SC1015
Mini Project

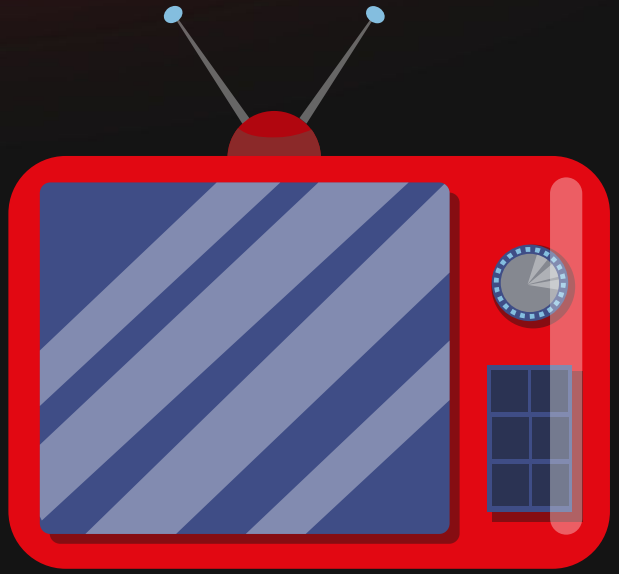
A132 GROUP 6

Phan Nguyen
Qiang Zhiqin
Rachel Phuar Yi Ling

01 Practical Motivation

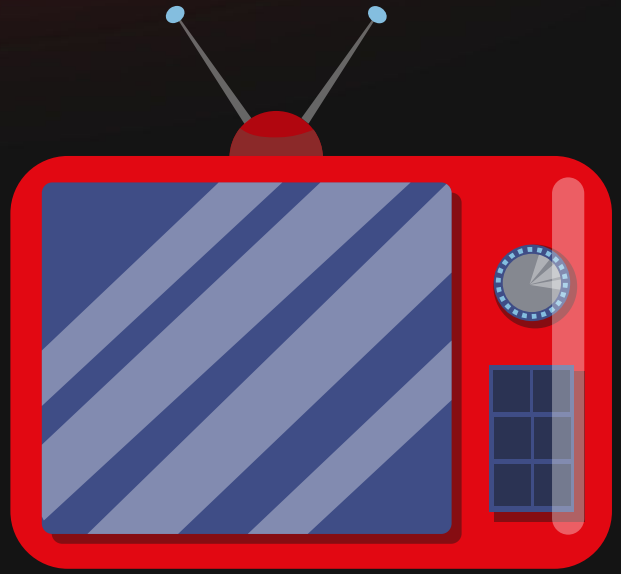
Real-Life Problem

Difficult to determine the success of a movie



Problem Statement

Is it possible to predict the success of a movie based on pre and post production elements?



Breaking Down the Problem

Pre-production elements

- Budget
- Director
- Genre
- Cast

Post-production elements

- Revenue
- Popularity
- Ratings

Dataset

‘The Movie Dataset’ by TMDb
4803 movies
Contains 2 CSV

tmdb_5000_movies.csv

Budget : company's budget
genres : genres of the movie
homepage : link to the homepage
id : movie id
keywords : keywords of the data
original_language : shows the original language of the movie
original_title : original title of the movie
overview : movie overview
popularity : popularity rate of the movie
production_companies production company name
production_countries :production countries names
release_date : date of release
revenue : movie's revenue
runtime : runtime of the movie
spoken_languages : list of spoken languages
status : status of the movie (ex. released)
tagline : tagline of the movie
vote_average : average of vote grade
vote_count : number of votes

tmdb_5000_credits.csv

movie_id : id of the movie
title : title of the movie
cast : information of cast in the movie
crew : information of cast in the movie

Summary of the merged data set, with 4803 records.

Budget : budget of the company
genres : show genres of the movie
homepage : link to the homepage
id : id of the movie
keywords : show keywords of the data
original_language : show the language of the movie
original_title : original title of the movie
overview : overview of the movie
popularity : popularity rate of the movie
production_companies : name of production company
production_countries : name of production countries
release_date : date of release
revenue : revenue of the movie
runtime : runtime of the movie
spoken_languages : list of spoken language
status : status of the movie (ex. released)
tagline : tagline of the movie
title : title of the movie
vote_average : average of vote grade
vote_count : number of vote
movie_id : id of the movie
cast : information of cast in the movie
crew : information of cast in the movie

Exploratory Data Analysis

02

Data cleaning

Missing values will be filled with default values (NULL or 0)

```
[ ] moviesdata['homepage'] = moviesdata['homepage'].fillna('')
    moviesdata['overview'] = moviesdata['overview'].fillna('')
    moviesdata['release_date'] = moviesdata['release_date'].fillna('')
    moviesdata['runtime'] = moviesdata['runtime'].fillna(0)
    moviesdata['tagline'] = moviesdata['tagline'].fillna('')
```

Homepage, overview, runtime and tagline have missing values but they are not important factors to predicting the success of the movie
-> Fill in the missing values with default values

```
Number of missing value in  budget :  0
Number of missing value in  genres :  0
Number of missing value in  homepage :  3096
Number of missing value in  id :  0
Number of missing value in  keywords :  0
Number of missing value in  original_language :  0
Number of missing value in  original_title :  0
Number of missing value in  overview :  3
Number of missing value in  popularity :  0
Number of missing value in  production_companies :  0
Number of missing value in  production_countries :  0
Number of missing value in  release_date :  1
Number of missing value in  revenue :  0
Number of missing value in  runtime :  2
Number of missing value in  spoken_languages :  0
Number of missing value in  status :  0
Number of missing value in  tagline :  844
Number of missing value in  title :  0
Number of missing value in  vote_average :  0
Number of missing value in  vote_count :  0
Number of missing value in  movie_id :  0
Number of missing value in  cast :  0
Number of missing value in  crew :  0
```

Selection of relevant columns

Pre-production elements

- Budget
- Director
- Genre
- Cast

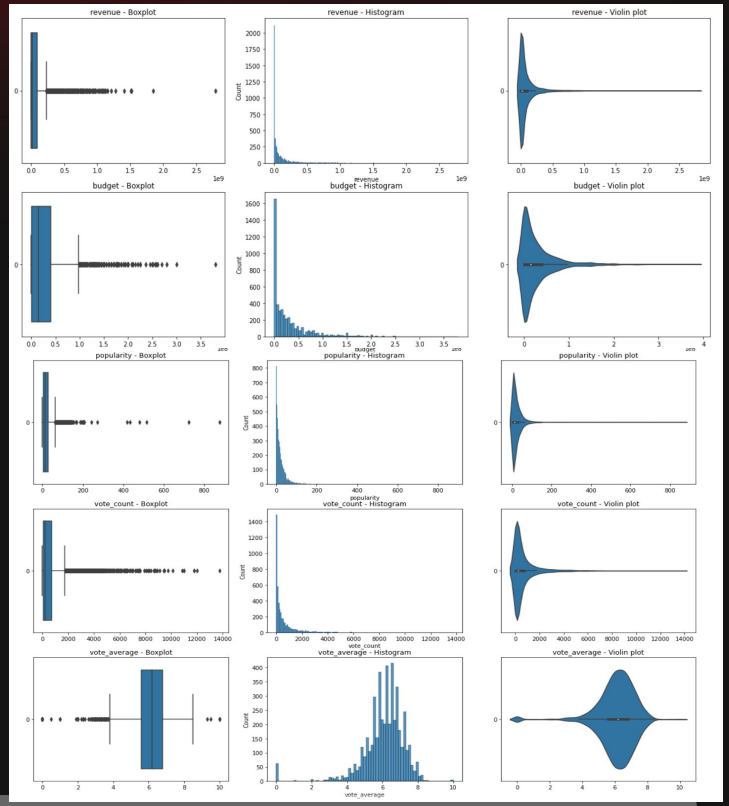
Post-production elements

- Revenue
- Popularity
- Ratings (vote count & vote average)

Budget : budget of the company
genres : show genres of the movie
homepage : link to the homepage
id : id of the movie
keywords : show keywords of the data
original_language : show the language of the movie
original_title : original title of the movie
overview : overview of the movie
popularity : popularity rate of the movie
production_companies : name of production company
production_countries : name of production countries
release_date : date of release
revenue : revenue of the movie
runtime : runtime of the movie
spoken_languages : list of spoken language
status : status of the movie (ex. released)
tagline : tagline of the movie
title : title of the movie
vote_average : average of vote grade
vote_count : number of vote
movie_id : id of the movie
cast : information of cast in the movie
crew : information of cast in the movie

Preparation of dataset

Here we plot the chosen features, including revenue, budget, popularity, vote count, vote average to observe how normal and reliable they are.



Then we measure the skewness and kurtosis of data and decided to:

- Normalise the distribution of variables
- Remove outliers

Measure the skewness and kurtosis of data

```
data_o.loc[:, ['revenue', 'budget', 'popularity', 'vote_count', 'vote_average']].skew()
```

```
revenue          4.442112
budget           2.438385
popularity       9.721524
vote_count       3.823107
vote_average    -1.961065
dtype: float64
```

```
[ ] data_o.loc[:, ['revenue', 'budget', 'popularity', 'vote_count', 'vote_average']].kurt()
```

```
revenue          33.097489
budget            7.667528
popularity      192.078712
vote_count       19.914038
vote_average      7.803744
dtype: float64
```

We remove all records with '0' value, as when we transform to log scale (because log(0) will go to -infinity).

```
[ ] data_process = data_process[data_process['budget'] > 0]
    data_process = data_process[data_process['revenue'] > 0]
    data_process = data_process[data_process['vote_count'] > 0]
    data_process = data_process[data_process['popularity'] > 0]
    data_process.shape
```

```
(3230, 27)
```

We will use log-transformation to reduce the skewness of data and make it more uniform:

- First, we remove all records with “0” values.

- Next, we use function np.log function to take the log of each feature.
- We remove outliers by using function called “zscore” from scipy. If the distance of the point is more than 3 times of standard deviation, we consider it outlier and remove it.

Transform the dataset into log scale to make it more uniform

```
[ ] budget_log = pd.DataFrame(np.log(data_process['budget']))
    revenue_log = pd.DataFrame(np.log(data_process['revenue']))
    popularity_log = pd.DataFrame(np.log(data_process['popularity']))
    vote_count_log = pd.DataFrame(np.log(data_process['vote_count']))
```

```
[ ] data_log = pd.concat([budget_log, revenue_log, num_cast_d, num_crew_d, genres_d, popularity_log, vote_count_log, vot
    data_log.columns = ['budget', 'revenue', 'num_cast', 'num_crew', 'sciencefiction','action','western','comedy','myste
    data_log.shape
```

```
(3230, 27)
```

Here, we use a function called ***zscore*** from stats in scipy. The function of this function is to to measure the distance of the point to mean over the standard deviation. If more than 3 times or the point lies out of 3 standard deviation, we cut that record.

```
[ ] data_log = data_log[(np.abs(stats.zscore(data_log['budget'])) < 3)]
    data_log = data_log[(np.abs(stats.zscore(data_log['revenue'])) < 3)]
    data_log = data_log[(np.abs(stats.zscore(data_log['popularity'])) < 3)]
    data_log = data_log[(np.abs(stats.zscore(data_log['vote_count'])) < 3)]
    data_log.shape
```

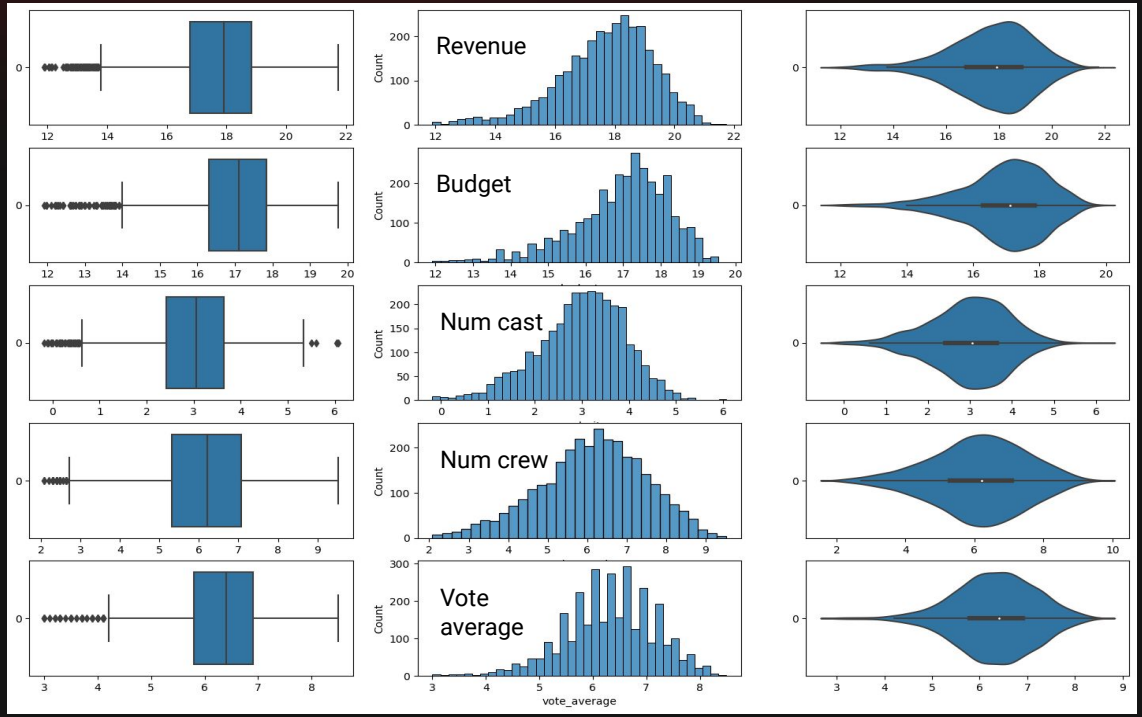
```
(3100, 27)
```

	budget	revenue	num_cast	num_crew	popularity	vote_count	vote_average
count	3100.000000	3100.000000	3100.000000	3100.000000	3100.000000	3100.000000	3100.000000
mean	16.950823	17.716700	26.573226	34.846129	2.987727	6.133300	6.325903
std	1.270268	1.575329	21.662035	35.366078	0.930901	1.343900	0.843885
min	11.918391	11.894112	0.000000	1.000000	-0.179585	2.079442	3.000000
25%	16.300417	16.784724	15.000000	12.000000	2.421539	5.303305	5.800000
50%	17.111347	17.910894	20.000000	21.000000	3.057103	6.212604	6.400000
75%	17.845287	18.832801	31.000000	45.000000	3.641061	7.077497	6.900000
max	19.755682	21.748578	224.000000	435.000000	6.073686	9.528940	8.500000

AFTER NORMALISATION

- Statistics of relevant variables
  -> data is normalised
  -> forms a normal distribution so as to reduce the effect of outliers and skewed data

Now, we can clearly observe that the data is more uniform and ready to be fed into the model



Both the statistical skewness and kurtoris are decent

```
data_log.loc[:, ['revenue', 'budget', 'popularity', 'vote_count', 'vote_average']].skew()
```

```
revenue         -0.678232
budget          -0.829586
popularity      -0.414438
vote_count      -0.287159
vote_average    -0.360412
dtype: float64
```

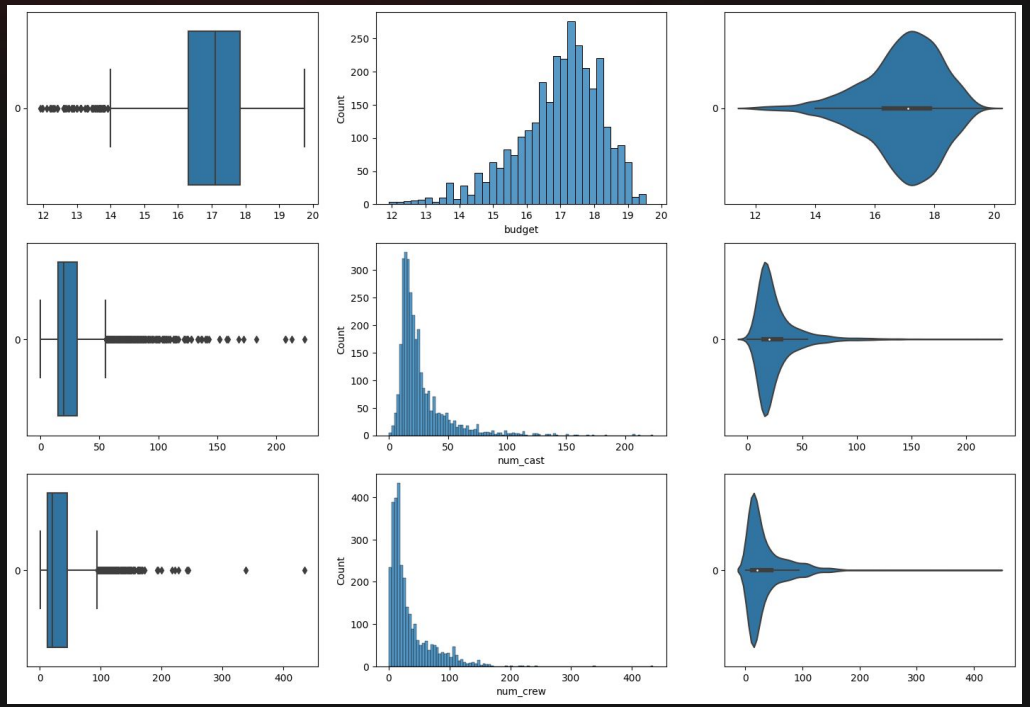
```
data_log.loc[:, ['revenue', 'budget', 'popularity', 'vote_count', 'vote_average']].kurt()
```

```
revenue          0.578514
budget           0.850353
popularity       0.196128
vote_count      -0.184214
vote_average     0.335789
dtype: float64
```

Univariate Exploratory Analysis with Pre-Elements

Now, we have 4 pre-production elements:

- Budget
- Num_cast
- Num_crew
- Genres: 20 genres are used as 20 different features

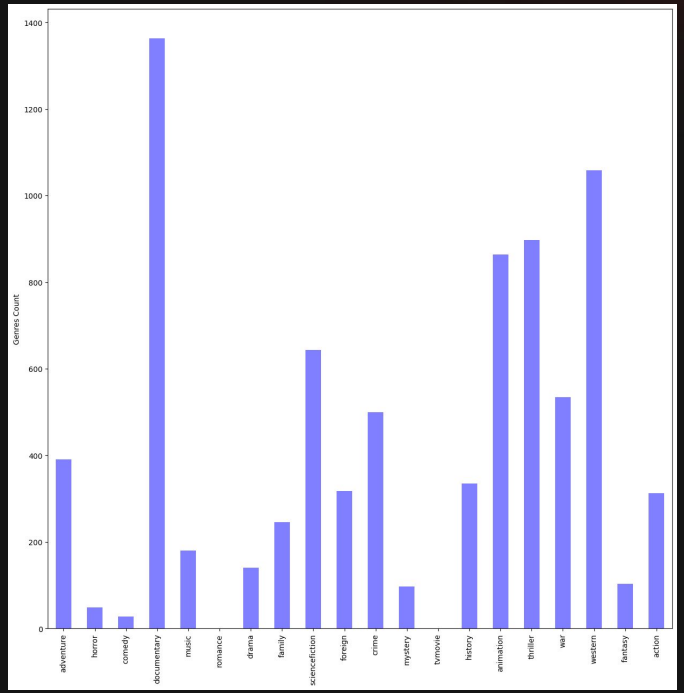


```
[ ] genres_d.head()
```

	drama	history	foreign	music	horror	action	comedy	thriller	crime	war	fantasy	romance	mystery	documentary	sciencefiction	western	family	animation	adventure	tvmovie
0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	1	0	0	0	1	0
1	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	1	0
2	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	1	0
3	1	0	0	0	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	1	0

Before moving on, we perform one-hot encoding representations of the genres to make all related data have the same numerical form

Univariate Exploratory Analysis with Pre-Elements



20 Genres

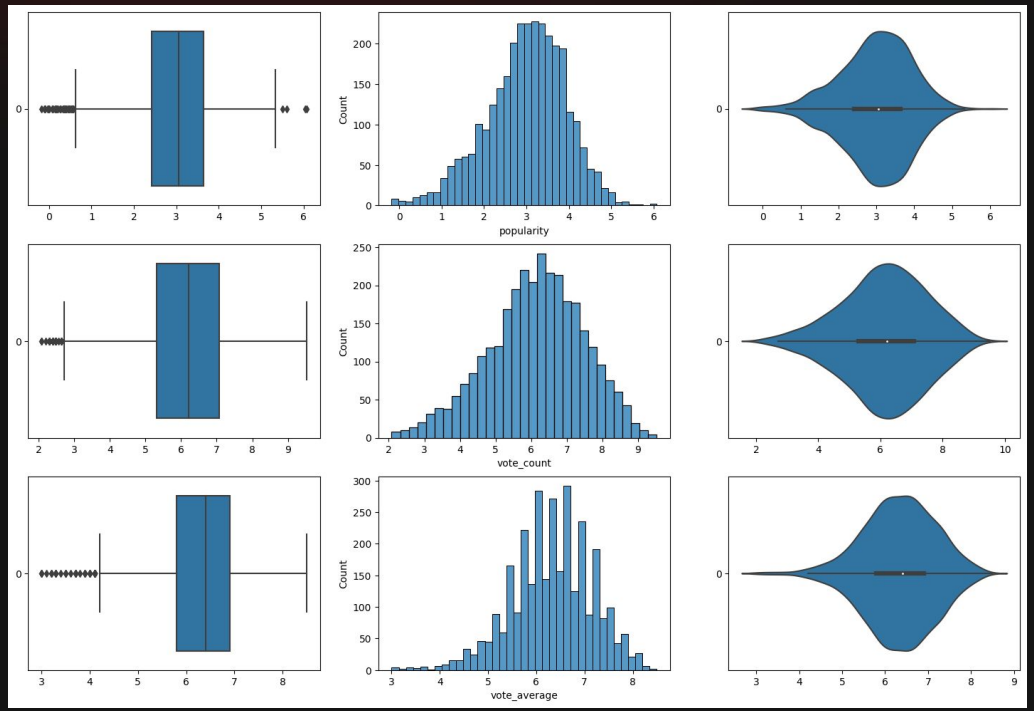
```
num_genres = data_log[genres
num_genres
```

```
adventure          390
horror              49
comedy              28
documentary       1363
music              180
romance              0
drama              140
family             245
sciencefiction     643
foreign            318
crime              500
mystery             97
tvmovie              0
history            335
animation          864
thriller           897
war                534
western           1058
fantasy            103
action             312
dtype: int64
```

Univariate Exploratory Analysis with Post-Elements

We have 3 post-production elements:

- Popularity
- Vote_count
- Vote_average



Multivariate Exploratory Analysis

Find correlations with revenue

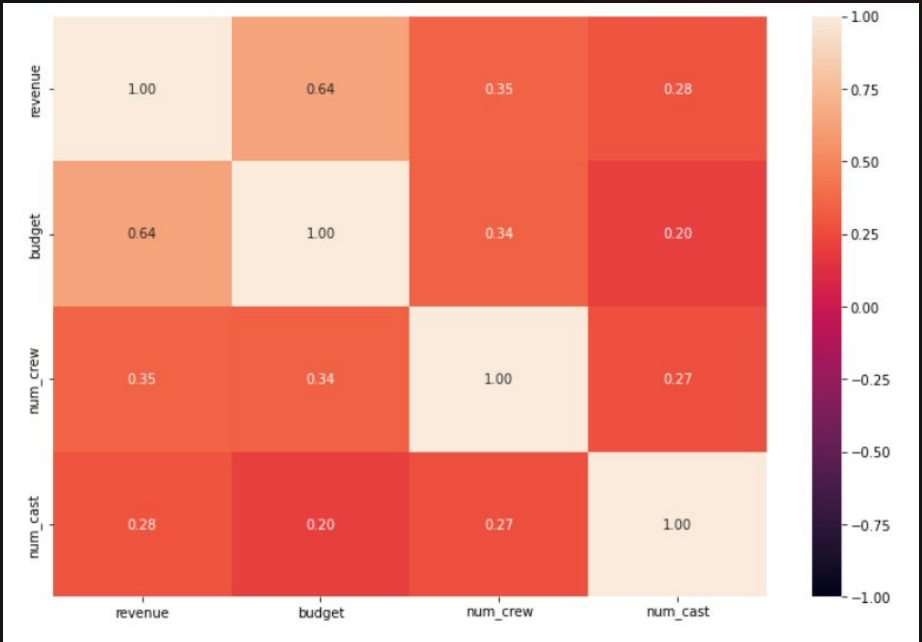
Pre-elements:
1. num_cast
2. num_crew
3. budget

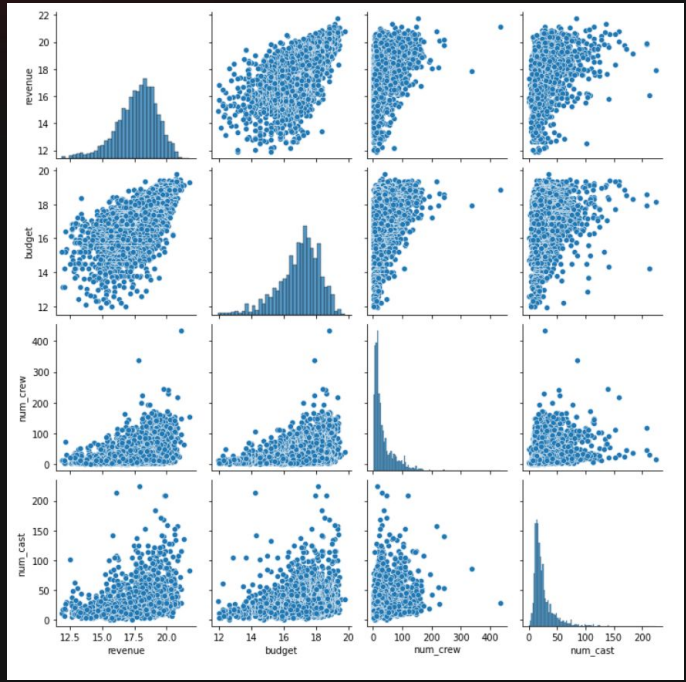
Post-elements:
1. Popularity
2. vote_count
3. vote_average ✗

sciencefiction	-0.203814
drama	-0.084991
family	-0.072423
animation	-0.070343
mystery	-0.067070
adventure	-0.057618
action	-0.031282
comedy	-0.027223
documentary	-0.005469
tvmovie	-0.005327
romance	-0.003028
history	0.009936
music	0.095573
vote_average	0.125474
war	0.161976
crime	0.163064
horror	0.169485
fantasy	0.195995
foreign	0.261552
num_cast	0.281805
num_crew	0.349295
budget	0.637824
popularity	0.651487
vote_count	0.707363
revenue	1.000000
western	NaN
thriller	NaN

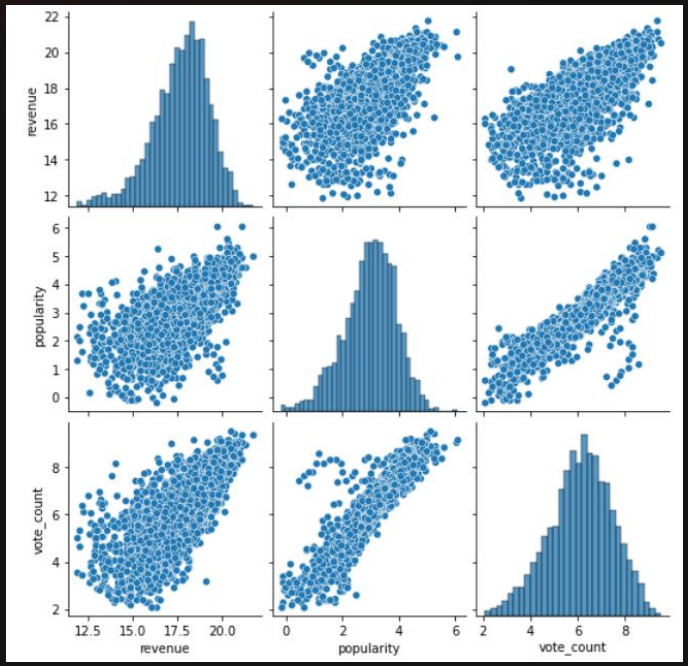
Name: revenue, dtype: float64

Multivariate Exploratory Analysis: Pre-elements





Multivariate Exploratory Analysis: Post-elements

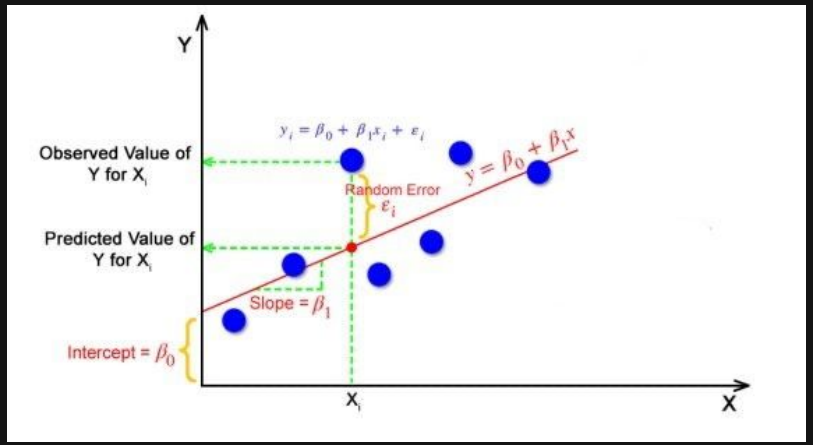


03

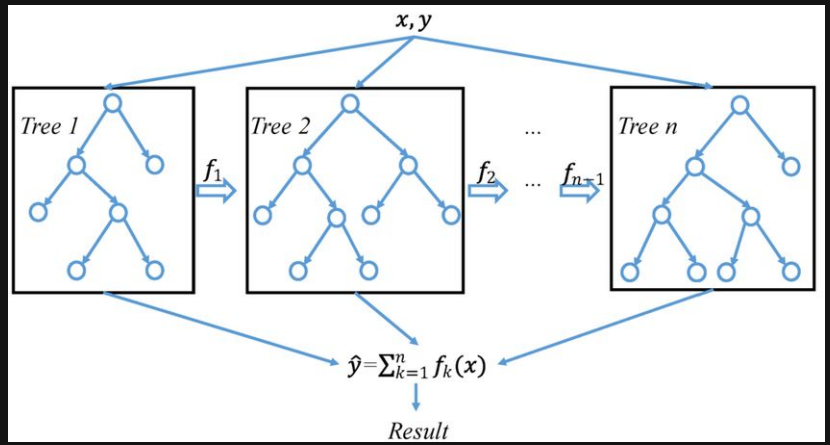
CORE ANALYSIS

Building our models for our prediction

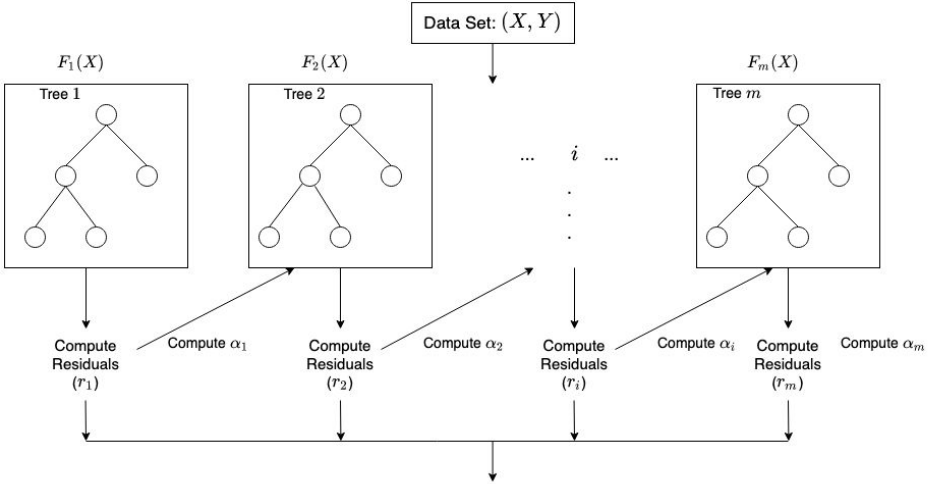
Model 1: Linear Regression



Model 2: Extreme Gradient Boosting (XGBoost)



About Extreme Gradient Boosting (XGBoost)

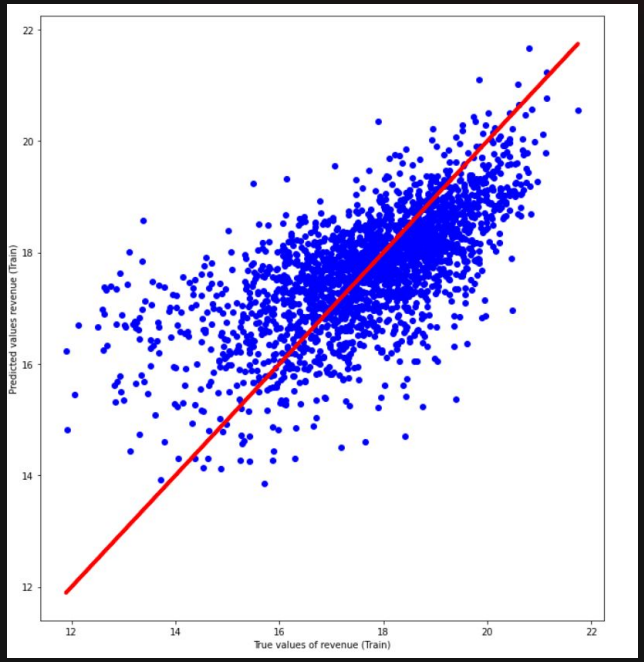


$$F_m(X) = F_{m-1}(X) + \alpha_m h_m(X, r_{m-1}),$$

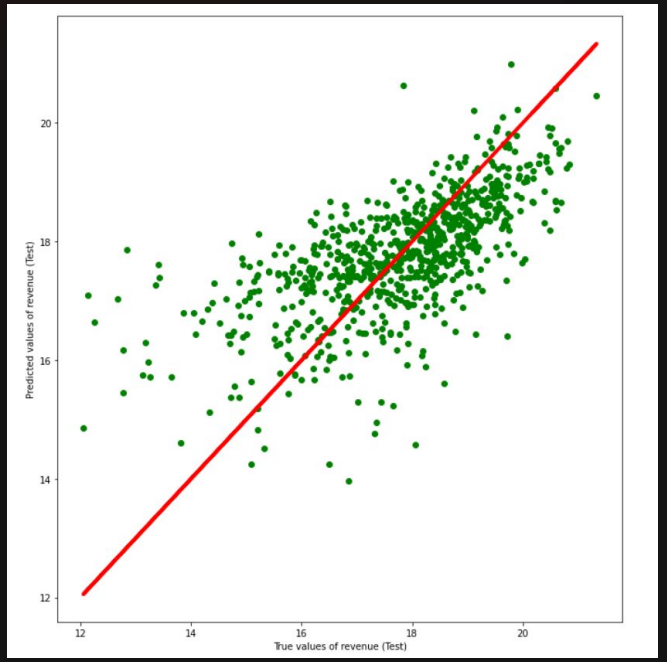
where α_i, and r_i are the regularization parameters and residuals computed with the i^{th} tree respectfully, and h_i is a function that is trained to predict residuals, r_i using X for the i^{th} tree. To compute α_i we use the residuals computed, r_i and compute the following: $arg \min_{\alpha} = \sum_{i=1}^{m} L(Y_i, F_{i-1}(X_i) + \alpha h_i(X_i, r_{i-1}))$ where $L(Y, F(X))$ is a differentiable loss function.

Linear Regression on Pre-production elements

```
Goodness of Fit of Model          Train Dataset
Explained Variance (R^2)          : 0.4433556114630096
Mean Squared Error (MSE)          : 1.354955983071493

Goodness of Fit of Model          Test Dataset
Explained Variance (R^2)          : 0.44262774888073997
Mean Squared Error (MSE)          : 1.4608688853249576
```



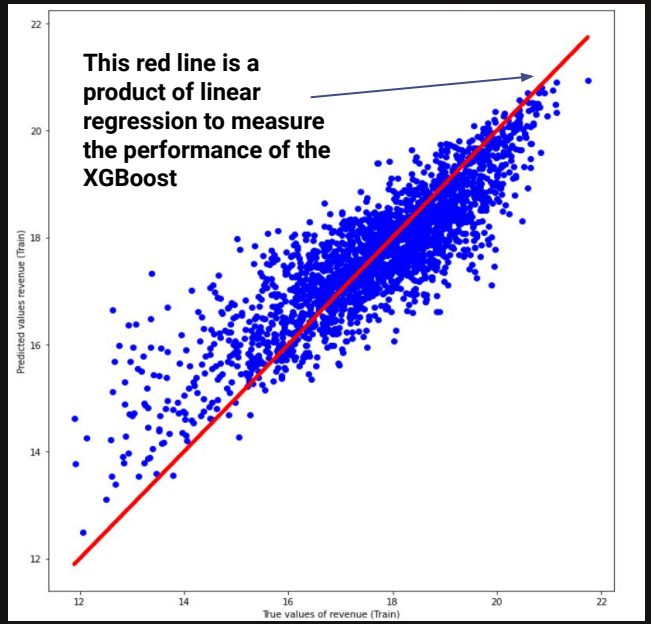
Train data set



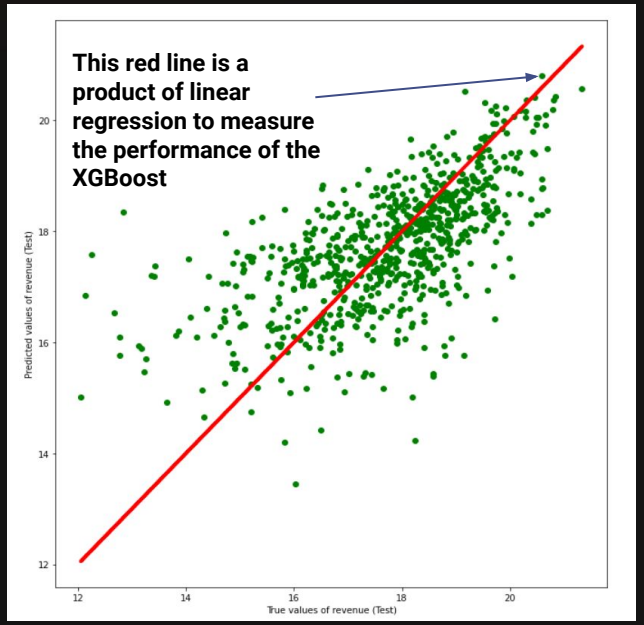
Test data set

XGBoost on Pre-production elements

```
Goodness of Fit of Model          Train Dataset
Explained Variance (R^2)          : 0.744739681438667
Mean Squared Error (MSE)          : 0.6213419249306414

Goodness of Fit of Model          Test Dataset
Explained Variance (R^2)          : 0.4026442394605799
Mean Squared Error (MSE)          : 1.5656653920055728
```



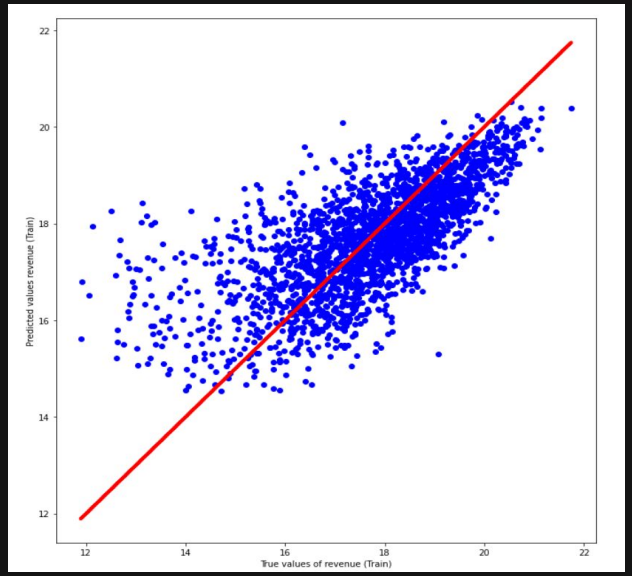
Train data set



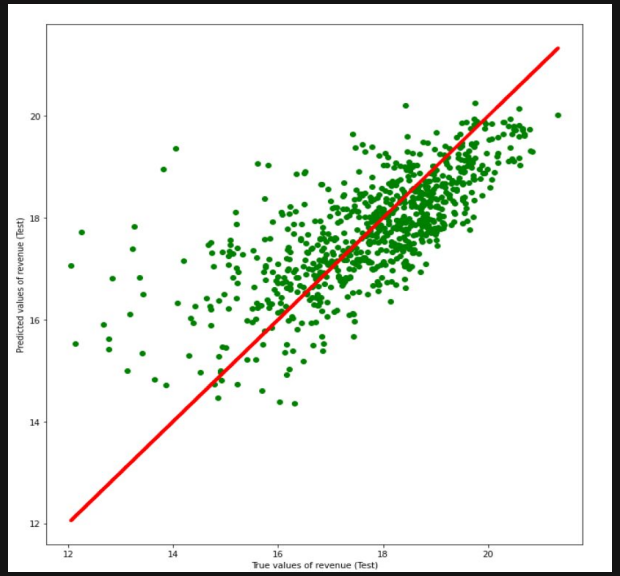
Test data set

Linear Regression on Post-production elements

```
Goodness of Fit of Model          Train Dataset
Explained Variance (R^2)          : 0.491441114305065
Mean Squared Error (MSE)          : 1.2379086524658875

Goodness of Fit of Model          Test Dataset
Explained Variance (R^2)          : 0.525010975377914
Mean Squared Error (MSE)          : 1.2449430081024675
```



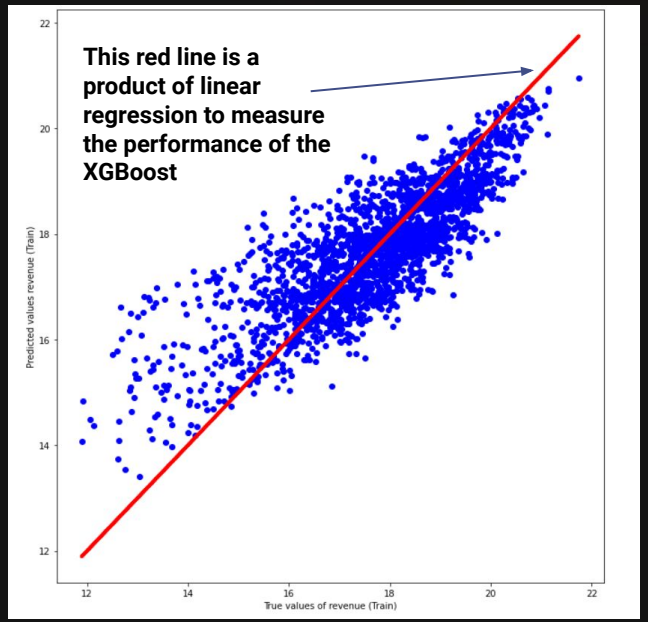
Train data set



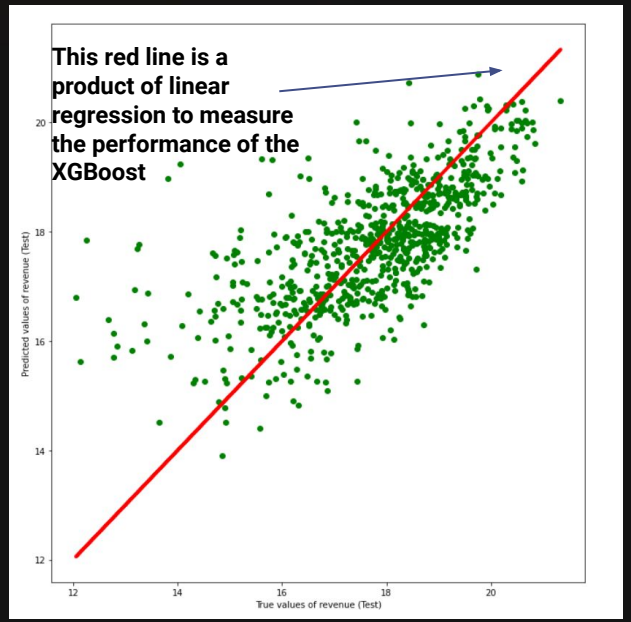
Test data set

XGBoost on Post-production elements

```
Goodness of Fit of Model          Train Dataset
Explained Variance (R^2)          : 0.6848402068755204
Mean Squared Error (MSE)          : 0.7671462357501346

Goodness of Fit of Model          Test Dataset
Explained Variance (R^2)          : 0.478093311495867
Mean Squared Error (MSE)          : 1.3679138865410347
```



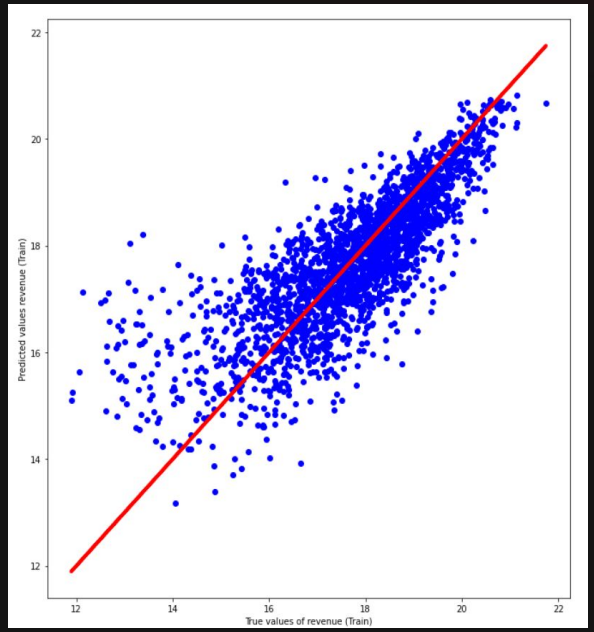
Train data set



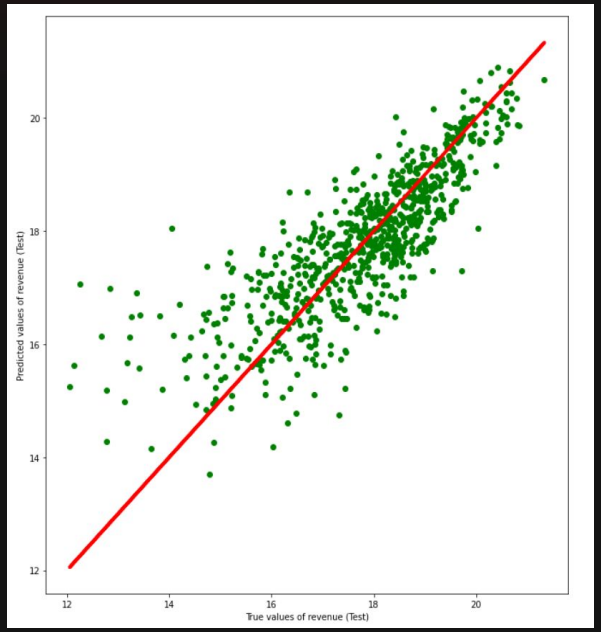
Test data set

Linear Regression on Pre & Post-production elements

```
Goodness of Fit of Model          Train Dataset
Explained Variance (R^2)          : 0.6480805676433314
Mean Squared Error (MSE)          : 0.8566247145399991

Goodness of Fit of Model          Test Dataset
Explained Variance (R^2)          : 0.6712293906070198
Mean Squared Error (MSE)          : 0.8617055346889937
```



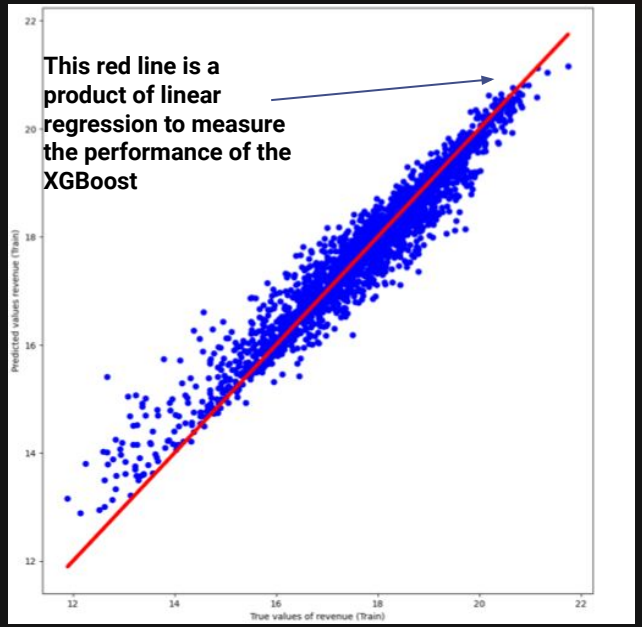
Train data set



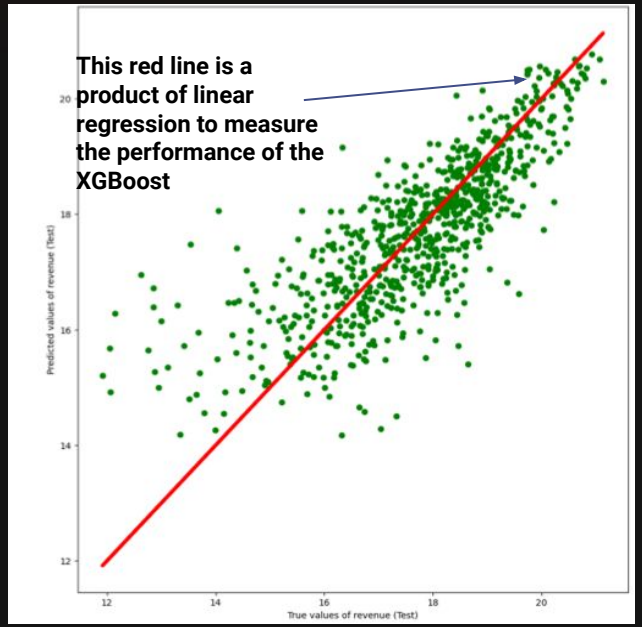
Test data set

XGBoost on Combination of Pre-production and Post-production elements

```
Goodness of Fit of Model          Train Dataset
Explained Variance (R^2)          : 0.926415280141226
Mean Squared Error (MSE)          : 0.17911625175515447

Goodness of Fit of Model          Test Dataset
Explained Variance (R^2)          : 0.6469888011559795
Mean Squared Error (MSE)          : 0.9252399550334754
```



Train data set



Test data set

XGBoost on Combination of Pre-production and Post-production elements - With Fine Tuning

Fine tune XGBoost

```
[ ] from sklearn.model_selection import GridSearchCV

    X_train = train.drop(['revenue'], axis = 1)
    y_train = pd.DataFrame(train['revenue'])
    X_test = test.drop(['revenue'], axis = 1)
    y_test = pd.DataFrame(test['revenue'])

    params = { 'max_depth': [4,7,10],
               'learning_rate': [0.01, 0.05, 0.1],
               'n_estimators': [100, 500, 1000],
               'colsample_bytree': [0.3, 0.7]}
    xgbr = xgboost.XGBRegressor(seed = 20)
    clf = GridSearchCV(estimator=xgbr,
                       param_grid=params,
                       scoring='neg_mean_squared_error',
                       verbose=1)
    clf.fit(X_train,y_train)
    print("Best parameters:", clf.best_params_)
    print("Lowest RMSE: ", (-clf.best_score_)**(1/2.0))
```

```
Fitting 5 folds for each of 54 candidates, totalling 270 fits
Best parameters: {'colsample_bytree': 0.7, 'learning_rate': 0.01, 'max_depth': 4, 'n_estimators': 100
Lowest RMSE:  0.9432090032427589
```

04 Outcome

Can a movie's success be determined ?

PERHAPS

What we have learnt

- Data Extraction & Cleaning
- Data Normalization
- Linear Regression Model
- XGBoost Algorithm
- Github
- ....

05 BONUS

MOVIE RECOMMENDATION :
CONTENT BASED FILTERING

Movie Recommendation System 1

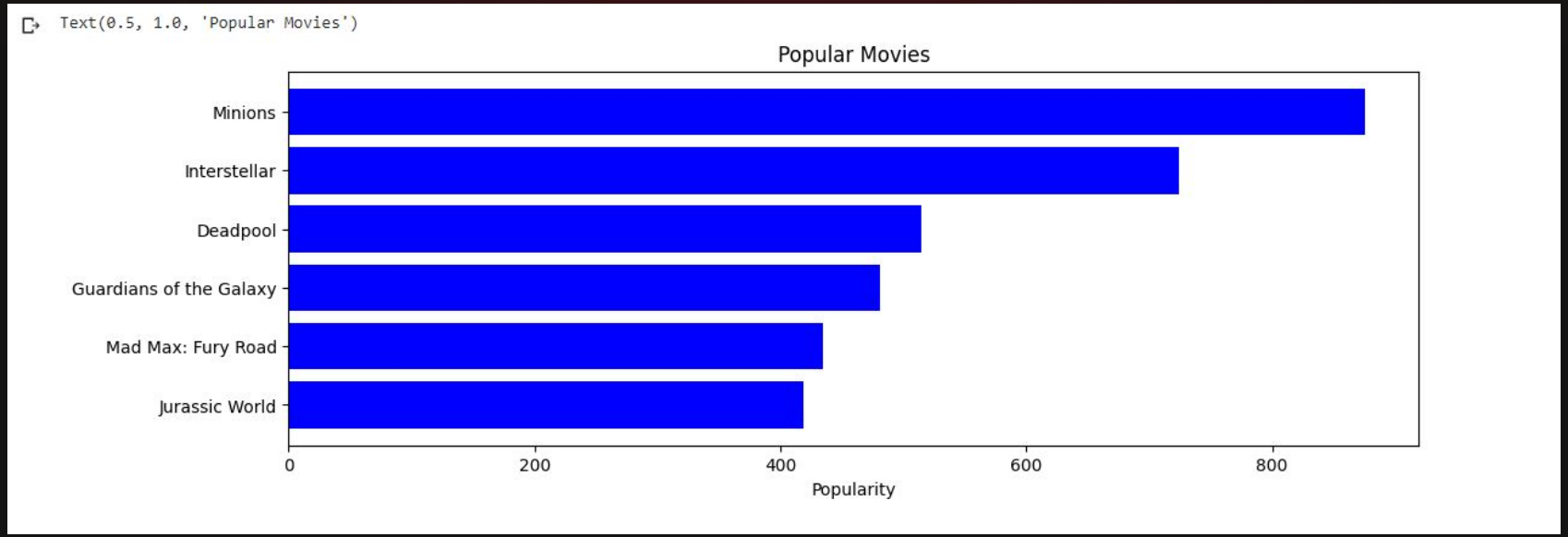
```
[70] C= moviesdata['vote_average'].mean()
     C

     6.092514036182159

[71] m= moviesdata['vote_count'].quantile(0.9)
     m

     1842.199999999998

[72] q_movies = moviesdata.copy().loc[moviesdata['vote_count'] >= m]
     q_movies.shape

     (481, 43)

[73] def weighted_rating(x, m=m, C=C):
         v = x['vote_count']
         R = x['vote_average']
         # Calculation based on the IMDB formula
         return (v/(v+m) * R) + (m/(m+v) * C)
```

- **Weighted rating calculated using the IMDB formula**

	title	vote_count	vote_average	score
1887	The Shawshank Redemption	8205	8.5	8.058576
662	Fight Club	9413	8.3	7.938689
65	The Dark Knight	12002	8.2	7.919564
3237	Pulp Fiction	8428	8.3	7.904036
96	Inception	13752	8.1	7.862848
3342	The Godfather	5893	8.4	7.850454
95	Interstellar	10867	8.1	7.809015
809	Forrest Gump	7927	8.2	7.802587
329	The Lord of the Rings: The Return of the King	8064	8.1	7.726679
1996	The Empire Strikes Back	5879	8.2	7.697175
262	The Lord of the Rings: The Fellowship of the Ring	8705	8.0	7.666834
2917	Star Wars	6624	8.1	7.663182
1824	Schindler's List	4329	8.3	7.641031
3872	Whiplash	4254	8.3	7.632924
330	The Lord of the Rings: The Two Towers	7487	8.0	7.623336

- **The top 15 movies based on the IMDB formula**

```
Text(0.5, 1.0, 'Popular Movies')
```



Top 6 Most Popular Movies

- Top 10 movie recommendations based on movie similarities and their cosine similarity scores based on the movie overviews

```
[81] get_recommendations('Avatar')

3609                        Apollo 18
2136                    The American
634                       The Matrix
1344           The Inhabited Island
529                Tears of the Sun
1616                           Hanna
311     The Adventures of Pluto Nash
847                         Semi-Pro
775                        Supernova
2634             Blood and Chocolate
Name: title, dtype: object
```

```
[82] get_recommendations('The Avengers')

7               Avengers: Age of Ultron
3149                            Plastic
1721                            Timecop
4131                This Thing of Ours
3316             Thank You for Smoking
3038                    The Corruptor
588     Wall Street: Money Never Sleeps
2142         Team America: World Police
1474                       The Fountain
1289                        Snowpiercer
Name: title, dtype: object
```

Movie Recommendation System 2

```
[87] moviesdata[['title', 'cast', 'director', 'keywords', 'genres']].head(3)
```

	title	cast	director	keywords	genres
0	Avatar	[Sam Worthington, Zoe Saldana, Sigourney Weaver]	James Cameron	[culture clash, future, space war]	[Action, Adventure, Fantasy]
1	Pirates of the Caribbean: At World's End	[Johnny Depp, Orlando Bloom, Keira Knightley]	Gore Verbinski	[ocean, drug abuse, exotic island]	[Adventure, Fantasy, Action]
2	Spectre	[Daniel Craig, Christoph Waltz, Léa Seydoux]	Sam Mendes	[spy, based on novel, secret agent]	[Action, Adventure, Crime]

- To improve the recommendation system, we redefined the cosine similarity matrix based on adding ‘cast’, ‘director’, ‘keywords’, and ‘genres’ features.

```
get_recommendations('The Avengers', cosine_sim2)
```

```
7                  Avengers: Age of Ultron
26               Captain America: Civil War
79                               Iron Man 2
169      Captain America: The First Avenger
174                    The Incredible Hulk
85      Captain America: The Winter Soldier
31                               Iron Man 3
33                  X-Men: The Last Stand
68                                 Iron Man
94                  Guardians of the Galaxy
Name: title, dtype: object
```

```
get_recommendations('The Dark Knight Rises', cosine_sim2)
```

```
65                 The Dark Knight
119                  Batman Begins
4644    Amidst the Devil's Wings
1199                 The Prestige
3078           Romeo Is Bleeding
3331              Black November
1509                       Takers
1992                       Faster
303                      Catwoman
747                Gangster Squad
Name: title, dtype: object
```

- More accurate and relevant recommendations as it is based on a more comprehensive set of features
- Eg. “The Dark Knight Rises” does not even have “Batman” keyword but the recommendation system can suggest the film “Batman Begins”

THANKS!

Do you have any questions?

youremail@freepik.com
+91 620 421 838
yourwebsite.com