

# CP - TMDb Box Office Revenue Prediction

## I. Definition

### Project Overview

The proposed project resides in the film or motion picture industry domain as I enjoy the creativity used to inspire mass audiences globally to feel distinct emotions. The Movie Database (TMDb) and Kaggle teamed up to create a competition with metadata on over 7,000 past films to try and predict the movie's expected overall worldwide box office revenue based on data before the release of the movie in the box office.<sup>1</sup>

This competition was released a month ago and meets the various needs of the Capstone project including the benchmark models published in the leaderboard on Kaggle. The data is very engaging for me as it has a slew of numeric, text, and relational data that requires cleansing and transformation. In addition, no target/dependent variable was provided therefore I will use supervised and unsupervised learning techniques to deploy and put to test.

Through Kaggle, three datasets are provided which include sample\_submission.csv, test.csv, and train.csv. The sample submission file is provided with 2 attributes to illustrate the expected file layout to be uploaded on Kaggle to evaluate predicted revenue against actuals, which are the same accounts on the test file: 1) The output/submission file would consist of the film unique ID, which is also included in the test and train sample set, as well as 2) the revenue prediction, which is only included in the train sample set hence the additional attribute to the test sample set.

There are a total of 7,398 movies which require predictions split into a test file, which consists of 4,398 sample points and 22 attributes, and train file, which consists of 3,000 sample points and 23 attributes, which includes the actual revenue. All other 22 features are available in both the train and test files. Movies are uniquely labeled with an id as well as data points on cast, crew, plot keywords, budget, posters, release dates, languages, production companies, and countries. In addition, we are allowed to collect other publicly available data to use in our model predictions, but in the spirit of this competition, we should only use data that would have been available before a movie's release.<sup>2</sup> Ideally, given my tardiness in submitting the proposal, I will only use the information provided.

---

<sup>1</sup> TMDb Box Office Prediction competition on Kaggle: <https://www.kaggle.com/c/tmdb-box-office-prediction>

<sup>2</sup> <https://www.kaggle.com/c/tmdb-box-office-prediction/data>

## Problem Statement

The problem to be solved is predicting a movie's worldwide box office revenue pre-release. In the spirit of the competition, the actual revenue figures are not provided for about 4,398 sample points however one could easily connect to TMDb's API to download actual revenue data for all movies in the train and test sample provided. This regression problem is quantifiable, measurable, and replicable given that for each movie in the test set we need to predict the value of the revenue variable.

The solution would be to accurately predict the revenue with the amount of data provided. The first step is to explore the statistical data, including any missing information, and then visualize the data to get an understanding of the distribution, correlations, and outliers. Then I will perform feature extraction and generating new numeric variables from text to digest easily, if time permitting various dimensions. For training models, I plan to utilize clustering techniques to understand the relationship between data and may create segmentation and utilizing various regression models such as XGboost, LightGM, and CatBoost since we are predicting a continuous variable.

## Metrics

In layman's term and as to acknowledge words of an Udacity mentor, "Tying your metric choice into your particular problem and problem domain is actually the single most important thing to do in any machine learning project. If you optimize a model based on the incorrect metric, your model might not be suitable for the business goals."

Converting both budget and revenue to  $\log_{10}$  scales, as the final error metric is RMSLE

$$RMSLE = \sqrt{\frac{1}{N} \sum_{i=1}^N [\log_{10}(\hat{y}_i) - \log_{10}(y_i)]^2}$$

Source: TMDB-Anand (<https://www.kaggle.com/a4anandr/tmdb-anand>)

Kaggle has defined the evaluation metric for this competition. Submissions are evaluated on Root-Mean-Squared-Logarithmic-Error (RMSLE) between the predicted value and the actual revenue. Logs are taken to not overweight blockbuster revenue movies.<sup>3</sup> As another Udacity's mentor simply and cleverly put it, "Taking logs means that errors in predicting expensive revenue and low revenue will affect the result equally."

---

<sup>3</sup> <https://www.kaggle.com/c/tmdb-box-office-prediction/overview/evaluation>

## II. Analysis

### Data Exploration

“Garbage in, garbage out” is a great saying tied directly with any data collection exercise to ensure we do not blindly trust the input data. Best practice would be for the data to be clean, clear, and compliable for model development. Let’s take the data exploration literally and slice through to the important independent contributors to revenue prediction. First, let’s analyze the three aforementioned datasets, recall they were test.csv, train.csv, and sample\_submission.csv, and spend few minutes exploring and mining.

For my project, I have decided to utilize a free open-source application found within Google Cloud called Colaboratory, an upgraded Jupyter notebook leveraging Google Docs, python 3, and a GPU/TPU accelerator, to launch Tensorflow 2.0, Keras and various python libraries to explore, visualize, and model the provided data. For the benefit of those curious minds on the technical coding, I have shared a python project however henceforth this paper will be focused on insights from the final output.

The first dataset we will explore will be the most important file, which is train.csv, as our supervised baseline and regression model could be developed. This train file will be partitioned or split 80/20 randomly for model development and validation, respectively, before real-time verification through Kaggle sample submission.

Notice below figure 1 outlines the file layout of train.csv which consists of 3,000 unique movie entries and 23 unique feature columns, which consists of 2 float, 3 integer, and 18 object data types that are not all unique so our modeling efforts will not be as straightforward as we would like. In layman’s term, our data is not clean, clear, and compilable at its status quo state and more work will be required, as usually is the case in any real-time dimensional data domain.

By comparing train to Figure 2 test.csv file layout allows us to understand and focus on areas of data exploration prior to model building. Given there is a missing revenue feature and additional data points in test.csv, the data types are same and we can ball park simple statistics such as both files have 20% of movies belonging to a collection and all object data types need to be transformed to digestible features. Also importantly, we cannot assume all features are unique in both files so digging into the various (non-) integer features is important.

**Figure 1: File layout of train.csv**

```

train = pd.read_csv('../content/train.csv')
train.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3000 entries, 0 to 2999
Data columns (total 23 columns):
id                3000 non-null int64
belongs_to_collection  604 non-null object
budget            3000 non-null int64
genres            2993 non-null object
homepage          946 non-null object
imdb_id           3000 non-null object
original_language  3000 non-null object
original_title     3000 non-null object
overview          2992 non-null object
popularity        3000 non-null float64
poster_path       2999 non-null object
production_companies  2844 non-null object
production_countries  2945 non-null object
release_date      3000 non-null object
runtime           2998 non-null float64
spoken_languages  2980 non-null object
status            3000 non-null object
tagline           2403 non-null object
title             3000 non-null object
Keywords          2724 non-null object
cast              2987 non-null object
crew              2984 non-null object
revenue           3000 non-null int64
dtypes: float64(2), int64(3), object(18)
memory usage: 539.1+ KB

```

**Figure 2: File layout of test.csv**

```

test = pd.read_csv('../content/test.csv')
test.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4398 entries, 0 to 4397
Data columns (total 22 columns):
id                4398 non-null int64
belongs_to_collection  877 non-null object
budget            4398 non-null int64
genres            4382 non-null object
homepage          1420 non-null object
imdb_id           4398 non-null object
original_language  4398 non-null object
original_title     4398 non-null object
overview          4384 non-null object
popularity        4398 non-null float64
poster_path       4397 non-null object
production_companies  4140 non-null object
production_countries  4296 non-null object
release_date      4397 non-null object
runtime           4394 non-null float64
spoken_languages  4356 non-null object
status            4396 non-null object
tagline           3535 non-null object
title             4395 non-null object
Keywords          4005 non-null object
cast              4385 non-null object
crew              4376 non-null object
dtypes: float64(2), int64(2), object(18)
memory usage: 756.0+ KB

```

**Figure 3: File layout of sample\_submission.csv**

```

submission = pd.read_csv('../content/sample_submission.csv')
submission.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4398 entries, 0 to 4397
Data columns (total 2 columns):
id                4398 non-null int64
revenue           4398 non-null int64
dtypes: int64(2)
memory usage: 68.8 KB

```

Figure 3 is the submission file provided to be filled with the revenue prediction based on the movie information found in the test file which will be used once we train our model. To better understand in layman's term the features of these three files, please refer to Appendix A, which is a cumulative data dictionary of the train, test, and sample submission dataset and provides information on various features available.

Before we dive into each features, we need to understand how much missing information is present and whether comparable between both files containing at least 22 of the similar features. Comparing figure 4 and figure 5, yields that certain fields such as

belonging to a collection, having a homepage, having a tagline, or having production companies cannot be filled in using average mean or any other statistics. In addition, certain unique features such as id and imdb ID provide little value to overall model. As a result, we will investigate some of these object features in more details and keep only those that require no configuration for our benchmark model. Although not required, I will create a second benchmark model with more configuration, more details in the Methodology section.

Figure 4 - Train file missing info	Figure 5 - Test file missing info																																																																																														
<pre>train.isna().sum()</pre> <table> <tr><td>id</td><td>0</td></tr> <tr><td>belongs_to_collection</td><td>2396</td></tr> <tr><td>budget</td><td>0</td></tr> <tr><td>genres</td><td>7</td></tr> <tr><td>homepage</td><td>2054</td></tr> <tr><td>imdb_id</td><td>0</td></tr> <tr><td>original_language</td><td>0</td></tr> <tr><td>original_title</td><td>0</td></tr> <tr><td>overview</td><td>8</td></tr> <tr><td>popularity</td><td>0</td></tr> <tr><td>poster_path</td><td>1</td></tr> <tr><td>production_companies</td><td>156</td></tr> <tr><td>production_countries</td><td>55</td></tr> <tr><td>release_date</td><td>0</td></tr> <tr><td>runtime</td><td>2</td></tr> <tr><td>spoken_languages</td><td>20</td></tr> <tr><td>status</td><td>0</td></tr> <tr><td>tagline</td><td>597</td></tr> <tr><td>title</td><td>0</td></tr> <tr><td>Keywords</td><td>276</td></tr> <tr><td>cast</td><td>13</td></tr> <tr><td>crew</td><td>16</td></tr> <tr><td>revenue</td><td>0</td></tr> <tr><td>dtype: int64</td><td></td></tr> </table>	id	0	belongs_to_collection	2396	budget	0	genres	7	homepage	2054	imdb_id	0	original_language	0	original_title	0	overview	8	popularity	0	poster_path	1	production_companies	156	production_countries	55	release_date	0	runtime	2	spoken_languages	20	status	0	tagline	597	title	0	Keywords	276	cast	13	crew	16	revenue	0	dtype: int64		<pre>test.isna().sum()</pre> <table> <tr><td>id</td><td>0</td></tr> <tr><td>belongs_to_collection</td><td>3521</td></tr> <tr><td>budget</td><td>0</td></tr> <tr><td>genres</td><td>16</td></tr> <tr><td>homepage</td><td>2978</td></tr> <tr><td>imdb_id</td><td>0</td></tr> <tr><td>original_language</td><td>0</td></tr> <tr><td>original_title</td><td>0</td></tr> <tr><td>overview</td><td>14</td></tr> <tr><td>popularity</td><td>0</td></tr> <tr><td>poster_path</td><td>1</td></tr> <tr><td>production_companies</td><td>258</td></tr> <tr><td>production_countries</td><td>102</td></tr> <tr><td>release_date</td><td>1</td></tr> <tr><td>runtime</td><td>4</td></tr> <tr><td>spoken_languages</td><td>42</td></tr> <tr><td>status</td><td>2</td></tr> <tr><td>tagline</td><td>863</td></tr> <tr><td>title</td><td>3</td></tr> <tr><td>Keywords</td><td>393</td></tr> <tr><td>cast</td><td>13</td></tr> <tr><td>crew</td><td>22</td></tr> <tr><td>dtype: int64</td><td></td></tr> </table>	id	0	belongs_to_collection	3521	budget	0	genres	16	homepage	2978	imdb_id	0	original_language	0	original_title	0	overview	14	popularity	0	poster_path	1	production_companies	258	production_countries	102	release_date	1	runtime	4	spoken_languages	42	status	2	tagline	863	title	3	Keywords	393	cast	13	crew	22	dtype: int64	
id	0																																																																																														
belongs_to_collection	2396																																																																																														
budget	0																																																																																														
genres	7																																																																																														
homepage	2054																																																																																														
imdb_id	0																																																																																														
original_language	0																																																																																														
original_title	0																																																																																														
overview	8																																																																																														
popularity	0																																																																																														
poster_path	1																																																																																														
production_companies	156																																																																																														
production_countries	55																																																																																														
release_date	0																																																																																														
runtime	2																																																																																														
spoken_languages	20																																																																																														
status	0																																																																																														
tagline	597																																																																																														
title	0																																																																																														
Keywords	276																																																																																														
cast	13																																																																																														
crew	16																																																																																														
revenue	0																																																																																														
dtype: int64																																																																																															
id	0																																																																																														
belongs_to_collection	3521																																																																																														
budget	0																																																																																														
genres	16																																																																																														
homepage	2978																																																																																														
imdb_id	0																																																																																														
original_language	0																																																																																														
original_title	0																																																																																														
overview	14																																																																																														
popularity	0																																																																																														
poster_path	1																																																																																														
production_companies	258																																																																																														
production_countries	102																																																																																														
release_date	1																																																																																														
runtime	4																																																																																														
spoken_languages	42																																																																																														
status	2																																																																																														
tagline	863																																																																																														
title	3																																																																																														
Keywords	393																																																																																														
cast	13																																																																																														
crew	22																																																																																														
dtype: int64																																																																																															

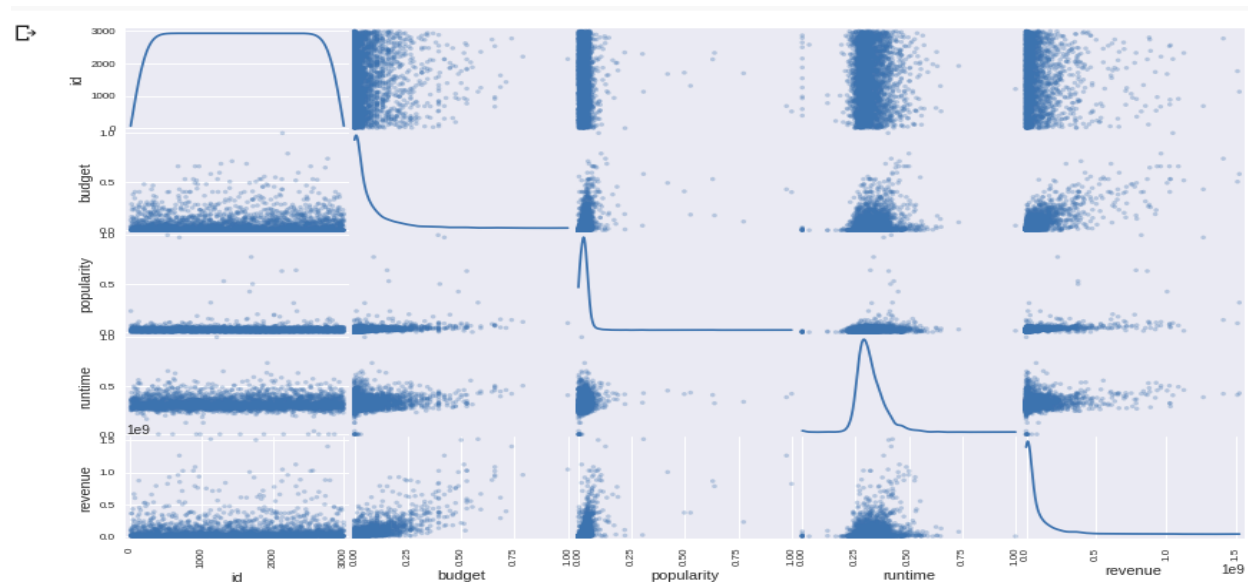
Figure 6 below is a sample from the train file that illustrates features such as original language the movie is in such as 'en' for English, 'hi' for Hindi, 'ko' for Korean etc and associated title of the movie. If we focus on tagline, for some Hindi and Korean movies, the database has missing, labelled as NaN, information. This is a great example for not filling in the missing information with any statistical measure because that would never make sense for the tagline variable. Therefore the usage of this feature could be converted into whether a tagline is available and understand the relationship to generating revenue. If we prove that having a tagline yields more revenue, we could then dig further into whether the choice of words within the tagline provides additional lift in predicting revenues.

**Figure 6: Data sample from Train file**

original_language	original_title	overview	popularity	...	release_date	runtime	spoken_languages	status	tagline
en	Hot Tub Time Machine 2	When Lou, who has become the "father of the In...	6.575393	...	2/20/15	93.0	[{"iso_639_1": "en", "name": "English"}]	Released	The Laws of Space and Time are About to be Vio...
en	The Princess Diaries 2: Royal Engagement	Mia Thermopolis is now a college graduate and ...	8.248895	...	8/6/04	113.0	[{"iso_639_1": "en", "name": "English"}]	Released	It can take a lifetime to find true love; she'...
en	Whiplash	Under the direction of a ruthless instructor, ...	64.299990	...	10/10/14	105.0	[{"iso_639_1": "en", "name": "English"}]	Released	The road to greatness can take you to the edge.
hi	Kahaani	Vidya Bagchi (Vidya Balan) arrives in Kolkata ...	3.174936	...	3/9/12	122.0	[{"iso_639_1": "en", "name": "English"}, {"iso...	Released	NaN
ko	마린보이	Marine Boy is the story of a former national s...	1.148070	...	2/5/09	118.0	[{"iso_639_1": "ko", "name": "한국어/조선말"}]	Released	NaN
en	Pinocchio and the Emperor of the Night	Pinocchio and his friends, a glow worm and a m...	0.743274	...	8/6/87	83.0	[{"iso_639_1": "en", "name": "English"}]	Released	NaN
en	The Possession	A young girl buys an antique box at a yard sal...	7.286477	...	8/30/12	92.0	[{"iso_639_1": "en", "name": "English"}]	Released	Fear The Demon That Doesn't Fear God

## Exploratory Visualization

Ideally, it is best practice to extract insights from all the features however as discussed certain features such as ID is unique to that data point and never any other data point. In addition, at this point the benchmark linear regression model only uses integer variables therefore for the benefit of this report, I will focus on few statistical insights strictly from a point of view that additional feature extraction at this point would not be beneficial.

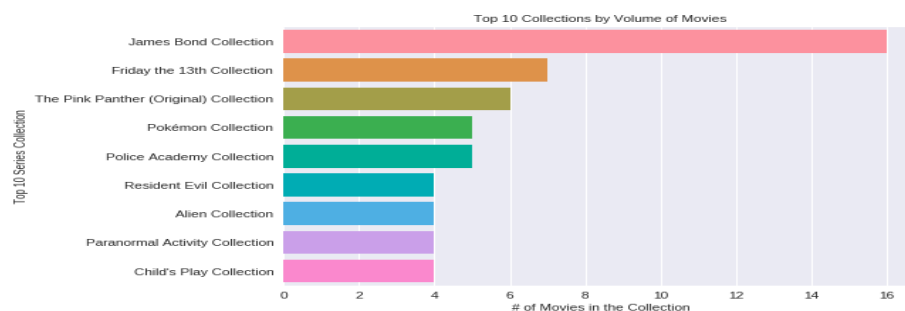
**Chart 1: Scatter matrix for each pair of features in the Train dataset**

We should not consider id as it is unique. Based on the scatter matrix 1 of the features, runtime, is distributed normally and the remaining 3 features; budget, popularity, and revenue, tend to be skewed closer to 0. The outliers seems to be the culprit in the

skewness in the data and the reported high deviation within each feature. The data for those features distributed are in favour of outliers such as big budget movies which carry out highly paid actors produced in varying fancy high budget location. Once we segment out for these sort of enterprises, the data may become more normalized and new patterns may emerge. From Chart 1 or by simply running a correlation function would suggest following pairs exhibit some degree of strong to mild positive correlation: Revenue-Budget, Revenue-Popularity, Budget-Popularity, Budget-Runtime, and Popularity-Runtime, respectively.

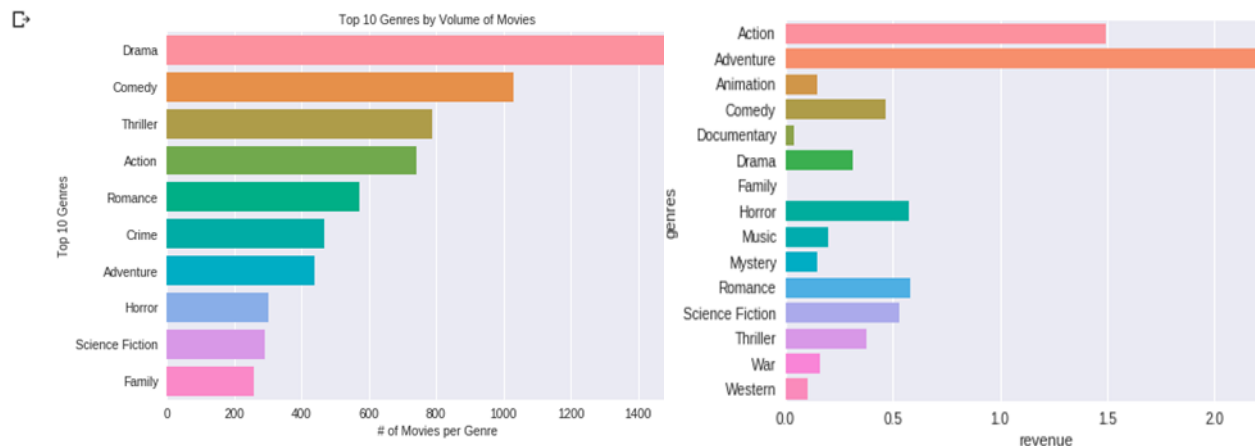
Below are few other categorical visualisations that could be beneficial to predicting revenue from an intuition perspective. For example, from Chart 2 we could gather that a movie producer would not giving producing collections of 007 James Bond movies if it was not revenue generating otherwise some of the collection will discontinue.

**Chart 2: Top 10 Collections by Volume of Movies**



Who does not love drama, comedy, thriller, action, and romance packed altogether? To any producer reading, make action and adventure mandatory as evidence supports higher chances of revenue generation and explains the 007 James Bond collection aforementioned. We will test our hypothesis later in our modeling.

**Chart 3: Top 10 Genres by Volume of Movies and by Revenue generated**



Additional visualizations are available in the supporting python document. Need to extend my many thanks to the open-source Kaggle members for sharing their various data exploration techniques and insights, which I utilized directly or modified as required for this proposal.

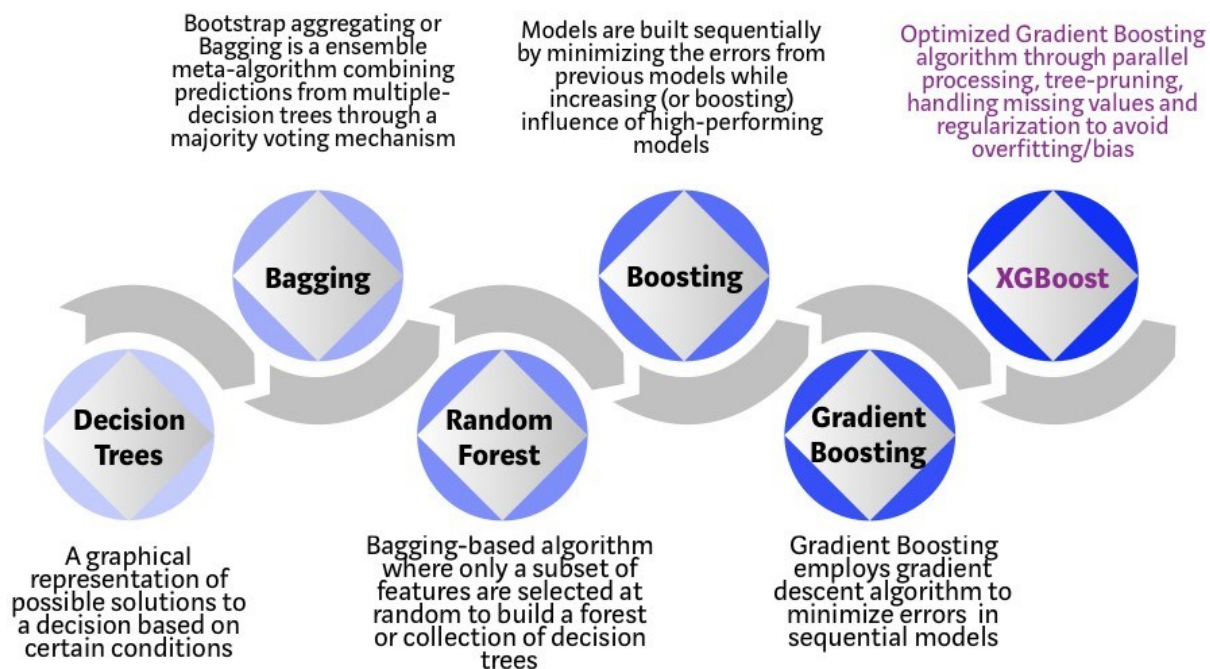


## Algorithms and Techniques

For model development, I plan to use the training file to utilize clustering techniques to understand the relationship between data and may create segmentation and utilizing regression models. Given the problem is to predict a continuous variable, revenue, and given our data has few features with outliers, non-standardized, collinearity, and missing values, we need to use regression models that would ideally consider our constraints. A handful of regression models, namely XGBoost, LightGM, CatBoost, and random forest, which we will utilize, requires minimum effort to take care of the various feature engineering intricacies required for the training data. In addition, cross-validation to the test file will help in realizing the best performing model and will help to refine any parameters and features. As the revenue variable is not provided in the test file, we will use the predicted revenue from various models and submit on Kaggle to obtain the RMSLE.

Figure 7 below illustrates the sort of models we could deploy to solve our problem, from simple Decision Trees to more advance XGBoost models.

**Figure 7: Evolution of Tree-Based Regression Algorithms**







Source: <https://towardsdatascience.com/https-medium-com-vishalmorde-xgboost-algorithm-long-she-may-rein-edd9f99be63d>

In layman's term, Random Forest fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and



control over-fitting.<sup>4</sup> XGBoost, LightGBM, and CatBoost is a decision-tree-based ensemble Machine Learning algorithm that uses a gradient boosting framework in addition to handling categorical features, missing values, and regularizing to avoid overfitting.

I chose the following configuration for each of aforementioned regression models and corresponding RMSLE score will be provided in Results section:

Model	Parameters
<a href="#">Random Forest</a>	 <pre>RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=10,                         max_features='auto', max_leaf_nodes=None,                         min_impurity_decrease=0.0, min_impurity_split=None,                         min_samples_leaf=1, min_samples_split=2,                         min_weight_fraction_leaf=0.0, n_estimators=100, n_jobs=None,                         oob_score=False, random_state=None, verbose=0, warm_start=False)</pre> <p>Mostly used default setting except limited depth of tree to 10 due to the sample and feature limitations as well as avoid overfitting.</p>
<a href="#">XGBoost</a>	 <pre>XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=0.5,               colsample_bytree=0.7, eval_metric='rmse', gamma=1.45,               importance_type='gain', learning_rate=0.01, max_delta_step=0,               max_depth=5, min_child_weight=1, missing=None, n_estimators=10000,               n_jobs=1, nthread=None, objective='reg:linear', random_seed=42,               random_state=0, reg_alpha=0, reg_lambda=1, scale_pos_weight=1,               seed=2019, silent=True, subsample=0.8)</pre> <p>Choose a conservative tree based algorithm to help develop clusters with limited depth to the tree to avoid overfitting.</p>
<a href="#">LightGM</a>	 <pre>LGBMRegressor(bagging_fraction=0.9, bagging_freq=1, bagging_seed=2019,                boosting='dart', boosting_type='gbdt', class_weight=None,                colsample_bytree=0.9, feature_fraction=0.9, importance_type='gain',                lambda_l1=0.2, learning_rate=0.5, max_bin=5, max_depth=5,                metric='rmse', min_child_samples=100, min_child_weight=0.001,                min_data_in_leaf=0, min_split_gain=0.0, min_sum_hessian_in_leaf=100,                n_estimators=10000, n_jobs=-1, num_leaves=255, num_threads=16,                num_trees=100, objective='regression', random_state=None,                reg_alpha=0.0, reg_lambda=0.0, silent=True, subsample=0.8,                subsample_for_bin=200000, subsample_freq=0, use_best_model=True)</pre> <p>A bagging enabled traditional gradient boosting decision tree with dropouts meeting multiple additive regression tree.</p>
<a href="#">CatBoost</a>	 <pre>catmodel = cat.CatBoostRegressor(iterations=10000,                                   learning_rate=0.01,                                   depth=5,                                   eval_metric='RMSE',                                   colsample_bylevel=0.8,                                   bagging_temperature = 0.2,                                   metric_period = None,                                   early_stopping_rounds=200,                                   random_seed=2019)</pre> <p>10,000 iterations to classify and develop a limited depth tree, and tested various learning rates and settled for 0.01 to be optimal. CatBoost will set initial weights to 0.2. This yielded the best results for the time I had available.</p>

<sup>4</sup> <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html>

## Benchmark Model

In the spirit of the Machine Learning Engineer course, my goal for this project is to develop a simple decision tree, and time permitting Ridge and Lasso, regression model to illustrate how simpler models compare to advanced modeling techniques. Recall, a decision tree is a single tree developed on our train dataset.

The simple decision tree regression model training on an 80/20 split, appropriate split given the feature and sample size, on the train dataset it yielded a RMSLE score of 3.07533, which is high or not very predictive.

```
# Create a decision tree regressor and fit it to the training set
from sklearn.tree import DecisionTreeRegressor
regressor = DecisionTreeRegressor(random_state=42)
regressor.fit(X_train,y_train)

# Make predictions using the testing data
test_pred = regressor.predict(features_test)
features_test['revenue'] = test_pred

# Report the score of the prediction using the testing set
score = regressor.score(X_test,y_test)
```

## III. Methodology

### Data Preprocessing

As aforementioned in the Data Exploration section, we need to conduct feature transformations for object features in the input data. I would like to share some of my findings when we extract the Train dataset:

- **id:** There are 3000 unique movie IDs
- **belongs\_to\_collection:** 604 or 20% of the movies belong to a collection and because more than 1 series of collections reduces the unique count to 422. The belong\_to\_collection feature is in a [JSON](#) format as is many other variables, which requires special attention.
- **budget:** The minimum is 0, which needs our attention unless the movie was produced and released pro-bono. The average budget is 22.5M USD with a standard deviation of 37M USD, which is high. The maximum budget spent was 380M USD. The unique count shows NaN therefore assume missing data is possible and a movie with missing budget should be giving little value, no pun intended.
- **genres:** Most records have this information and some 3 do not. This is in JSON format.
- **homepage:** Only 946 or 31% of the movies have a homepage. This is in a text format so we will need to convert into a binary integer tagging whether a movie has a homepage allowing this variable to visualize against revenue and evaluate usage in the model.
- **imdb\_id:** unique tag so one could connect to the website to crawl more information. We will not be doing that as part of this project as it doesn't help with the prediction as is.

- **original\_language**: all movies have a defined language with 36 unique categories of languages.
- **original\_title**: surprisingly not all movies have unique titles. One could use text analytics to determine whether
- **overview** - 8 movies have a missing brief description of the movie. We will use some text mining to count # of words in each movie's overview

There are many possibilities based on the input data, for example as one could download the poster of the movie, analyze the image and extract features such as if a presence of a pet, certain color combination, choice of words, or certain things and people may yield a lift to the revenue prediction model. Due to missing poster information for some movies, I have not explored certain features and transformed others.

Here are things I have done during the data preprocessing section to improve model performance:

1. Extracted Release Date into release year and day of release
2. Created ratio variables such as budget-runtime ratio, budget-popularity ratio, budget-year ratio, popularity-year ratio
3. Created binary variables such as whether belonging to a movie collection, whether has homepage, tagline, released
4. Log of revenue, popularity and budget to normalize
5. Used JSON to extract various spoken languages, production countries, top genres, and production companies
6. Created count variables on count # of words in title, overview, and tagline
7. Converted float variables to integer for model digestion

If we dig into the budget or revenue information, we see there is misrepresentation in the dataset such as movie ID 5, 8, 9, 12, 18, 23-26, etc have a budget of 0 while movie ID 348, 1755, 1875, 1918 only made \$1 in revenue. As mentioned earlier, connecting to TMDB API would enable us to verify if there was an issue with the input file or the book of records. Due to limited time, I have focused on the low hanging fruits with some effort.

The final output from the data preprocessing includes 31 columns, below is a sample of 5 records and few of the features:

	year	dayofrelease	log_budget	runtime	belongs_to_collection	homepage	has_tag	released	popularity	US production	...
0	2015	4	16	93	1	0	1	1	1	1	...
1	2004	4	17	113	1	0	1	1	2	1	...
2	2014	4	15	105	0	1	1	1	4	1	...
3	2012	4	13	122	0	1	0	1	1	0	...
4	2009	3	0	118	0	0	0	1	0	0	...

5 rows x 31 columns

The following 31 features were utilized for model development:

```
RangeIndex: 3000 entries, 0 to 2999
Data columns (total 31 columns):
year                3000 non-null int64
dayofrelease        3000 non-null int64
log_budget          3000 non-null int64
runtime             3000 non-null int64
belongs_to_collection 3000 non-null int64
homepage            3000 non-null int64
has_tag             3000 non-null int64
released            3000 non-null int64
popularity           3000 non-null int64
US production       3000 non-null int64
no_production_countries 3000 non-null int64
Big production      3000 non-null int64
no_production_companies 3000 non-null int64
Top genres          3000 non-null int64
no_languages         3000 non-null int64
english_spoken       3000 non-null int64
title_word_count     3000 non-null int64
overview_word_count  3000 non-null int64
tagline_word_count   3000 non-null int64
cn                   3000 non-null int64
en                   3000 non-null int64
es                   3000 non-null int64
fr                   3000 non-null int64
hi                   3000 non-null int64
it                   3000 non-null int64
ja                   3000 non-null int64
ko                   3000 non-null int64
others               3000 non-null int64
ru                   3000 non-null int64
zh                   3000 non-null int64
log_revenue          3000 non-null int64
dtypes: int64(31)
memory usage: 726.6 KB
```

## Implementation

As we explore the implementation steps, I faced couple main complications during the coding process mostly around feature engineering and finding the most optimal parameters for the various gradient boosting models. There are plenty of features we could extract from the data and with what I have extracted, I find there are opportunities to improve on my work. Finding the optimal parameters for advance boosting models is not very straight forward. A suggestion from Udacity mentor to make implementation smoother would be to implement more advanced techniques, such as combining scaling, fitting, gridSearch, etc. with the notion of [Pipelining](#).

### 1. Read and assign the three datasets

```
train = pd.read_csv('../content/train.csv') #read train data
test = pd.read_csv('../content/test.csv') #reading test data
submission = pd.read_csv('../content/sample_submission.csv') #read data
```

2. Extract the 31 features, as aforementioned in the pre-processing section for both training and testing files. See python code for more details, too long to share here and requires better documentation. Majority of the code was sourced from Kamal Chhirang<sup>5</sup>.
3. Partition the training dataset, which has the revenue information, into 80/20 split. I chose a random state of 42 because it is the answer to life's question.

```
# Split the 'features' and 'income' data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(train_copy,
                                                    revenue,
                                                    test_size = 0.2,
                                                    random_state = 42)

# Show the results of the split
print("Training set has {:,} samples.".format(X_train.shape[0]))
print("Testing set has {:,} samples.".format(X_test.shape[0]))
```

Training set has 2,400 samples.  
Testing set has 600 samples.

4. We will now use [Random Forest Regression](#) model to fit the model and obtain the RSMLE of 2.19280 on the test dataset upon submission on Kaggle, which calculates the RSMLE metric as per our file submission. We could also develop an RMSLE model internally on the train dataset.

#### ▼ Random Forest

```
[164] # Create a random forest tree regressor and fit it to the training set
rf_reg = RandomForestRegressor(criterion = 'mse', max_depth = 10, n_estimators = 100)
rf_reg.fit(X_train,y_train)
```

```
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=10,
                       max_features='auto', max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=100, n_jobs=None,
                       oob_score=False, random_state=None, verbose=0, warm_start=False)
```

```
[ ] # Use the trained model to predict revenue on test dataset
test_copy['log_revenue_pred'] = rf_reg.predict(test_copy)
```

```
[165] # Report the score of the prediction using the testing set
score = rf_reg.score(X_test,y_test)
print(score)
```

```
0.47916892785353293
```

```
[ ] # We use this to convert log revenue into revenue through the exponential function and develop the submission file for Kaggle
def file_for_submission(test_df):
    test_df['id'] = np.arange(3001,7399)
    test_df['revenue'] = test_df['log_revenue_pred'].apply(np.exp)
    test_df[['id','revenue']].to_csv('submission.csv',index = False)
```

```
[ ] file_for_submission(test_copy)
```

Yields RSMLE score of 2.19280 according to Kaggle

5. We will now use [LightGB](#) model to fit the model and obtain the RSMLE of 2.22461 on the test dataset upon submission on Kaggle, which calculates the RSMLE metric as per our file submission.

<sup>5</sup> <https://www.kaggle.com/kamalchhirang/eda-feature-engineering-lgb-xgb-cat#Feature-Engineering-&-Prediction>

## LGBModel

```
[166] lgbmodel12 = lgb.LGBMRegressor(n_estimators=10000, objective='regression', metric='rmse',
                                     max_depth = 5, num_leaves = 255, num_trees = 100,
                                     num_threads = 16, min_child_samples = 100, learning_rate=0.5,
                                     boosting = 'dart', min_data_in_leaf= 0, max_bin = 5,
                                     min_sum_hessian_in_leaf = 100, feature_fraction = 0.9,
                                     bagging_freq = 1, bagging_fraction = 0.9,
                                     importance_type='gain', lambda_l1 = 0.2,
                                     bagging_seed=2019, subsample=.8, colsample_bytree=.9,
                                     use_best_model=True)
```

```
[167] lgbmodel12.fit(X_train,y_train,
                    eval_set=[(X_train, y_train), (X_test, y_test)], eval_metric='rmse',
                    verbose=1000, early_stopping_rounds=200)
```

```
↳ LGBMRegressor(bagging_fraction=0.9, bagging_freq=1, bagging_seed=2019,
                 boosting='dart', boosting_type='gbdt', class_weight=None,
                 colsample_bytree=0.9, feature_fraction=0.9, importance_type='gain',
                 lambda_l1=0.2, learning_rate=0.5, max_bin=5, max_depth=5,
                 metric='rmse', min_child_samples=100, min_child_weight=0.001,
                 min_data_in_leaf=0, min_split_gain=0.0, min_sum_hessian_in_leaf=100,
                 n_estimators=10000, n_jobs=-1, num_leaves=255, num_threads=16,
                 num_trees=100, objective='regression', random_state=None,
                 reg_alpha=0.0, reg_lambda=0.0, silent=True, subsample=0.8,
                 subsample_for_bin=200000, subsample_freq=0, use_best_model=True)
```

```
[170] test_copy['log_revenue_pred'] = lgbmodel12.predict(test_copy)
```

```
[172] # Report the score of the prediction using the testing set
      score = lgbmodel12.score(X_test,y_test)
      print(score)
```

```
↳ 0.4555178716392949
```

```
[ ] file_for_submission(test_copy)
```

Yields RSMLE score of 2.22461 according to Kaggle.



6. We will now use [XGBoost](#) model to fit the model and obtain the RSMLE of 2.17154 on the test dataset upon submission on Kaggle, which calculates the RSMLE metric as per our file submission.

## ▾ XGBoost Model

```
[173] xgbmodel = xgb.XGBRegressor(max_depth=5,
                                learning_rate=0.01,
                                n_estimators=10000,
                                objective='reg:linear',
                                eval_metric='rmse',
                                gamma=1.45,
                                seed=2019,
                                silent=True,
                                subsample=0.8,
                                colsample_bytree=0.7,
                                colsample_bylevel=0.5,
                                random_seed = 42)
```

```
[174] xgbmodel.fit(X_train,y_train)
```

```
↳ XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=0.5,
               colsample_bytree=0.7, eval_metric='rmse', gamma=1.45,
               importance_type='gain', learning_rate=0.01, max_delta_step=0,
               max_depth=5, min_child_weight=1, missing=None, n_estimators=10000,
               n_jobs=1, nthread=None, objective='reg:linear', random_seed=42,
               random_state=0, reg_alpha=0, reg_lambda=1, scale_pos_weight=1,
               seed=2019, silent=True, subsample=0.8)
```

```
[177] test_copy['log_revenue_pred'] = xgbmodel.predict(test_copy)
```

```
[178] # Report the score of the prediction using the testing set
      score = xgbmodel.score(X_test,y_test)
      print(score)
```

```
↳ 0.4703233332266797
```

```
[ ] file_for_submission(test_copy)
```

```
[176] #test_copy = test_copy.drop(['log_revenue_pred', 'revenue', 'id'], axis=1)
      test_copy = test_copy.drop(['log_revenue_pred'], axis=1)
```

Yields RSMLE score of 2.17154 according to Kaggle.

7. We will now use [CatBoost regressor](#) model to fit the model and obtain the RSMLE of 2.07148 on the test dataset upon submission on Kaggle, which calculates the RSMLE metric as per our file submission.

### ▼ CatBoost Model

```
[ ] catmodel = cat.CatBoostRegressor(iterations=10000,
                                     learning_rate=0.01,
                                     depth=5,
                                     eval_metric='RMSE',
                                     colsample_bylevel=0.8,
                                     bagging_temperature = 0.2,
                                     metric_period = None,
                                     early_stopping_rounds=200,
                                     random_seed=2019).
```

```
catmodel.fit(X_train,y_train)
```

```
0:   learn: 16.0618618      total: 54ms      remaining: 9m
1:   learn: 15.9078182      total: 60.1ms    remaining: 5m
2:   learn: 15.7560495      total: 65.2ms    remaining: 3m 37s
3:   learn: 15.6049686      total: 70.2ms    remaining: 2m 55s
4:   learn: 15.4554993      total: 74.9ms    remaining: 2m 29s
5:   learn: 15.3083235      total: 79.6ms    remaining: 2m 12s
6:   learn: 15.1623932      total: 84.4ms    remaining: 2m
7:   learn: 15.0195519      total: 89.1ms    remaining: 1m 51s
8:   learn: 14.8775382      total: 94ms      remaining: 1m 44s
9:   learn: 14.7348836      total: 99.7ms    remaining: 1m 39s
10:  learn: 14.5940226      total: 102ms     remaining: 1m 32s
11:  learn: 14.4547535      total: 107ms     remaining: 1m 29s
12:  learn: 14.3156167      total: 110ms     remaining: 1m 24s
13:  learn: 14.1779241      total: 113ms     remaining: 1m 20s
14:  learn: 14.0425188      total: 118ms     remaining: 1m 18s
15:  learn: 13.9104372      total: 122ms     remaining: 1m 16s
16:  learn: 13.7779206      total: 127ms     remaining: 1m 14s
17:  learn: 13.6467785      total: 132ms     remaining: 1m 13s
18:  learn: 13.5167923      total: 137ms     remaining: 1m 11s
```

```
4833: learn: 1.8538535      total: 33.7s     remaining: 36s
4834: learn: 1.8538526      total: 33.7s     remaining: 36s
4835: learn: 1.8537820      total: 33.7s     remaining: 36s
4836: learn: 1.8537799      total: 33.7s     remaining: 36s
4837: learn: 1.8537797      total: 33.8s     remaining: 36s
4838: learn: 1.8537796      total: 33.8s     remaining: 36s
4839: learn: 1.8537796      total: 33.8s     remaining: 36s
4840: learn: 1.8537794      total: 33.8s     remaining: 36s
4841: learn: 1.8537290      total: 33.8s     remaining: 36s
4842: learn: 1.8537288      total: 33.8s     remaining: 36s
4843: learn: 1.8537110      total: 33.8s     remaining: 36s
```

```
[195] test_copy['log_revenue_pred'] = catmodel.predict(test_copy)
```

```
# Report the score of the prediction using the testing set
score = catmodel.score(X_test,y_test)
print(score)
```

```
[ ] file_for_submission(test_copy)
```

Yields RSMLE score of 2.07148 according to Kaggle. This model yields the best results.

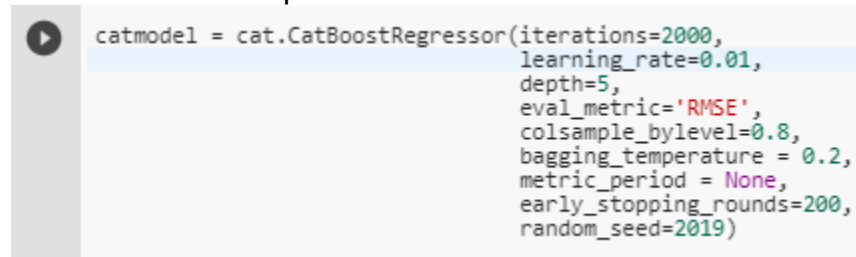
## Refinement

We could use Google Cloud AutoML to eliminate the refinement process as it selects the most optimal parameters as it claims to select the most optimal parameters. I am pretty content with the results given the limited amount of features extracted. If we were to look for an opportunity to improve a model then I just realized that I could have trained CatBoost at various learning rates with reduced iterations of 2,000 from 10,000. As a result, CatBoost at 0.1 yields a RSMLE score of 2.0977869, 0.01 yields a RSMLE score of 2.0495644, and 0.001 yields a RSMLE score of 3.2740926. This tells us that reduced iterations provided us with a slightly better lift in predicting revenue.

## IV. Results

### Model Evaluation and Validation

The final CatBoost parameters from the model is



```
catmodel = cat.CatBoostRegressor(iterations=2000,
                                  learning_rate=0.01,
                                  depth=5,
                                  eval_metric='RMSE',
                                  colsample_bylevel=0.8,
                                  bagging_temperature = 0.2,
                                  metric_period = None,
                                  early_stopping_rounds=200,
                                  random_seed=2019)
```

CatBoost model is reasonable and aligns with solution expectations. The final parameters are appropriate as we were able to get a lift from the refinement process. I have tested this model by submitting on Kaggle and ranked 400 out of over 1000 members, so pretty content. Appendix C illustrates results from a small perturbations in parameters to this model as at 10 fold scores the mean of 2.1737 and standard deviation of 0.1619. The model is robust enough for the problem and small perturbations yields marginal benefit however if it was a case of millions of dollars, small changes could yield few thousands USD swings.

### Justification

The CatBoost model yields a RSMLE score of 2.049 versus the baseline linear regression model yields a RSMLE score of 3.075 meaning we are able to better predict revenue with a more robust model. I was ranked close to 750 out of over 1000 members on Kaggle with the baseline model and jumped to 418 with the final model. The final solution is not significant enough to have solved the problem as I would ideally like to be on the top 10 of Kaggle Leaderboard however grateful to know the model is better at predicting than flipping a coin.

## V. Conclusion

## Free-Form Visualization

Coming from a risk management domain, public relation risk is very important as it would significantly reduce brand loyalty and as a result brand equity and bottom line. There have been articles about the movie industry being male dominant and if we look at the common names of all cast in 3,000 movies the results does support those articles. i.e. David, James, Mark, William, Tom, John, Peter, Richard, Michael, Robert.



## Reflection

In summary the entire end-to-end problem solution could be described as a process of feature exploration, feature engineering, and modeling. Within the feature exploration, we discovered so many different insights into the faults that a machine learning engineer will need to discover and tackle carefully. In the feature engineering, we discovered that taking the input data as is and developing a baseline model is not as accurate as extracting information from object features. Finally, modeling has its own intricacies with parameter and model selections. Each of the stages are equally important and understanding how to work with the data to drive insights is extremely important.

## Improvement

The biggest opportunity for improvement I see is in feature engineering as I have only extracted 31 features from the endless features that could be extracted when connecting to TMDb API to imagine analysis. Spending additional time understanding how to use various feature engineering techniques in python would help. On the flip side, I have never used LightGB and CatBoost model so this was an excellent opportunity to implement these algorithms. Spending more time reading up on parameters of these model would help build even better solution. It is possible as there is a #1 seed on Kaggle Leaderboard and it is not my name. Great opportunity to keep learning and evolving in the Machine Learning domain. Thank you to Udacity and everyone supporting to make this course extremely engaging within this exponential domain of Machine Learning.

# Appendix

## Appendix A - Feature Column Data Dictionary

Kamal Chhirang (<https://www.kaggle.com/kamalchhirang/eda-feature-engineering-lgb-xgb-cat>)

**id** - Unique id assigned to each movie

**belongs\_to\_collection** - If the movie belongs to a collection then additional JSON format details on parent movie such as id, movie name, poster path and backdrop URL of a movie

**budget**: Budget of a movie in dollars. 0 values mean unknown

**genres** : Contains all the genres the movie belongs to including genre ID and genre in JSON format

**homepage** - Contains the official homepage URL of a movie

**imdb\_id** - IMDB id of a movie (string). You can visit the IMDB Page by adding any imdb id to the end of this URL:

<https://www.imdb.com/title/>

**original\_language** - Two digit code of the original language, in which the movie was made. Like: en = English, fr = french.

**original\_title** - The original title of a movie. Title & Original title may differ, if the original title is not in English

**overview** - Brief description of the movie

**popularity** - Popularity of the movie

**poster\_path** - Poster path of a movie. You can see the full image by adding the path to the end of this URL:

<https://image.tmdb.org/t/p/original/>

**production\_companies** - All production company name(s) including associated production company id in JSON format

**production\_countries** - Two digit code and full name of the production company in JSON format

**release\_date** - Release date of a movie in mm/dd/yy format

**runtime** - Total runtime of a movie in minutes

**spoken\_languages** - Two digit code and full name of the spoken language

**status** - The status is populated either as released or rumored

**tagline** - Tagline of a movie

**title** - Title of the movie

**Keywords** - Unique tag id and associated name of all the tagged keywords in JSON format.

**cast** - All acting cast TMDB id, name, character name, gender (1 = Female, 2 = Male), real name, and ordered in JSON format

**crew** - All supporting crew members with their unique credit TMDB id, department of work, gender (1 = Female, 2 = Male), job TMDB id, job title, crew name, and profile path of various kind of crew members job like Director, Writer, Art, Sound etc

**revenue** - Total revenue earned by a movie in dollars

## Appendix B – ReadME Python Libraries utilized

To learn more about each of the libraries utilized, please search on [Python website](#).

```
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python docker image: https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load in

#Import libraries necessary for this project
from __future__ import absolute_import, division, print_function

import pathlib
import math
import random
import matplotlib.pyplot as plt #pretty chart displays

from IPython.display import display # Allows the use of display() for DataFrames

from wordcloud import WordCloud

from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import RobustScaler
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler

from sklearn.model_selection import cross_validate
from sklearn.model_selection import train_test_split

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt
import seaborn as sns
import ast
import xgboost as xgb
import lightgbm as lgb
import catboost as cat
from sklearn.metrics import mean_squared_log_error
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge
from sklearn.ensemble import RandomForestRegressor
import warnings
from tqdm import tqdm
from datetime import datetime
from collections import Counter
import json
from sklearn.preprocessing import LabelEncoder
warnings.filterwarnings("ignore")

# Input data files are available in the "../content/" directory.
import os
print(os.listdir("../content/"))

['.config', 'submission.csv', 'test.csv', 'train.csv', 'sample_submission.csv', 'sample_data']
```



**Appendix C – Validation of Final Model, CatBoost, using KFold CV (k=10)**

```

cat_params = {'learning_rate': 0.01,
              'depth': 5,
              'l2_leaf_reg': 10,
              'colsample_bylevel': 0.8,
              'bagging_temperature': 0.2,
              'od_type': 'Iter',
              'od_wait': 100,
              'random_seed': 11,
              'allow_writing_files': True}

oof_cat, prediction_cat = train_model(train_copy, test_copy, revenue, params=cat_params, model_type='cat', plot_feature_importance=True)

Fold 0 started at Sat May 18 01:54:17 2019
0:   learn: 15.5825187   test: 15.9022517   best: 15.9022517 (0)   total: 55.2ms   remaining: 18m 24s
1:   learn: 15.4348337   test: 15.7556001   best: 15.7556001 (1)   total: 61.9ms   remaining: 10m 19s
2:   learn: 15.2891359   test: 15.6093430   best: 15.6093430 (2)   total: 69.6ms   remaining: 7m 44s
3:   learn: 15.1449325   test: 15.4649169   best: 15.4649169 (3)   total: 75.7ms   remaining: 6m 18s
4:   learn: 15.0023395   test: 15.3215157   best: 15.3215157 (4)   total: 80.8ms   remaining: 5m 23s
5:   learn: 14.8601697   test: 15.1799417   best: 15.1799417 (5)   total: 85.8ms   remaining: 4m 45s
1163: learn: 2.0771355   test: 2.1936674 best: 2.1924752 (1066) total: 6.16s   remaining: 1m 39s
1164: learn: 2.0769341   test: 2.1935702 best: 2.1924752 (1066) total: 6.16s   remaining: 1m 39s
1165: learn: 2.0768492   test: 2.1935667 best: 2.1924752 (1066) total: 6.17s   remaining: 1m 39s
1166: learn: 2.0767649   test: 2.1935639 best: 2.1924752 (1066) total: 6.18s   remaining: 1m 39s
Stopped by overfitting detector (100 iterations wait)

bestTest = 2.192475232
bestIteration = 1066

Shrink model to first 1067 iterations.
Fold 1 started at Sat May 18 01:54:23 2019
0:   learn: 15.6158874   test: 15.6019707   best: 15.6019707 (0)   total: 6.11ms   remaining: 2m 2s
1:   learn: 15.4679095   test: 15.4535020   best: 15.4535020 (1)   total: 11.6ms   remaining: 1m 55s
3330: learn: 1.9836255   test: 1.9217383 best: 1.9210898 (3230) total: 17.2s   remaining: 1m 26s
Stopped by overfitting detector (100 iterations wait)

bestTest = 1.921089799
bestIteration = 3230

Shrink model to first 3231 iterations.
Fold 2 started at Sat May 18 01:54:42 2019
0:   learn: 15.6027145   test: 15.7241324   best: 15.7241324 (0)   total: 6.29ms   remaining: 2m 5s
1:   learn: 15.4548791   test: 15.5751182   best: 15.5751182 (1)   total: 12.1ms   remaining: 2m 1s
2:   learn: 15.3089616   test: 15.4275723   best: 15.4275723 (2)   total: 17.3ms   remaining: 1m 55s
3:   learn: 15.1648444   test: 15.2828067   best: 15.2828067 (3)   total: 22.3ms   remaining: 1m 51s
4:   learn: 15.0225554   test: 15.1397065   best: 15.1397065 (4)   total: 27.3ms   remaining: 1m 49s
1550: learn: 2.0941586   test: 2.0817133 best: 2.0815783 (1452) total: 7.99s   remaining: 1m 35s
1551: learn: 2.0941397   test: 2.0817025 best: 2.0815783 (1452) total: 8s   remaining: 1m 35s
1552: learn: 2.0939436   test: 2.0816388 best: 2.0815783 (1452) total: 8s   remaining: 1m 35s
Stopped by overfitting detector (100 iterations wait)

bestTest = 2.08157831
bestIteration = 1452

Shrink model to first 1453 iterations.
Fold 3 started at Sat May 18 01:54:50 2019
0:   learn: 15.6355969   test: 15.4262088   best: 15.4262088 (0)   total: 6.3ms   remaining: 2m 6s
1:   learn: 15.4873995   test: 15.2777886   best: 15.2777886 (1)   total: 12.2ms   remaining: 2m 1s
2:   learn: 15.3411814   test: 15.1309138   best: 15.1309138 (2)   total: 17.3ms   remaining: 1m 55s
3:   learn: 15.1969804   test: 14.9858788   best: 14.9858788 (3)   total: 22.7ms   remaining: 1m 53s
4:   learn: 15.0543513   test: 14.8427371   best: 14.8427371 (4)   total: 27.5ms   remaining: 1m 49s
1889: learn: 2.0468001   test: 2.0545027 best: 2.0538393 (1791) total: 9.71s   remaining: 1m 33s
1890: learn: 2.0466113   test: 2.0545195 best: 2.0538393 (1791) total: 9.72s   remaining: 1m 33s
1891: learn: 2.0465785   test: 2.0545449 best: 2.0538393 (1791) total: 9.72s   remaining: 1m 33s
Stopped by overfitting detector (100 iterations wait)

bestTest = 2.053839279
bestIteration = 1791

Shrink model to first 1792 iterations.
Fold 4 started at Sat May 18 01:55:01 2019
0:   learn: 15.6101430   test: 15.6581527   best: 15.6581527 (0)   total: 6.08ms   remaining: 2m 1s
1:   learn: 15.4621303   test: 15.5103028   best: 15.5103028 (1)   total: 12.2ms   remaining: 2m 1s
2:   learn: 15.3167986   test: 15.3648202   best: 15.3648202 (2)   total: 17.1ms   remaining: 1m 54s
3:   learn: 15.1725940   test: 15.2196972   best: 15.2196972 (3)   total: 22.3ms   remaining: 1m 51s

```

1999: learn: 2.0355402 test: 2.3045196 best: 2.3038760 (1901) total: 10.4s remaining: 1m 33s  
 2000: learn: 2.0355394 test: 2.3045231 best: 2.3038760 (1901) total: 10.4s remaining: 1m 33s  
 2001: learn: 2.0355385 test: 2.3045265 best: 2.3038760 (1901) total: 10.4s remaining: 1m 33s  
 Stopped by overfitting detector (100 iterations wait)

bestTest = 2.30387596  
 bestIteration = 1901

Shrink model to first 1902 iterations.

Fold 5 started at Sat May 18 01:55:12 2019

0: learn: 15.6220566 test: 15.5479470 best: 15.5479470 (0) total: 6.43ms remaining: 2m 8s  
 1: learn: 15.4748193 test: 15.4008623 best: 15.4008623 (1) total: 12.9ms remaining: 2m 8s  
 2: learn: 15.3288806 test: 15.2547952 best: 15.2547952 (2) total: 18.5ms remaining: 2m 3s  
 3: learn: 15.1843569 test: 15.1114440 best: 15.1114440 (3) total: 23.6ms remaining: 1m 58s  
 2886: learn: 1.9971555 test: 1.9751098 best: 1.9746520 (2788) total: 15.2s remaining: 1m 30s  
 2887: learn: 1.9971551 test: 1.9751103 best: 1.9746520 (2788) total: 15.2s remaining: 1m 30s  
 2888: learn: 1.9969565 test: 1.9750070 best: 1.9746520 (2788) total: 15.2s remaining: 1m 30s  
 Stopped by overfitting detector (100 iterations wait)

bestTest = 1.974651959  
 bestIteration = 2788

Shrink model to first 2789 iterations.

Fold 6 started at Sat May 18 01:55:28 2019

0: learn: 15.6137970 test: 15.6222533 best: 15.6222533 (0) total: 13.2ms remaining: 4m 24s  
 1: learn: 15.4655970 test: 15.4751003 best: 15.4751003 (1) total: 19.4ms remaining: 3m 14s  
 2: learn: 15.3193869 test: 15.3298486 best: 15.3298486 (2) total: 25.5ms remaining: 2m 50s  
 4481: learn: 1.8899648 test: 2.4644170 best: 2.4643354 (4382) total: 24.6s remaining: 1m 25s  
 4482: learn: 1.8898244 test: 2.4645399 best: 2.4643354 (4382) total: 24.6s remaining: 1m 25s  
 Stopped by overfitting detector (100 iterations wait)

bestTest = 2.464335405  
 bestIteration = 4382

Shrink model to first 4383 iterations.

Fold 7 started at Sat May 18 01:55:55 2019

0: learn: 15.6395826 test: 15.3890291 best: 15.3890291 (0) total: 6.9ms remaining: 2m 17s  
 1: learn: 15.4911355 test: 15.2412574 best: 15.2412574 (1) total: 13.8ms remaining: 2m 18s  
 2: learn: 15.3447952 test: 15.0966701 best: 15.0966701 (2) total: 20ms remaining: 2m 13s  
 4052: learn: 1.9358156 test: 2.3345520 best: 2.3342738 (3954) total: 21.2s remaining: 1m 23s  
 4053: learn: 1.9357983 test: 2.3345097 best: 2.3342738 (3954) total: 21.2s remaining: 1m 23s  
 4054: learn: 1.9356337 test: 2.3345362 best: 2.3342738 (3954) total: 21.2s remaining: 1m 23s  
 Stopped by overfitting detector (100 iterations wait)

bestTest = 2.334273796  
 bestIteration = 3954

Shrink model to first 3955 iterations.

Fold 8 started at Sat May 18 01:56:18 2019

0: learn: 15.6256204 test: 15.5155452 best: 15.5155452 (0) total: 6.39ms remaining: 2m 7s  
 1: learn: 15.4774265 test: 15.3678348 best: 15.3678348 (1) total: 12.3ms remaining: 2m 3s  
 2: learn: 15.3313603 test: 15.2215918 best: 15.2215918 (2) total: 17.4ms remaining: 1m 56s  
 3: learn: 15.1865816 test: 15.0773783 best: 15.0773783 (3) total: 22.3ms remaining: 1m 51s  
 4: learn: 15.0437377 test: 14.9358561 best: 14.9358561 (4) total: 27.6ms remaining: 1m 50s  
 5: learn: 14.9010464 test: 14.7941569 best: 14.7941569 (5) total: 32.9ms remaining: 1m 49s  
 772: learn: 2.1592840 test: 2.2633477 best: 2.2620639 (672) total: 4s remaining: 1m 39s  
 Stopped by overfitting detector (100 iterations wait)

bestTest = 2.262063894  
 bestIteration = 672

Shrink model to first 673 iterations.

Fold 9 started at Sat May 18 01:56:22 2019

0: learn: 15.5994665 test: 15.7527696 best: 15.7527696 (0) total: 6.45ms remaining: 2m 8s  
 1: learn: 15.4516563 test: 15.6043333 best: 15.6043333 (1) total: 12.5ms remaining: 2m 4s  
 1140: learn: 2.1262433 test: 2.1494636 best: 2.1486138 (1042) total: 5.96s remaining: 1m 38s  
 1141: learn: 2.1261732 test: 2.1494835 best: 2.1486138 (1042) total: 5.96s remaining: 1m 38s  
 1142: learn: 2.1261547 test: 2.1495043 best: 2.1486138 (1042) total: 5.97s remaining: 1m 38s  
 Stopped by overfitting detector (100 iterations wait)

bestTest = 2.148613779  
 bestIteration = 1042

Shrink model to first 1043 iterations.  
 CV mean score: 2.1737, std: 0.1619.