

Table of Contents

- ▼ [1 Team Information](#)
 - [1.1 Title: TMDb Box Office Revenue Prediction](#)
 - [1.2 Group number 2](#)
 - [1.3 Alex Bzdel - \[abzdel@bryant.edu\]\(mailto:abzdel@bryant.edu\) \(<mailto:abzdel@bryant.edu>\)](#)
 - [1.4 Michael Fornal - \[mforal@bryant.edu\]\(mailto:mforal@bryant.edu\) \(<mailto:mforal@bryant.edu>\)](#)
 - [1.5 Eric Gilbert - \[egilbert@bryant.edu\]\(mailto:egilbert@bryant.edu\) \(<mailto:egilbert@bryant.edu>\)](#)
- [2 Abstract](#)
- [3 Project Description \(data and tasks\)](#)
- [4 Data Import & notebook preparation](#)
- ▼ [5 EDA TMDb](#)
 - ▼ [5.0.1 Distribution of the target column](#)
 - [5.0.1.1 Take log of target variable \(revenue\)](#)
- ▼ [6 Preprocessing](#)
 - ▼ [6.1 Genre, Belongs to Collection and Homepage Binarization](#)
 - [6.1.1 Production Companies](#)
 - [6.2 Keywords, cast, crew](#)
 - [6.3 Fixing Release Date](#)
- [7 Pipeline](#)
- [8 Modeling](#)
- ▼ [9 Evaluation, reporting and analysis](#)
 - [9.1 Kaggle Submission](#)
- [10 Discussion](#)
- [11 Conclusion](#)

1 Team Information

1.1 Title: TMDb Box Office Revenue Prediction

1.2 Group number 2

1.3 Alex Bzdel - abzdel@bryant.edu (<mailto:abzdel@bryant.edu>)



**1.4 Michael Fornal - mfornal@bryant.edu
(<mailto:mfornal@bryant.edu>)**



**1.5 Eric Gilbert - egilbert@bryant.edu
(<mailto:egilbert@bryant.edu>)**



2 Abstract

In phase three of our project, we are improving on our baseline pipeline from phase 2 by introducing the following new features.

- `hash_feats = ['production_countries_count', 'production_companies_count', 'spoken_languages_count', 'keyword_count', 'cast_count', 'crew_count']`

The main goal of this phase is to optimize our pipeline to produce the best Kaggle score using our additional features. We did this by searching for the best regression model and parameters that will optimally score our model. We found that the optimal regression model is XGB and, when fit with our pipeline, we received a private score of 2.13772 on Kaggle.

3 Project Description (data and tasks)

We are working with data from the TMDb box office, which describes a number of categorical features (eg. `release_year`, `spoken_languages`) and numeric features (eg. `budget`, `runtime`, `homepage`). Additionally we will be adding new hash features into our pipeline (eg. `production_companies`, `languages`). We will use these features to predict the revenue of each movie in the test set. The challenge here is to predict the worldwide revenue for 4,398 movies in the test file given various information about the movie. The tasks to be completed in phase 3 require adding the hash features of production countries, spoken languages, keywords, along with the cast and crew. We also plan to integrate a decision tree regression model (xgboost) to help improve our score and accuracy.

```
In [1]: ▶ DATA_DIR = "./tmdb-box-office-prediction"  #same level as course repo in the
#DATA_DIR = os.path.join('./dddd/')
!mkdir $DATA_DIR
```

```
mkdir: cannot create directory './tmdb-box-office-prediction': File exists
```

```
In [2]: ▶ !ls -l $DATA_DIR
```

```
total 0
-rwxr-xr-x 1 root root 139134 Mar 27 2019 TestAdditionalFeatures.csv
-rwxr-xr-x 1 root root 94918 Mar 27 2019 TrainAdditionalFeatures.csv
-rwxr-xr-x 1 root root 61585 Feb 7 2019 sample_submission.csv
-rwxr-xr-x 1 root root 41868556 Feb 7 2019 test.csv
-rwxr-xr-x 1 root root 28311747 Feb 7 2019 train.csv
```

```
In [3]: ▶ # ! kaggle competitions download tmdb-box-office-prediction -p $DATA_DIR
```

4 Data Import & notebook preparation

```

In [4]: ▶ from sklearn.pipeline import Pipeline, FeatureUnion, make_pipeline
from sklearn.compose import ColumnTransformer
from sklearn.model_selection import train_test_split
import seaborn as sns
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split # sklearn.cross_validation
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.linear_model import LogisticRegression, LinearRegression
from sklearn.impute import SimpleImputer
from sklearn.base import BaseEstimator, TransformerMixin
from time import time
import xgboost as xgb
from sklearn.metrics import mean_squared_error

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 of
ds_name = 'train'

df_train = load_data(os.path.join(DATA_DIR, f'{ds_name}.csv'), ds_name)
#datasets[ds_name] = trainOrig

ds_name = 'test'
df_test = load_data(os.path.join(DATA_DIR, f'{ds_name}.csv'), ds_name)
#datasets[ds_name] = testOrig

```

```

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

```
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]
```

5 EDA TMDb

```
In [6]: ▶ df_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: 539.2+ KB
```

In [7]: `df_train.describe()` *#only 4 numerical features*

Out[7]:

	id	budget	popularity	runtime	revenue
count	3000.000000	3.000000e+03	3000.000000	2998.000000	3.000000e+03
mean	1500.500000	2.253133e+07	8.463274	107.856571	6.672585e+07
std	866.169729	3.702609e+07	12.104000	22.086434	1.375323e+08
min	1.000000	0.000000e+00	0.000001	0.000000	1.000000e+00
25%	750.750000	0.000000e+00	4.018053	94.000000	2.379808e+06
50%	1500.500000	8.000000e+06	7.374861	104.000000	1.680707e+07
75%	2250.250000	2.900000e+07	10.890983	118.000000	6.891920e+07
max	3000.000000	3.800000e+08	294.337037	338.000000	1.519558e+09

In [10]: `df_train.describe(include='all')` *#Look at all categorical and numerical*

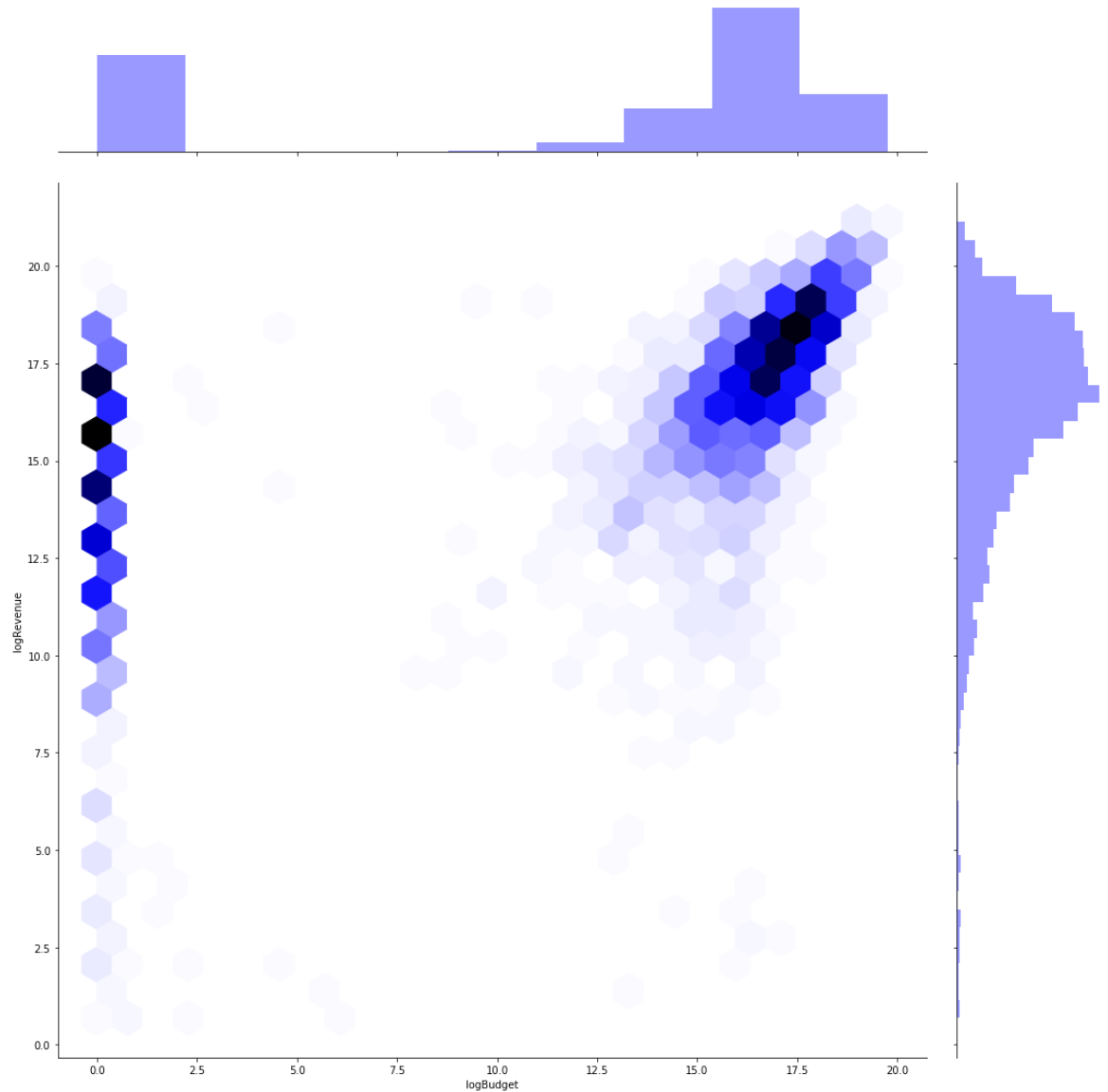
Out[10]:

	id	belongs_to_collection	budget	genres	homepage
count	3000.000000		604	3.000000e+03	2993
unique	NaN		422	NaN	872
top	NaN	[[{'id': 645, 'name': 'James Bond Collection', ...	NaN	[[{'id': 18, 'name': 'Drama'}]]	http://www.transformersmovie.com
freq	NaN		16	NaN	266
mean	1500.500000		NaN	2.253133e+07	NaN
std	866.169729		NaN	3.702609e+07	NaN
min	1.000000		NaN	0.000000e+00	NaN
25%	750.750000		NaN	0.000000e+00	NaN
50%	1500.500000		NaN	8.000000e+06	NaN
75%	2250.250000		NaN	2.900000e+07	NaN
max	3000.000000		NaN	3.800000e+08	NaN

11 rows × 23 columns

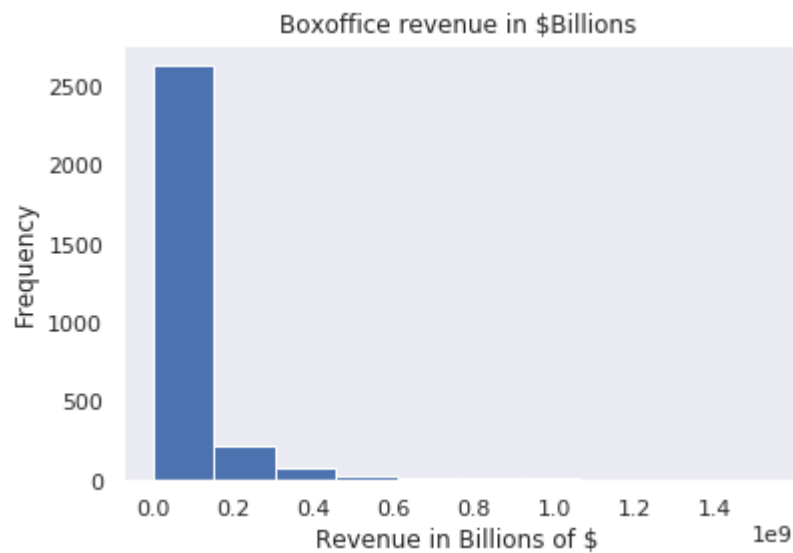
```
In [7]: ▶ df_train['logRevenue']=np.log1p(df_train['revenue'])  
df_train['logBudget']=np.log1p(df_train['budget'])  
sns.jointplot(x="logBudget", y="logRevenue", data=df_train, height=15, ratio=
```

Out[7]: <seaborn.axisgrid.JointGrid at 0x7f24318fb150>

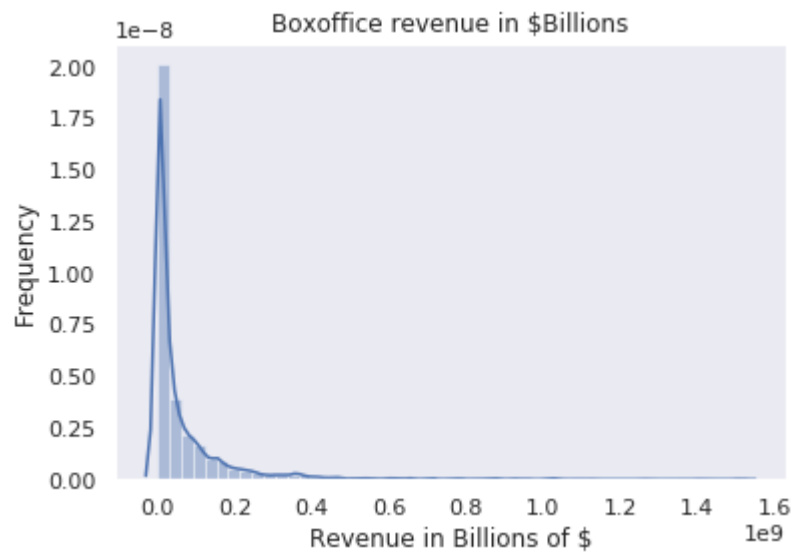


5.0.1 Distribution of the target column

```
In [8]: # 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 [9]: %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()
```

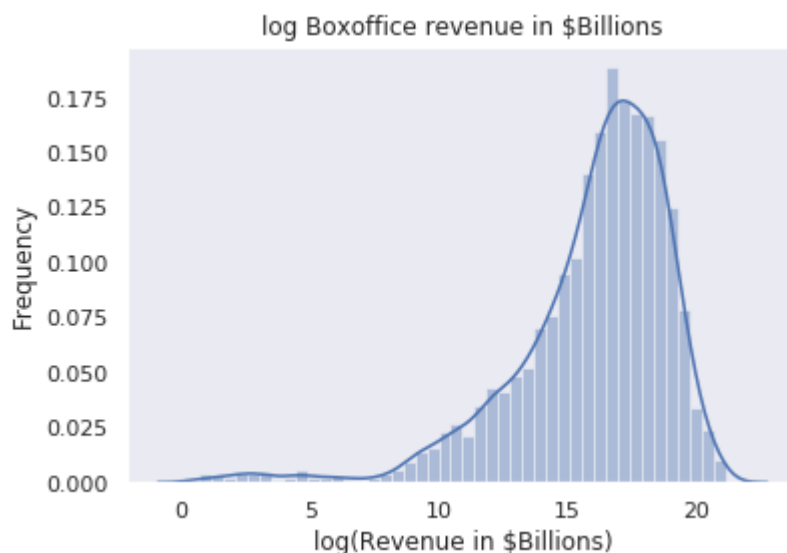


5.0.1.1 Take log of target variable (revenue)

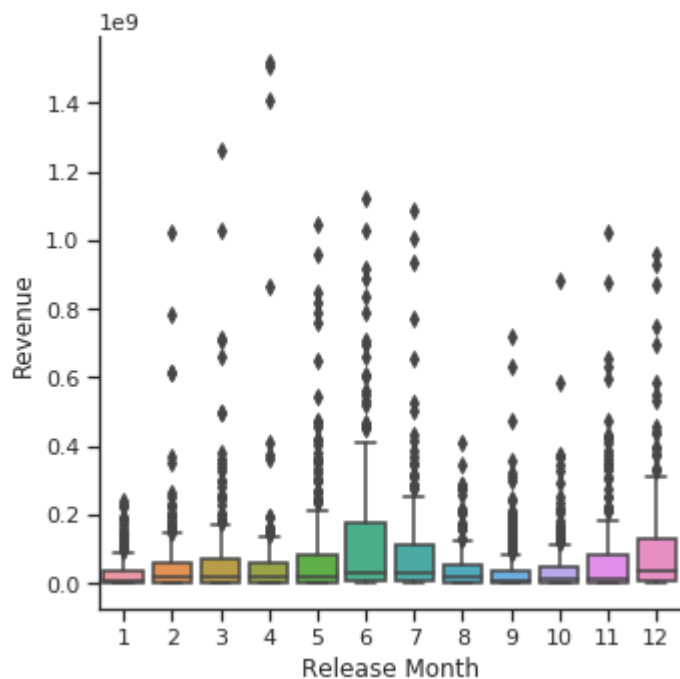
Because revenue variable is skewed, let's calculate log of it.

```
In [10]: ▶ 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")
```

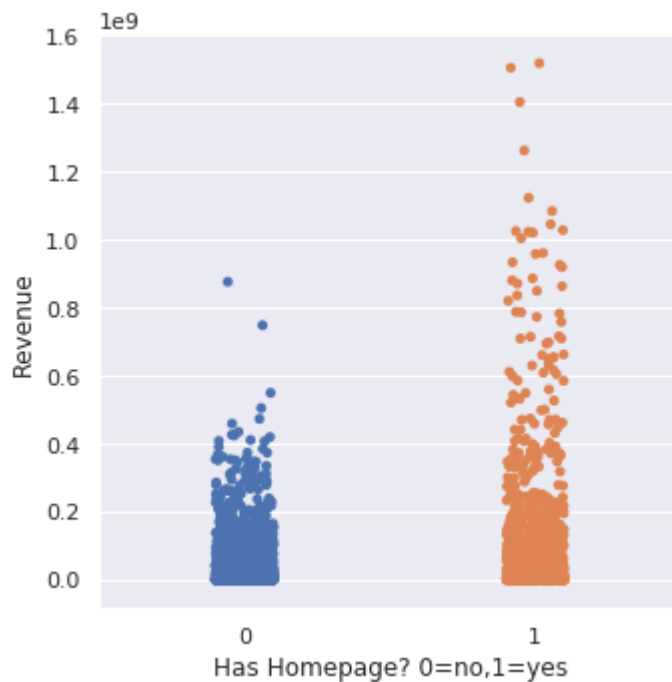
Out[10]: Text(0.5, 1.0, 'log Boxoffice revenue in \$Billions')



```
In [61]: ▶ #Run cells below before running
with sns.axes_style(style='ticks'):
    g = sns.catplot("release_month", "revenue", data=df_train, kind="box")
    g.set_axis_labels("Release Month", "Revenue");
```

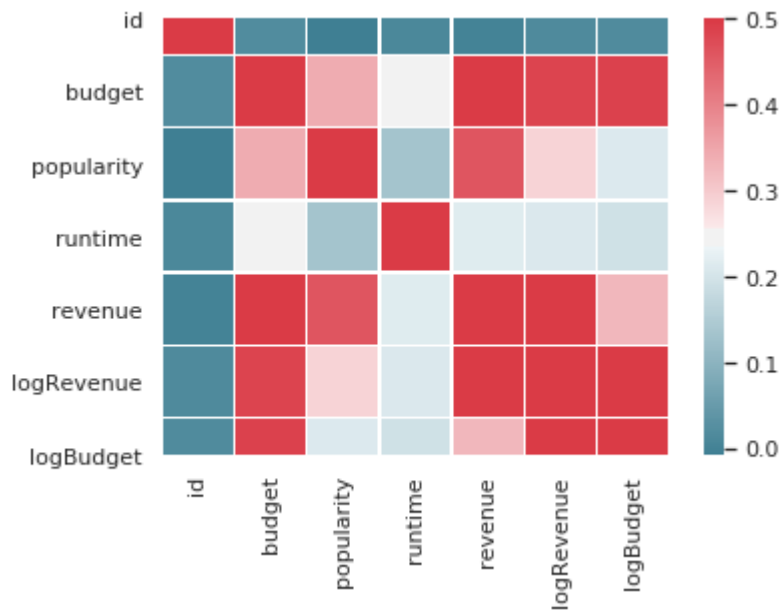


```
In [59]: #Run cells below before running
g=sns.catplot(x="has_homepage", y="revenue", data=df_train);
g.set_axis_labels("Has Homepage? 0=no,1=yes", "Revenue");
```



```
In [14]: corr=df_train.corr()
cmap = sns.diverging_palette(220, 10, as_cmap=True)
sns.heatmap(corr, cmap=cmap, vmax=.5, square=True, linewidths=.5)
```

Out[14]: <matplotlib.axes._subplots.AxesSubplot at 0x7f242f0d1990>



6 Preprocessing

```

In [5]: df_train['budget'].fillna('median')
df_train['popularity'].fillna('median')
df_train['runtime'].fillna('median')

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 Cookout
df_train.loc[df_train['id'] == 451, 'revenue'] = 12000000 # Chasing Libe
df_train.loc[df_train['id'] == 464, 'budget'] = 20000000 # Parenthood
df_train.loc[df_train['id'] == 470, 'budget'] = 13000000 # The Karate K
df_train.loc[df_train['id'] == 513, 'budget'] = 930000 # From Prada t
df_train.loc[df_train['id'] == 797, 'budget'] = 8000000 # Welcome to D
df_train.loc[df_train['id'] == 819, 'budget'] = 90000000 # Alvin and th
df_train.loc[df_train['id'] == 850, 'budget'] = 90000000 # Modern Times
df_train.loc[df_train['id'] == 1112, 'budget'] = 7500000 # An Officer c
df_train.loc[df_train['id'] == 1131, 'budget'] = 4300000 # Smokey and t
df_train.loc[df_train['id'] == 1359, 'budget'] = 10000000 # Stir Crazy
df_train.loc[df_train['id'] == 1542, 'budget'] = 1 # All at Once
df_train.loc[df_train['id'] == 1570, 'budget'] = 15800000 # Crocodile Du
df_train.loc[df_train['id'] == 1571, 'budget'] = 4000000 # Lady and the
df_train.loc[df_train['id'] == 1714, 'budget'] = 46000000 # The Recruit
df_train.loc[df_train['id'] == 1721, 'budget'] = 17500000 # Cocoon
df_train.loc[df_train['id'] == 1865, 'revenue'] = 25000000 # Scooby-Doo 2
df_train.loc[df_train['id'] == 2268, 'budget'] = 17500000 # Madea Goes t
df_train.loc[df_train['id'] == 2491, 'revenue'] = 6800000 # Never Talk t
df_train.loc[df_train['id'] == 2602, 'budget'] = 31000000 # Mr. Holland
df_train.loc[df_train['id'] == 2612, 'budget'] = 15000000 # Field of Dre
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 Panth
df_test.loc[df_test['id'] == 5704, 'budget'] = 4300000 # French Connect
df_test.loc[df_test['id'] == 6109, 'budget'] = 281756 # Dogtooth
df_test.loc[df_test['id'] == 7242, 'budget'] = 10000000 # Addams Family
df_test.loc[df_test['id'] == 7021, 'budget'] = 17540562 # Two Is a Fami
df_test.loc[df_test['id'] == 5591, 'budget'] = 4000000 # The Orphanage
df_test.loc[df_test['id'] == 4282, 'budget'] = 20000000 # Big Top Pee-we

```

```
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['id']

```

In [6]:

```
target = "revenue"

X = df_train.drop(target, axis=1)
y = df_train[target]

X_train, X_valid, y_train, y_valid = train_test_split(X, y, train_size=0.7, t
X_train, X_test, y_train, y_test = train_test_split(X_train, y_train, test_si
```

In [7]:

```
dict_columns = ['belongs_to_collection', 'production_companies',
                'production_countries', 'spoken_languages', 'Keywords', 'cast

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

df_train = text_to_dict(df_train)
df_test = text_to_dict(df_test)
```

6.1 Genre,Belongs to Collection and Homepage Binarization

In [8]:

```

def proc_json(string, key):
    try:
        data = eval(string)
        return ",".join([d[key] for d in data])
    except:
        return ''

df_train.genres = df_train.genres.apply(lambda x: proc_json(x, 'name'))
df_test.genres = df_test.genres.apply(lambda x: proc_json(x, 'name'))

genres = []
for idx, val in df_train.genres.iteritems():
    gen_list = val.split(',')
    for gen in gen_list:
        if gen == '':
            continue

        if gen not in genres:
            genres.append(gen)

genre_column_names = []
for gen in genres:
    col_name = 'genre_' + gen.replace(' ', '_')
    df_train[col_name] = df_train.genres.str.contains(gen).astype('uint8')
    df_test[col_name] = df_test.genres.str.contains(gen).astype('uint8')
    genre_column_names.append(col_name)

```

In [9]:

```

# de-jsonify production_countries
df_train.production_countries = df_train.production_countries.apply(lambda x:
df_test.production_countries = df_test.production_countries.apply(lambda x:

```

In [10]:

```

# collection

df_train['belongs_to_collection'] = df_train['belongs_to_collection'].apply(lambda
df_test['belongs_to_collection'] = df_test['belongs_to_collection'].apply(lambda

```

```
In [11]: # homepage, released

df_train['has_homepage'] = 1
df_train.loc[pd.isnull(df_train['homepage']), "has_homepage"] = 0

df_test['has_homepage'] = 1
df_test.loc[pd.isnull(df_test['homepage']), "has_homepage"] = 0

df_train['released'] = 1
df_train.loc[pd.isnull(df_train['status']), "released"] = 0

df_test['released'] = 1
df_test.loc[pd.isnull(df_test['status']), "released"] = 0
```

```
In [12]: # de-jsonify production_countries
df_test.production_countries = df_test.production_countries.apply(lambda x: p
```

Let's get a count of how many production countries are involved in a production

```
In [13]: df_train['production_countries_count'] = df_train['production_countries'].app
df_test['production_countries_count'] = df_test['production_countries'].apply
```

6.1.1 Production Companies

```
In [14]: # take counts of each
df_train['production_companies_count'] = df_train['production_companies'].app
df_train['spoken_languages_count'] = df_train['spoken_languages'].apply(lambda

df_test['production_companies_count'] = df_test['production_companies'].apply
df_test['spoken_languages_count'] = df_test['spoken_languages'].apply(lambda
```

6.2 Keywords, cast, crew

```
In [15]: # number of keywords
df_train['keyword_count'] = df_train['Keywords'].apply(lambda x : len(x))
df_test['keyword_count'] = df_test['Keywords'].apply(lambda x : len(x))
```

```
In [16]: # count of crew
df_train['crew_count'] = df_train['crew'].apply(lambda x : len(x))
df_test['crew_count'] = df_test['crew'].apply(lambda x : len(x))
```

```
In [17]: # count of cast members
df_train['cast_count'] = df_train['cast'].apply(lambda x : len(x))
df_test['cast_count'] = df_test['cast'].apply(lambda x : len(x))
```

```
In [18]: df_train.runtime = df_train.runtime.fillna(0) # two null values here
```

6.3 Fixing Release Date

```
In [19]: df_test.release_date.isnull().sum()
```

```
Out[19]: 1
```

```
In [20]: df_test['release_date'] = df_test['release_date'].fillna('7/14/07')
```

```
In [21]: df_test.release_date.isnull().sum()
```

```
Out[21]: 0
```

```
In [22]: import re
def yearfix(x): # run year fix, then date fix
    """Regular expression to pull out the two digit year from release_date and
    r = re.match(r"(\d+/\d+)/(\d+)", x)[2]
    return int(r)
def datefix(x):
    """The dates only provide two digits for year. This is meant to fix this
    The youngest movie is from 2019, so we'll say any year digits less than 1
    Otherwise, we'll say they are from the 1900s."""
    if x < 19:
        x = x + 2000
        return x
    if x >= 19:
        x = x + 1900
        return x
```

```
In [23]: df_test['release_date'] = df_test['release_date'].fillna('7/14/07') # fill the
```

```
In [24]: df_train['release_year'] = df_train['release_date'].apply(lambda x: yearfix(x))
df_train['release_year'] = df_train['release_year'].apply(lambda x: datefix(x))
```

```
In [25]: df_train["release_date"] = pd.to_datetime(df_train["release_date"]) # set to datetime
df_train['release_month'] = df_train["release_date"].dt.month # month of release

df_test['release_year'] = df_test['release_date'].apply(lambda x: yearfix(x))
df_test['release_year'] = df_test['release_year'].apply(lambda x: datefix(x))
```

```
In [26]: df_test["release_date"] = pd.to_datetime(df_test["release_date"]) # set to datetime
df_test['release_month'] = df_test["release_date"].dt.month # month of release
```



```
In [27]: ▶ y = df_train['revenue']
X = df_train.drop(['revenue'],
                  axis = 1)

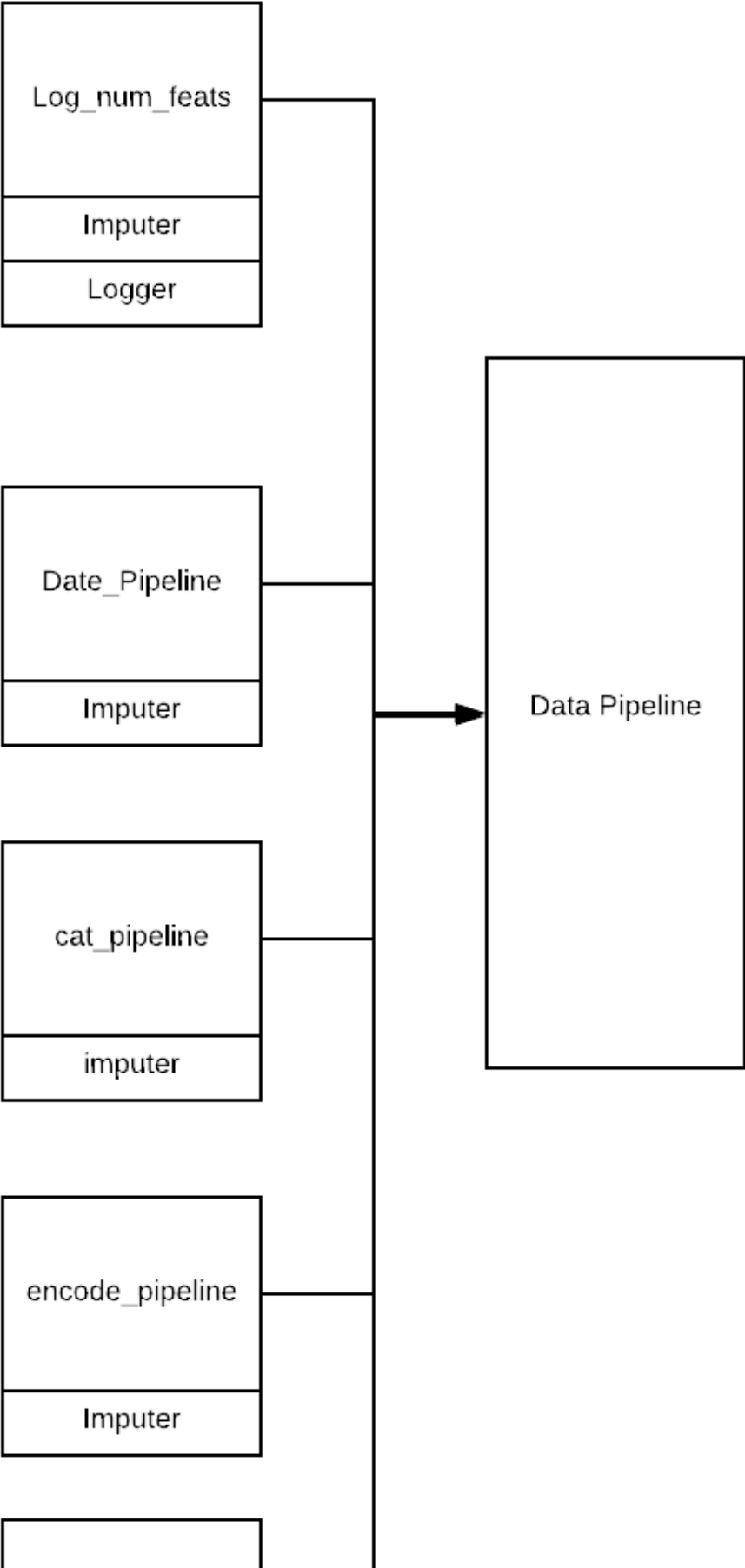
X_train, X_valid, y_train, y_valid = train_test_split(X, y, train_size=0.7, t
X_train, X_test, y_train, y_test = train_test_split(X_train, y_train, test_si
```

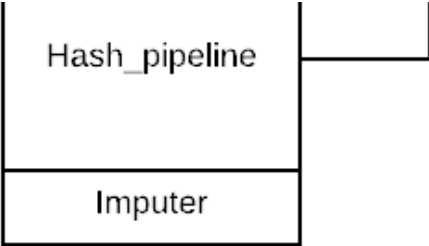
```
In [28]: ▶ class Log1pTransformer(BaseEstimator, TransformerMixin):
    def __init__(self, features=None):
        self.features = features

    def fit(self, X, y=None):
        return self

    def transform(self, X):
        return np.log1p(X)
```

7 Pipeline





```

In [29]: from sklearn.feature_extraction.text import HashingVectorizer

log_num_feats = ["budget", "popularity"]
# num_feats = ["runtime"] not required, will explore later
date_feats = ['release_year', "release_month"]
cat_feats = ['belongs_to_collection', 'has_homepage', 'released']
encode_feats = ['genre_Comedy', 'genre_Drama', 'genre_Family', 'genre_Romance',
                 'genre_Animation', 'genre_Adventure', 'genre_Horror', 'genre_Doc',
                 'genre_Science_Fiction', 'genre_Mystery', 'genre_Foreign', 'genre',
                 'genre_History', 'genre_TV_Movie']
hash_feats = ['production_countries_count', 'production_companies_count', 'sp',
              'keyword_count', 'cast_count', 'crew_count']

log_pipe = Pipeline([
    ('imputer', SimpleImputer(missing_values=np.nan, strategy='median')),
    ('logger', Log1pTransformer()),
    #('scaler', StandardScaler()) # might delete later - just a test
])

# num_pipeline = Pipeline ([
#     ('imputer', SimpleImputer(strategy='median'),
#     ('scaler', StandardScaler()))
# ])

date_pipeline = Pipeline ([
    ('imputer', SimpleImputer(missing_values=np.nan, strategy='median'))
])

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

encode_pipeline = Pipeline([
    ('imputer', SimpleImputer(strategy='most_frequent')),
])

hash_pipe = Pipeline([
    ('imputer', SimpleImputer(strategy='most_frequent'))
])

data_pipeline = ColumnTransformer([
    # name, pipeline, features
    ('log_num_feats', log_pipe, log_num_feats),
    # ('numerical_feats', num_pipeline, num_feats),
    ('date_feats', date_pipeline, date_feats),
    ('cat_feats', cat_pipeline, cat_feats),
])

```

```

        ('encode_feats', encode_pipeline, encode_feats),
        ('hash_feats', hash_pipe, hash_feats)
    ],
    remainder='drop', #passthrough or drop?
    n_jobs=-1
)

df_train_processed = data_pipeline.fit_transform(X_train)

```

```

In [30]: X_std = data_pipeline.fit_transform(df_train)
         model = data_pipeline.fit_transform(X)

```

```

In [31]: y_std = np.log1p(y)

```

```

In [32]: X_train, X_test, y_train, y_test = train_test_split(X_std, y_std, test_size=0.2)

```

8 Modeling

```

In [33]: from sklearn.linear_model import LinearRegression
         from sklearn.neighbors import KNeighborsRegressor
         models = {'linear': LinearRegression(),
                   'kn': KNeighborsRegressor()}

```

```

In [34]: def mean_absolute_percentage_error(y_test, y_pred):
         y_test, y_pred = np.array(y_test), np.array(y_pred)
         return np.mean(np.abs((y_test - y_pred) / y_test)) * 100
         results = pd.DataFrame(columns=["ExpID", "Train RMSLE", "Test RMSLE", "Dollars over"])
         def rmsle(y, y_pred):
             return np.sqrt(mean_squared_error(y, y_pred))

         def get_results(model, X, y, X_test, y_test, name='model_name', desc='experiment'):
             start = time()
             model.fit(X, y)
             y_pred = model.predict(X)
             train_rmsle = np.sqrt(((y_pred-y)**2).mean())
             y_pred_test = model.predict(X_test)
             test_rmsle = np.sqrt(((y_pred_test-y_test)**2).mean())
             dollarsover= test_rmsle*1000000000
             train_time = np.round(time() - start, 4)
             results.loc[results.shape[0]+1] = [name, np.round(train_rmsle,2), np.round(test_rmsle,2), dollarsover, train_time]

```

```
In [35]: ▶ for name, model in models.items():
           get_results(model, X_train, y_train, X_test, y_test, 'Initial Search', 'U
           display(results)
```

	ExpID	Train RMSLE	Test RMSLE	Dollars Over	Train Time(s)	Experiment description
1	Initial Search	2.23	1.79	1.788279e+09	0.2475	Untuned linear
2	Initial Search	2.08	2.09	2.086000e+09	0.2607	Untuned kn

```
In [36]: ▶ from sklearn.metrics import make_scorer
           # define custom scorer
           def rmsle(y_true, y_pred):
               rmsle = np.sqrt(((np.log1p(y_true)-np.log1p(y_pred))**2).mean())
               return rmsle

           scorer = make_scorer(rmsle)
```

9 Evaluation, reporting and analysis

```
In [64]: from sklearn.model_selection import GridSearchCV
from xgboost.sklearn import XGBRegressor
start = time()
xgb1 = XGBRegressor()
parameters = {'objective': ['reg:squarederror'],
              'learning_rate': [.03, 0.05, .07], #so called `eta` value
              'max_depth': [5, 6, 7],
              'min_child_weight': [3,4],
              'silent': [1],
              'subsample': [0.7,0.8,0.9],
              'colsample_bytree': [0.7],
              'n_estimators': [500,1000,2800]}

grid = GridSearchCV(xgb1,
                    parameters,
                    cv = 2,
                    n_jobs = -1,
                    verbose=True)

grid.fit(X_train, y_train)
print("Best parameters: {}".format(grid.best_params_))
train_time = np.round(time() - start, 4)
train_rmsle = grid.best_score_
y_pred = grid.predict(X_test)
test_rmsle = rmsle(y_test, y_pred)
test_mape = mean_absolute_percentage_error(y_test, y_pred)
dollarsover= test_rmsle*1000000000
results.loc[results.shape[0]+1] = ['Best Model: XGB Regressor', np.round(train_time, 4),
                                   train_time, "XGB Regressor"]

display(results)
```

Fitting 2 folds for each of 162 candidates, totalling 324 fits

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done 46 tasks      | elapsed: 6.2min
[Parallel(n_jobs=-1)]: Done 196 tasks    | elapsed: 27.3min
[Parallel(n_jobs=-1)]: Done 324 out of 324 | elapsed: 34.3min finished
/usr/local/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \
```

```
Best parameters: {'colsample_bytree': 0.7, 'learning_rate': 0.03, 'max_depth': 5, 'min_child_weight': 4, 'n_estimators': 500, 'objective': 'reg:squarederror', 'silent': 1, 'subsample': 0.8}
```

	ExpID	Train RMSLE	Test RMSLE	Dollars Over	Train Time(s)	Experiment description
1	Initial Search	2.23	1.79	1.788279e+09	0.2475	Untuned linear
2	Initial Search	2.08	2.09	2.086000e+09	0.2607	Untuned kn
3	Best Model: KN Regressor	0.22	0.18	1.758187e+08	25.5587	{'leaf_size': 30, 'n_neighbors': 5, 'p': 2, 'w...

	ExpID	Train RMSLE	Test RMSLE	Dollars Over	Train Time(s)	Experiment description
4	Best Model: KN Regressor	0.22	0.18	1.758187e+08	31.3203	{'leaf_size': 30, 'n_neighbors': 5, 'p': 2, 'w...
5	Best Model: XGB Regressor	0.52	0.14	1.391409e+08	2063.4620	XGB Regressor

9.1 Kaggle Submission

In [66]: ▶

```
test_std = data_pipeline.transform(df_test)
final_model = XGBRegressor(objective = 'reg:squarederror',
                           eta = 0.01,
                           max_depth = 6,
                           min_child_weight = 3,
                           subsample = 0.8,
                           colsample_bytree = 0.7,
                           eval_metric = 'rmse',
                           seed = 1,
                           n_estimators = 2800)

final_model.fit(X_std, y_std)
predictions = np.expm1(final_model.predict(test_std))
pd.DataFrame({'revenue': predictions.reshape(-1,)}, index=df_test.id).to_csv(
```

Your most recent submission

Name	Submitted	Wait time	Execution time	Score
best_model_xgb.csv	7 minutes ago	0 seconds	0 seconds	2.13772

Complete

[Jump to your position on the leaderboard ▼](#)

In [67]:

```

# Time and score test predictions
start = time()
final_model.fit(X_train, y_train)
train_time = np.round(time() - start, 4)
trainAcc = final_model.score(X_train, y_train)
start = time()
testAcc = final_model.score(X_test, y_test)
test_time = np.round(time() - start, 4)
experimentLog = pd.DataFrame(columns=["Pipeline", "Dataset", "TrainAcc", "TestAcc", "Train Time(s)", "Test Time(s)", "Description"])

try: experimentLog
except : experimentLog = pd.DataFrame(columns=["Pipeline", "Dataset", "TrainAcc", "TestAcc", "Train Time(s)", "Test Time(s)", "Description"])

experimentLog.loc[len(experimentLog)] = ["Pipeline with Optimised KNeighbors Regressor", "Movie Classes", 96.46%, 67.20%, 4.7014, 0.0322, "KNN pipeline with included features"]

experimentLog

```

Out[67]:

	Pipeline	Dataset	TrainAcc	TestAcc	Train Time(s)	Test Time(s)	Description
0	Pipeline with Optimised KNeighbors Regressor	Movie Classes	96.46%	67.20%	4.7014	0.0322	KNN pipeline with included features

10 Discussion

With the new features added to the pipeline, we were able to slightly increase our scores from phase 2. Our new features include:

- production_countries_count
 - production_companies_count
 - keyword_count
 - cast_count
 - crew_count
- Some of these features involved using our `proc_json` function (defined in phase 2), while some only involved taking the length of each list.

The results we were able to get through our new pipeline show a test accuracy of 64.06% which is an improvement from the 55.38% in phase 2. In comparison, our phase 1 baseline pipeline had 28.64% test accuracy for log baseline pipeline. We were unable to get a Kaggle score in phase 1, but in phase 2 our Kaggle score was 2.40318. We successfully improved our previous score from phase 2 thanks to our additional features and new gridsearch optimised XRB Regressor.

11 Conclusion

With this project, our focus was to be able to predict box office revenue for any movie. In order to do this, we created machine learning pipelines with custom features to predict the revenue from the box office. As we continue to improve our pipeline from the previous phase, features like 'production_companies_count', 'spoken_languages_count', 'keyword_count', 'cast_count', and

'crew_count' have been added and have allowed our pipelines to produce better accuracy scores which allowed our test dataset to be 64.06% accurate. We also used a different prediction model in XGBRegressor, which gave us more accurate scores. When submitting the scores to Kaggle, our score of 2.13772 shows that we are quite close to a model that can accurately predict these revenues.