What Factors Affect A Movie's Revenue Performance?

Team members: Xuanchen Liu, Simeng Alexandra Cai

Motivation

Why movies data?

Our project is analyzing movies listed in the Full MovieLens Datasets to explore the factors that influence a movie's box office performance in the worldwide scale.

Movies are one of the most popular forms of entertainment and make us learn something. However, factors that contribute to a movie's success (we define a movie's success mainly as its revenue in this scenario) are not that familiar with all audiences. Both of our team members are film enthusiasts and are interested in understanding what factors can impact a movie's revenue performance and what factors normally have more impact than others.

Our goal

Specifically, we intend to see factors -such as director gender/popularity, primary cast gender/popularity, secondary cast gender/popularity, budget, release dates, production companies, genre, etc. -that influence a movie's revenue. In addition, among these features, we would like to explore whether one or some specific factors can have more impact on a movie's performance than others.

Data Source

Primary Dataset

The primary dataset will be "movies_metadata" and "credits" datasets from Kaggle. The "movies_metadata" includes features such as language, budget, revenue, release dates, production countries from early 20th century to the end of 2020. The "credits" consists of film cast and crew information for all movies. The "movies_metadata" dataset is 32.85 megabytes, and the "credits" dataset is 181.12 megabytes. The Movies Datasets are in a csv format and can be downloaded at https://www.kaggle.com/rounakbanik/the-movies-dataset.

Secondary Dataset

The secondary dataset is retrieved from the TMDB website https://developers.themoviedb.org/3/people/get-person-details to get the people popularity feature, including the popularity for movie directors and cast. This dataset contains cast/crew id, gender, and popularity. A total of 46360 records (rows) were retrieved.

For each cast or crew member, his/her popularity number will be retrieved in a JSON format through the TMDB API. The estimated size for a single JSON file is 3kb.

Data Manipulation Methods

Step 1: Extract "Director Id", "Primary Cast Id" and "Secondary Cast Id" from Credits Dataset (Primary dataset)

-Director id of each movie is extracted from the crew column in the credits dataframe by filtering the value of "job" and setting its crew value as "Director". We created a function get_director_id to achieve this.

- Considering large amounts of cast each movie has, only Primary Cast and Secondary Cast will be retrieved in this context, which are normally taken as the most important cast and leading

actors/actresses for each movie. Specifically, Primary Cast id is extracted from the cast column by filtering the value of “order” of each cast as “0”, and Secondary Cast id is extracted by filtering the value of “order” of each cast as “1”. According to the dataframe, the lower the number, the more important the cast is. We created a function called get_cast_ids to achieve this goal.
-New columns director id, primary cast id and secondary cast id are created for the “credit” dataframe.

```
import pandas as pd
import requests
import json
import ast
```

```
credits = pd.read_csv('credits.csv')
def get_director_id(crews):
    crews = ast.literal_eval(crews)
    director = [crew.get('id') for crew in crews if crew.get('job', '') == 'Director']
    return director[0] if director else -1

def get_cast_ids(casts):
    casts = ast.literal_eval(casts)
    top2 = [cast.get('id') for cast in casts if cast.get('order', 9999) < 2]
    primary = top2[0] if top2 else -1
    secondary = top2[1] if len(top2) >= 2 else -1
    return pd.Series([primary, secondary])

credits['director_id'] = credits['crew'].apply(get_director_id)
credits[['primary_cast','secondary_cast']] = credits['cast'].apply(get_cast_ids)
credits.head()
```

Step2: Get popularities and Gender for Each Id from Secondary Dataset

-We created get_popularities function to retrieve the popularities for all directors, primary cast and secondary cast based on their ids from the secondary dataset TMDB website.
- A new dataframe “popularities” is created consisting columns of id, gender and popularity.

```
def get_popularities(ids):
    popularity_gender_list = []
    for id_ in ids:
        try:
            resp = requests.get(
                f"https://api.themoviedb.org/3/person/{id_}?api_key=1b0d5d30689a88c8c88f2586e12885de&language=en
-US")
            popularity_gender_entry = resp.json()
            entries = [id_, popularity_gender_entry.get('gender', -1), popularity_gender_entry.get('popularity',
-1)]
            popularity_gender_list.append(entries)
        except:
            print(f"Failed to pull popularity for person id #{id_}")
            continue
    people = pd.DataFrame(popularity_gender_list, columns = ['id', 'gender', 'popularity'])
    return people
```

```
popularities = get_popularities(person_ids)
```

```
director_id_set = set(credits['director_id'].to_list())
primary_cast_id_set = set(credits['primary_cast'].to_list())
secondary_cast_id_set = set(credits['secondary_cast'].to_list())
```

Step3: Merging dataframes “credits” (Primary Dataset) and “popularities” (Secondary Dataset) Based on their common column “Id”

-We merged two dataframes—“credits” and “popularities” based on their common ids of directors, primary cast and secondary cast.

-We used left join to merge the “credits” (left) with the “popularities” (right) with director id (for “credits” the column is called “director_id” and for “popularities” the column is called “id”). We then got a dataframe with original data from “credits” as well as the popularity and gender information for directors from “popularities”. Similar process was applied to merge primary & secondary cast popularity and gender to the “credits”.

```
#Join dataframes based on director id, this addes director gender and popularity to the credits dataframe
mergedf1=pd.merge(credits, popularities, left_on='director_id', right_on='id',how="left")
mergedf1.rename(columns={'gender': 'director_gender', 'popularity': 'director_popularity'}, inplace=True)
```

```
# join dataframes based on primary cast id, this addes columns of
#primary cast gender and popularity to the credits dataframe
mergedf2_primarycast=pd.merge(mergedf1, popularities, left_on='primary_cast', right_on='id',how="left")
mergedf2_primarycast=mergedf2_primarycast.drop(['id_y','id'], axis=1)
mergedf2_primarycast.rename(columns={'id_x': 'movie_id', 'gender': 'primary_cast_gender',
                                     'popularity':'primary_cast_popularity'}, inplace=True)
```

```
#join dataframes based on secondary cast id, this addes columns of
#secondary cast gender and popularities to the credits dataframe
merged3_secondarycast=pd.merge(mergedf2_primarycast,popularities,left_on='secondary_cast', right_on='id',how="left")
merged3_secondarycast.rename(columns={ 'gender': 'secondary_cast_gender','popularity':'secondary_cast_popularity'},
                             inplace=True)
merged3_secondarycast.head()
```

Step4: Data Cleaning for “merged3_secondarycast” dataframe

- Since useful information has been extracted, we will drop columns (‘cast’, and ‘crew’) that we don’t need for our analysis from “merged3_secondarycast” we got from step 3 after merging the primary and secondary dataset.

-Drop null value: we dropped rows with any null values in the dataframe using dropna().

-Check data type for each column by using df.info() and use astype() to convert each column into appropriate data type.

```
merged3_secondarycast=merged3_secondarycast.drop(["cast",'crew'],axis=1)
merged3_secondarycast=merged3_secondarycast.dropna(how='any')
```

```
merged3_secondarycast.info()
```

```
# convert data type of gender columns into integer
merged3_secondarycast.director_gender = merged3_secondarycast.director_gender.astype(int)
merged3_secondarycast.primary_cast_gender = merged3_secondarycast.primary_cast_gender.astype(int)
merged3_secondarycast.secondary_cast_gender = merged3_secondarycast.secondary_cast_gender.astype(int)
```

-We also used min() , max(), describe(), and value_counts() to understand the features. We found that 1 represents female, 2 represents male, and 3 represents non-binary. We dropped rows with -1 and 0 used for null values of gender.

```
count_primarycastgender = merged3_secondarycast['primary_cast_gender'].value_counts()
```

```
print(count_primarycastgender)
```

```
merged3_secondarycast = merged3_secondarycast[(merged3_secondarycast.primary_cast_gender != -1)&
                                              (merged3_secondarycast.primary_cast_gender != 0)]
merged3_secondarycast = merged3_secondarycast[(merged3_secondarycast.secondary_cast_gender != -1)&
                                              (merged3_secondarycast.secondary_cast_gender != 0)]
merged3_secondarycast = merged3_secondarycast[(merged3_secondarycast.director_gender != -1)&
                                              (merged3_secondarycast.director_gender != 0)]
```

-We dropped rows for “popularity” columns that have null value of -1.

```
#director_popularity:  minimu value is -1
# popularity has no upperbond, we decide it is better to just retain the original data
#drop rows if value is -1 for popularity columns
merged3_secondarycast = merged3_secondarycast[merged3_secondarycast.director_popularity != -1]
# 156 values are -1 for column directory_popularity
merged3_secondarycast= merged3_secondarycast[merged3_secondarycast.primary_cast_popularity != -1]
merged3_secondarycast= merged3_secondarycast[merged3_secondarycast.secondary_cast_popularity != -1]
```

Step 5: Merging Dateframes “mered3_secondarycast” from step 4 and “movies_metadata.csv” (one of the primary dataset) based on common column “movie id”.

-we cleaned the id column and converted the datatype into integer. We got error messages for values that can’t be converted because they are date-time values mistakenly input into the wrong column—therefore, astype() was used to identify invalid values for data cleaning.

```
movies_metadata=pd.read_csv("movies_metadata.csv")
movies_metadata= movies_metadata[movies_metadata.id != '1997-08-20']
movies_metadata= movies_metadata[movies_metadata.id != '2012-09-29']
movies_metadata= movies_metadata[movies_metadata.id != '2014-01-01']
movies_metadata['id']=movies_metadata.id.astype(int)
```

```
df=pd.merge(merged3_secondarycast,movies_metadata,left_on='movie_id',right_on='id',how="left")
```

Step6: Selecting Columns(features) in “df” dataframe that we will be analyzing

```
whitelist = ['director_gender','director_popularity', 'primary_cast_gender','primary_cast_popularity',
             'secondary_cast_gender','secondary_cast_popularity','adult', 'budget', 'genres', 'original_language',
             'original_title', 'popularity', 'production_companies', 'production_countries', 'release_date',
             'revenue', 'runtime', 'spoken_languages', 'status', 'title', 'video', 'vote_average', 'vote_count']
df = df[whitelist]
```

Step7: Further Data Cleaning for “df” dataframe

-We filtered out noisy data for video column.

-We created a function get_first_from() to get the first item from the embedded json list for columns, including “genres”, “production_companies”, “production_countries”, “spoken_language” in the dataframe.

-We converted the column values to appropriate datatype for analysis. For example, we used one-hot-encoding for categorical variables with only a couple of unique values, and converted categorical variables to ordinal forms for columns with many unique values.

-we used min() and describe() to figure out that there is non-negative values for numerical columns and because there is no upper bound, we decided it is better to just retain the original data instead of removing outliers using z-score.

```
# filter out noisy data and only keep the boolean true and false value
df = df[(df.video == True) | (df.video == False)]
df.dropna(inplace=True)
```

```
#get first item from the embedded json list
def get_first_from(value):
    list_ = ast.literal_eval(value)
    return list_[0].get('name') if list_ else None
columns = ['genres', 'production_companies','production_countries','spoken_languages']
for c in columns:
    df[f'major_{c}'] = df[c].apply(get_first_from)
df.drop(columns=columns, axis=1, inplace=True)
```

```
#check if title has changed
df['title_changed'] = (df.original_title != df.title)
df.drop(columns=['original_title'], axis=1, inplace=True)
```

```
#convert categorical variables to numeric for data analysis
column = 'status'
df[column] = df[column].astype('category')
df[f'{column}_code'] = df[column].cat.codes
df.drop(columns=[column], axis=1, inplace=True)
```

```
#convert categorical variables to ordinal if there are many unique values
columns = [
    'original_language',
    'major_genres',
    'major_production_companies',
    'major_production_countries',
    'major_spoken_languages',
]
df[columns] = df[columns].fillna('nan')
for c in columns:
    df_by = df[[c, 'revenue']].groupby(by=c).mean()
    df_by.sort_values(by='revenue', inplace=True)
    df_by.reset_index(inplace=True)

    mapping = pd.Series(df_by.index.values, index=df_by[c]).to_dict()
    df.replace({c: mapping}, inplace=True)
    df[c] = df[c].astype(int)
```

```
#convert categorical variables to one-hot encoding if there are only a couple of unique values
columns = [
    'director_gender',
    'primary_cast_gender',
    'secondary_cast_gender',
    'status_code'
]
dfs = [df]
for c in columns:
    df_one_hot = pd.get_dummies(df[c], prefix=c)
    dfs.append(df_one_hot)
df = pd.concat(dfs, axis=1)
df.drop(columns=columns, axis=1, inplace=True)
#convert dates to numeric
df.release_date = pd.to_datetime(df.release_date)
epoch = datetime.date(1970, 1, 1)
df['release_date_epoch'] = df.release_date.apply(lambda x: (x.date() - epoch).days)
df.drop(columns=['release_date'], inplace=True)
#change datatypes
df.adult = df.adult.astype(int)
df.runtime = df.runtime.astype(int)
df.video = df.video.astype(int)
df.vote_count = df.vote_count.astype(int)
df.title_changed = df.title_changed.astype(int)
```

Analysis and Visualization:

Step 1: From Figure 1 below we can see that there tends to show a linear trend between “revenue” and “budget”. At the same time, values of variables such as “director_popularity”, “primary_cast_popularity”, “secondary_cast_poluarity” and “popularity”(representing the movie popularity) tend to get slanted to the left(these variables are right-skewed), which gives us some hints that we can do some log transformation for further exploration to see if there's any relationship between these variables and revenue.

```
#Data visualization
df1 = df[['revenue','director_popularity','primary_cast_popularity','secondary_cast_popularity',
    'budget','popularity']]
pd.plotting.scatter_matrix(df1, alpha=0.2, figsize=(15, 15))
```

Step2: From Figure 2 we can see a linear trend showing between “revenue” and “vote_count”. Values of runtime tend to focus on the left side(right-skewed), which we can also do log transformation to further explore the relationship between revenue and runtime.

```
df2 = df[['revenue','runtime','vote_average','vote_count','original_language']]
pd.plotting.scatter_matrix(df2, alpha=0.2, figsize=(15, 15))
```

We'll take log of the following variables to explore further since we can not see any obvious regression trend at this moment.

```
df3 = df[['revenue','major_genres','major_production_companies','major_production_countries','major_spoken_languages',
    'release_date_epoch']]
pd.plotting.scatter_matrix(df3, alpha=0.2, figsize=(15, 15))
```

Figure 1

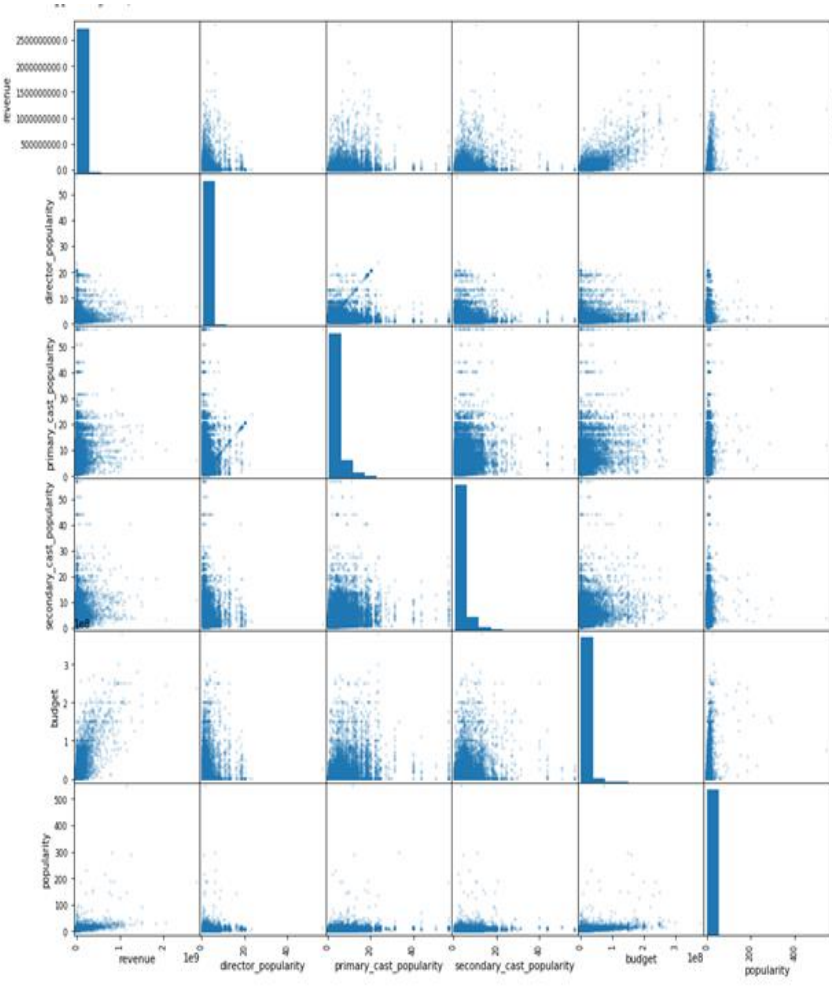
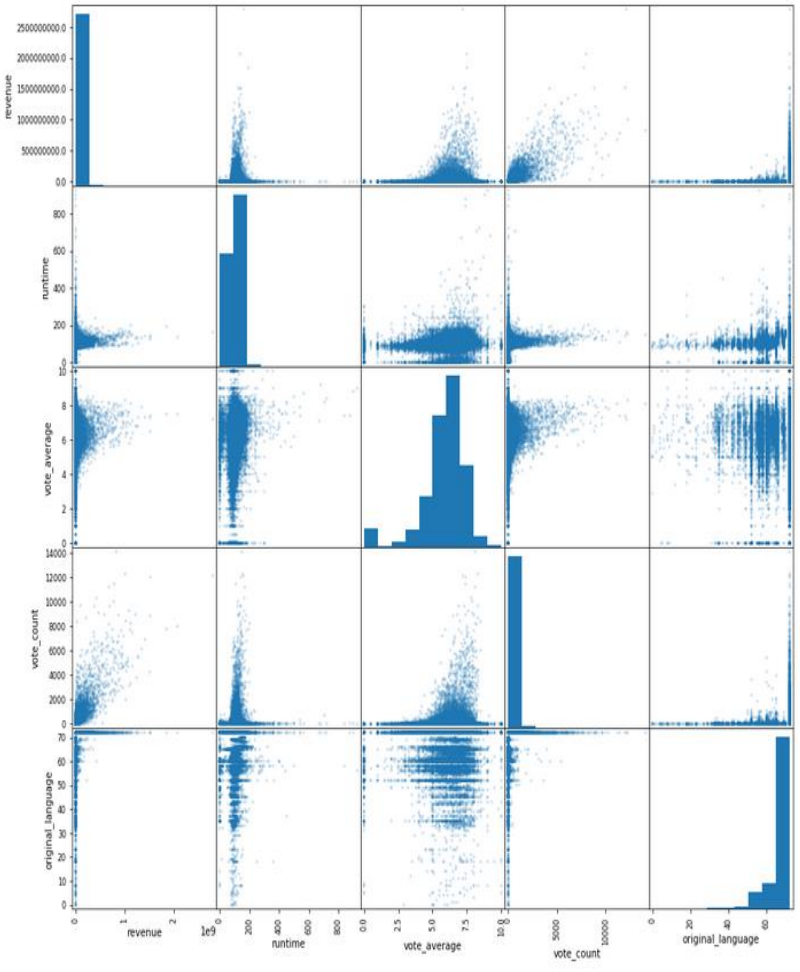


Figure 2



Step3: Log transformation

-If the variable is right-skewed, take their log.

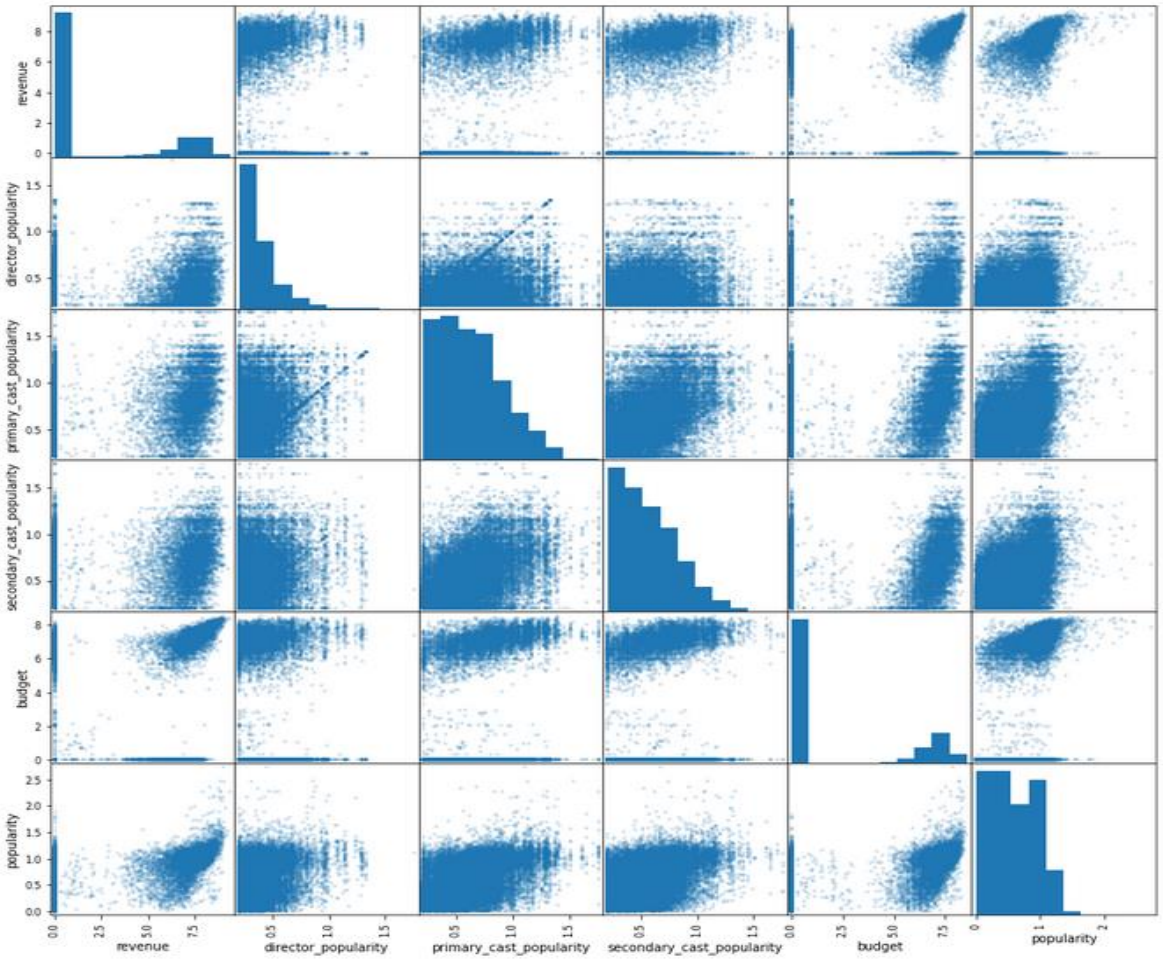
-After taking log, we can see some variables that seem unconnected with revenue before tend to show some linear trend, such as “director_popularity”, “primary_cast_popularity”, “secondary_cast_popularity”, “budget”, and “popularity”. Revenue tends to increase when values of these independent variables increases. See below.

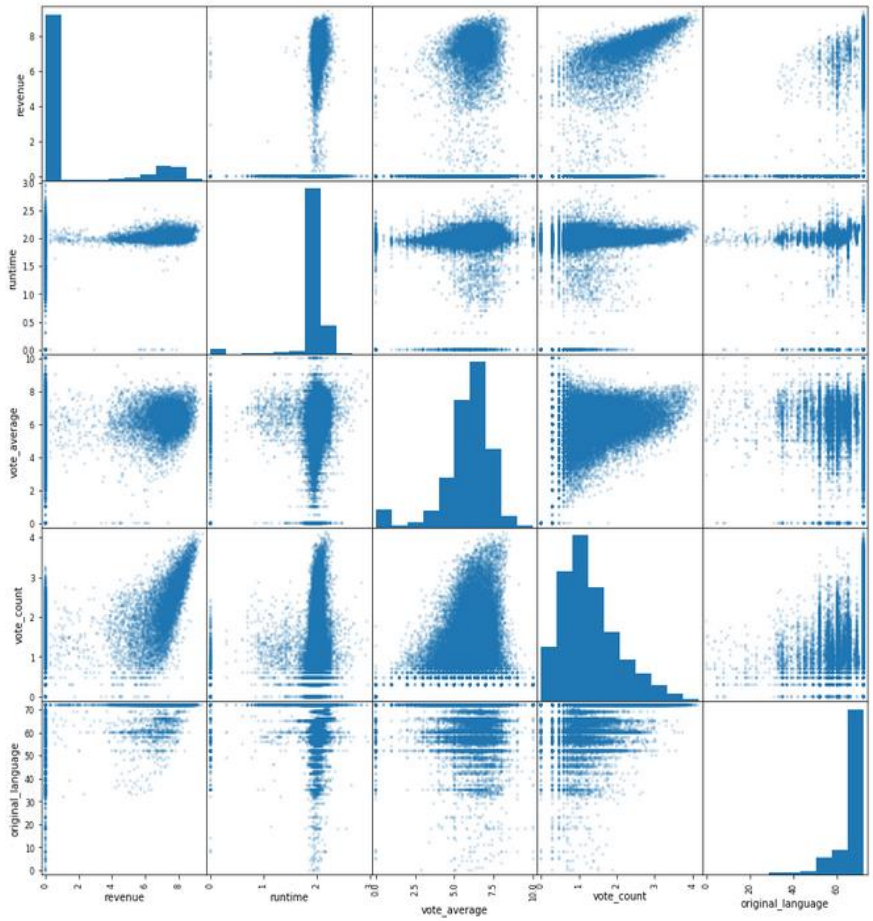
```
columns = ['revenue','director_popularity','primary_cast_popularity','secondary_cast_popularity',
    'budget','popularity','runtime','vote_count',]
for c in columns:
    df[c] = np.log10(df[c]+1)
```

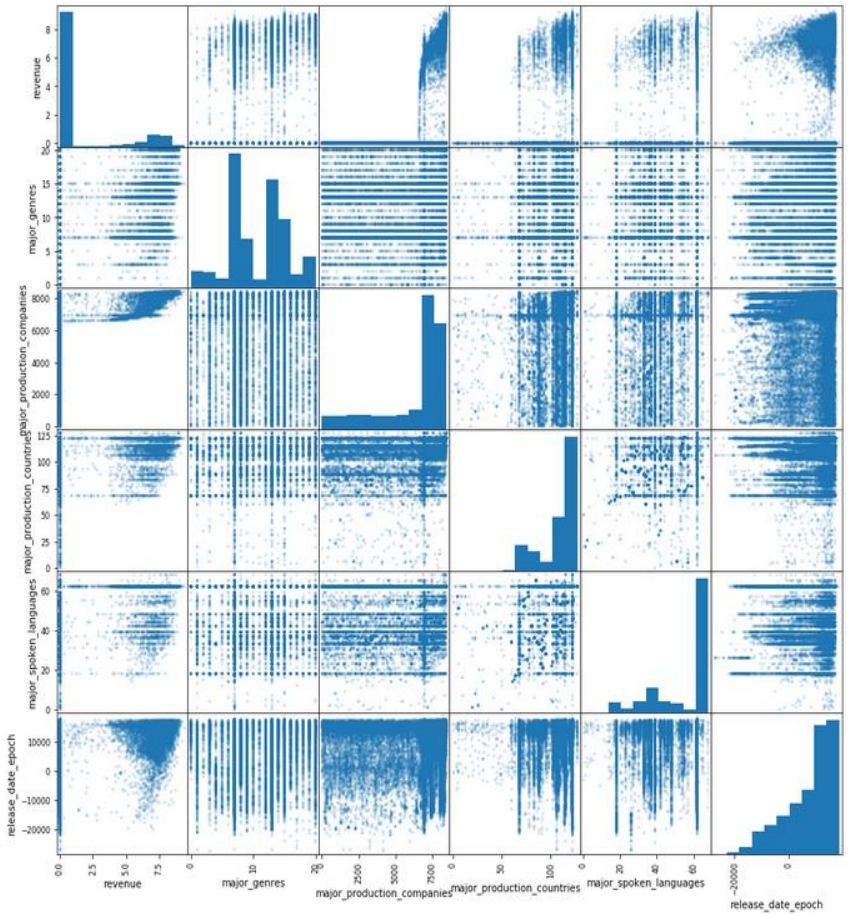
```
df1 = df[['revenue','director_popularity','primary_cast_popularity','secondary_cast_popularity','budget',
    'popularity',]]
pd.plotting.scatter_matrix(df1, alpha=0.2, figsize=(15, 15))
```

```
df2 = df[['revenue','runtime','vote_average','vote_count','original_language']]
pd.plotting.scatter_matrix(df2, alpha=0.2, figsize=(15, 15))
```

```
df3 = df[['revenue','major_genres','major_production_companies','major_production_countries',
          'major_spoken_languages','release_date_epoch']]
pd.plotting.scatter_matrix(df3, alpha=0.2, figsize=(15, 15))
```




revenue
runtime
vote_average
vote_count
original_language


revenue
major_genres
major_production_companies
major_production_countries
major_spoken_languages
release_date_epoch

Step4: Regression Analysis

-we used linear regression model to examine the linear relationship between independent variables such as “budget”, “popularity”, “gender” and the dependent variable “revenue”.

```
from sklearn.linear_model import LinearRegression
columns = df.columns
columns = [c for c in columns if c not in {'revenue', 'title'}]
X = df[columns]
y = df.revenue
reg = LinearRegression()
reg.fit(X, y)
reg.score(X, y)
```

```
0.6156337538777461
```

```
import statsmodels.api as sm
```

```
X2 = sm.add_constant(X)
est = sm.OLS(y, X2)
est2 = est.fit()
```

```
est2.summary()
```

The summary of the OLS regression results shows the coefficient for each independent variable in the model and the corresponding p-value.

Step 5: We eliminated features with p-value being greater than 0.05 because we only want to keep variables that have a statistically significant effect on the dependent variable in our OLS regression model and we built the model again until we have a model with p-values of all features less than 0.05.

-The OLS regression results show that the model is:

Revenue=3.0063+0.2329*director_popularity+0.3256*primary_cast_popularity+0.4087*secondary_cast_popularity+0.3735*budget-0.5639*popularity+0.2766*runtime-0.1023*vote_average+1.7337*vote_count+0.0003*major_production_companies

-Features that have positive impact on revenue ordered from highest to lowest impact—higher coefficients mean higher impact:

vote_count> secondary_cast_popularity> budget> primary_cast_popularity> runtime> director_popularity> major_production_companies

- Features that have negative impact on revenue: vote_average & popularity

```
#drop any feature if the corresponding p-value > 0.05
dropped = {'adult','original_language','video','major_genres','major_spoken_languages','director_gender_1',
           'director_gender_2', 'director_gender_3','primary_cast_gender_2','primary_cast_gender_3',
    'secondary_cast_gender_1','secondary_cast_gender_2','secondary_cast_gender_3','status_code_0',
           'status_code_1', 'status_code_2', 'status_code_3','release_date_epoch'}
columns = [c for c in columns if c not in dropped]
X = df[columns]
X2 = sm.add_constant(X)
est = sm.OLS(y, X2)
est2 = est.fit()
est2.summary()
```

Step6: Lasso Regression

Using lasso model to further potentially simplify the model to get the most significant features. As we can see from the below result, **the most important features are “vote_count”, “budget”, and “major_production_companies”.**

```
#drop any feature if the corresponding p-value > 0.05
dropped = {'major_production_countries','title_changed',
           'primary_cast_gender_1'},
columns = [c for c in columns if c not in dropped]
X = df[columns]
X2 = sm.add_constant(X)
est = sm.OLS(y, X2)
est2 = est.fit()
est2.summary()
```

OLS Regression Results

Dep. Variable:	revenue	R-squared:	0.615
Model:	OLS	Adj. R-squared:	0.615
Method:	Least Squares	F-statistic:	3853.
Date:	Mon, 24 May 2021	Prob (F-statistic):	0.00
Time:	07:11:42	Log-Likelihood:	-46160.
No. Observations:	21701	AIC:	9.234e+04
Df Residuals:	21691	BIC:	9.242e+04
Df Model:	9		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	-3.0063	0.145	-20.765	0.000	-3.290	-2.723
director_popularity	0.2329	0.080	2.897	0.004	0.075	0.390
primary_cast_popularity	0.3256	0.058	5.577	0.000	0.211	0.440
secondary_cast_popularity	0.4087	0.062	6.578	0.000	0.287	0.531
budget	0.3735	0.006	65.430	0.000	0.362	0.385
popularity	-0.5639	0.080	-7.018	0.000	-0.721	-0.406
runtime	0.2766	0.068	4.051	0.000	0.143	0.410
vote_average	-0.1023	0.011	-9.334	0.000	-0.124	-0.081
vote_count	1.7337	0.041	42.146	0.000	1.653	1.814
major_production_companies	0.0003	8.53e-06	39.526	0.000	0.000	0.000

Omnibus:	529.266	Durbin-Watson:	1.900
Prob(Omnibus):	0.000	Jarque-Bera (JB):	1209.068
Skew:	0.080	Prob(JB):	2.85e-263
Kurtosis:	4.145	Cond. No.	5.43e+04

Step5:OLS regression

```
#------------------Lasso regression
from sklearn.linear_model import LassoCV
lassocv = LassoCV(cv = 10, max_iter = 100000, normalize = True,
                  random_state=42)
lassocv.fit(X, y)
lassocv.score(X, y)
```

```
0.6151972078885197
```

```
{k: v for k, v in sorted(dict(zip(columns, lassocv.coef_)).items(),
                         key=lambda item: abs(item[1]),
                         reverse=True)}
```

```
{'vote_count': 1.7097479428337283,
 'popularity': -0.507675618562757,
 'secondary_cast_popularity': 0.4030252906589698,
 'budget': 0.3736078249729487,
 'primary_cast_popularity': 0.32058159041841994,
 'runtime': 0.26490762374262256,
 'director_popularity': 0.22424421337412478,
 'vote_average': -0.09909599268920079,
 'major_production_companies': 0.0003361481252357648}
```

```
from sklearn.linear_model import Lasso
lasso = Lasso(alpha=0.1, random_state=42)
lasso.fit(X, y)
{k: v for k, v in sorted(dict(zip(columns, lasso.coef_)).items(),
                         key=lambda item: abs(item[1]),
                         reverse=True)}
```

```
{'vote_count': 1.296766502338385,
 'budget': 0.4147185553274118,
 'major_production_companies': 0.0003733590478715816,
 'director_popularity': 0.0,
 'primary_cast_popularity': 0.0,
 'secondary_cast_popularity': 0.0,
 'popularity': 0.0,
 'runtime': 0.0,
 'vote_average': -0.0}
```

```
lasso.score(X, y)
```

```
0.6085389281576881
```

Step6: Lasso regression

Statement of Work:

Data Cleaning: Xuanchen Liu & Simeng Cai Data visualization and Analysis: Xuanchen Liu Report Writing: Simeng Cai