1 Homework Task

Created by Dr. James G. Shanahan

By now we have learned how to solve a problem from end to end using SKLearn, and using pipelines. For example, we learned how to predict house prices in California, how to classify images, etc. In this homework, we push pipelines to the next level as we incorporate feature engineering directly in to the modeling workflow via SKlearn's `Pipeline` class. This will be done in the context of the Titanic Survival classification problem. The following figure gives a quick overview of a possible classification pipeline that you will be building to solve this challenge.

Table of Contents

2 Project overview: TMDB Box Office Prediction on Kaggle

2.1 Challenge: predict the worldwide revenue for 4,398 movies (test data)

This project involves working with movie data to predict the worldwide takings/revenue for each movie. This project is based on the TMDB Box Office Prediction Competition on Kaggle (https://www.kaggle.com/c/tmdb-box-office-prediction) So far we have and lab we developed some sophisticate prediction pipelines. For example, we developed a pipeline to predict house prices in California. In this homework, we will build on these past efforts and adopt these existing machine learning pipeline to tackle the TMDB Box Office Prediction.

In a world… where movies made an estimated $41.7 billion in 2018, the film industry is more popular than ever. But what movies make the most money at the box office? How much does a director matter? Or the budget? For some movies, it's "You had me at 'Hello.'" For others, the trailer falls short of expectations and you think "What we have here is a failure to communicate."

In this dataset, you are provided with 7,398 movies (3,000 for training and 4,398 for testing) and a variety of metadata obtained from The Movie Database (TMDB). The goal to try and predict their overall worldwide box office revenue for each movie. Each movie is associate with a unique id. Each row in the dataset corresponds to a movie, its corresponding input features, target feature which corresponds to the worldwide takings/revenue for that movie. This input features for a movie include the cast, crew, plot keywords, budget, posters, release dates, languages, production companies, and countries.

Task: the challenge here is to predict the worldwide revenue for 4,398 movies in the test file given various information about the movie.

Note - many movies are remade over the years, therefore it may seem like multiple instance of a movie may appear in the data, however they are different and should be considered separate movies. In addition, some movies may share a title, but be entirely unrelated.

E.g.

- The Karate Kid (id: 5266) was released in 1986, while a clearly (or maybe just subjectively) inferior remake (id: 1987) was released in 2010.
- Also, while the Frozen (id: 5295) released by Disney in 2013 may be the household name, don't forget about the less-popular Frozen (id: 139) released three years earlier about skiers who are stranded on a chairlift...

Feel free to use the Kaggle API (https://github.com/Kaggle/kaggle-api) for downloading the dataset or submitting to the competition. It is not mandatory to use the package but it would be interesting to explore.

You will need to:

- **Important:** Make sure your results are reproducible
- **Important:** Use the training data set provided by the competition to create a training set(70%), validation set (15%) and a test set (15%)

- **EDA.** Identify the types of data available, evaluate basic statistical information about the data and determine whether you have any missing or misformated data.
- **Feature Engineering.** Develop at least one new feature based on the existing features of the dataset
- **Pre-processing.** All work must be performed using pipelines. You can adapt code from above or develop your own.
- **Modeling.**Evaluate at least two appropriate algorithms (estimators) for generating predictions.
  - Use grid search to tune hyperparameters.
  - Use crossfold evaluation (cv=5).
- **Evaluation.** Select appropriate metrics for the problem to evaluate your models.
- **Reporting.** Record all experiments in a table of results (pandas dataframe) including at least the following information:
  - description of the model (algorithim, notable processing steps)
  - key hyperparameters
  - results (using one or more appropriate metrics)
  - run time for each experiment (train and test results)
  - hardware used
- **Submit your best model to Kaggle** Provide a screenshot of the kaggle submission
- **Comment your code and provide explanations of how you're proceeding in each part**

In [1]:
```
import numpy as np
```

In [2]:
```
print(f"{2.200000e-01:,}")
print(f"{np.expm1(2.200000e-01)*10**9:,}")
np.expm1(2.200000e-01)
```

```
0.22
246,076,730.58738083
```

Out[2]:
```
0.24607673058738083
```

In [3]:
```
leaderboard_RMLSE = 0.6877
print(f"{leaderboard_RMLSE:,}")
print(f"{np.expm1(leaderboard_RMLSE)*10**9:,}")
```

```
0.6877
989,135,256.8536093
```

3 Team Info

TMDB Box Office Prediction

Group 1

Names: Sarah Freeman, Steven Grivers, Varun Arvind

Emails: [sfreeman1@bryant.edu (mailto:sfreeman1@bryant.edu)](mailto:sfreeman1@bryant.edu), [sgrivers@bryant.edu (mailto:sgrivers@bryant.edu)](mailto:sgrivers@bryant.edu), [varvind@bryant.edu (mailto:varvind@bryant.edu)](mailto:varvind@bryant.edu)

Photos:

4 Data Description

In this project we are examining 7,398 movies the the movie dataset. The 7,398 movies is split into 3,000 for training and 4,398 for testing. The testing data includes many features about the movies along with the revenue for the movies. The testing data includes all of the information about the movies but it does not include the revenue. Some of the information that is given about the movies are the collection the movie belongs to, the budget, the genre, the original language, the title, the popularity, the release data, the cast, the crew, an overview of the movie, the runtime, some keywords associated with the movies, the tagline, the languages the movie is spoken in, the production company and many more.

5 Task at Hand

The task that we are tackling in the third phase of this project is to improve our Phase 2 model with a few more features. The features that we are going to add are production countries, production companies,spoken languages, keywords, cast, and crew. We are going to do some feature engineering on these variables so they can be useful in helping to predict the revenue of movies. We will make two models - one predicting the raw box revenue and one predicting the log of the box revenue. We plan to take the root mean square (log) error, the average number of dollars we overpredicted/underpredicted for a movie, and the mean absolute percentage error. Then we will continue doing kaggle submissions to see how these new features improve our score.

6 Abstract

In phase 3 of the Kaggle TMDB Box Office Dataset Competition, our team attempted to improve our model to predict the revenue of Hollywood movies. In this 'improved' model, we did some feature engineering on the JSON features. We also implemented a transformer in order to clean up the log features such as log_budget and log_popularity.. We added them to the data_pipeline, and ran a KNNRegressor on this preprocessed data. Our new model proved to be slightly better than the second one, with a Kaggle score of 2.49487, which would place us in about 900th place on the leaderboard out of about 1400. With more time, we would've like to fully implement grid search, but this is a successful final model.

7 Downloading the files via Kaggle API (feel free to skip as I have already downloaded it for you)

Create a base directory:

```
DATA_DIR = "../../../Data/tmdb-box-office-prediction"   #same level as c
ourse repo in the data directory
```

Please download the project data files and data dictionary and unzip them using either of the following approaches:

1. Click on the `Download` button on the following [Data Webpage (https://www.kaggle.com/c/tmdb-box-office-prediction/data)](https://www.kaggle.com/c/tmdb-box-office-prediction/data) and unzip the zip file to the `BASE_DIR`
2. If you plan to use the Kaggle API, please use the following steps.

In [2]:
```
DATA_DIR = "./tmdb-box-office-predictions"   #same level as course repo in
#DATA_DIR = os.path.join('./ddddd/')
!mkdir $DATA_DIR
```

```
mkdir: cannot create directory ‘./tmdb-box-office-predictions’: File exists
```

In [3]:

```
!ls -l $DATA_DIR
```

```
total 68700
-rwxr-xr-x 1 root root    94918 Apr 23 12:19 TrainAdditionalFeatures.xls
-rwxr-xr-x 1 root root    61585 Apr 23 12:20 sample_submission.xls
-rwxr-xr-x 1 root root 41868556 Apr 23 12:18 test.csv
-rwxr-xr-x 1 root root 28311747 Apr 23 12:18 train.csv
```

In [6]:

```
# ! kaggle competitions download tmdb-box-office-prediction -p $DATA_DIR
```

7.1 Data Import & notebook preperation

In [4]:

```
import pandas as pd
import os
def load_data(in_path, name):
    df = pd.read_csv(in_path)
    print(f"{name}: shape is {df.shape}")
    print(df.info())
    display(df.head(5))
    return df

datasets={}  # lets store the datasets in a dictionary so we can keep track
ds_name = 'train'
df_train = load_data(os.path.join(DATA_DIR, f'{ds_name}.csv'), ds_name)
#datasets[ds_name] = df_train
ds_name = 'test'
df_test = load_data(os.path.join(DATA_DIR, f'{ds_name}.csv'), ds_name)
#datasets[ds_name] = df_test
```

```
train: shape is (3000, 23)
<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: 539.2+ KB
None
```

	id	belongs_to_collection	budget	genres	homepage	imdb_id
0	1	[{'id': 313576, 'name': 'Hot Tub Time Machine ...	14000000	[{'id': 35, 'name': 'Comedy'}]	NaN	tt2637294

	id	belongs_to_collection	budget	genres	homepage	imdb_id
1	2	[{'id': 107674, 'name': 'The Princess Diaries ...	40000000	[{'id': 35, 'name': 'Comedy'}, {'id': 18, 'nam...	NaN	tt0368933
2	3	NaN	3300000	[{'id': 18, 'name': 'Drama'}]	http://sonyclassics.com/whiplash/	tt2582802
3	4	NaN	1200000	[{'id': 53, 'name': 'Thriller'}, {'id': 18, 'n...	http://kahaanithefilm.com/	tt1821480
4	5	NaN	0	[{'id': 28, 'name': 'Action'}, {'id': 53, 'nam...	NaN	tt1380152

5 rows × 23 columns

```
test: shape is (4398, 22)
<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
None
```

	id	belongs_to_collection	budget	genres	homepag
0	3001	[{'id': 34055, 'name': 'Pokémon Collection', '...	0	[{'id': 12, 'name': 'Adventure'}, {'id': 16, '...	http://www.pokemon.com/us/movies/movie pokemon.
1	3002	NaN	88000	[{'id': 27, 'name': 'Horror'}, {'id': 878, 'na...	Nal
2	3003	NaN	0	[{'id': 35, 'name': 'Comedy'}, {'id': 10749, '...	Nal
3	3004	NaN	6800000	[{'id': 18, 'name': 'Drama'}, {'id': 10752, 'n...	http://www.sonyclassics.com/incendies
4	3005	NaN	2000000	[{'id': 36, 'name': 'History'}, {'id': 99, 'na...	Nal

5 rows × 22 columns

In [5]:

```
ds_name = 'train'
print(f'dataset {ds_name:24}: [ {df_train.shape[0]:10,}, {df_train.shape[1]
ds_name = 'test'
print(f'dataset {ds_name:24}: [ {df_test.shape[0]:10,}, {df_test.shape[1]}]
```

```
dataset train                   : [      3,000, 23]
dataset test                    : [      4,398, 22]
```

- **Important:** Remember that x and y should be split into a training set (70% of the original dataset), a validation set (15% of the original dataset) and a test set (15% of the original dataset.
- test_data will be only used for the kaggle submission

8 Evaluation metrics

There has been a lot of evaluation metrics when it comes to Regression problem and Root Mean Square Error or RMSE, in short, has been among the “goto” methods for the evaluation of regression problems and has been around since forever.

But recently, there has been a wildcard entry among the evaluation metrics for regression problems, especially in the Data Science competitions, and is referred to as Root Mean Squared Log Error (RMSLE).

$$\text{RMSLE} = \sqrt{(\frac{1}{n})\sum_{i=1}^{n}(\log(\hat{y}_i + 1) - \log(y_i + 1))^2}$$

At first glance, it would seem like there is just a difference of the keyword “Log” in the name of the metric.

In case of RMSLE, you take the log of the predictions and actual values. So basically, what changes is the variance that you are measuring. I believe RMSLE is usually used when you don't want to penalize huge differences in the predicted and the actual values when both predicted and true values are huge numbers.

- If both predicted and actual values are small: RMSE and RMSLE is same.
- If either predicted or the actual value is big: RMSE > RMSLE
- If both predicted and actual values are big: RMSE > RMSLE (RMSLE becomes almost negligible)

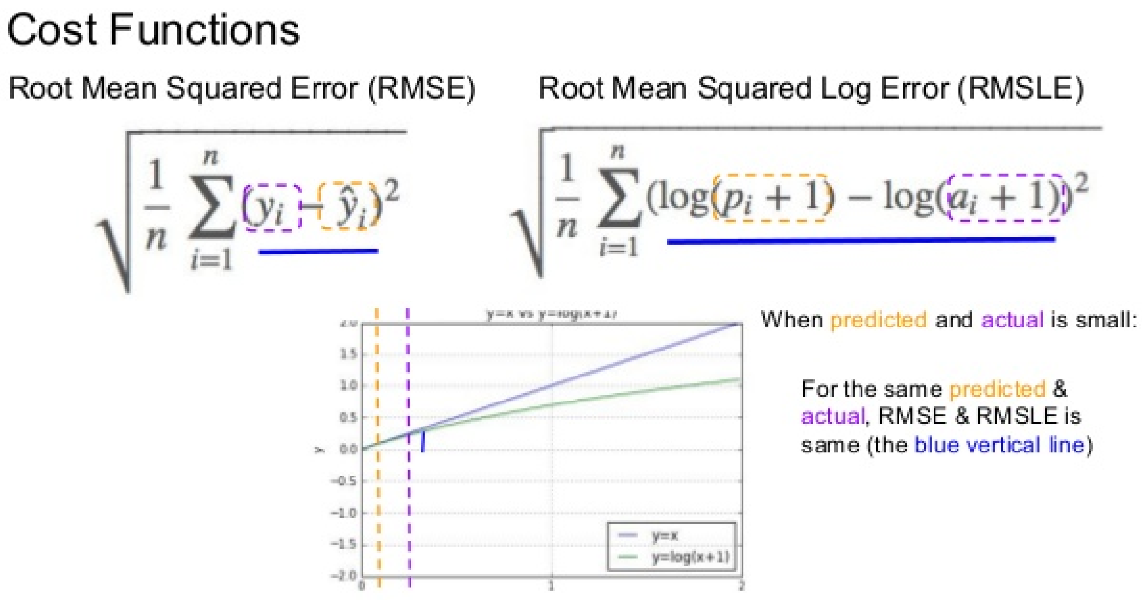
The Robustness of RMSLE to the outliers, the property of calculating the relative error between the Predicted and Actual Values, the most unique property of the RMLSE that it penalizes the underestimation of the actual value more severely than it does for the Overestimation.

8.1 Reference

- https://www.kaggle.com/c/ashrae-energy-prediction/discussion/113064 (https://www.kaggle.com/c/ashrae-energy-prediction/discussion/113064)
- https://medium.com/analytics-vidhya/root-mean-square-log-error-rmse-vs-rmlse-935c6cc1802a (https://medium.com/analytics-vidhya/root-mean-square-log-error-rmse-vs-rmlse-935c6cc1802a)
- https://towardsdatascience.com/metrics-and-python-850b60710e0c (https://towardsdatascience.com/metrics-and-python-850b60710e0c)

In [5]:

```
import math

#A function to calculate Root Mean Squared Logarithmic Error (RMSLE)
def rmsle(y, y_pred):
    assert len(y) == len(y_pred)
    terms_to_sum = [(math.log(y_pred[i] + 1) - math.log(y[i] + 1)) ** 2.0 f
    return (sum(terms_to_sum) * (1.0/len(y))) ** 0.5
```



9 Preprocessing

In [6]:
```
df_train.columns
```

Out[6]:
```
Index(['id', 'belongs_to_collection', 'budget', 'genres', 'homepage',
       'imdb_id', 'original_language', 'original_title', 'overview',
       'popularity', 'poster_path', 'production_companies',
       'production_countries', 'release_date', 'runtime', 'spoken_language
s',
       'status', 'tagline', 'title', 'Keywords', 'cast', 'crew', 'revenu
e'],
      dtype='object')
```

9.1 Import Libraries

In [7]:
```
from sklearn.pipeline import Pipeline, FeatureUnion, make_pipeline
from sklearn.compose import ColumnTransformer
import numpy as np
import pandas as pd
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer, make_column_transformer
from sklearn.model_selection import train_test_split  # sklearn.cross_valid
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.linear_model import LinearRegression
from sklearn.impute import SimpleImputer
from time import time
from sklearn.metrics import mean_squared_log_error
import datetime
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.feature_extraction.text import HashingVectorizer, TfidfVectori
from sklearn.model_selection import GridSearchCV
```

9.2 Date Transformer

In [8]:

```
class DateTransformer(TransformerMixin):
    '''
    This date transformer takes the release date that was originally in a m
    the date into year, month, day, and quarter all in their own columns. T
    in evaluating how the year and date can affect the revenue for a movie.
    '''
    def __init__(self,features=None):
        self.features = features
    def fit(self, X, y=None):
        self.col = X.columns
        return self
    def transform(self, X):
        df = pd.DataFrame(X.copy(), columns=self.features)
        print(f'shape is: {df.shape}')
        df['release_date'] = pd.to_datetime(df['release_date'])
        df = pd.DataFrame({#'year':df[self.col[0]].dt.year,
                           'month':df[self.col[0]].dt.month})#,
                          #'day':df[self.col[0]].dt.day,
                          #'quarter':df[self.col[0]].dt.quarter})
        print(f'Shape is now {df.shape}')
        return df

date_feature = ["release_date"]

test_pipeline = make_pipeline(DateTransformer(date_feature))

display(test_pipeline.fit_transform(df_train[date_feature]).head())
test_pipeline.fit_transform(df_train[date_feature]).count()
```

```
shape is: (3000, 1)
Shape is now (3000, 1)
```

	month
0	2
1	8
2	10
3	3
4	2

```
shape is: (3000, 1)
Shape is now (3000, 1)
```

Out[8]:

```
month    3000
dtype: int64
```

9.3 Json

In [9]:
```
# JSON based features
json_columns = ['genres', 'production_countries', 'production_companies','s

## decode/deserialize JSON base features
#  replace missing values for multivalued features to {}

def decode_json_features(df):
    for column in json_columns:
        df[column] = df[column].apply(lambda x: {} if pd.isna(x) else eval(
    return df

df_train = decode_json_features(df_train)
df_test = decode_json_features(df_test)
```

In [10]:
```
class ListValueTransformer(BaseEstimator, TransformerMixin):
    """
    Parses deserialized JSON objects (previously in the data preprocessing
    we decoded JSON  to list of dictionaries using the eval() function).
    # JSON based features
    # json_columns = ['belongs_to_collection', 'genres', 'production_compan
                'production_countries', 'spoken_languages', 'Keywords', 'ca
    E.g., extra a CSV list  of genres from the JSON decoded list of genre i
       Go FROM [{'id': 53, 'name': 'Thriller'}, {'id': 18, 'name': 'Drama'}
        TO Drama,Thriller
    Parameters
    -------
        X : DataFrame
            assumes X is a DataFrame

    Returns
    -------
        DataFrame
    """

    def fit(self, X, y=None):
        # stateless transformer for now but in the future consider the foll
        # TODO convert to a stateful Transformer
        # e.g., drop low frequency items
        return self

    def transform(self, X):    #CSV String of names
        return X.applymap(lambda x: ','.join(sorted([i['name'] for i in x])
```

9.4 Counts of Features

In [13]:

```
df_train['Spoken_languages']= parser.fit_transform(df_train[['spoken_langua
df_test['Spoken_languages']= parser.fit_transform(df_test[['spoken_language
df_train['Production_countries']= parser.fit_transform(df_train[['productio
df_test['Production_countries']= parser.fit_transform(df_test[['production_
df_train['Production_countries']
df_train['Production_companies']= parser.fit_transform(df_train[['productio
df_test['Production_companies']= parser.fit_transform(df_test[['production_
df_train['keywords']= parser.fit_transform(df_train[['Keywords']])
df_test['keywords']= parser.fit_transform(df_test[['Keywords']])
df_train['Cast']= parser.fit_transform(df_train[['cast']])
df_test['Cast']= parser.fit_transform(df_test[['cast']])
df_train['Crew']= parser.fit_transform(df_train[['crew']])
df_test['Crew']= parser.fit_transform(df_test[['crew']])
df_train['Crew']
```

Out[13]:

```
0       Adam Blum,Allison Gordin,Andrew Panay,Annabell...
1       Bruce Green,Charles Minsky,Debra Martin Chase,...
2       Alicia Hadaway,Andy Ross,Barbara Harris,Ben Wi...
3                     Sujoy Ghosh,Sujoy Ghosh,Sujoy Ghosh
4                            Jong-seok Yoon,Jong-seok Yoon
                              ...
2995    Christian Wagner,Dan Gilroy,David Wisnievitz,D...
2996    Anna Anthony,Christian Wikander,Coco Moodysson...
2997    Alan Silvestri,Allen Hall,Geena Davis,Guillerm...
2998    Alex Daniels,Anders Bard,Andrew Bracken,Andrew...
2999    Alan Lee,Alison Evans,Amanda Jenkins,Brad Mart...
Name: Crew, Length: 3000, dtype: object
```

9.5 Genres and Production Countries

In [12]:

```
#returns all of the genres per movie separated by a comma
parser = ListValueTransformer()
json_columns = ['genres', 'production_countries', 'production_companies','s

df_train['Genres'] = parser.fit_transform(df_train[['genres']])
df_test['Genres'] = parser.fit_transform(df_test[['genres']])
df_train['Genres']
df_train['Production_countries'] = parser.fit_transform(df_train[['producti
df_test['Production_countries'] = parser.fit_transform(df_test[['production
```

In [88]:

Out[88]:

	id	belongs_to_collection	budget	genres	homepage	imdb_
0	1	[{'id': 313576, 'name': 'Hot Tub Time Machine ...	14000000	[{'id': 35, 'name': 'Comedy'}]	NaN	tt263729
1	2	[{'id': 107674, 'name': 'The Princess Diaries ...	40000000	[{'id': 35, 'name': 'Comedy'}, {'id': 18, 'nam...	NaN	tt036893
2	3	NaN	3300000	[{'id': 18, 'name': 'Drama'}]	http://sonyclassics.com/whiplash/	tt258280
3	4	NaN	1200000	[{'id': 53, 'name': 'Thriller'}, {'id': 18, 'n...	http://kahaanithefilm.com/	tt182148
4	5	NaN	0	[{'id': 28, 'name': 'Action'}, {'id': 53, 'nam...	NaN	tt138015
...	...	...	...	...	...	
2995	2996	NaN	0	[{'id': 35, 'name': 'Comedy'}, {'id': 10749, '...	NaN	tt010940
2996	2997	NaN	0	[{'id': 18, 'name': 'Drama'}, {'id': 10402, 'n...	NaN	tt236497
2997	2998	NaN	65000000	[{'id': 80, 'name': 'Crime'}, {'id': 28, 'name...	NaN	tt011690
2998	2999	NaN	42000000	[{'id': 35, 'name': 'Comedy'}, {'id': 10749, '...	http://www.alongcamepolly.com/	tt034313
2999	3000	NaN	35000000	[{'id': 53, 'name': 'Thriller'}, {'id': 28, 'n...	http://www.abductionthefilm.com/	tt160019

3000 rows × 65 columns

In [14]:

```
df_train['firstGenres'] = df_train["Genres"].str.extract('([A-Z[a-z]+)')
df_train['firstGenres']
df_test['firstGenres'] = df_test["Genres"].str.extract('([A-Z[a-z]+)')

df_train['Production_countries_first'] = df_train["Production_countries"].s
df_train['Production_countries_first']
df_test['Production_countries_first'] = df_test["Production_countries"].str
```

9.6 Log Transformer

In [15]:

```
class logTransformer(BaseEstimator, TransformerMixin):
    def __init__(self, features = None):
        self.features = features
    def fit(self, X, y = None):
        return self
    def transform(self, X):
        df = pd.DataFrame(X.copy(), columns = self.features)
        df['log budget'] = np.log1p(df[self.features[0]])
        df['log popularity'] = np.log1p(df[self.features[1]])
        df.drop(self.features, axis = 1, inplace = True)
        return df
log_feature = ["budget", 'popularity']

test_pipeline = make_pipeline(logTransformer(log_feature))

display(test_pipeline.fit_transform(df_train[log_feature]).head())
test_pipeline.fit_transform(df_train[log_feature]).count()
```

	log budget	log popularity
0	16.454568	2.024905
1	17.504390	2.224504
2	15.009433	4.178992
3	13.997833	1.429099
4	0.000000	0.764570

Out[15]:

```
log budget        3000
log popularity    3000
dtype: int64
```

9.7 Count Transformer

In [38]:
```
class countTransformer(BaseEstimator, TransformerMixin):
    def __init__(self, features = None):
        self.features = features
    def fit(self, X, y = None):
        return self
    def transform(self, X):
        df = pd.DataFrame(X.copy(), columns = self.features)
        #df['production_companies_count'] = df[self.features[0]].apply(lamb
        df['cast_count'] = df[self.features[0]].apply(lambda x: len(x))
        df['crew_count'] = df[self.features[1]].apply(lambda x: len(x))
        df.drop(self.features, axis = 1, inplace = True)
        return df
count_feature = [ 'cast', 'crew', 'Production_companies']

test_pipeline = make_pipeline(countTransformer(count_feature))

display(test_pipeline.fit_transform(df_train[count_feature]).head())
test_pipeline.fit_transform(df_train[count_feature]).count()
```

	cast_count	crew_count
0	24	72
1	20	9
2	51	64
3	7	3
4	4	2

Out[38]:
```
cast_count    3000
crew_count    3000
dtype: int64
```

9.8 Cleaning Up Data

In [16]:
```
df_train['release_date'] = pd.to_datetime(df_train['release_date'])
df_test['release_date'] = pd.to_datetime(df_test['release_date'])
```

In [17]:

```
# fix release dates
df_train.iloc[df_train[df_train.release_date > '06/01/2019'].release_date.i
df_train[df_train.release_date > '06/01/2019'].release_date.apply(lambda x:

# data fixes from https://www.kaggle.com/somang1418/happy-valentines-day-an
df_train.loc[df_train['id'] == 16,'revenue'] = 192864        # Skinning
df_train.loc[df_train['id'] == 90,'budget'] = 30000000       # Sommersby
df_train.loc[df_train['id'] == 118,'budget'] = 60000000      # Wild Hogs
df_train.loc[df_train['id'] == 149,'budget'] = 18000000      # Beethoven
df_train.loc[df_train['id'] == 313,'revenue'] = 12000000     # The Cookou
df_train.loc[df_train['id'] == 451,'revenue'] = 12000000     # Chasing Li
df_train.loc[df_train['id'] == 464,'budget'] = 20000000      # Parenthood
df_train.loc[df_train['id'] == 470,'budget'] = 13000000      # The Karate
df_train.loc[df_train['id'] == 513,'budget'] = 930000        # From Prada
df_train.loc[df_train['id'] == 797,'budget'] = 8000000       # Welcome to
df_train.loc[df_train['id'] == 819,'budget'] = 90000000      # Alvin and
df_train.loc[df_train['id'] == 850,'budget'] = 90000000      # Modern Tim
df_train.loc[df_train['id'] == 1112,'budget'] = 7500000      # An Officer
df_train.loc[df_train['id'] == 1131,'budget'] = 4300000      # Smokey and
df_train.loc[df_train['id'] == 1359,'budget'] = 10000000     # Stir Crazy
df_train.loc[df_train['id'] == 1542,'budget'] = 1            # All at Onc
df_train.loc[df_train['id'] == 1570,'budget'] = 15800000     # Crocodile
df_train.loc[df_train['id'] == 1571,'budget'] = 4000000      # Lady and t
df_train.loc[df_train['id'] == 1714,'budget'] = 46000000     # The Recrui
df_train.loc[df_train['id'] == 1721,'budget'] = 17500000     # Cocoon
df_train.loc[df_train['id'] == 1865,'revenue'] = 25000000    # Scooby-Doo
df_train.loc[df_train['id'] == 2268,'budget'] = 17500000     # Madea Goes
df_train.loc[df_train['id'] == 2491,'revenue'] = 6800000     # Never Talk
df_train.loc[df_train['id'] == 2602,'budget'] = 31000000     # Mr. Hollan
df_train.loc[df_train['id'] == 2612,'budget'] = 15000000     # Field of D
df_train.loc[df_train['id'] == 2696,'budget'] = 10000000     # Nurse 3-D
df_train.loc[df_train['id'] == 2801,'budget'] = 10000000     # Fracture
df_test.loc[df_test['id'] == 3889,'budget'] = 15000000     # Colossal
df_test.loc[df_test['id'] == 6733,'budget'] = 5000000      # The Big Sick
df_test.loc[df_test['id'] == 3197,'budget'] = 8000000      # High-Rise
df_test.loc[df_test['id'] == 6683,'budget'] = 50000000     # The Pink Pan
df_test.loc[df_test['id'] == 5704,'budget'] = 4300000      # French Conne
df_test.loc[df_test['id'] == 6109,'budget'] = 281756       # Dogtooth
df_test.loc[df_test['id'] == 7242,'budget'] = 10000000     # Addams Famil
df_test.loc[df_test['id'] == 7021,'budget'] = 17540562     #  Two Is a Fa
df_test.loc[df_test['id'] == 5591,'budget'] = 4000000      # The Orphanag
df_test.loc[df_test['id'] == 4282,'budget'] = 20000000     # Big Top Pee-

power_six = df_train.id[df_train.budget > 1000][df_train.revenue < 100]

for k in power_six :
    df_train.loc[df_train['id'] == k,'revenue'] =  df_train.loc[df_train['i
```

10 EDA TMDB

In [18]:

```
df_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3000 entries, 0 to 2999
Data columns (total 32 columns):
id                          3000 non-null int64
belongs_to_collection       604 non-null object
budget                      3000 non-null int64
genres                      3000 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        3000 non-null object
production_countries        3000 non-null object
release_date                3000 non-null datetime64[ns]
runtime                     2998 non-null float64
spoken_languages            3000 non-null object
status                      3000 non-null object
tagline                     2403 non-null object
title                       3000 non-null object
Keywords                    3000 non-null object
cast                        3000 non-null object
crew                        3000 non-null object
revenue                     3000 non-null int64
Genres                      3000 non-null object
Production_countries        3000 non-null object
Spoken_languages            3000 non-null object
Production_companies        3000 non-null object
keywords                    3000 non-null object
Cast                        3000 non-null object
Crew                        3000 non-null object
firstGenres                 2993 non-null object
Production_countries_first  2945 non-null object
dtypes: datetime64[ns](1), float64(2), int64(3), object(26)
memory usage: 750.1+ KB
```

In [19]:

```
df_train.describe() #only 4 numerical features
```

Out[19]:

	id	budget	popularity	runtime	revenue
count	3000.000000	3.000000e+03	3000.000000	2998.000000	3.000000e+03
mean	1500.500000	2.270393e+07	8.463274	107.856571	6.672303e+07
std	866.169729	3.703865e+07	12.104000	22.086434	1.374996e+08
min	1.000000	0.000000e+00	0.000001	0.000000	1.000000e+00
25%	750.750000	0.000000e+00	4.018053	94.000000	2.437773e+06
50%	1500.500000	8.000000e+06	7.374861	104.000000	1.692863e+07
75%	2250.250000	3.000000e+07	10.890983	118.000000	6.877599e+07
max	3000.000000	3.800000e+08	294.337037	338.000000	1.519558e+09

In [20]:

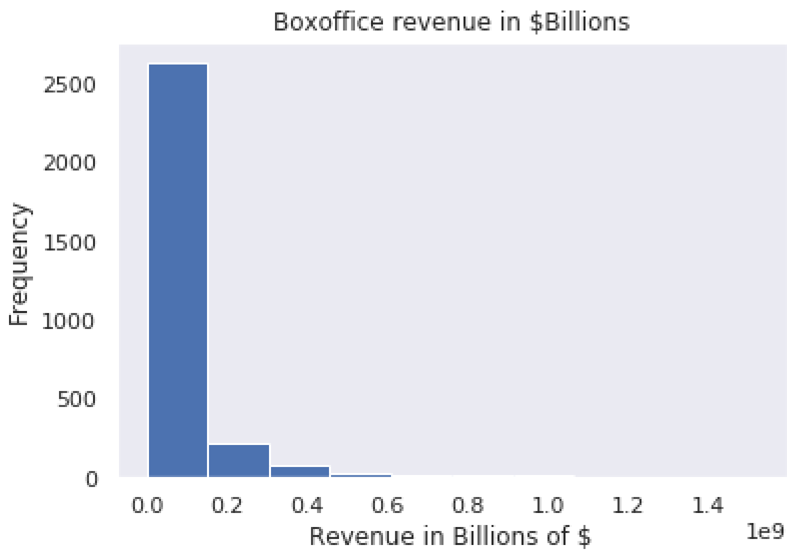
```
### determine the categorical and numerical features

numerical_ix = df_train.select_dtypes(include=['int64', 'float64']).columns
categorical_ix = df_train.select_dtypes(include=['object', 'bool']).columns
print(numerical_ix)
print(categorical_ix)
```

```
Index(['id', 'budget', 'popularity', 'runtime', 'revenue'], dtype='object')
Index(['belongs_to_collection', 'genres', 'homepage', 'imdb_id',
       'original_language', 'original_title', 'overview', 'poster_path',
       'production_companies', 'production_countries', 'spoken_languages',
       'status', 'tagline', 'title', 'Keywords', 'cast', 'crew', 'Genres',
       'Production_countries', 'Spoken_languages', 'Production_companies',
       'keywords', 'Cast', 'Crew', 'firstGenres',
       'Production_countries_first'],
      dtype='object')
```

10.0.1 Distribution of the target column

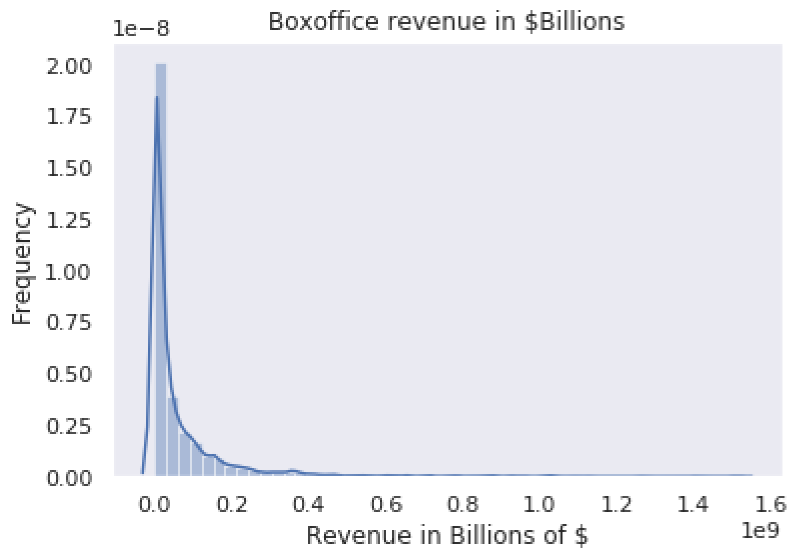
In [21]:

```
# necessary imports
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns; sns.set()
df_train['revenue'].astype(float).plot.hist()
plt.xlabel("Revenue in Billions of $")
plt.ylabel("Frequency")
plt.title("Boxoffice revenue in $Billions")
plt.grid()
```



In [22]:

```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns; sns.set()
sns.distplot(df_train['revenue'] )
plt.xlabel("Revenue in Billions of $")
plt.ylabel("Frequency")
plt.title("Boxoffice revenue in $Billions")
plt.grid()
```

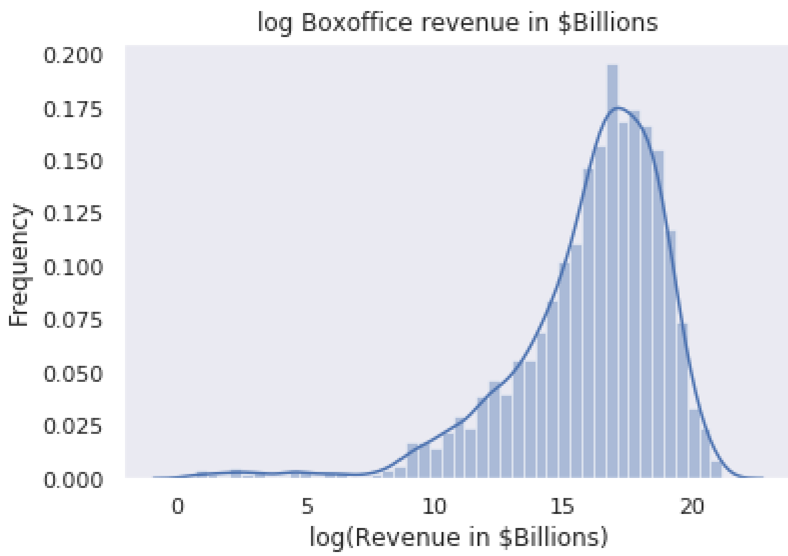


10.0.1.1 Take log of target variable (revenue)

Because revenue variable is skewed, let's calculate log of it.

In [23]:
```
df_train['logRevenue'] = np.log1p(df_train['revenue'])
sns.distplot(df_train['logRevenue'] )
plt.grid()
plt.xlabel("log(Revenue in $Billions)")
plt.ylabel("Frequency")
plt.title("log Boxoffice revenue in $Billions")
```

Out[23]:
```
Text(0.5, 1.0, 'log Boxoffice revenue in $Billions')
```



10.1 Genre and Revenue

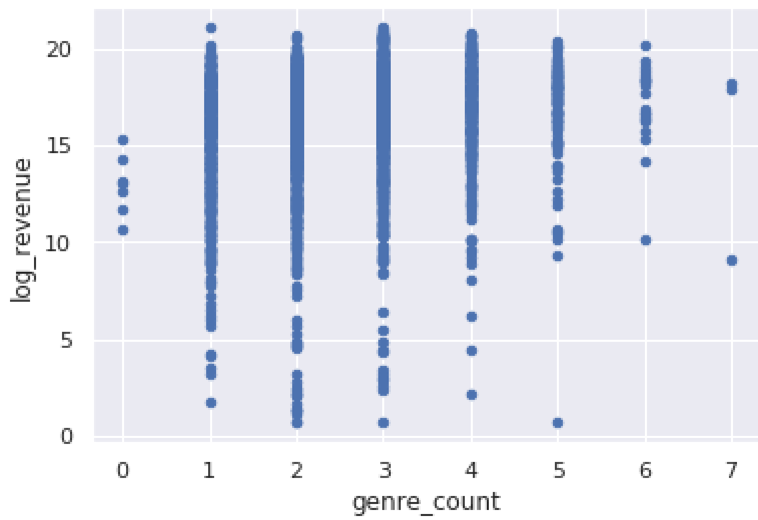
In [24]:

```
pd.DataFrame({'log_revenue': (np.log1p(df_train.revenue.values)),
              'genre_count': (df_train['genres'].apply(lambda x: len(x) if x
```

```
'c' argument looks like a single numeric RGB or RGBA sequence, which should
be avoided as value-mapping will have precedence in case its length matches
with 'x' & 'y'.  Please use a 2-D array with a single row if you really wan
t to specify the same RGB or RGBA value for all points.
```

Out[24]:

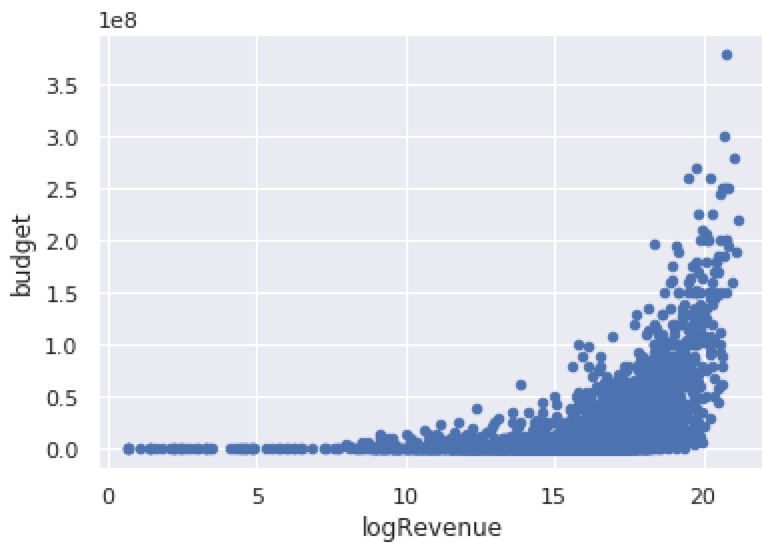
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8b75b89e10>
```

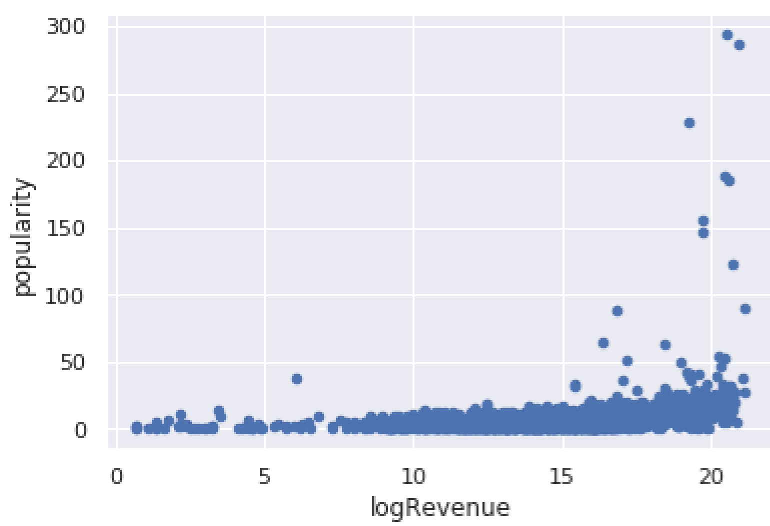


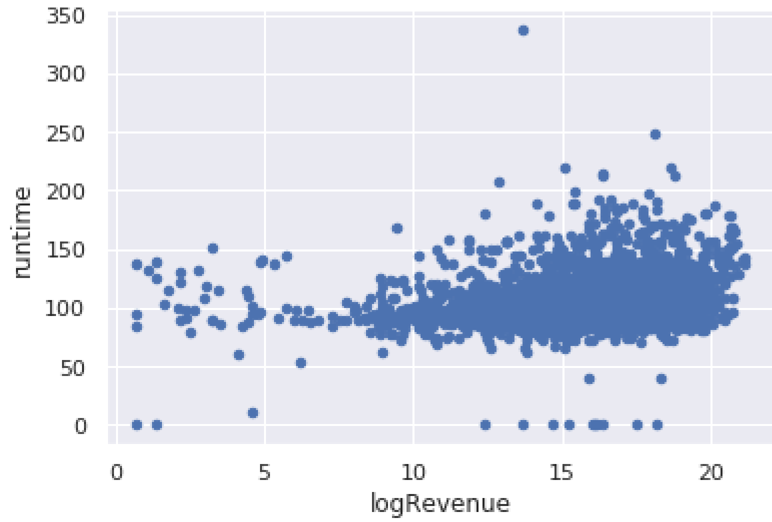
In [25]:
```
df_train.plot.scatter(x='logRevenue',y='budget')
df_train.plot.scatter(x='logRevenue',y='popularity')
df_train.plot.scatter(x='logRevenue',y='runtime')
```

```
'c' argument looks like a single numeric RGB or RGBA sequence, which should
be avoided as value-mapping will have precedence in case its length matches
with 'x' & 'y'.  Please use a 2-D array with a single row if you really wan
t to specify the same RGB or RGBA value for all points.
'c' argument looks like a single numeric RGB or RGBA sequence, which should
be avoided as value-mapping will have precedence in case its length matches
with 'x' & 'y'.  Please use a 2-D array with a single row if you really wan
t to specify the same RGB or RGBA value for all points.
'c' argument looks like a single numeric RGB or RGBA sequence, which should
be avoided as value-mapping will have precedence in case its length matches
with 'x' & 'y'.  Please use a 2-D array with a single row if you really wan
t to specify the same RGB or RGBA value for all points.
```

Out[25]: <matplotlib.axes._subplots.AxesSubplot at 0x7f8b75a85a50>

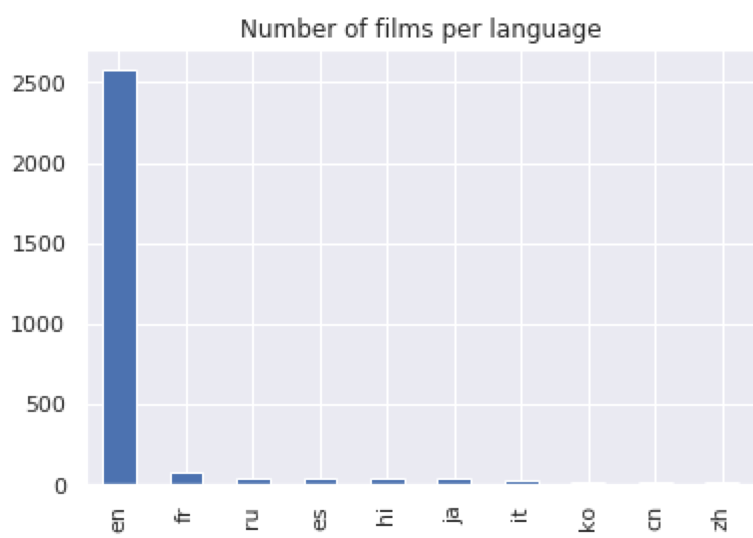






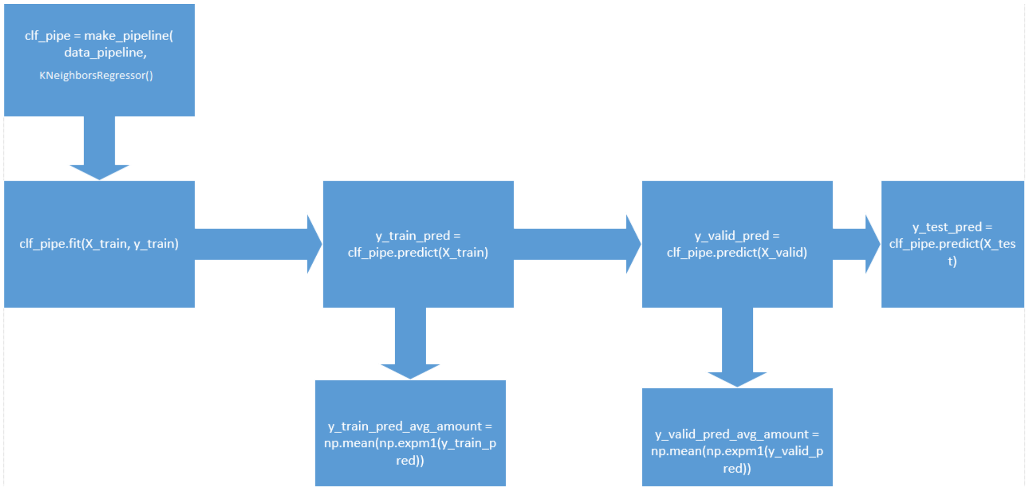
In [26]:
```
df_train.original_language.value_counts()[:10].plot.bar()
plt.title("Number of films per language")
```

Out[26]: Text(0.5, 1.0, 'Number of films per language')



11 Block Diagrams

A block diagram is a clear way to neatly outline your processes.



12 Metrics

In [27]:
```
def rmsle(y_true, y_pred):
    assert len(y_true) == len(y_pred)
    return np.sqrt(np.mean((np.log1p(y_pred) - np.log1p(y_true))**2))
```

In [28]:
```
def mape(y_true, y_pred):
    assert len(y_true) == len(y_pred)
    return np.mean(np.abs((y_pred-y_true)/y_true))*100
```

13 Baseline Model Raw Box Revenue

In [57]:

```
from sklearn.preprocessing import MinMaxScaler
from sklearn.neighbors import KNeighborsRegressor
df_train['Belongs_to_collection'] = (df_train["belongs_to_collection"].repl
df_train['Homepage'] = (df_train["homepage"].replace('([a-z]+)', 1, regex =

X = df_train.drop(['id', 'imdb_id','genres', "Genres",'belongs_to_collectio
y = df_train['revenue']

#split into train/valid/test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, ra
X_train, X_valid, y_train, y_valid = train_test_split(X_train, y_train, tes


print(f"X train          shape: {X_train.shape}")
print(f"X validation     shape: {X_valid.shape}")
print(f"X test           shape: {X_test.shape}")


#categorical/numeric features
numerical_ix = X.select_dtypes(include=['int64', 'float64']).columns
categorical_ix = X.select_dtypes(include=['object', 'bool']).columns
print(numerical_ix)
print(categorical_ix)

#pipelines
numerical_features = [
    'budget',
    'popularity'
]

num_pipeline =  Pipeline([
            ('impute', SimpleImputer(strategy='median')),
            ('std_scaler', MinMaxScaler(feature_range = (0,1)))
            ])


categorical_features = [
    'Belongs_to_collection',
    'Homepage',
    "original_language",
    "status",
    "firstGenres",
    "Production_countries_first",
    'keywords'
]


cat_values = [
    list(set(X["original_language"])), # language
    ['Released', 'Rumored'] # status
]

cat_pipeline = Pipeline([
        ('imputer', SimpleImputer(strategy='most_frequent')),
        ('ohe', OneHotEncoder(sparse=False, handle_unknown="ignore"))
    ])
```

```
date_feature = ['release_date']

date_pipeline = Pipeline([
        ('date_transformer',DateTransformer()),
        ('imputer', SimpleImputer(strategy='most_frequent')),
        ('ohe', OneHotEncoder(sparse=False, handle_unknown="ignore"))

])
count_features = ["production_companies", 'spoken_languages']

count_pipeline = Pipeline([
        ('count_transformer',countTransformer(count_features)),
        ('imputer', SimpleImputer(strategy='most_frequent'))
    ])



genre_feature = ["genres"]



data_pipeline = ColumnTransformer( transformers= [
        ("num_pipeline", num_pipeline, numerical_features),
        ("cat_pipeline", cat_pipeline, categorical_features),
        ('date_pipeline', date_pipeline, date_feature),
        ('count_pipeline', count_pipeline, count_features)
],
         remainder='drop',
        n_jobs=-1
    )

#dispaying pipeline results into a dataframe
X_train_transformed = data_pipeline.fit_transform(X_train)
column_names = numerical_features  + \
               list(data_pipeline.transformers_[1][1].named_steps["ohe"].ge
               list(data_pipeline.transformers_[2][1].named_steps["ohe"].ge
                count_features

display(pd.DataFrame(X_train_transformed,  columns=column_names).head())
number_of_inputs = X_train_transformed.shape[1]

clf_pipe = make_pipeline(
    data_pipeline,
    KNeighborsRegressor())


param_grid = {
    'kneighborsregressor__n_neighbors': list(range(1,6)),
    'kneighborsregressor__weights': ['uniform', 'distance'],
    'kneighborsregressor__algorithm': ['ball_tree', 'kd_tree', 'brute'],
    'kneighborsregressor__leaf_size': list(range(29,32)),
    'kneighborsregressor__p': list(range(1,4))
            }

grid = GridSearchCV(estimator=clf_pipe, param_grid=param_grid,
                    cv=3, scoring='neg_mean_squared_error' ,n_jobs=-1)
```

```
start = time()
grid.fit(X_train,y_train)
train_time = np.round(time() - start, 4)

print(grid.best_params_)

y_train_pred = grid.best_estimator_.predict(X_train)
y_valid_pred = grid.best_estimator_.predict(X_valid)
y_test_pred = grid.best_estimator_.predict(X_test)




# Time and score test predictions
start = time()
clf_pipe.fit(X_train, y_train)
train_time = np.round(time() - start, 4)

y_train_pred = clf_pipe.predict(X_train)
y_valid_pred = clf_pipe.predict(X_valid)
y_test_pred = clf_pipe.predict(X_test)

#Root mean square error
trainRMSLE  = rmsle(y_train,y_train_pred)
validRMSLE  = rmsle(y_valid,y_valid_pred)
start = time()
testRMSLE  = rmsle(y_test,y_test_pred)
test_time = np.round(time() - start, 4)

#average value over/under predicted
train_avg  = np.mean(y_train_pred-y_train)
valid_avg  = np.mean(y_valid_pred-y_valid)
test_avg  = np.mean(y_test_pred-y_test)

#mean absolute percentage error
trainMAPE  = mape(y_train,y_train_pred)
validMAPE  = mape(y_valid,y_valid_pred)
testMAPE  = mape(y_test,y_test_pred)


#del experimentLog
try: experimentLog
except : experimentLog = pd.DataFrame(columns=["Pipeline", "Dataset", "Trai
                                                "Train Time(s)",  "Test Time
experimentLog.loc[len(experimentLog)] =[f"Baseline Raw with {number_of_inpu
                                        f"{trainRMSLE:.4f}", f"{validRMSLE:
                                        train_time, test_time,
                                        "Baseline 1 pipeline"]



experimentLog
```

```
X train           shape: (1785, 18)
X validation      shape: (315, 18)
```

```
X test           shape: (900, 18)
Index(['budget', 'popularity', 'runtime', 'Belongs_to_collection', 'Home
page'], dtype='object')
Index(['original_language', 'production_companies', 'spoken_languages',
       'status', 'Production_countries', 'Spoken_languages',
       'Production_companies', 'keywords', 'Cast', 'Crew', 'firstGenre
s',
       'Production_countries_first'],
      dtype='object')
```

	budget	popularity	Belongs_to_collection_0.0	Belongs_to_collection_1.0	Homepage_0.0	H
0	0.315789	0.027460	1.0	0.0	0.0	
1	0.000000	0.021214	1.0	0.0	1.0	
2	0.039474	0.039126	1.0	0.0	0.0	
3	0.003947	0.001554	1.0	0.0	1.0	
4	0.002237	0.011610	1.0	0.0	1.0	

5 rows × 1705 columns

Out[57]:

	Pipeline	Dataset	TrainRMSLE	ValidRMSLE	TestRMSLE	trainAVG	validAV(
0	Baseline Log with 5306 inputs	IMDB dataset	0.0000	0.2693	0.2010	68487336.5652	28542669.737
1	Baseline Log with 3533 inputs	IMDB dataset	0.0000	0.2697	0.2009	68487336.5652	28573410.754
2	Baseline Kaggle with 8118 inputs	IMDB dataset	0.0000	0.2697	0.2009	68487336.5652	28573410.754
3	Baseline Log with 3535 inputs	IMDB dataset	0.0000	0.2672	0.2178	68487336.5652	24521071.281
4	Baseline Kaggle with 8120 inputs	IMDB dataset	0.0000	0.2672	0.2178	68487336.5652	24521071.281
5	Baseline Log with 3535 inputs	IMDB dataset	0.0000	0.2635	0.2016	68487336.5652	29780683.296
6	Baseline Kaggle with 8120 inputs	IMDB dataset	0.0000	0.2635	0.2016	68487336.5652	29780683.296

	Pipeline	Dataset	TrainRMSLE	ValidRMSLE	TestRMSLE	trainAVG	validAV
7	Baseline Log with 3535 inputs	IMDB dataset	0.0000	0.2456	0.1944	68487336.5652	32264681.758
8	Baseline Kaggle with 8120 inputs	IMDB dataset	0.0000	0.2456	0.1944	68487336.5652	32264681.758
9	Baseline Log with 3535 inputs	IMDB dataset	0.1811	0.2455	0.1950	0.0928	0.355
10	Baseline Kaggle with 8120 inputs	IMDB dataset	0.1811	0.2455	0.1950	0.0928	0.355
11	Baseline Raw with 1705 inputs	IMDB dataset	2.6900	3.2372	2.6844	-2450424.1109	-519284.796

14 Baseline Model Log Box Revenue

In [51]:

```
from sklearn.pipeline import Pipeline, FeatureUnion, make_pipeline
from sklearn.compose import ColumnTransformer
import numpy as np
import pandas as pd
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer, make_column_transformer
from sklearn.model_selection import train_test_split  # sklearn.cross_valid
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.linear_model import LinearRegression
from sklearn.impute import SimpleImputer
from time import time
from sklearn.metrics import mean_squared_log_error
from sklearn.neighbors import KNeighborsRegressor


df_train['logRevenue'] = np.log1p(df_train['revenue'])

X = df_train.drop(['id','imdb_id', 'genres', 'homepage', 'tagline','belongs
y = df_train['logRevenue']

#split into train/valid/test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, ra
X_train, X_valid, y_train, y_valid = train_test_split(X_train, y_train, tes


print(f"X train           shape: {X_train.shape}")
print(f"X validation      shape: {X_valid.shape}")
print(f"X test            shape: {X_test.shape}")


#categorical/numeric features
numerical_ix = X.select_dtypes(include=['int64', 'float64']).columns
categorical_ix = X.select_dtypes(include=['object', 'bool']).columns
print(numerical_ix)
print(categorical_ix)

#pipelines
numerical_features = [
    'runtime'
]

num_pipeline =  Pipeline([
            ('impute', SimpleImputer(strategy='median')),
            ('std_scaler', StandardScaler())
            ])
log_features = [
    'budget',
    'popularity'
]

log_pipeline =Pipeline([
            ('log_transformer', logTransformer(log_features)),
            ('impute', SimpleImputer(strategy='median')),
            ('std_scaler', StandardScaler())
            ])
```

```
categorical_features = [
    'Belongs_to_collection',
    'Homepage',
    "original_language",
    "status",
    "Spoken_languages",
    "Production_countries",
    'Production_companies',
    'keywords'
]


cat_values = [
    list(set(X["original_language"])), # language
    ['logReleased', 'Rumored'] # status
]

cat_pipeline = Pipeline([
        ('imputer', SimpleImputer(strategy='most_frequent')),
        ('ohe', OneHotEncoder(sparse=False, handle_unknown="ignore"))
    ])

date_feature = ['release_date']

date_pipeline = Pipeline([
        ('date_transformer',DateTransformer()),
        ('imputer', SimpleImputer(strategy='most_frequent')),
        ('ohe', OneHotEncoder(sparse=False, handle_unknown="ignore"))

])

genre_feature = ["genres"]
count_features = [  'cast', 'crew']

count_pipeline = Pipeline([
        ('count_transformer',countTransformer(count_features)),
        ('imputer', SimpleImputer(strategy='median')),
        ('std_scaler', StandardScaler())
   ])




data_pipeline = ColumnTransformer( transformers= [
        ("num", num_pipeline, numerical_features),
        ("cat_pipeline", cat_pipeline, categorical_features),
        ('date_pipeline', date_pipeline, date_feature),
        ('log_transformer', log_pipeline, log_features),
        ('count_pipeline', count_pipeline, count_features)
],
         remainder='drop',
        n_jobs=-1
    )

X_train_transformed = data_pipeline.fit_transform(X_train)
column_names = numerical_features  + \
```

```
               list(data_pipeline.transformers_[1][1].named_steps["ohe"].ge
                list(data_pipeline.transformers_[2][1].named_steps["ohe"].g
                log_features +\
                count_features

display(pd.DataFrame(X_train_transformed,  columns=column_names).head())
number_of_inputs = X_train_transformed.shape[1]

clf_pipe = make_pipeline(
    data_pipeline,
    KNeighborsRegressor())


param_grid = {
    'kneighborsregressor__n_neighbors': list(range(1,6)),
    'kneighborsregressor__weights': ['uniform', 'distance'],
    'kneighborsregressor__algorithm': ['ball_tree', 'kd_tree', 'brute'],
    'kneighborsregressor__leaf_size': list(range(29,32)),
    'kneighborsregressor__p': list(range(1,4))
            }

grid = GridSearchCV(estimator=clf_pipe, param_grid=param_grid,
                    cv=3, scoring='neg_mean_squared_error' ,n_jobs=-1)

start = time()
grid.fit(X_train,y_train)
train_time = np.round(time() - start, 4)

print(grid.best_params_)

y_train_pred = grid.best_estimator_.predict(X_train)
y_valid_pred = grid.best_estimator_.predict(X_valid)
y_test_pred = grid.best_estimator_.predict(X_test)




# Time and score test predictions
start = time()
clf_pipe.fit(X_train, y_train)
train_time = np.round(time() - start, 4)

y_train_pred = clf_pipe.predict(X_train)
y_valid_pred = clf_pipe.predict(X_valid)
y_test_pred = clf_pipe.predict(X_test)

#Root mean square error
trainRMSLE  = rmsle(y_train,y_train_pred)
validRMSLE  = rmsle(y_valid,y_valid_pred)
start = time()
testRMSLE  = rmsle(y_test,y_test_pred)
test_time = np.round(time() - start, 4)

#average value over/under predicted
train_avg  = np.mean(y_train_pred-y_train)
valid_avg  = np.mean(y_valid_pred-y_valid)
test_avg  = np.mean(y_test_pred-y_test)
```

```
#mean absolute percentage error
trainMAPE  = mape(y_train,y_train_pred)
validMAPE  = mape(y_valid,y_valid_pred)
testMAPE  = mape(y_test,y_test_pred)

#del experimentLog
try: experimentLog
except : experimentLog = pd.DataFrame(columns=["Pipeline", "Dataset", "Trai
                                               "Train Time(s)",  "Test Time
experimentLog.loc[len(experimentLog)] =[f"Baseline Log with {number_of_inpu
                                        f"{trainRMSLE:.4f}", f"{validRMSLE:
                                        train_time, test_time,
                                        "Baseline 1 pipeline"]


experimentLog
```

```
X train          shape: (1785, 23)
X validation     shape: (315, 23)
X test           shape: (900, 23)
Index(['budget', 'popularity', 'runtime', 'Belongs_to_collection', 'Homepag
e'], dtype='object')
Index(['original_language', 'production_companies', 'production_countries',
       'spoken_languages', 'status', 'Keywords', 'cast', 'crew', 'Genres',
       'Production_countries', 'Spoken_languages', 'Production_companies',
       'keywords', 'Cast', 'Crew', 'firstGenres',
       'Production_countries_first'],
      dtype='object')
```

	runtime	Belongs_to_collection_0.0	Belongs_to_collection_1.0	Homepage_0.0	Homepage_1
0	0.520813	1.0	0.0	0.0	1
1	-0.124874	1.0	0.0	1.0	0
2	-0.032633	1.0	0.0	0.0	1
3	2.503996	1.0	0.0	1.0	0
4	-0.032633	1.0	0.0	1.0	0

5 rows × 3535 columns

```
{'kneighborsregressor__algorithm': 'ball_tree', 'kneighborsregressor__leaf_
size': 29, 'kneighborsregressor__n_neighbors': 5, 'kneighborsregressor__p':
1, 'kneighborsregressor__weights': 'uniform'}
```

Out[51]:

	Pipeline	Dataset	TrainRMSLE	ValidRMSLE	TestRMSLE	trainAVG	validAVG	
0	Baseline Log with 5306 inputs	IMDB dataset	0.0000	0.2693	0.2010	68487336.5652	28542669.7377	40

	Pipeline	Dataset	TrainRMSLE	ValidRMSLE	TestRMSLE	trainAVG	validAVG	
1	Baseline Log with 3533 inputs	IMDB dataset	0.0000	0.2697	0.2009	68487336.5652	28573410.7547	40
2	Baseline Kaggle with 8118 inputs	IMDB dataset	0.0000	0.2697	0.2009	68487336.5652	28573410.7547	40
3	Baseline Log with 3535 inputs	IMDB dataset	0.0000	0.2672	0.2178	68487336.5652	24521071.2819	30
4	Baseline Kaggle with 8120 inputs	IMDB dataset	0.0000	0.2672	0.2178	68487336.5652	24521071.2819	30
5	Baseline Log with 3535 inputs	IMDB dataset	0.0000	0.2635	0.2016	68487336.5652	29780683.2967	38
6	Baseline Kaggle with 8120 inputs	IMDB dataset	0.0000	0.2635	0.2016	68487336.5652	29780683.2967	38
7	Baseline Log with 3535 inputs	IMDB dataset	0.0000	0.2456	0.1944	68487336.5652	32264681.7587	43
8	Baseline Kaggle with 8120 inputs	IMDB dataset	0.0000	0.2456	0.1944	68487336.5652	32264681.7587	43
9	Baseline Log with 3535 inputs	IMDB dataset	0.1811	0.2455	0.1950	0.0928	0.3559	

15 Kaggle Submission

In [54]:

```
from sklearn.pipeline import Pipeline, FeatureUnion, make_pipeline
from sklearn.compose import ColumnTransformer
import numpy as np
import pandas as pd
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer, make_column_transformer
from sklearn.model_selection import train_test_split  # sklearn.cross_valid
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.linear_model import LinearRegression
from sklearn.impute import SimpleImputer
from time import time
from sklearn.metrics import mean_squared_log_error
from sklearn.linear_model import LogisticRegression

df_test2 =pd.DataFrame(df_test, columns=df_test.columns)

#df_test2['genres'] = df_test2['genres'].str.extract('([A-Z][a-z]+)')

df_test['Belongs_to_collection'] = (df_test["belongs_to_collection"].replac
df_test['Homepage'] = (df_test["homepage"].replace('([a-z]+)', 1, regex = T


X_test_orig = df_test.drop(['id','genres', 'imdb_id', 'belongs_to_collectio
X_test = df_test.drop(['id', 'imdb_id', 'genres', 'belongs_to_collection','


numerical_ix = X_test.select_dtypes(include=['int64', 'float64']).columns
categorical_ix = X_test.select_dtypes(include=['object', 'bool']).columns
print(numerical_ix)
print(categorical_ix)

numerical_features = [
    'runtime'
]

num_pipeline =  Pipeline([
            ('impute', SimpleImputer(strategy='median')),
            ('std_scaler',MinMaxScaler(feature_range = (0,1)))
            ])


categorical_features = [
    'Belongs_to_collection',
    'Homepage',
    "original_language",
    "status",
    "Spoken_languages",
    "Production_countries",
    "Production_companies",
    'keywords'
]


cat_values = [
    list(set(X_test["original_language"])),# language
    ['Released', 'Rumored'] # status
```

```
]

cat_pipeline = Pipeline([
        ('imputer', SimpleImputer(strategy='most_frequent')),
        ('ohe', OneHotEncoder(sparse=False, handle_unknown="ignore"))
    ])

date_feature = ['release_date']

date_pipeline = Pipeline([
        ('date_transformer',DateTransformer()),
        ('imputer', SimpleImputer(strategy='most_frequent')),
        ('ohe', OneHotEncoder(sparse=False, handle_unknown="ignore"))

])
log_features = [
    'budget',
    'popularity'
]

log_pipeline =Pipeline([
            ('log_transformer', logTransformer(log_features)),
            ('impute', SimpleImputer(strategy='median')),
            ('std_scaler', StandardScaler())
            ])

genre_feature = ["genres"]

count_features = [ 'cast', 'crew']

count_pipeline = Pipeline([
        ('count_transformer',countTransformer(count_features)),
        ('imputer', SimpleImputer(strategy='most_frequent')),
        ('Std Scaler', StandardScaler())
    ])

data_pipeline = ColumnTransformer( transformers= [
        ("num", num_pipeline, numerical_features),
        ("cat_pipeline", cat_pipeline, categorical_features),
        ('date_pipeline',date_pipeline, date_feature),
        ('log_pipeline', log_pipeline, log_features),
        ('count_pipeline', count_pipeline, count_features)
],
         remainder='drop',
        n_jobs=-1
    )

X_train_transformed = data_pipeline.fit_transform(X_test)
column_names = numerical_features  + \
              list(data_pipeline.transformers_[1][1].named_steps["ohe"].ge
              list(data_pipeline.transformers_[2][1].named_steps["ohe"].ge
              log_features +\
               count_features

display(pd.DataFrame(X_train_transformed,  columns=column_names).head())
number_of_inputs = X_train_transformed.shape[1]
```

```python
clf_pipe = make_pipeline(
    data_pipeline,
    KNeighborsRegressor(n_neighbors=5, weights='uniform', algorithm='ball_t
                        p=1))


param_grid = {
    'kneighborsregressor__n_neighbors': list(range(1,6)),
    'kneighborsregressor__weights': ['uniform', 'distance'],
    'kneighborsregressor__algorithm': ['ball_tree', 'kd_tree', 'brute'],
    'kneighborsregressor__leaf_size': list(range(29,32)),
    'kneighborsregressor__p': list(range(1,4))
            }

grid = GridSearchCV(estimator=clf_pipe, param_grid=param_grid,
                  cv=3, scoring='neg_mean_squared_error' ,n_jobs=-1)

start = time()
grid.fit(X_train,y_train)
train_time = np.round(time() - start, 4)

print(grid.best_params_)


y_test_pred = grid.best_estimator_.predict(X_test)




# # Time and score test predictions
# start = time()
# clf_pipe.fit(X_train, y_train)
# train_time = np.round(time() - start, 4)

# # Time and score test predictions
# start = time()
# clf_pipe.fit(X_train,y_train)
# train_time = np.round(time() - start, 4)

#y_test_pred = clf_pipe.predict(X_test)
#print(y_test_pred)




#del experimentLog
try: experimentLog
except : experimentLog = pd.DataFrame(columns=["Pipeline", "Dataset", "Trai
                                               "Train Time(s)",  "Test Time
experimentLog.loc[len(experimentLog)] =[f"Baseline Kaggle with {number_of_i
                                        f"{trainRMSLE:.4f}", f"{validRMSLE:
                                        train_time, test_time,
                                        "Baseline 1 pipeline"]


experimentLog
```

```
Index(['budget', 'popularity', 'runtime', 'Belongs_to_collection', 'Homepag
e'], dtype='object')
Index(['original_language', 'production_companies', 'production_countries',
       'spoken_languages', 'status', 'Keywords', 'cast', 'crew', 'Genres',
       'Production_countries', 'Spoken_languages', 'Production_companies',
       'keywords', 'Cast', 'Crew', 'firstGenres',
       'Production_countries_first'],
      dtype='object')
```

	runtime	Belongs_to_collection_0.0	Belongs_to_collection_1.0	Homepage_0.0	Homepage_1.
0	0.281250	0.0	1.0	0.0	1.
1	0.203125	1.0	0.0	1.0	0.
2	0.312500	1.0	0.0	1.0	0.
3	0.406250	1.0	0.0	0.0	1.
4	0.287500	1.0	0.0	1.0	0.

5 rows × 8120 columns

Out[54]:

	Pipeline	Dataset	TrainRMSLE	ValidRMSLE	TestRMSLE	trainAVG	validAVG	
0	Baseline Log with 5306 inputs	IMDB dataset	0.0000	0.2693	0.2010	68487336.5652	28542669.7377	4
1	Baseline Log with 3533 inputs	IMDB dataset	0.0000	0.2697	0.2009	68487336.5652	28573410.7547	4
2	Baseline Kaggle with 8118 inputs	IMDB dataset	0.0000	0.2697	0.2009	68487336.5652	28573410.7547	4
3	Baseline Log with 3535 inputs	IMDB dataset	0.0000	0.2672	0.2178	68487336.5652	24521071.2819	3
4	Baseline Kaggle with 8120 inputs	IMDB dataset	0.0000	0.2672	0.2178	68487336.5652	24521071.2819	3
5	Baseline Log with 3535 inputs	IMDB dataset	0.0000	0.2635	0.2016	68487336.5652	29780683.2967	3

	Pipeline	Dataset	TrainRMSLE	ValidRMSLE	TestRMSLE	trainAVG	validAVG	
6	Baseline Kaggle with 8120 inputs	IMDB dataset	0.0000	0.2635	0.2016	68487336.5652	29780683.2967	3
7	Baseline Log with 3535 inputs	IMDB dataset	0.0000	0.2456	0.1944	68487336.5652	32264681.7587	4
8	Baseline Kaggle with 8120 inputs	IMDB dataset	0.0000	0.2456	0.1944	68487336.5652	32264681.7587	4
9	Baseline Log with 3535 inputs	IMDB dataset	0.1811	0.2455	0.1950	0.0928	0.3559	
10	Baseline Kaggle with 8120 inputs	IMDB dataset	0.1811	0.2455	0.1950	0.0928	0.3559	

In [55]:

```
y_test_pred = np.expm1((y_test_pred))


#np.delete(y_test_pred, np.isinf(y_test_pred) == True)
sub_test = df_test.assign(revenue=y_test_pred)
print(y_test_pred)
print(y_test_pred.max())
print(y_test_pred.min())
sub_test['revenue']


# 2.) Extract a table of ids and their revenue predictions
sub_test_y = sub_test[['id','revenue']].set_index('id')

# 3.) save that table to a csv file. On Kaggle, the file will be visible in
sub_test_y.to_csv("submission.csv")

# 4.) output the head of our file her to check if it looks good :)
pd.read_csv("submission.csv").head()
```

```
[14647943.82513129 11970380.18077584  5317216.47314081 ...
 60219206.29981972 39641235.9689453    416544.25455275]
886355526.1972706
990.539283005745
```

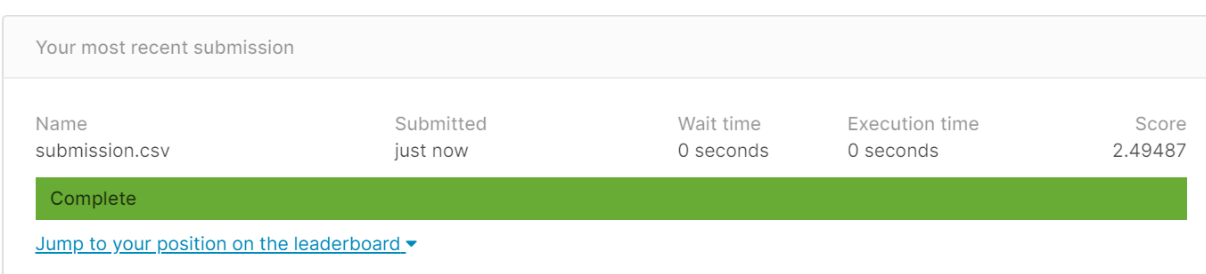
Out[55]:

	id	revenue
0	3001	1.464794e+07
1	3002	1.197038e+07
2	3003	5.317216e+06
3	3004	1.626683e+06
4	3005	2.033833e+05

Make sure this cell is only run once, otherwise it becomes an overflow error!

15.1 Conclusion: Evaluation, Discussion, and Analysis

Our Kaggle score on submission 3 is: 2.49487, placing us slightly higher on the Kaggle leaderboard than before. This score would put us at about 900 out of 1395 on the leaderboard.



Our phase 2 score was 2.53479, so our model improved slightly in Phase 3. When we learned that the linear regression we were using previously to predict the model so we switched to using KNN to predict the revenues for the movies. If we had more time, we would learn better ways to engineer the Phase 3 features for modeling, and run a longer GridSearch on different models and hyperparameters.