



Mining Kickstarter Data: Analysis and Prediction



```
In [1]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

Loading / Preprocessing

Let's load the data and see what type of features we're dealing with.

```
In [2]: #The last few columns of the CSV from Kaggle are blank, so we'll ignore them
kick = pd.read_csv("ksprojects.csv", usecols=range(13))
```

```
In [3]: kick.dtypes
```

Out[3]:

```
In [4]: kick.head()
```

Out[4]:

There's a bunch of whitespace in a few of the column heads, so we'll need to remove that first.

```
In [5]: kick.rename(columns=lambda x: x.strip(), inplace=True)
```

Let's also convert some of the date/numeric columns to their proper data type.

There seems to be a few rows that weren't entered correctly, and we need to remove them. We can set the error parameter in the `pd.to_datetime` or `pd.to_numeric` functions to `coerce`, which will set the value as `NaN`. Then we can remove all Null values from the dataframe.

```
In [6]: #Convert the deadline and launched dates to datetime objects
kick['deadline'] = pd.to_datetime(kick['deadline'], errors='coerce')
kick['launched'] = pd.to_datetime(kick['launched'], errors='coerce')
```

```
In [7]: #Convert the goal, pledged, and backers columns to numeric
kick['goal'] = pd.to_numeric(kick['goal'], errors='coerce')
kick['pledged'] = pd.to_numeric(kick['pledged'], errors='coerce')
kick['usd pledged'] = pd.to_numeric(kick['usd pledged'], errors='coerce')
kick['backers'] = pd.to_numeric(kick['backers'], errors='coerce')
```

```
In [8]: #Check that everything worked smoothly
kick.dtypes
```

Version 1 of 1

Notebook

Mining Kickstarter Data:
Analysis And Prediction

Input (1)

Execution Info

Log

Comments (0)

```
kick.dtypes
```

```
Out[8]:
```

```
In [9]:
```

```
#Now, drop all of the rows that have NaN in them
print("Pre-drop: " + str(len(kick)))
kick.dropna(inplace=True)
print("Post-drop: " + str(len(kick)))
```

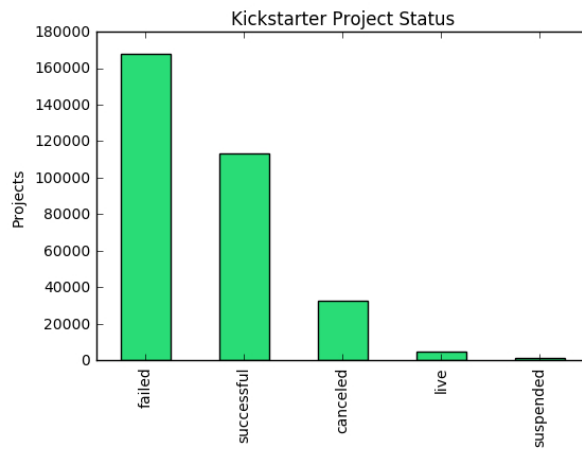
Basic Analysis and Visualization

Before running prediction, let's explore the data. We'll start with some summary visualizations.

```
In [10]:
```

```
#Distribution of project status
kick['state'].value_counts().plot(kind='bar', color='#2ADC75')
plt.title('Kickstarter Project Status')
plt.ylabel('Projects')
```

```
Out[10]:
```

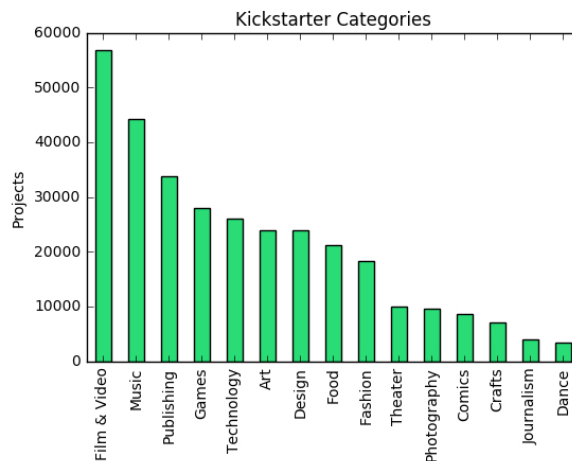


In this dataset, the plurality of projects failed, and a non-trivial amount were cancelled. Very few are live or suspended.

```
In [11]:
```

```
#Distribution of main categories
kick['main_category'].value_counts().plot(kind='bar', color='#2ADC75')
plt.title('Kickstarter Categories')
plt.ylabel('Projects')
```

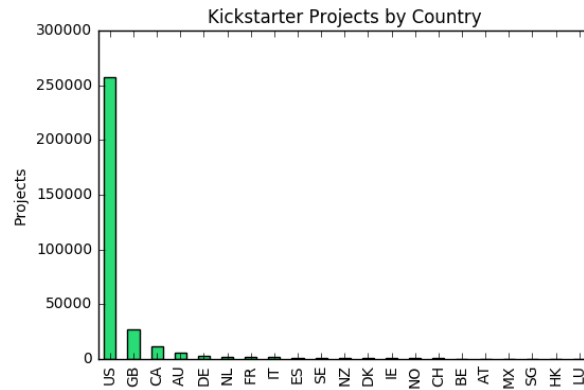
```
Out[11]:
```



The arts (Film, Video, Music) are the most popular categories. Technology is only 5th, which is surprising given how much those kinds of projects grab the news headlines.

```
In [12]: #Distribution of countries
kick['country'].value_counts().plot(kind='bar', color='#2ADC75')
plt.title('Kickstarter Projects by Country')
plt.ylabel('Projects')
```

Out[12]:



English speaking companies totally dominate.

```
In [13]: #Distribution of goals
pd.set_option('display.float_format', lambda x: '%.2f' % x)
kick['goal'].describe()
```

Out[13]:

The average goal is around 50K, but the standard deviation is more than 1M – and the median is 5K. Let's check out how these compare to the actual amount pledged.

```
In [14]: kick['pledged'].describe()
```

Out[14]:

These numbers are noticeably lower across the board. We can get a sense for how succesful each project generally is by creating a pledged/goal ratio on a per group basis. First, across state:

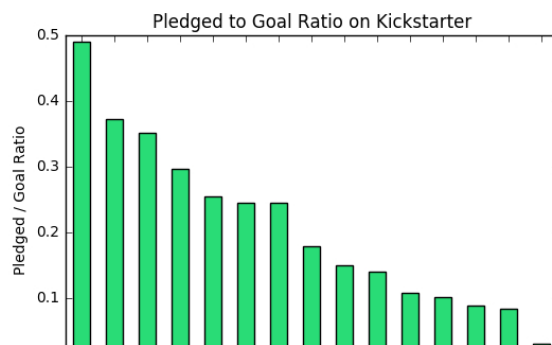
```
In [15]: stateRatio = kick.groupby('state').agg({'pledged': np.mean, 'goal': np.mean})
stateRatio['ratio'] = stateRatio['pledged'] / stateRatio['goal']
stateRatio
```

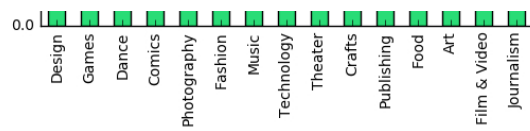
Out[15]:

Now, across categories:

```
In [16]: catRatio = kick.groupby('main_category').agg({'pledged': np.mean, 'goal': np.mean})
catRatio['ratio'] = catRatio['pledged'] / catRatio['goal']
catRatio['ratio'].sort_values(ascending=False).plot(kind='bar', color='#2ADC75')
plt.title('Pledged to Goal Ratio on Kickstarter')
plt.xlabel('')
plt.ylabel('Pledged / Goal Ratio')
```

Out[16]:

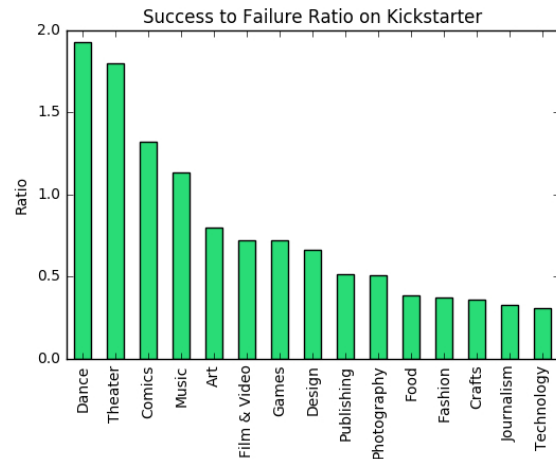




We can do the same analysis with succesful to failed projects to get a more macro feel for each category.

```
In [17]: catPivot = kick.pivot_table(index='main_category', columns='state', values='ID', ag
gfunc='count')
catPivot['WlRatio'] = catPivot['successful'] / catPivot['failed']
catPivot['WlRatio'].sort_values(ascending=False).plot(kind='bar', color='#2ADC75')
plt.title('Success to Failure Ratio on Kickstarter')
plt.xlabel('')
plt.ylabel('Ratio')
```

Out[17]:



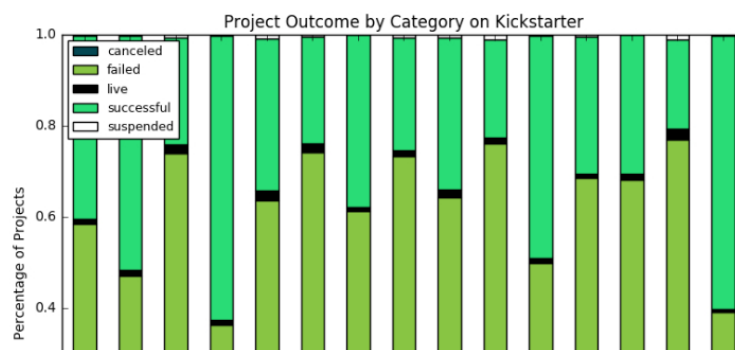
Finally, let's create a comprehensive visualization that maps project outcomes across all categories.

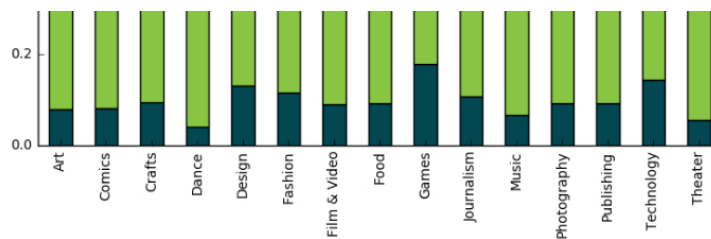
```
In [18]: #First, turn our pivot table columns into percentages instead of absolute numbers
catPivot = kick.pivot_table(index='main_category', columns='state', values='ID', ag
gfunc='count')
catPivot['total'] = catPivot.sum(axis=1)
```

```
In [19]: #Change all columns to percentages of total
for column in catPivot.columns[:5]:
    catPivot[column] = catPivot[column] / catPivot['total']
```

```
In [20]: #Plot
catPivot.iloc[:, :5].plot(kind='bar', stacked=True, figsize=(9,6),
color=['#034752', '#88C543', 'black', '#2ADC75', 'white'])
plt.title('Project Outcome by Category on Kickstarter')
plt.legend(loc=2, prop={'size': 9})
plt.xlabel('')
plt.ylabel('Percentage of Projects')
```

Out[20]:





Prediction – What Drives Successful Projects?

Imagine you're starting out on the Kickstarter platform and want to make sure that your project has the highest possible chance of success. What do you do?

There are a number of ways we can approach this as a prediction problem:

- Classification: what drives a "success" in the state column?
- Regression: what drives highest "pledged" values?
- Regression: what drives the highest number of backers?

We'll run through all of these and see what kinds of accuracy we can achieve. Which model is right for any given situation depends on what the creator is optimizing for.

Classification

Some relevant features might be:

- category
- main_category
- goal
- country

There's also some potential to mind project names for words that correlate with success. We'll try a few different algorithms and see which gets the best accuracy: Logistic Regression, Decision Trees, and Naive Bayes.

First, let's encode our categorical variables.

```
In [60]: from sklearn.preprocessing import LabelEncoder
enc = LabelEncoder()

kickML = kick.copy()
for column in ['category', 'main_category', 'country']:
    kickML[column] = enc.fit_transform(kickML[column])
```

Since we're really only interested in success, we'll narrow down the dataset to only two outcomes – success and failure.

```
In [61]: kickML = kickML[kickML['state'].apply(lambda x: x in ['successful', 'failed'])]
```

We'll also split the data into train, test, and cross validation sets (70% train, 15% each test and CV).

```
In [62]: from sklearn.utils import shuffle
kickML = shuffle(kickML)
```

```
In [202]: def dataSplit(features, target):
    trainx = kickML.iloc[:223526][features]
    trainy = kickML.iloc[:223526][target]
    testx = kickML.iloc[223527:271425][features]
    testy = kickML.iloc[223527:271425][target]
    cvx = kickML.iloc[271426:len(kickML)][features]
    cvy = kickML.iloc[271426:len(kickML)][target]
```

```
return trainx, trainy, testx, testy, cvx, cvy
```

```
In [204]: trainx, trainy, testx, testy, cvx, cvy = dataSplit(['category', 'main_category', 'goal', 'country'], 'state')
```

Logistic Regression

We'll start with Logistic Regression.

```
In [205]: from sklearn import linear_model
model = linear_model.LogisticRegression()
```

```
In [206]: model.fit(trainx, trainy)
```

Out[206]:

We'll define a function for confusion matrix analysis, since we'll use it a few times.

```
In [207]: from sklearn.metrics import confusion_matrix
```

```
In [208]: def printCM(y, y_pred):
    tn, fp, fn, tp = confusion_matrix(y, y_pred).ravel()
    print("True positives: " + str(tp))
    print("False positives: " + str(fp))
    print("True negatives: " + str(tn))
    print("False negatives: " + str(fn))
    print('\n')
    print("Overall accuracy: " + str((tp+tn)/float((tp+tn+fp+fn))))
    print("Precision (tp/tp+fp): " + str(tp/float((tp+fp))))
    print("Recall (tp/tp+fn): " + str(tp/float((tp+fn))))
```

```
In [209]: printCM(trainy, model.predict(trainx))
```

Our model is predicting exclusively "failed." This is strange, and will only leave us with accuracy along the base rate:

```
In [210]: kickML['state'].value_counts()['failed'] / float(len(kickML))
```

Out[210]:

Let's try a decision tree (or a few).

Decision Trees

```
In [72]: from sklearn import tree
clf = tree.DecisionTreeClassifier(max_depth=10)
```

```
In [73]: clf = clf.fit(trainx, trainy)
```

```
In [74]: printCM(trainy, clf.predict(trainx))
```

Training accuracy is a respectable 67%, but both precision and recall are lower than that number.

```
In [75]: printCM(cvy, clf.predict(cvx))
```

Accuracy on the cross validation set is less encouraging. Let's try an ensemble model.

```
In [76]: from sklearn.ensemble import RandomForestClassifier
        clfF = RandomForestClassifier(max_depth=2, random_state=0, n_estimators=10)
```

```
In [77]: clfF.fit(trainx, trainy)
```

Out[77]:

```
In [78]: printCM(cvy, clfF.predict(cvx))
```

This classifier has a lower overall accuracy than the single tree, but a 13% point higher precision. The recall has dropped from 47% to just 4% though, as there's a very high proportion of false negatives. In fact the number of positives predicted dropped from 1849 to 160, so this model is much more conservative.

Naive Bayes

Let's try a classic, the Naive Bayes Classifier.

```
In [79]: from sklearn.naive_bayes import GaussianNB
        gnb = GaussianNB()
```

```
In [80]: printCM(trainy, gnb.fit(trainx, trainy).predict(trainx))
```

Overall accuracy is much lower than other models, but recall is extremely high with a value of 99%. The same holds true for the test set:

```
In [81]: printCM(testy, gnb.fit(testx, testy).predict(testx))
```

Regression

The feature set should be largely similar, but instead of trying to predict a binary (success/failure) we'll look at a different target variable: money pledged as a percentage of the goal. This is a better target variable than absolute money raised, since it is normalized by the goal.

We'll start by creating a column for percentage of goal achieved to use as our target variable.

```
In [84]: kickML['completion'] = kickML['usd pledged'] / kickML['goal']
```

And now, as above, we'll split our data into train, test, and cross validation.

```
In [211]: trainx, trainy, testx, testy, cvx, cvy = dataSplit(['category', 'main_category', 'goal', 'country'], 'completion')
```

Linear Regression

```
In [212]: from sklearn.linear_model import LinearRegression
        from sklearn.feature_selection import f_regression
        import sklearn.metrics
        lr = LinearRegression()
```

```
In [213]: model = lr.fit(trainx, trainy)
```

Let's define a function to analyze metrics for our regressions.

```
In [214]: def regScore(trainy, predy):
```

```
print("R^2: " + str(m.r2_score(trainy, predy)))
print("MSE: " + str(m.mean_squared_error(trainy, predy)))
print("MAE: " + str(m.mean_absolute_error(trainy, predy)))
```

```
In [215]: regScore(trainy, model.predict(trainx))
```

An R^2 of sub 1% is not very good. Let's take a look at the coefficients and their significance levels:

```
In [216]: pd.DataFrame({'coefficient': model.coef_,
                      'F-score': f_regression(trainx, trainy)[0],
                      'p-value': f_regression(trainx, trainy)[1]},
                  index = trainx.columns)
```

```
Out[216]:
```

The category and country features have p-values of .08, which is slightly above the traditional .05 threshold but still means that they have non-trivial predictive power for completion. Interestingly enough, the size of the goal has no impact on whether it will be achieved or not.

Decision Tree

```
In [217]: from sklearn.tree import DecisionTreeRegressor
          dtr = DecisionTreeRegressor(max_depth=5)
```

```
In [218]: dtr.fit(trainx, trainy)
```

```
Out[218]:
```

```
In [141]: regScore(trainy, dtr.predict(trainx))
```

This decision tree achieves a much better R^2 of 22%. We can also try a random forest:

```
In [143]: from sklearn.ensemble import RandomForestRegressor
          rfr = RandomForestRegressor()
```

```
In [144]: rfr.fit(trainx, trainy)
```

```
Out[144]:
```

```
In [146]: regScore(trainy, rfr.predict(trainx))
```

Looks like the Random Forest Regressor achieves the highest overall R^2 . Let's plot how our key metrics change as we vary the max_depth parameter in this decision tree.

```
In [197]: def graphRFR2(trainx, trainy):

          #For adjusting max_depth
          rsmx_depth = []
          for depth in np.arange(2, 112, 10):
              rfr = RandomForestRegressor(max_depth=depth)
              rsmx_depth.append(m.r2_score(trainy, rfr.fit(trainx, trainy).predict(
                  trainx)))

          #For adjusting max_leaf_nodes
          rsmx_leaf_nodes = []
          for nodes in np.arange(2, 112, 10):
              rfr = RandomForestRegressor(max_leaf_nodes=nodes)
              rsmx_leaf_nodes.append(m.r2_score(trainy, rfr.fit(trainx, trainy).predict(
                  trainx)))

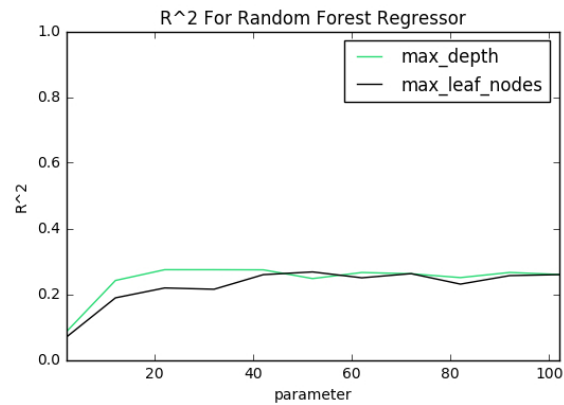
          return pd.DataFrame({'max_depth': rsmx_depth, 'max_leaf_nodes': rsmx_leaf_nod
                              es}, index=np.arange(2, 112, 10))
```

```
In [198]: plotR = graphRFR2(trainx, trainy)
```


In [201]:

```
plotR.plot(color=['#2ADC75', 'black'])
plt.title('R^2 For Random Forest Regressor')
plt.xlabel('parameter')
plt.ylabel('R^2')
plt.ylim(0,1)
```

Out[201]:



Summary / Conclusion

Overall, we weren't able to achieve any measure of great accuracy on our prediction tasks. Our classifier maxed out at around 67%, and our regressions maxed out around an R^2 of 25%.

Some ideas to improve our prediction accuracy across both classification and regression:

- Mine project names for new features
- Gather new informative data like a project description and attached images
- More complex algorithms

This Notebook has been released under the [Apache 2.0](#) open source license.

Did you find this Notebook useful?
Show your appreciation with an upvote



Input

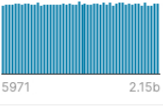
103.72 MB

Data Sources

- ▼ Kickstarter Projects
 - ks-projects-201612.csv
 - ks-projects-201801.csv

ks-projects-201612.csv (44.35 MB)



Detail	Compact	Column	10 of 16 columns		
ID	name	category	main_c		
	321616 unique values	Product Design	5%	Film & Vide	
		Documentary	5%	Music	
		Other (291382)	90%	Other (219	
100002330	The Songs of Adelaide & Abdullah	Poetry		Publishin	
100004038	Where is Hank?	Narrative Film		Film & Vi	
100007540	ToshiCapital Rekordz Needs Help to Complete Album	Music		Music	
100011046	Community Film Project: The Art of Neighborhood Filmmaking	Film & Video		Film & Vi	
100014025	Monarch Espresso Bar	Restaurants		Food	
100023410	Support Solar Roasted Coffee & Green Energy! SolarCoffee.co	Food		Food	
100030581	Chaser Strips. Our Strios make Shots	Drinks		Food	

	their B*tch!		
1000034518	SPIN - Premium Retractable In-Ear Headphones with Mic	Product Design	Design

Execution Info

Succeeded	False	Run Time	64.5 seconds
Exit Code	1	Timeout Exceeded	False
Used All Space	False	Output Size	0
Environment	Container Image (Dockerfile)	Accelerator	None
Failure Message	The kernel returned an unsuccessful exit code (1).		

Log

[Download Log](#)

```

Time Line # Log Message
1 1 {
2 2 "data": "[NbConvertApp] Converting notebook __temp_notebook_source__.ipynb to
html\n",
3 3 "stream_name": "stderr",
4 4 "time": 3.668371606967412
5 5 }, {
6 6 "data": "/opt/conda/lib/python3.6/site-
packages/nbconvert/filters/datatypefilter.py:41: UserWarning: Your element with
minetype(s) dict_keys([]) is not able to be represented.\n
minetypes=output.keys()[0])\n[NbConvertApp] Support files will be in
__results__files/\n[NbConvertApp] Making directory __results__files\n",
7 7 "stream_name": "stderr",
8 8 "time": 4.184072192991152
9 9 }, {
10 10 "data": "[NbConvertApp] Making directory __results__files\n[NbConvertApp] Making
directory __results__files\n[NbConvertApp] Making directory __results__files\n[NbConvertApp] Making directory
__results__files\n[NbConvertApp] Making directory __results__files\n[NbConvertApp] Making directory
__results__files\n[NbConvertApp] Writing 333615 bytes to __results__html\n",
11 11 "stream_name": "stderr",
12 12 "time": 4.218606963986531
13 13 }
14 14
15 15 Failed. Exited with code 1.

```

Comments (0)

Sort by 

Comment here. Be patient, be friendly, and focus on ideas. We're all here to learn and improve!



Post Comment