Milestone 2 Report:

Feature Engineering, Feature Selection, Model Selection & Performance, Take-Aways

Feature Engineering:

In order to prepare the datasets I needed to perform additional feature engineering on the text columns. I decided to leverage Pipeline, Function Transformer, and Feature Union from sklearn to streamline feature engineering and modeling.

To summarize how they're used:

- Pipelines: A pipeline "a pipeline bundles preprocessing and modeling steps so you can use the whole bundle as if it were a single step."
- Function Transformer: Takes a python processing job and turns it into an object that a sci-kit learn Pipeline can understand. Using Function Transformer I wrote three lambda functions to select the text, numeric, and categorical features in order to create separate Pipelines for each.
- Feature Union: Allows us to join the arrays generated by the pipelines as a single array that will be the input to our classifier.

In [20]:

```
# Goal of FunctionTransformer: takes a Python job and turn it into an object that the scikit-Learn Pipeline can understand
# Write 3 functions for pipeline preprocessing
# 1) Takes entire DataFrame and returns numeric columns
# 2) Takes entire DataFrame and returns text columns
# 3) Takes entire DataFrame and returns categorical columns
# Using these function transformer objects, we can build a separate Pipeline for our numeric data, text data, and categorical

#Parameter: validate = False => tells scikit-learn it doesn't need to check for NaNs or va;idate the dtypes of the input

from sklearn.base import TransformerMixin
from sklearn.pipeline import Pipeline

from sklearn.preprocessing import FunctionTransformer
from sklearn.pipeline import FeatureUnion

get_text_data = FunctionTransformer(lambda x: x['combined'], validate = False)

get_numeric_data = FunctionTransformer(lambda x: x[['goal','time_to_launch', 'launch_to_deadline', 'launched_month','launched_

# Took out creator_clean
get_categorical_data = FunctionTransformer(lambda x: x[['category_clean', 'location_clean_country','location_clean_name', 'loc
```

Once the columns were selected, I then added the necessary feature engineering steps required for each type of data (numeric, text, categorical).

For the first iteration of the model the numeric pipeline included SimpleImputer where nan's were filled with 0 in order to flag missing data. The text pipeline included CountVectorizer() to create a Bag of Words representation and the categorical pipe included an imputer with the fill

value = ‘No Value’ and OneHotEncoder (handle unknown set to ‘ignore’ in order to account for the case where features were present in the test but not the training dataset).

The pipelines were further bundled into a single master pipeline that included instantiation of a LogisticRegression() object (for example).

```
# Feature Union allows us to place the arrays generated by the pipelines together
# as a single array that will be the input to our classifier

from sklearn.pipeline import FeatureUnion
from sklearn.preprocessing import OneHotEncoder
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
import numpy as np
from sklearn.impute import SimpleImputer

#get_numeric_data created in prior block as FunctionTransformers
numeric_pipeline = Pipeline([('selector', get_numeric_data),
                             ('imputer', SimpleImputer(missing_values=np.nan, strategy='constant',fill_value =0))
                            ])

text_pipeline = Pipeline([('selector',get_text_data),
                          ('vectorizer',CountVectorizer())])

categorical_pipeline = Pipeline([('selector',get_categorical_data),
                                 ('imputer', SimpleImputer(missing_values=np.nan, strategy='constant',fill_value='No Value')),
                                ('onehot',OneHotEncoder(handle_unknown='ignore'))])
# handle_unknown: https://medium.com/hugo-ferreiras-blog/dealing-with-categorical-features-in-machine-learning-1bb70f07262d
# https://github.com/scikit-learn/scikit-learn/issues/12494

pl = Pipeline([
    ('union',FeatureUnion([
        ('numeric', numeric_pipeline),
        ('text',text_pipeline),
        ('categorical',categorical_pipeline)
    ])),
    ('logreg',LogisticRegression())
])
```

Feature Selection:

Given the short list of features and small data size I decided to use all features available and engineered in order to improve model prediction.

After creating the pipelines I then created a train-test split that allows us to fit the pipeline object and score. Using Pipeline ensures all feature engineering and modeling steps are taken care of and results in very clean code.

Model Selection & Performance:

Model choice in this case was driven by the need to handle the sparse matrix produced by the processing of the text columns. Especially given the small data sample size and even smaller

amount of text, we can have many words represented very few times resulting in a very wide table.

Using a random forest classifier and a linear svc seemed to make sense as both tend to perform well in high dimensional space and in cases where the number of dimensions is greater than the number of records.

After trying a number of combinations, it seems that while some models performed slightly better than others, for the most part the level of signal I could pull from the data set sat around 65%.

Summary of approaches used and performance:

Model Type	Bag of Words	TFIDF	N-grams	Hyperparameter Tuning
Logistic Regression	Model Variant 1: Accuracy: 65%			
	Model Variant 3: Accuracy: 66%			
Linear SVC	Model Variant 2: Accuracy: 64%			
Random Forest Classifier		Model Variant 4: Accuracy: 66%		Hyperparamter tuning: Accuracy: 68%

Four model variants were explored using various NLP techniques.

Model variant #1 utilized a logistic regression model and a Bag of Words approach to the text columns using CountVectorizer() from sklearn.

Model variant #2 utilized a linear svc model with Bag of Words.

Model variant #3 utilized a logistic regression with Bag of Words & TFIDF.

Model variant #4 utilized a random forest classifier with bi-grams and TFIDF. I also performed hyperparameter tuning on the random forest classifier, which resulted in a 2% lift.

Take-Aways:

After completing the project, a couple key take-aways emerged, especially around applying NLP to text data.

1) For text data to be useful, there needs to be a lot of it. Specifically, even a few sentences per campaign produced a very wide and sparse table which, while expected in NLP problems, resulted in the text data contributing very little in terms of lift for the base performance.
2) Spend less time on data engineering and try to find data produced in shorter intervals. The monthly scrapes combined with the average campaign only being a month long resulted in a large portion of the data needing to be discarded because of potentially biasing any models to be overly accurate due to leakage.
3) Pipelines are great but can cause downstream issues when attempting to unpack interpretability of results from the machine learning models.

However given enough data (especially around the text based features) could have yielded interesting insights into how campaign owners can craft effective messages in order to be successful in their campaigns. Ideally what I would do (time-permitting) is write a scraper to create weekly snapshots of Kickstarter campaigns and include more attributes.