Kickstarter Project - Final Report

Problem Statement

When Creators are seeking for ways to bring their project to life, they have several means of achieving this. Some projects are easy to bring to life based on stuff that creators already have, while others require a bit of financial assistance. For ideas that need help, they can go through a financer such as a publisher, for writers, or a studio, for game developers. However, not all creators want to go that route. For those, there is the crowdfunding path where their project can come to life with the help of other people who want to give money to see their creations.

Kickstarter's platform allows crowdfunding to easily happen. A Creator can post a project proposal, describing the idea, the path to the idea, and how the community can help. The community of crowdfunders can give money to the Creator to see their project come to life.

However, the Creator will only receive funding from the community if the project is 'fully backed', i.e. the monetary goal amount set by the Creator was reached by the community. Kickstarter states that the success of projects is only 37.06%. While not all ideas will, or should, be successfully funded, there are ways for Creators to improve their proposal, or "campaign", to reach their funding goal.

A prediction model can help both Kickstarter and the Creator by providing a way to increase the success rate and help Creators optimize their campaign. Campaigns can be edited to follow patterns of success of previous successful campaigns. They can also be optimized to release on certain days to reach a wide amount of audience members.

Kickstarter Data

Kickstarter, unfortunately, does not provide a public API and the community has limited access to their data. There are organizations, such as Web Robots, who, as a side project, scrape sites and provide the data they gathered to the public. I will be using the Kickstarter dataset provided by Web Robots to analyze successful Kickstarter campaigns. Web Robots provides their files in JSON or CSV with updates every month.

The initial kickstarter dataset is provided by webrobots.io from their free web scraping data projects. The dataset is pulled from the 2019-05-16 set and contains 210k rows with 37 columns. The data is split into 56 csv files.

Initial Cleaning

For the most part, the data set is generally clean. However there still needs to be some cleaning done. We'll start with combining the csv files into one single file. I'll be using the glob

module to find all the file names for reading into a dataframe using pandas. From there, I'll be using pd.concat to attach all the DataFrames together into one large data frame.

The first data point to clean would be the timestamps and the format that they're stored in. According to the webrobots.io website, the time format is in unix, or epoch time. Using the time module, we're able to easily convert the epoch timestamps stored on the 'created_at', 'deadline', 'launched_at', and 'state_changed_at' columns. We store the value into two separate columns, "x_date" and "x_time" where x is one of the original columns. By storing time and date into separate columns, we'll be able to do easier analysis later on when we're exploring the data.

Our next points to clean is the columns stored in JSON formats since the web scrape project pulled the information in JSON. To do this, we simply use the json module to read the string in as a dict and pull information that way. Since there are several columns that store format in this manner, we'll be using a dict to store column:category information. There are also some columns that contain multiple categories that we want to pull so this way will also address that. We'll then use a for loop over key-value pairs to pull information in an easy manner.

Doing a bit of simple analysis on the data set, there are several columns that contain a large number of nulls compared to our 210k rows. From our exploration, the columns 'friends', 'is_backing', 'is_starred', and 'permissions' look to be related to a certain account. It provides information that is not relevant to the overall analysis.

Additionally, there are other columns that, while informational, provide little value to an analysis. For example, there is the currency_symbol which shows which symbol the currency uses. This would be helpful but we already have a currency column which provides more detail than the symbol. For the "$", this would relate to USD, CAD, and AUS. The symbol is not able to tell us this information. Other columns such as "photo" also fall under the unusable list and are therefore dropped.

Finally, there is an opportunity to pull more information from Kickstarter using the URLs provided by the dataset. To prepare for this, we'll be cleaning up the URLs once more by reading the JSON into a dict and parsing through the data that way.

Web Scraping

This is easily the most frustrating portion of the wrangling process. The script to scrape information itself was simple. Using the requests library and the BeautifulSoup library, I was able to pull the description from the HTML of the Kickstarter. However, when running this script on 210,088 rows, each iteration took about 2 seconds. This totalled to an expected time of 116 hours or almost 5 days. This quickly became unreasonable.

I knew right away that there could be optimizations made to speed this process up. I broke down my script into three parts: requesting the HTML, parsing the HTML, and creating the text from the HTML.

I ended up comparing between two libraries: urllib and request. The urllib library ended up being way faster on several tests when pulling the HTML of the page. It also ended up being more consistent with what it would pull. The requests library would sometimes be unable to pull information from a page. As for the cause, I'm still unsure but I do know that it was worth the switch. This ended up making the script run way faster and averaged less than 1 second per iteration.

When parsing the html, I compared BeautifulSoup and the selectolax libraries. From my testing, the selectolax library ended up being faster by a few ms but with the benefit of BeautifulSoup being able to parse multiple tags, the minimal speed increase of selectolax was not enough to overcome BeautifulSoup. Additionally, I also ended up parsing the image and video tags to count how many images and videos are in each description. Time was not saved in this section.

For creating the description text, there was an optimization issue since each p tag was separated in our HTML parsing. To overcome this, I used list comprehension when joining text.

Overall, the time taken for the script was reduced in half.

To run this script on a personal computer is taxing. There runs the risk of the computer turning off, the laptop going into sleep mode, and a multitude of other issues. To overcome this, I searched for ways to run this on a cloud server. Google Cloud Platform became the answer for this. This will allow the script to run without worry of any computer outages or errors.

Using GCP's free trial, I spun up a Linux based compute engine to simply run my script. The overall time ran for 2.5 days, with error checking for each row to ensure that the script does not crash.

Further Cleaning

Once the script finished, there were additional steps to clean since the data was not always pulled in the nicest way. To parse through html, the parser script stored information into a comma delimited list resulting in one additional column. The cleaning process addressed this issue by moving to individual columns. Further cleaning also included removing of rows that resulted in errors during the parsing process. Since there were only 184 rows, and those rows were consistently returning errors, it made sense to remove them. Additional processes included removing NaNs and creating new columns such as 'percent goal' or 'video usage'.

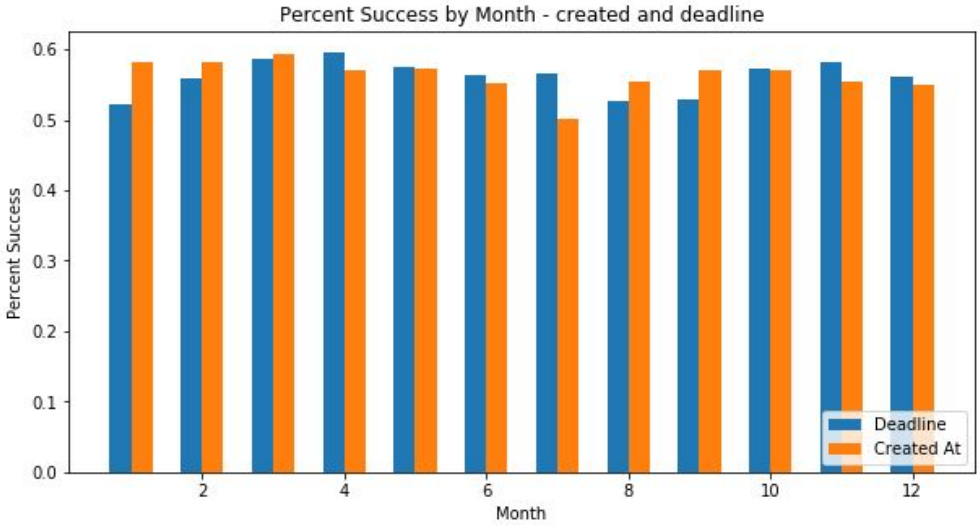
Past the cleaning, I realized that after certain procedures such as crafting the 'percent goal' or moving all monetary values to USD that other columns were not necessary for the analytics portion. This resulted in a new cleaning process where I dropped several of the tables that were not used for analytics such as the web urls or the original goal in JPY or GBP. I also made a final column to calculate the median of the rewards since that would be useful for analytics.

Exploratory Data Analysis

Diving deep into the data set, we can find several points that may lead to a solution for our problem.

Projects by Month

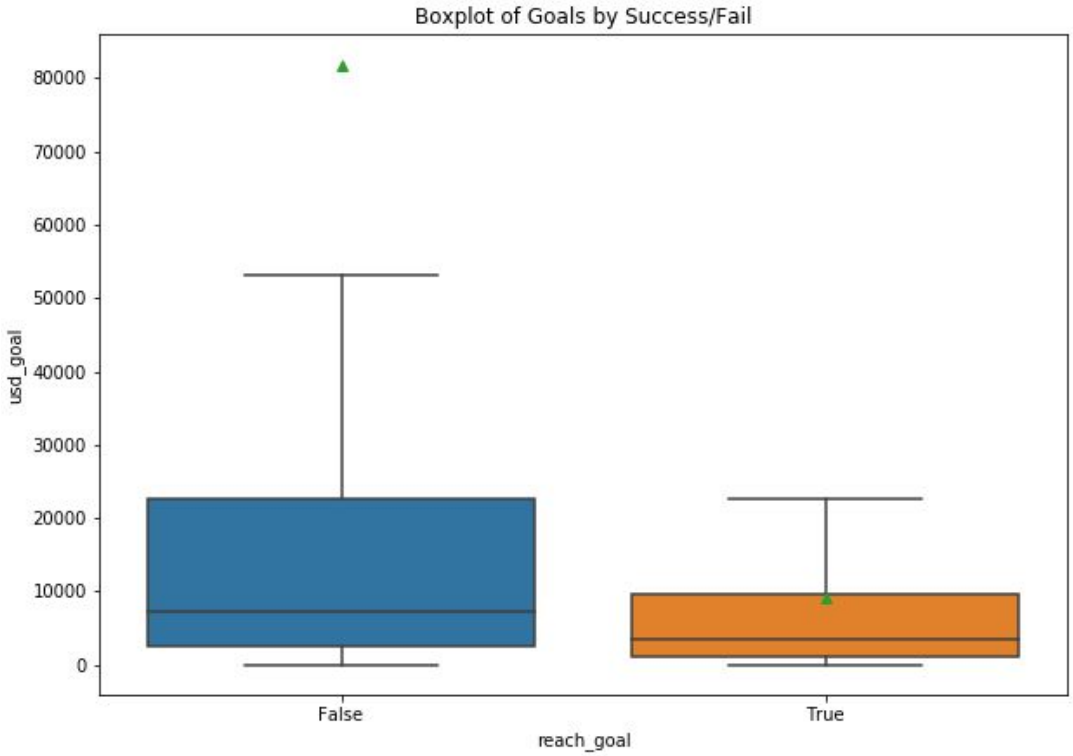
Kickstarter projects are being started no matter the time. Some months, namely January and December, have more projects stated during that time. Others have less projects starting at that time. However, frequency does not mean success.



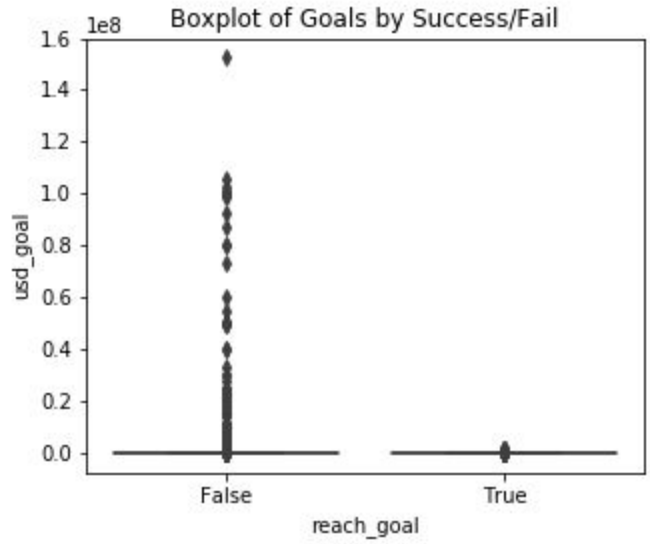
Taking a look at the success rate by month shows that projects started during March are the most successful while projects starter in July are the least. Deadline date also plays a large part in success rate where projects ending on April are the most successful.

Projects by Goal (USD)

Let's start by exploring the projects who have reached their goal and those who have not. A good comparison between them would be the Goal (USD) that each project has. Let's look at a boxplot comparing the two. For clarity, we've removed the outliers in the data.



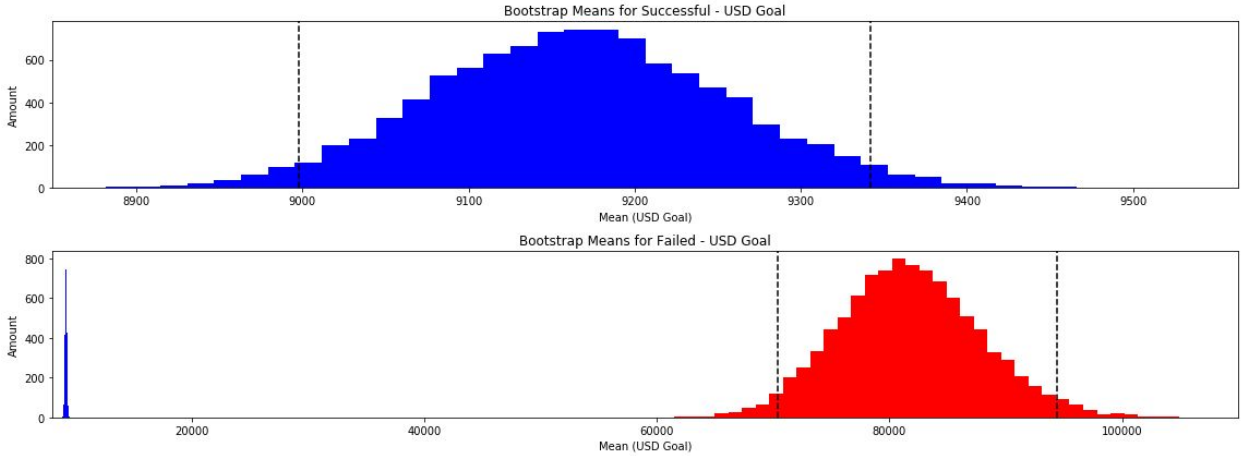
Here, we can see that the median Goal for successful projects is much less than projects that have failed. Intuitively this makes sense since smaller goals are easier to reach. Additionally, the range for projects that have succeeded is much smaller than the range for projects that have failed. If we take a quick look at the outliers, we can also see why the mean of failed projects is much higher (81,797.52) compared to successful projects (9,167.36).



It's clear that many of the projects that have incredibly high goals did not succeed, something definitely reasonable to expect. These are projects reaching 100 million.

Performing a t-test on the two groups, we get a value of -13.39 and a p-value of 6.49e-41. Since our p-value is close to 0, we can reasonably assume that there is a statistical difference between the goals of projects that fail and those that succeed.

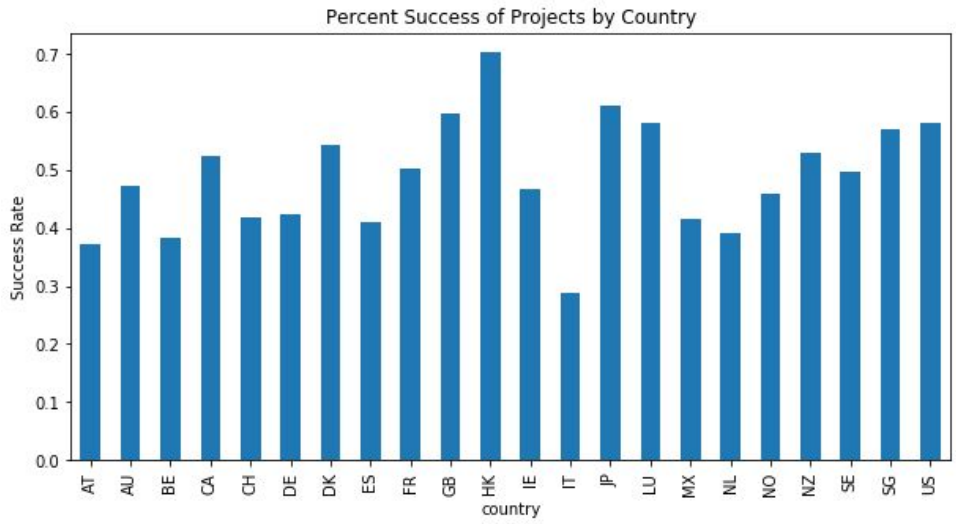
Additionally, we can see this graphically in bootstrap replicates of the mean goal between successes and failures.



The first graph shows the set of bootstrap replicates for the mean of goals (usd) for successful projects. The second one shows the failed projects compared to the successful ones. We can see a large difference in their means. If we perform a hypothesis test and assume that their means are equal and no different, how likely are we to get the observed difference of means between successful and failed projects.

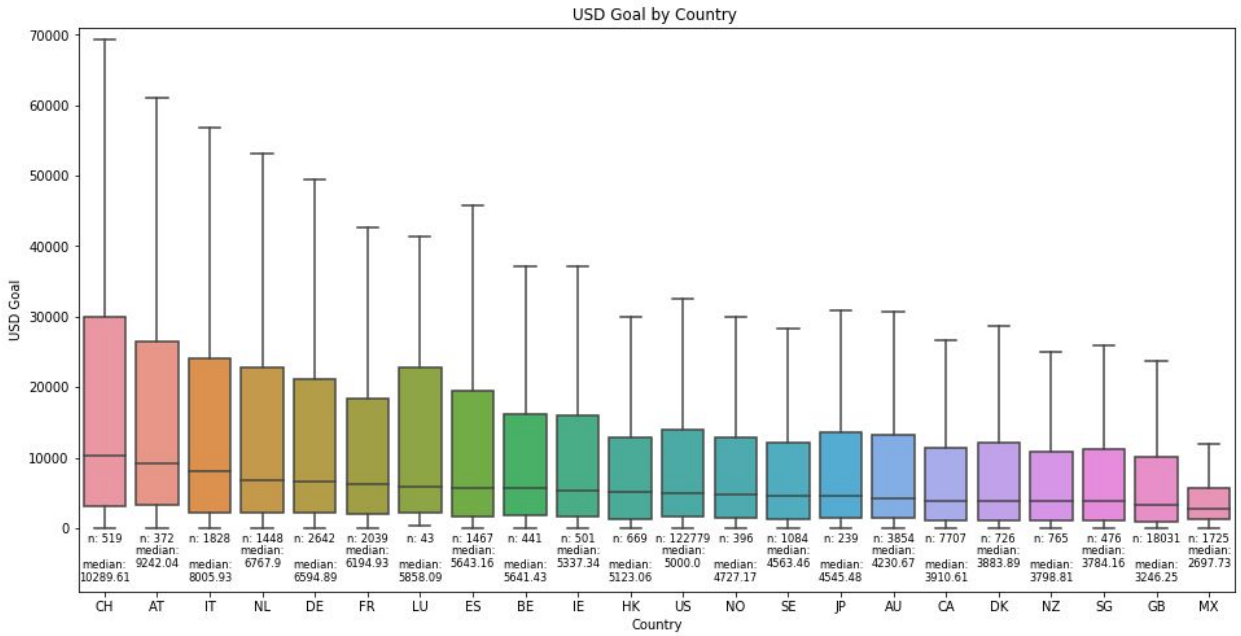
Projects by Country

Kickstarter is a global idea and so has projects being started all over the world. The main country where projects start out of is the United States where 72% of Kickstarter projects originated. However, does this necessarily mean that leads to a successful project?



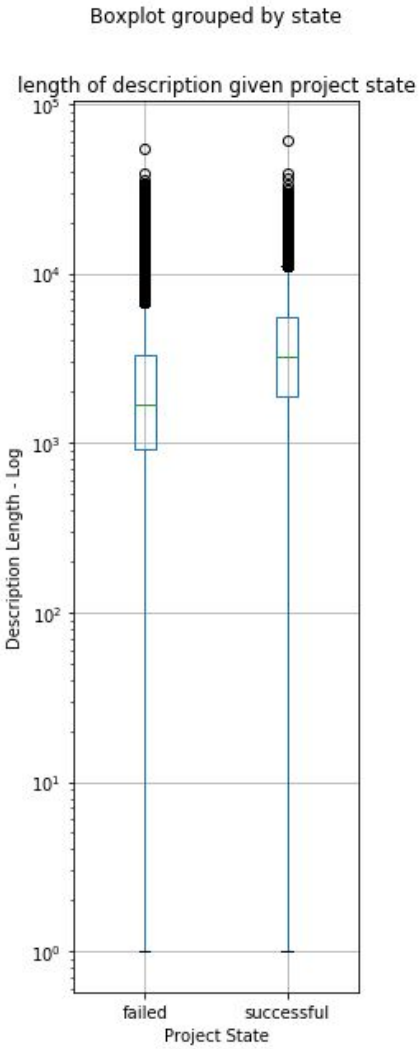
By taking a quick look at the success rate per Country, we can see that the US is actually at around a 58% success rate. The highest one is Hong Kong with a success rate of 74.7%. The country with the lowest success rate is Italy with a success rate of 32.8%.

We can also explore countries in relation to their USD Goal.



Here, we can see that China has the highest median goal followed by Austria then Italy. As the median goes down, we can also see the range decreasing.

Projects Descriptions - Length



Descriptions of projects are the best way to entice potential backers to support a Kickstarter. We can take a look at the lengths of descriptions and see if they determine success.

Here, projects that were successful have longer descriptions than projects that failed. The median description length for failed projects was 1564 characters compared to the 2782 characters for successful projects. Additionally, failed projects had a mean of 2434 while successful projects had a mean of 3773. Performing a t-test on the groups shows a p-value of 0.0. From this we can assume that there is a statistically significant difference between successful and failed projects.

Projects by Category

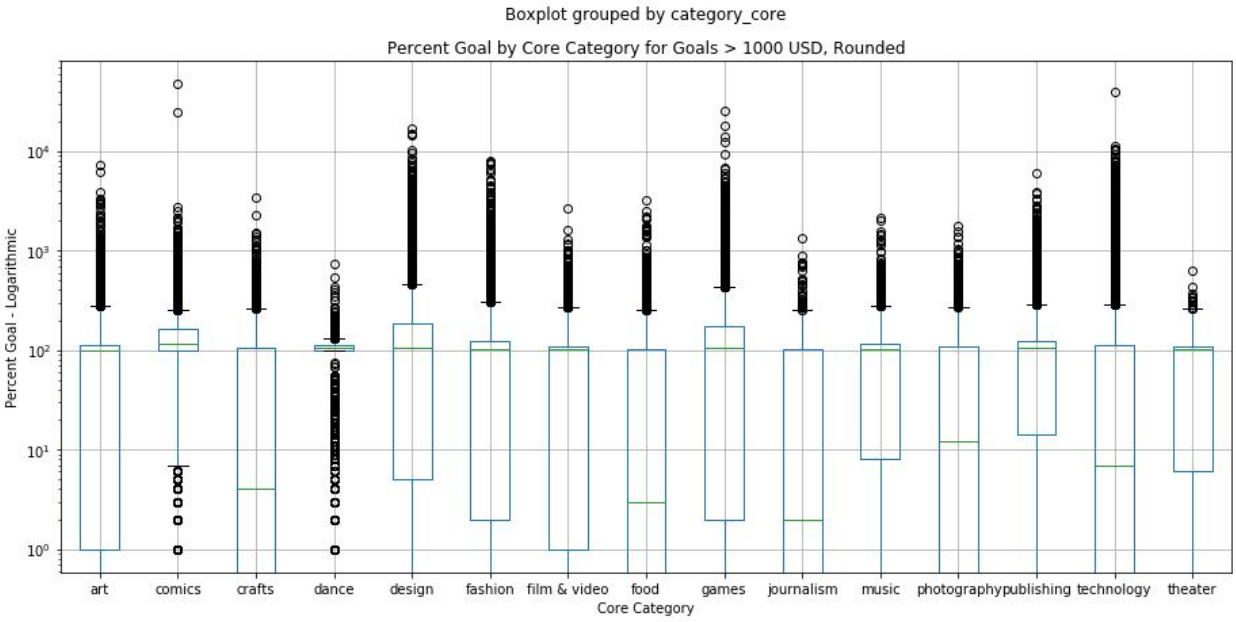
Category - Percent Goal

Each project on Kickstarter is broken down into one of 15 categories and further broken down into subcategories. These categories are selected by the creator and generally encompass

what the project entails. These range from “art” to “food” to “technology”. Some may overlap such as “film & video” “dance” project. It’s up to the creators to distinguish where their project belongs.

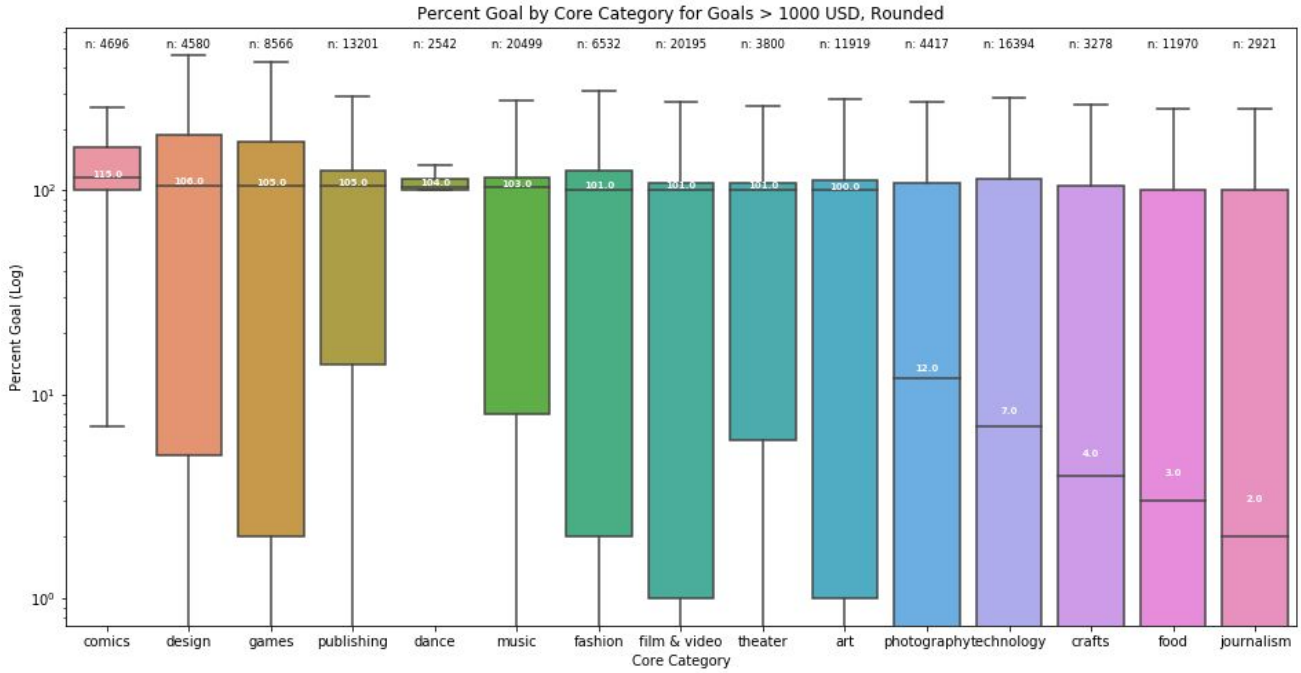
Are certain categories more successful? Does the final product drive backers to the project? To start, we can look at the percent success of projects based on their category.

For most categories, it looks like their average percent to goal is 100, meaning they’ve



hit their goal. For some, namely the crafts, food, journalism, photography, and technology categories, it looks like they’re more affected by other factors and bring their success rate down. Another interesting point is the dance category looks to center around 100% success rate. The highest ‘max’ between the categories look to be between design and games.

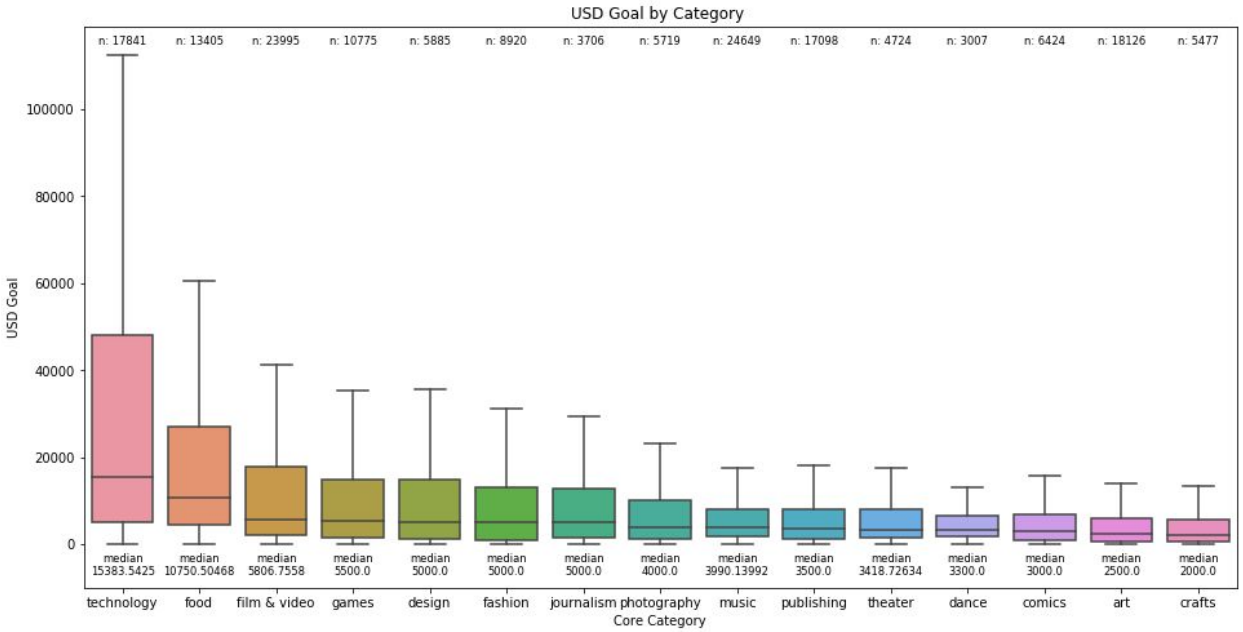
Let’s dive deeper into the categories.



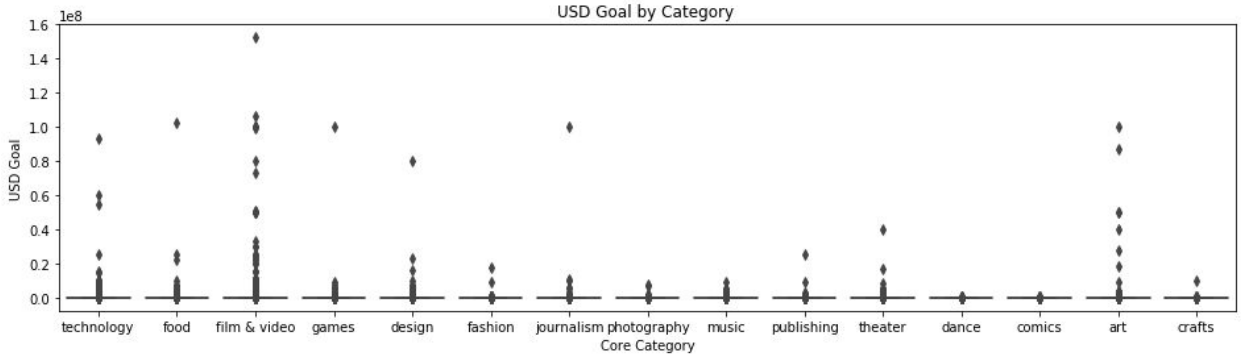
From this we can see that comics has the highest median success rate, followed by design and games. It's also interesting that these are some of the categories with the lower counts having smaller ranges compared to categories with higher counts with very large ranges.

Category - USD Goal

We'd also like to see how the categories reflect their USD Goals relative to other categories. We can do that with a boxplot.

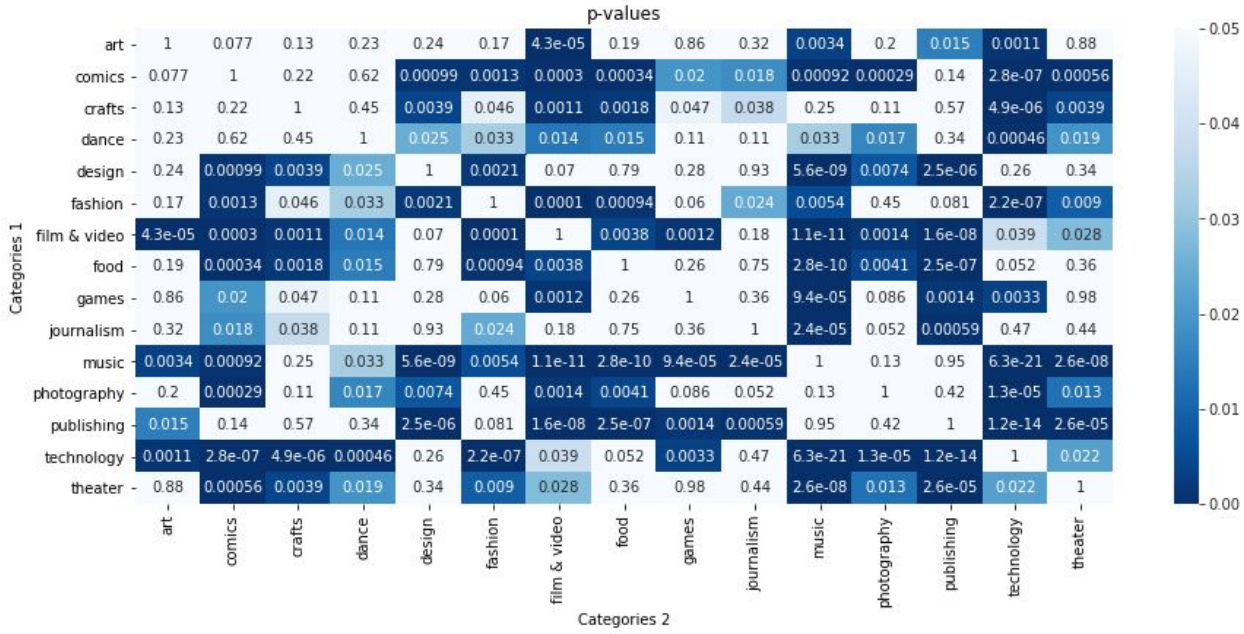


We can see that technology has the widest range and also the highest median goal. Other categories tend to be in the same range for their goals. We can also see that the size of each category fluctuates widely ranging from 24,000 to 3,000. A deep look into the categories would be to also look at their outliers and see how they affect each category.



It looks like Film & Video have the widest range of outliers. Technology also has a spread of outliers in the higher ranges. For a few categories such as food, games, and design, there look to be a cluster of data points with lower goals but have a few high reaching goals that may be affecting the mean. Art, similar to Technology, looks to also have a wide spread of data points in the higher goal range.

We'd also want to compare categories between each other to see whether or not the difference between them is significant in predicting success.



Using a 5% threshold, we can reject the hypothesis that the categories have similar average usd goals. This shows that simply comparing successful and failed projects as a whole is not enough to tell a complete story and that we would need to pay attention to the category of the project. If we performed a one way ANOVA test on our USD Goals with a null hypothesis that mean usd goals between categories are the same, we'd 2.185e-26, well below an alpha of

0.05. Since we're rejecting the null hypothesis, we can further explore to determine which categories differ the most by performing a Tukey test.

group1	group2	meandiff	lower	upper	reject
art	film & video	75098.7355	38120.565	112076.9061	True
comics	film & video	101276.2274	48490.4682	154061.9866	True
comics	technology	64089.8212	9415.2691	118764.3734	True
crafts	film & video	99138.275	42867.8098	155408.7403	True
crafts	technology	61951.8688	3905.8911	119997.8466	True
dance	film & video	101035.8861	28345.2743	173726.498	True
fashion	film & video	92658.3509	46060.9628	139255.7391	True
fashion	technology	55471.9448	6745.2902	104198.5993	True
film & video	food	-58887.1959	-99405.3324	-18369.0593	True
film & video	games	-72766.196	-116341.5353	-29190.8566	True
film & video	music	-97359.664	-131436.7193	-63282.6087	True
film & video	photography	-95079.9702	-150372.6429	-39787.2974	True
film & video	publishing	-97447.5457	-135053.6495	-59841.442	True
film & video	technology	-37186.4062	-74332.4488	-40.3636	True
film & video	theater	-72442.0367	-132252.4474	-12631.626	True
music	technology	60173.2578	23237.9634	97108.5523	True
photography	technology	57893.564	794.9656	114992.1623	True
publishing	technology	60261.1396	20046.8471	100475.432	True

Taking the results of the Tukey test and looking at those that it rejects based on an alpha of 0.5, we can see that film & video looks to be the most different to other categories in terms of mean usd goals. This is followed by technology which make up the other comparisons in this list (group1 and group2 include either tech or film/video). Several factors could be playing into this but it's worth noting that technology and film & video are one of the highest size categories that also have one of the higher ranges of usd goals.

Projects on Word Choice

We can also take a look at the categories and their most used description terms.

From this we can see some key terms that each category uses and some terms that are shared across categories. Immediately, we can see the term “one” being shared across multiple categories, regardless of relation. Taking a look, it seems as if the descriptions use “one” to describe their product in a way it stands above others (i.e. “one of a kind”, or “one perfect brew”). There are also other instances where one is just a passing term not meant to have any meaning related to the product (i.e. “one day”, or “one last thing”).

Some categories that require creativity use the word “design” seen in the design, fashion, and art categories. Some words like “app” or “device” live solely in the Technology category and words like “recipe” and “farm” belong to the food category.

In addition to descriptions, Kickstarter projects also include blurbs as a way to entice people to click into their project.

Similar to the descriptions, “one” becomes a key term used in a lot of these blurbs. Again, it’s use varies but most use it to describe their project in the superlative (“one of the most”).

The blurbs also use the word “world” frequently. The usage generally falls into two categories, the world as a whole or the world of the product they want to enter. Things like “world premier” or “around the world” describes the first category while phrases like “word of wine”, “art world”, or “word of sugar” describe the second.

Additionally, we’d like to explore the pull that each word has towards a certain category. In other words, given a category, what words are most likely to relate to that category? We can do that by using Multinomial Naive Bayes.

ART: Sculpture | 0.93 Mural | 0.91 Installation | 0.83 Sculptures | 0.83 Painting | 0.83	COMICS: Webcomic | 0.87 Comic | 0.78 Comics | 0.72 Graphic | 0.63 Manga | 0.58	CRAFTS: Candles | 0.92 Soaps | 0.79 Pens | 0.76 Candle | 0.74 Scented | 0.74
DANCE: Choreographers | 0.77 Ballet | 0.71 Dance | 0.66 Choreographer | 0.65 Choreography | 0.63	DESIGN: Font | 0.61 Titanium | 0.60 EDC | 0.58 Poster | 0.54 Logos | 0.53	FASHION: Footwear | 0.87 Sandals | 0.79 Clothing | 0.78 Shoes | 0.78 Apparel | 0.77

FILM & VIDEO: Webseries | 0.93 Cortometraje | 0.91 Film | 0.87 Mockumentary | 0.86 Animated | 0.84	FOOD: gourmet | 0.93 bakery | 0.92 sauces | 0.92 sauce | 0.89 brewery | 0.89	GAMES: platformer | 0.90 uspcc | 0.90 rpg | 0.88 28mm | 0.88 strategy | 0.86
JOURNALISM: journalism | 0.66 news | 0.50 journalists | 0.45 reporting | 0.41 coverage | 0.39	MUSIC: ep | 0.97 album | 0.97 cd | 0.94 lp | 0.92 recording | 0.92	PHOTOGRAPHY : photobook | 0.87 nudes | 0.77 photographing | 0.73 photographic | 0.71 photography | 0.66
PUBLISHING : poems | 0.82 chapbook | 0.79 literary | 0.76 rhyming | 0.74 essays | 0.74	TECHNOLOGY : arduino | 0.97 raspberry | 0.91 wireless | 0.89 bluetooth | 0.89 pi | 0.88	THEATER : fringe | 0.74 edinburgh | 0.69 theatre | 0.68 playwrights | 0.59 teatro | 0.58

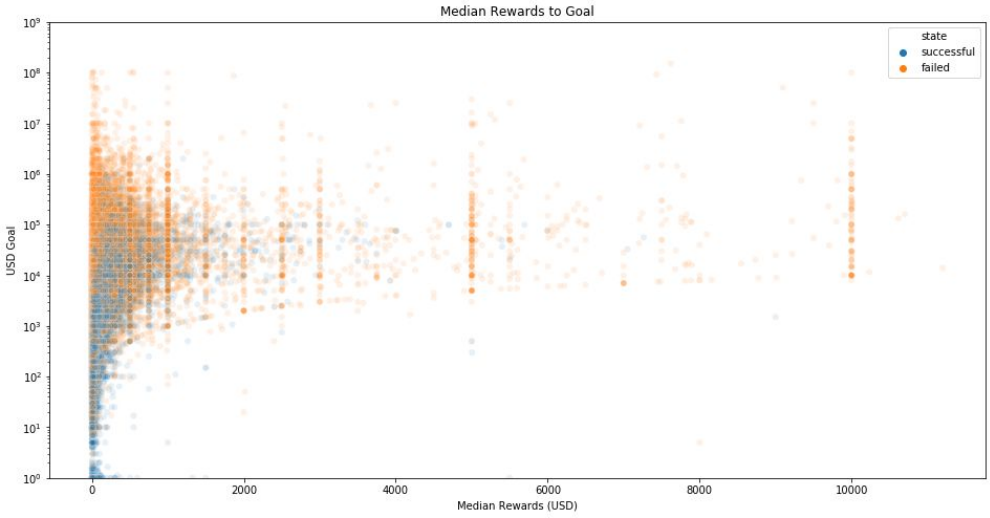
For the most part, most of the categories and their top 5 words make sense. They pretty much describe the category that tehy're in (i.e. the probability of art given that the word is 'sculpture' is 93%). It's surprising to see that not all categories have at least a few strongly predictive words (i.e. words giving probablity > 90%). The category "dance"'s most predictive word is "choreographers" at 77% and "design"s most preditive word is "font" at 61%.

Additionally, there are words that don't seem to make much sense such as "edc" predicting the "design" category and "28mm" predicting "games".

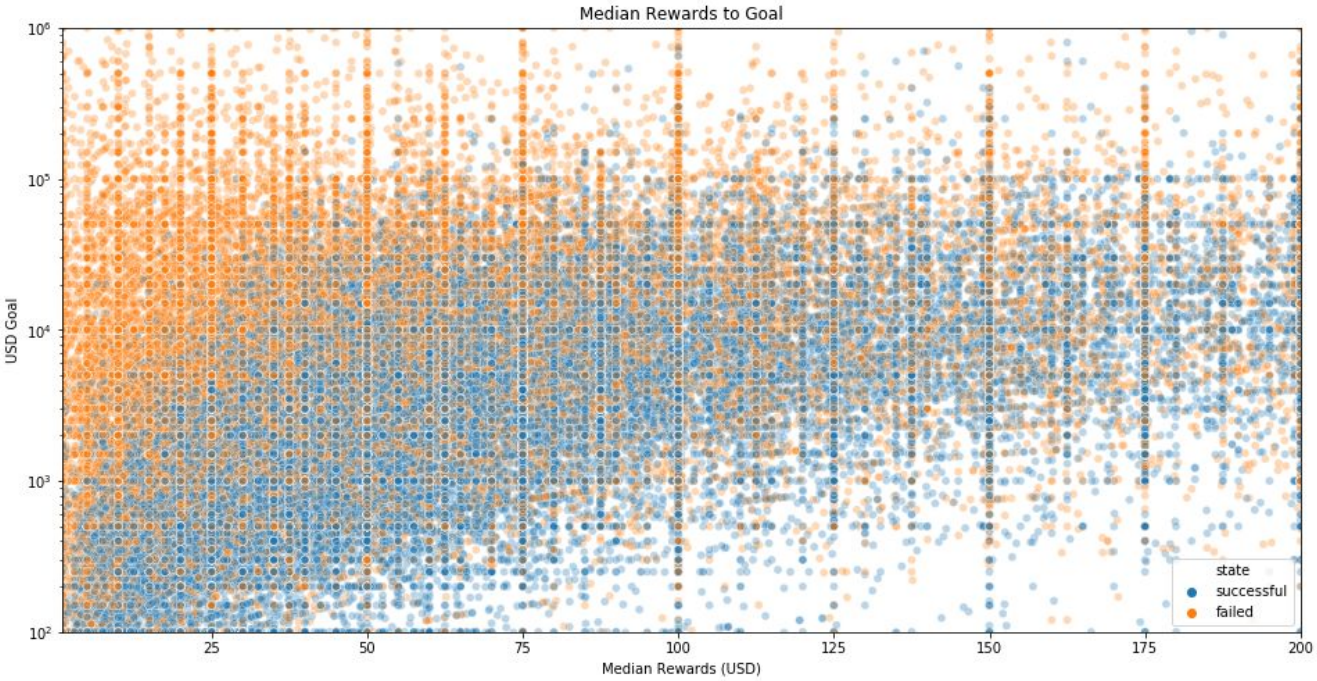
Since Kickstarters are international, there are also some words that are non-english being a big predictor of the category such as "teatro" for theater and "cortometraje"(short film) for film & video.

Projects by Rewards

Let's explore how rewards for Kickstarters can affect their progress.

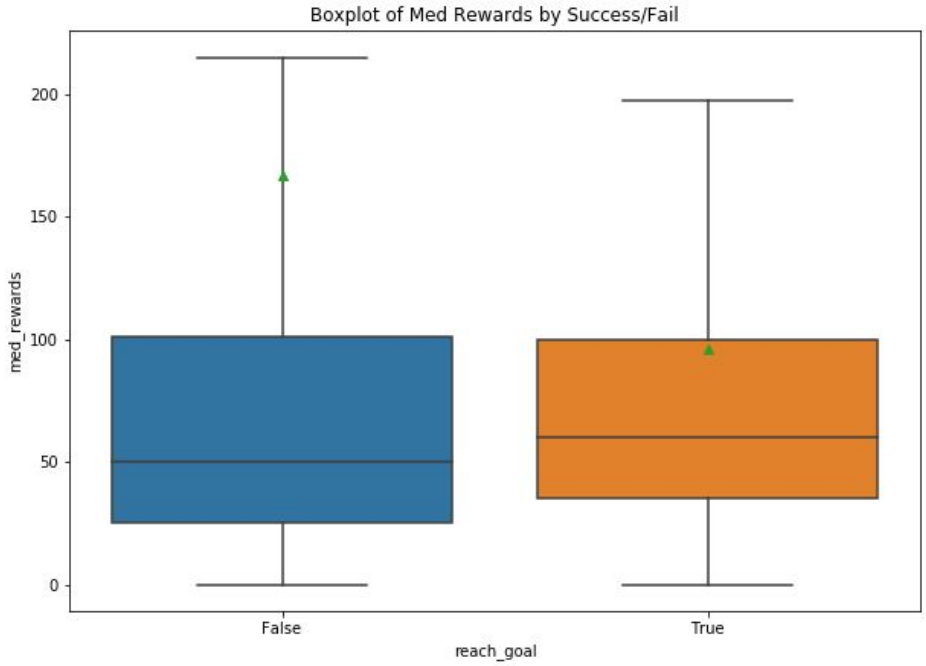


Here we've put the median reward price in relation to their goal. We can see that most Kickstarters that have a median reward of over 4000 usually fail. The median rewards over 200 actually encompass such a small amount of Kickstarters. Let's zoom in and see if we can focus on a core percent of projects.



Here, we're only looking at projects with a median reward less than 200. Right away, we can see that median rewards less than 25 and goals more than 10,000 usually fail. There's a nice curve of success vs fail as we increase in median rewards. It makes sense that if you have small reward prizes, it'll be hard to reach bigger goals. However, it's not all successful since there's still patches of failures between the successes if you're in the higher median v goal area.

We can also explore the differences between whether successful projects have a different range of median rewards compared to failed projects. Looking at their boxplots, we can see that for the most part, they look similar in their offerings.



The medians for failed and successful projects are 50 and 60, respectively. We can also see that their 25 and 75 quartiles are similar with successful projects have a slightly smaller range. What is noticeable is that their means (denoted in green triangles) are vastly different. This could be due to outliers for failed projects having median rewards higher than expected.

Performing a t-test on the groups, we get a p-value of 9.05e-269. From this, we can conclude that there exists a statistical difference between median rewards of successful projects and failed projects.

In-Depth Analysis of ML Models

The model creation process covered many different portions to create an acceptable algorithm to work with. From scoring, to feature selection, to tuning, each algorithm had their own different method. Here, we'll be exploring how the process lead to the most acceptable model and what future processes we could work with to improve the model.

To start, we'll be selecting which features we want to work with initially. These are features determine from our exploratory data analysis that would provide the most information. We'll be creating a 'features' variable which is a list of strings of the columns we want. We'll also have a 'dependent' variable which is a list of the string of our output variable. We also have a 'toscale' variable for variables we'd want to scale since some models are affected by the variance/ranges of variables.

Since a few of our features are categorical, they need to be transformed into dummy variables. We do that by using the built-in pandas get_dummies().

```
## Getting dummy variables
X = pd.get_dummies(X, drop_first = True)
y = y.values.ravel()
```

We can then split our X and y into a training and testing set to limit our chances of overfitting our data and make sure we're applying our scoring metric properly to ensure that the model can be generalized.

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size = 0.3, random_state =
42)
```

We set a random_state of 42 to enable reproducibility. We can now start testing models.

In addition to our train/test set, we'd want a train/test for our scaled variables.

```
## Implementing A Robust Scaler
robust_transformer = Pipeline(steps = [ ('robust', RobustScaler()) ] )
preprocessor = ColumnTransformer(remainder = 'passthrough',
                  transformers = [('rb', robust_transformer, toscale)])
preprocessor.fit(X)
X_scale = preprocessor.transform(X)

Xs_train, Xs_test, ys_train, ys_test = train_test_split(X_scale, y,
                                                  test_size = 0.3, random_state = 42)
```

We utilize RobustScaler since our variables are not normally distributed. We also establish a new set of train/test splits. Note that the y_train/test set and the ys_train/test set are the same since these are just the outputs of each sample.

We can willy-nilly start applying fit_predict methods to classifiers but we would not be getting any insight from them. This is where we apply a scoring method. The usual scoring method of accuracy_score would work as a base but we have a specific business case. We

want our algorithm to be used to alert users that there may be something wrong with their project and therefore we need them to know if it is a fail. Simply classifying fail/success and getting the ratio of that isn't enough. We'd want to focus on the precision and recall of our algorithm. Moreover, we'd want to focus on the Precision of our algorithm since we'd like to improve how relevant the predictions to failing are overall. To do that, we'll use the fbeta_score from scikit-learn. We'll also be implementing and using the make_scorer function to enable us to use our scoring methods in gridsearches and randomsearches.

```
from sklearn.metrics import make_scorer, fbeta_score
f_precision = make_scorer(fbeta_score, beta = .5)
```

The first model we'd like to test is the Logistic Regression (LogReg) model. Since the LogReg model is susceptible to feature scaling, we'll be utilizing the scaled train/test sets.

```
## Initial LogReg Model using scaled features
clf = LogisticRegression(solver='liblinear')
clf.fit(Xs_train, ys_train)

training = clf.predict(Xs_train)
y_pred = clf.predict(Xs_test)

train_acc = fbeta_score(ys_train, training, beta = 0.5)
test_acc = fbeta_score(ys_test, y_pred, beta = 0.5)
```

From our initial model, we receive a training score of 72.93% and a testing score of 73.06%. It seems our model is generalized and not overfitting on the training set. However, we can start tuning our hyperparameters to see if we can improve our testing scores. We can use GridSearchCV.

```
## Utilizing GridSearchCV
params = {'C':[0.0001, 0.001, 0.1, 1, 10, 100]}
clf = LogisticRegression(solver='liblinear')
gs_cv = GridSearchCV(clf, param_grid=params, cv = 5, scoring = f_precision)
gs_cv.fit(Xs_train, ys_train)
```

From the GridSearch, we get our best C as 0.001 and a score of 74.04%. We can further validate our model by utilizing cross validation on the training set and predicting on the testing set.

```
## Utilize KFold to split Train set
score = 0
folds = 10
logreg = LogisticRegression(solver='liblinear', C=0.0001)
for train, test in KFold(n_splits = folds).split(Xs_train):
    logreg.fit(Xs_train[train], ys_train[train])
    pred = logreg.predict(Xs_test)
    fb = fbeta_score(ys_test, pred, beta = 0.5)
    score += fb
```

```
Score = score / folds
```

Here, we get a score of 74.40%. We have a decently optimal solution using Logistic Regression. However, let's explore some other algorithms to see what information we can gain.

Let's explore using a support vector machine using LinearSVC.

```
## Initial SVM Model
svc = LinearSVC()
svc.fit(Xs_train, ys_train)
y_pred = svc.predict(Xs_test)
score = fbeta_score(ys_test, y_pred, beta = 0.5)
```

It takes quite a bit to fit our model since SVMs need to compare distances between several points to ensure their boundaries are good. We get an initial score of 72.85%. However, we also get a warning stating our model did not converge. Testing this on iterations up to 4000, we get slight improvements of our score to 74% range but we still do not converge. For efficiency, an SVM won't work for our case since we have a large amount of samples.

We can also explore K-Nearest Neighbors. However, right off the bat, we can tell that this won't work. KNN stores all information regarding the data points to make a prediction. Again, for our data set containing 169k points, it isn't reasonable to wait to get predictions.

We can explore an ensemble method for classification through RandomForests. Utilize the RandomForestClassifier, we do not need to use the scaled features since Decision Trees, the basis of a random forest, are not affected by a features variance/range.

```
## Base RF classifier
clf = RandomForestClassifier()
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
score = fbeta_score(y_test, y_pred, beta = 0.5)
```

We get a score of 74.97% on our base classifier. This is already an improvement over our tuned LogisticRegression classifier. Let's iterate over our RF classifier. Our first iteration is changing the number of estimators to the default in future sklearn versions. From this, we get a score of 77.38%. We get immediate improvements. We can start tuning the hyperparameters to see if we can improve this a bit more. Since Random Forests have much many more hyper parameters to tune than the LogisticRegression, we'll be utilizing RandomizedSearchCV.

```
params = {'bootstrap': [True, False],
 'max_depth': [10, 12, 14, 16, 18, 21, 23, 25, 27, 30, None],
 'max_features': ['auto', 'sqrt'],
 'min_samples_leaf': [1, 2, 4],
 'min_samples_split': [2, 5, 10],
 'n_estimators': [100, 200, 300, 400, 500]}
clf = RandomForestClassifier()
rf_random = RandomizedSearchCV(estimator=clf,param_distributions=params,
```

```
                           cv = 3,scoring=f_precision, n_jobs = -1, verbose=3)
rf_random.fit(X_train, y_train)
```

Here, we get the best parameters of

```
{'n_estimators': 500,
 'min_samples_split': 2,
 'min_samples_leaf': 2,
 'max_features': 'sqrt',
 'max_depth': 30,
 'bootstrap': True}
```

We'll implement this into a random forest and check the score

```
## RF with optimized hyperparameters
clf = RandomForestClassifier(n_estimators = 500, min_samples_split = 2,
                        min_samples_leaf = 2, max_features = 'sqrt',
                        max_depth = 30, bootstrap = True)
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
score = fbeta_score(y_test, y_pred, beta = 0.5)
```

With the best parameters, we get a score of 77.72%. To validate our score, we'll perform additional cross validation on the classifier. With this, we get a score of 77.65%. We have more confidence that our classifier will reach this range. Overall, this was highly improved over the logistic regression and will be our final model.

Next Steps

With a fbeta score of 77.6%, we can use our tuned RandomForestClassifier as our final model. This will give future Kickstarter Creators a glimpse into their project by providing our model some of the project details when setting up a Kickstarter. If the model predicts that it will fail, this allows the Creator the opportunity to make adjustments to their campaign to ensure success.

While this is a good first step, there are many more pieces of information we can improve on. We would want to start breaking down the descriptions of projects to get a better sense of keywords that attract success. Given the proper tools, we'd also be able to use other pieces of information such as background information of the Creator or prior projects started. These pieces of information could boost the algorithm to a higher prediction success rate.

We would also want to improve the model itself. While it is nice to know that a project at this point will fail, it will be more impactful to know what specifically about the project that causes it to fail.

It would be enjoyable to iterate over this model in the future as I acquire the skills to gather these pieces of information.