(kind='bar', title='Project Country Histograms')

# ### Remove country with invalid value, N,0"
kickproj = kickproj[kickproj['country'] != 'N,0"]
kickproj.groupby(['country']).size().reset_index(name='counts')
kickproj['country'].value_counts().plot(kind='bar', title='Project Country Histograms')

# ### Check null value
null_columns=kickproj.columns[kickproj.isnull().any()]
null_columns
kickproj[null_columns].isnull().sum()
kickproj[kickproj["name"].isnull()][null_columns]

# ### Replace nan with Unknow for name

```

```

kickproj["name"].fillna('Unknown', inplace=True)

null_columns=kickproj.columns[kickproj.isnull().any()]

null_columns

# ### Apply correct data types to DataFrame:

print 'Data types do not align with the data types defined in the data dictionary:\n\n', kickproj.dtypes

# Columns that are of date data type:

datecols = ['deadline','launched']

# Columns that are of int data type:

intcols = ['usd_pledged_real','usd_goal_real']

for col in datecols:

    kickproj[col] = pd.to_datetime(kickproj[col])

    kickproj[col] = [d.date().toordinal() for d in kickproj[col]]

kickproj[intcols] = kickproj[intcols].fillna(0).astype(np.int64)

kickproj['duration'] = abs(kickproj['deadline']-kickproj['launched'])

print 'Review converted data types:\n\n', kickproj.dtypes

# ### Find out correlation among variables

# 1. ** Feature X - backer (0.33), duration (0.11), usd_pledged_real(0.45), usd_goal_real (-0.07),
currency (-0.05) and country (-0.04) are strongly correlated with State (Target Variable) **

# 2. ** Currency and Country are highly correlated (0.94). Only one should be used in the model. We
decide to go with Country **

# 3. ** Duration is derived from deadline and launched. Duration will be used instead of deadline and
launched in the model.**

kickproj.apply(lambda x : pd.factorize(x)[0]).corr(method='pearson', min_periods=1)

print('Heat Map of Correlation Coefficients:')

sns.heatmap(kickproj.apply(lambda x : pd.factorize(x)[0]).corr(method='pearson', min_periods=1),
cmap=sns.diverging_palette(10, 220, sep=80, n=7), linewidths=0.1, annot=True, vmin=-1, vmax=1)

```

```

#### Feature Selection Method One: Recursive Feature Elimination

# ** Observation: according to this method, the most important features are:<br>backer, country,
duration and main_category<br>**

#Encode non-numeric variables - needed to run most of the models, understand anything feature
importance:

le = preprocessing.LabelEncoder

d = defaultdict(le)

le_df = kickproj.drop(['ID', 'currency', 'deadline', 'launched'], axis=1).apply(lambda x:
d[x.name].fit_transform(x))

le_df.head(10)

#Split data into two subsets - features and target - looking to predict state of the project:

features = le_df[le_df.columns.drop('state')]

target = le_df['state']

#We'll look at recursive feature elimination (RFE) with logistic regression and select three features:

LogReg_RFE = RFE(LogisticRegression(), 3).fit(features, target)

print('The three most important features according to Logistic
Regression:\n'),(np.array(features.columns)[LogReg_RFE.support_])

#We'll look at recursive feature elimination (RFE) with linear regression and select three features:

LinReg_RFE = RFE(LinearRegression(), 3).fit(features, target)

print('The three most important features according to Linear
Regression:\n'),(np.array(features.columns)[LinReg_RFE.support_])

#We'll use extra trees classifier to calculate feature importance:

ETC = ExtraTreesClassifier().fit(features, target)

```

```

feat_imp_df = pd.DataFrame({'Columns':pd.Series(features.columns)})
feat_imp_df['Feature Importance'] = pd.Series(ETC.feature_importances_)
feat_imp_df.set_index(['Columns'],inplace=True)

# ### Feature Selection Method Two: Feature Importance
# ** Observation: according to the extra trees classifier, the most important features
are:<br>usd_pledged_real, usd_goal_real, backers<br>**

print('Column Names and Associated Feature Importance:')
feat_imp_df

feat_imp_df.plot(kind="bar", title="Feature Importance Results", legend = False)

# ### Check the range of usd_pledged_real and usd_goal_real
binrange = range(1, roundup(max(kickproj['usd_pledged_real']),100000), 5000000)
binrange

min(kickproj['usd_goal_real'])
max(kickproj['usd_goal_real'])

min(kickproj['usd_pledged_real'])
max(kickproj['usd_pledged_real'])

# ### Check whether All successful records have usd_pledged_real > 0 - Outliners to be removed
# All successful records have usd_pledged_real > 0? - There is one record with excpetion and we remove
it
min(kickproj['usd_pledged_real'])
x = kickproj[(kickproj['usd_pledged_real']==0) & (kickproj['state']=='successful')].index
kickproj.drop(x, inplace=True)
kickproj[(kickproj['usd_pledged_real']==0) & (kickproj['state']=='successful')]

```

```

#### Shuffle the dataset and create training and test datasets
shffled_kickproj = kickproj.sample(frac=1)

##### Convert the value of state to True (success) or False (Other)
shffled_kickproj['state_cd'] = shffled_kickproj['state'].apply(lambda a: True if a == 'successful' else False)
shffled_kickproj.head(5)

##### Convert each country to a number
le = preprocessing.LabelEncoder()
le.fit(shffled_kickproj['country'])
shffled_kickproj['country_cd'] = le.transform(shffled_kickproj['country'])
shffled_kickproj.head(5)

num = shffled_kickproj.shape[0]/2

train_x, train_y = shffled_kickproj.iloc[0:num, [8,9,10,11,12,14]], shffled_kickproj.iloc[0:num, 13]
test_x, test_y = shffled_kickproj.iloc[num:, [8,9,10,11,12,14]], shffled_kickproj.iloc[num:, 13]
train_x.head(2)
train_x.shape
train_y.head(2)
train_y.shape
test_x.head(2)
test_x.shape
test_y.head(2)
test_y.shape

#### Train training set
##### Convert country using oneHotEncoder
temp_features_train = train_x['country_cd'].reshape(-1, 1) # Needs to be the correct shape
temp_features_test = test_x['country_cd'].reshape(-1, 1) # Needs to be the correct shape

```



```

ohe = preprocessing.OneHotEncoder(sparse=False) #Easier to read
#fit on training set only
ohe.fit(temp_features_train)

countryDF_train = DataFrame(ohe.transform(temp_features_train), columns = ohe.active_features_,
index = train_x.index)

countryDF_test = DataFrame(ohe.transform(temp_features_test), columns = ohe.active_features_, index
= test_x.index)

countryDF_train.head(10)
countryDF_test.head(10)


train_x.shape
countryDF_train.shape
test_x.shape
countryDF_test.shape


train_X1 = pd.merge(train_x.drop(['country','country_cd'], axis=1), countryDF_train, left_index=True,
right_index=True)
train_X1.head(10)
train_X1.shape


test_X1 = pd.merge(test_x.drop(['country','country_cd'], axis=1), countryDF_test, left_index=True,
right_index=True)
test_X1.head(10)
test_X1.shape


result_Dic = {}

#Call function to run KNeighborsClassifier with different n_neighbors settings (up to 20) and store the f1
score results

result_Dic = runKNN(train_X1, train_y.values.ravel(), test_X1, test_y.values.ravel(), 18, trainSetName =
'Basic&Country', dic_result_knn = result_Dic)


#Call function to run LogisticRegression with different class_weight settings (None or Balance) and store
the f1 score results

```

```

result_Dic = runLogistic(train_X1, train_y.values.ravel(), test_X1, test_y.values.ravel(), trainSetName =
'Basic&Country', dic_result_log = result_Dic)

#Store number of classes
n_classes = np.unique(shffled_kickproj['state_cd'])
n_classes

resultDF = formatResult(result_Dic, np.append(n_classes, 'micro'))
resultDF.head(10)

# Print out the top 10 Micro (overall) F1 score from all settings
resultDF.drop('sum').nlargest(5, 'micro')
resultDF.drop('sum').nlargest(5, 'Sum of Class F1')

knnClr = KNeighborsClassifier(n_neighbors=5)
knnClr.fit(train_X1, train_y.values.ravel())
final_y_pred = knnClr.predict(test_X1)

# Compute confusion matrix
cnf_matrix = confusion_matrix(test_y, final_y_pred)
np.set_printoptions(precision=2)

# Plot non-normalized confusion matrix
plt.figure()
plot_confusion_matrix(cnf_matrix, classes=[False, True],
                      normalize=False,
                      title='Confusion matrix, without normalization (True = successful; False = Other)')

stats = show_statistics(test_y, final_y_pred, cnf_matrix)
strName = map((lambda a: 'successful' if a == True else 'Failed/Cancelled/Live/Suspended'), [False,
True])
print "Classificaiton Reprt:"
print classification_report(test_y, final_y_pred, target_names=strName, digits=2)

```