UNITEDWORLD SCHOOL OF TECHNOLOGY (UIT)

Summative Assessment (SA)

Submitted BY

KHUSHI PAGARIA

(Enrl. No.: 20220701059)

Course Code and Title: -21BSAI99E32

INTRODUCTION TO MACHINE LEARNING

B.Sc. (Hons.) Computer Science / Data Science / AIML

III Semester – July – Nov 2023

Nov/Dec 2023

INDEX

Predicting Box Office Revenue Using TMDB Dataset

Project Objective:

The aim of this project is to develop a machine learning model that can forecast the box office revenue of films according to many criteria such as budget, genre, release date, runtime, and so on.
Description of the dataset:
As part of the research, a dataset containing information on different movies, including their box office earnings, release dates, budgets, and genres, will be analyzed. A prediction model is developed using the dataset to project the potential box office receipts that a movie might earn upon its release. The model will employ linear regression and the Random Forest method as machine learning techniques to estimate revenue after being trained on historical data.

DATASET DESCRIPTION

id: A special identification for every film.
belongs_to_collection: If appropriate, details about the film series that the film is a part of.
budget: The sum allotted to the film's production.
genre(s): The film's chosen genre(s).
homepage: If the movie's homepage URL is available.
The movie's IMDb ID is imdb_id.
original_language: The film's native tongue.
original title: Originally intended title of the film.
summary: A concise synopsis or summary of the film.
popularity: An indicator of how well-liked the film is.
release_date: The movie's release date.
runtime: The number of minutes the film lasts.
Spoken Languages: The languages that are used during the film.
status: The film's current state (e.g., released).
tagline: The film's tagline or slogan.
title: The movie's title.
Keywords: Terms connected to the film.
cast: Details regarding the film's cast.
crew: Details about the film's crew, such as the director and producer.
revenue: The movie's box office receipts (target variable).

Importing Libraries:

importing necessary libraries such as numpy, pandas, matplotlib, seaborn, and scikit-learn modules for data manipulation, visualization, and machine learning tasks.

```
[56] import numpy as np
     import pandas as pd
     import matplotlib.pyplot as plt
     import seaborn as sns
     import ast
     from sklearn.ensemble import RandomForestRegressor # ML
     from sklearn.inspection import permutation_importance # computing feature importance
     from sklearn import preprocessing
     from sklearn.model_selection import train_test_split
     from sklearn.metrics import mean_absolute_error
     from xgboost import XGBRegressor
     import missingno as msno
```

Loading Data:

load the training and testing datasets from CSV files into Pandas DataFrame objects.

```
[57] train = pd.read_csv('train.csv')
     test = pd.read_csv('test.csv')
```

```
train.head()
```

	id	belongs_to_collection	budget	genres	homepage	imdb_id	original_language	original_title	overview	popularity	...	release_date	runtime	spoken_languages	status	tagline	title	Keywords	cast	crew	revenue
0	1	[{'id': 313576, 'name': 'Hot Tub Time Machine ...	14000000	[{'id': 35, 'name': 'Comedy'}]	NaN	tt2637294	en	Hot Tub Time Machine 2	When Lou, who has become the "father of the In...	6.575393	...	2/20/15	93.0	[{'iso_639_1': 'en', 'name': 'English'}]	Released	The Laws of Space and Time are About to be Vio...	Hot Tub Time Machine 2	[{'id': 4379, 'name': 'time travel'}, {'id': 9...	[{'cast_id': 4, 'character': 'Lou', 'credit_id...	[{'credit_id': '59ac067c92514107aff02c8c8', 'de...	12314651
1	2	[{'id': 107674, 'name': 'The Princess Diaries ...	40000000	[{'id': 35, 'name': 'Comedy'}, {'id': 18, 'nam...	NaN	tt0368933	en	The Princess Diaries 2: Royal Engagement	Mia Thermopolis is now a college graduate and ...	8.248895	...	8/6/04	113.0	[{'iso_639_1': 'en', 'name': 'English'}]	Released	It can take a lifetime to find true love; she'...	The Princess Diaries 2: Royal Engagement	[{'id': 2505, 'name': 'coronation'}, {'id': 42...	[{'cast_id': 1, 'character': 'Mia Thermopolis'...	[{'credit_id': '52fe43fe9251416c7502563d', 'de...	95149435
2	3	NaN	3300000	[{'id': 18, 'name': 'Drama'}]	http://sonyclassics.com/whiplash/	tt2582802	en	Whiplash	Under the direction of a ruthless instructor, ...	64.299990	...	10/10/14	105.0	[{'iso_639_1': 'en', 'name': 'English'}]	Released	The road to greatness can take you to the edge.	Whiplash	[{'id': 1416, 'name': 'jazz'}, {'id': 1523, 'n...	[{'cast_id': 5, 'character': 'Andrew Neimann',...	[{'credit_id': '54c5356ec3a3683da0000039', 'de...	13092000
3	4	NaN	1200000	[{'id': 53, 'name': 'Thriller'}, {'id': 18, 'n...	http://kahaanithefilm.com/	tt1821480	hi	Kahaani	Vidya Bagchi (Vidya Balan) arrives in Kolkata ...	3.174936	...	3/9/12	122.0	[{'iso_639_1': 'en', 'name': 'English'}, {'iso...	Released	NaN	Kahaani	[{'id': 10092, 'name': 'mystery'}, {'id': 1054...	[{'cast_id': 1, 'character': 'Vidya Bagchi',' ...	[{'credit_id': '52fe48779251416c9108d6eb', 'de...	16000000
4	5	NaN	0	[{'id': 28, 'name': 'Action'}, {'id': 53, 'nam...	NaN	tt1380152	ko	마른 보이	Marine Boy is the story of a former national s...	1.148070	...	2/5/09	118.0	[{'iso_639_1': 'ko', 'name': '한국어/조선말'}]	Released	NaN	Marine Boy	NaN	[{'cast_id': 3, 'character': 'Chun-soo', 'cred...	[{'credit_id': '52fe464b9251416c75073b43', 'de...	3923970

5 rows × 23 columns

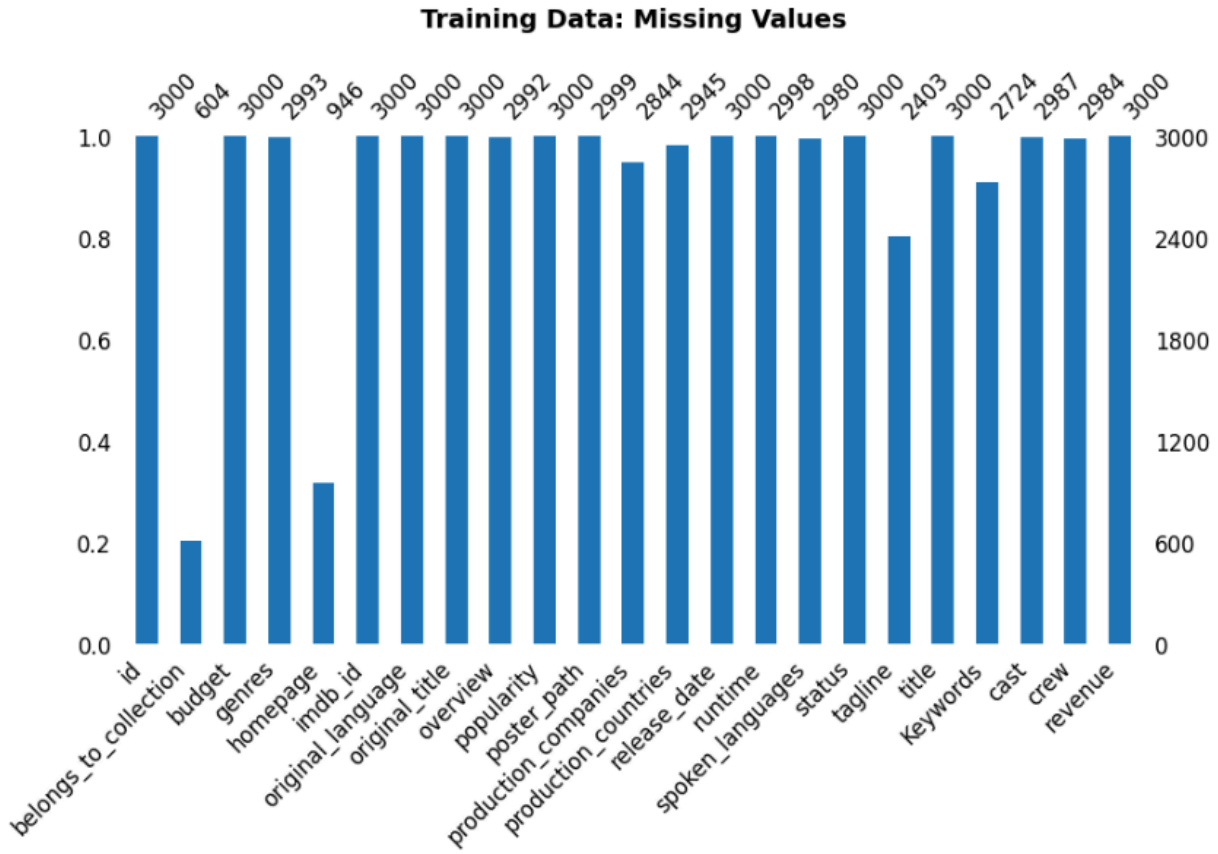
```
[59] train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3000 entries, 0 to 2999
Data columns (total 23 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   id                     3000 non-null   int64  
 1   belongs_to_collection  604 non-null    object 
 2   budget                 3000 non-null   int64  
 3   genres                 2993 non-null   object 
 4   homepage               946 non-null    object 
 5   imdb_id                3000 non-null   object 
 6   original_language      3000 non-null   object 
 7   original_title         3000 non-null   object 
 8   overview               2992 non-null   object 
 9   popularity             3000 non-null   float64
 10  poster_path            2999 non-null   object 
 11  production_companies   2844 non-null   object 
 12  production_countries   2945 non-null   object 
 13  release_date           3000 non-null   object 
 14  runtime                2998 non-null   float64
 15  spoken_languages       2980 non-null   object 
 16  status                 3000 non-null   object 
 17  tagline                2403 non-null   object 
 18  title                  3000 non-null   object 
 19  Keywords               2724 non-null   object 
 20  cast                   2987 non-null   object 
 21  crew                   2984 non-null   object 
 22  revenue                3000 non-null   int64  
dtypes: float64(2), int64(3), object(18)
memory usage: 539.2+ KB
```

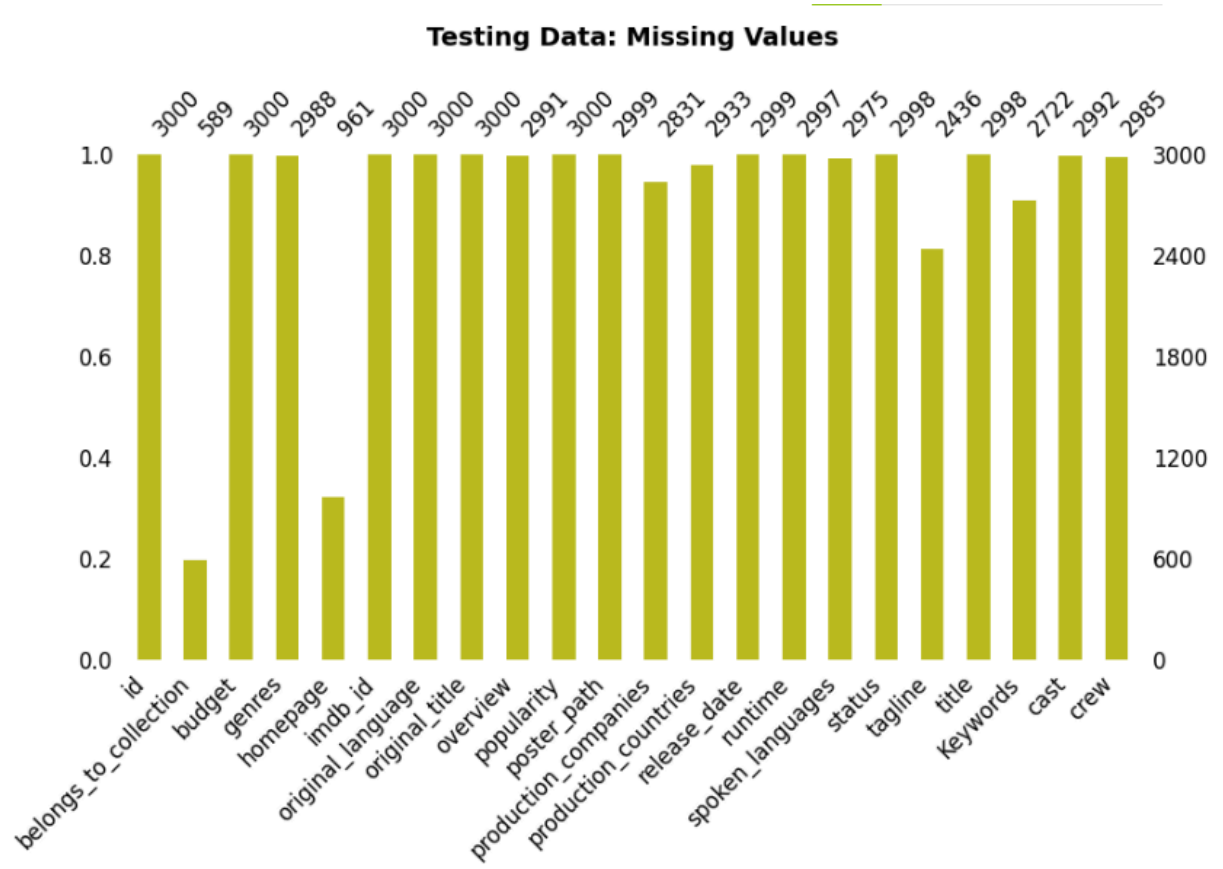
Handling Missing Data

visualizing missing values in the training and testing datasets using the `missingno` library. Then filling the missing values

```
train_miss = msno.bar(train, figsize=(10,5), fontsize=12, color= 'C0').set_title('Training Data: Missing Values' '\n',
                                                                                  fontweight="bold", fontsize=14)
```



```
test_miss = msno.bar(test, figsize=(10,5), fontsize=12, color= 'C8').set_title('Testing Data: Missing Values' '\n', fontweight="bold", fontsize=14)
```



converting the 'release_date' column to a datetime format and extracting features like year, day, and month from it.

```
test[test["release_date"].isnull()]
```

	id	belongs_to_collection	budget	genres	homepage	imdb_id	original_language	original_title	overview	popularity	...	production_countries	release_date	runtime	spoken_languages	status	tagline	title	Keywords	cast	crew
828	3829	NaN	0	[{'id': 18, 'name': 'Drama'}]	NaN	tt0210130	en	Jails, Hospitals & Hip-Hop	Jails, Hospitals & Hip-Hop is a cinematic ...	0.009057	...	NaN	NaN	90.0	NaN	NaN	three worlds / two million voices / one genera...	Jails, Hospitals & Hip-Hop	NaN	[]	NaN

1 rows × 22 columns

```
# Addin the release date 05/01/2020, which I found through a quick online search
test.loc[test['release_date'].isnull()==True, 'release_date']= '5/1/00'
test[test["release_date"]== '5/1/00']
```

	id	belongs_to_collection	budget	genres	homepage	imdb_id	original_language	original_title	overview	popularity	...	production_countries	release_date	runtime	spoken_languages	status	tagline	title	Keywords	cast
828	3829		NaN	0	[{'id': 18, 'name': 'Drama'}]	NaN	tt0210130	en	Jails, Hospitals & Hip-Hop	Jails, Hospitals & Hip-Hop is a cinematic ...	0.009057	...	NaN	5/1/00	90.0	NaN	NaN	three worlds / two million voices / one genera...	Jails, Hospitals & Hip-Hop	NaN	[]

1 rows × 22 columns

```
# For nominal data, replacing the missing values with "none"

train[['genres',
       'original_language',
       'spoken_languages',
       'status',
       'production_countries',
       'production_companies',
       'cast',
       'crew']] = train[['genres',
         'original_language',
         'spoken_languages',
         'status',
         'production_countries',
         'production_companies',
         'cast',
         'crew']].fillna("none")
```

```
test[['genres',
       'original_language',
       'spoken_languages',
       'status',
       'production_countries',
       'production_companies',
       'cast',
       'crew']] = test[['genres',
         'original_language',
         'spoken_languages',
         'status',
         'production_countries',
         'production_companies',
         'cast',
         'crew']].fillna("none")
```

```
# For numerical data, replacing the missing values with the mean
train['runtime'] = train['runtime'].fillna(train['runtime'].mean())
test['runtime'] = test['runtime'].fillna(train['runtime'].mean())
train['runtime'].isnull().any()
```

```
False
```

```
# Converting the format of the date and creating new year, day, and month columns

train['release_date'] = pd.to_datetime(train['release_date'])
test['release_date'] = pd.to_datetime(test['release_date'])

train["release_year"] = pd.to_datetime(train["release_date"]).dt.year.astype(int)
train["release_day"] = pd.to_datetime(train["release_date"]).dt.dayofweek.astype(int)
train["release_month"] = pd.to_datetime(train["release_date"]).dt.month.astype(int)
test["release_year"] = pd.to_datetime(test["release_date"]).dt.year.astype(int)
test["release_day"] = pd.to_datetime(test["release_date"]).dt.dayofweek.astype(int)
test["release_month"] = pd.to_datetime(test["release_date"]).dt.month.astype(int)
```

```
# Since this competition was in 2019, there shouldn't be a release that after 2019
train['release_year'].max()
```

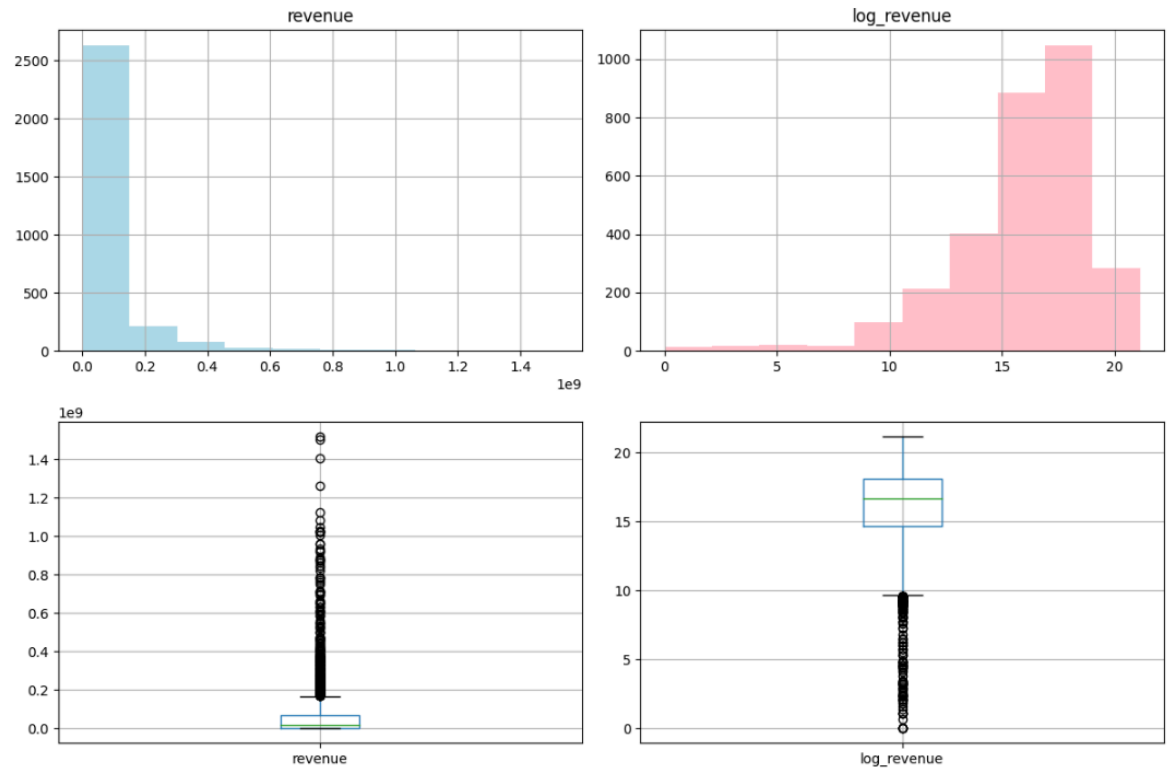
```
2073
```

Exploratory Data Analysis (EDA):

the distribution of key variables such as revenue, budget, popularity, runtime, and release year through histograms, boxplots, and time series plots.

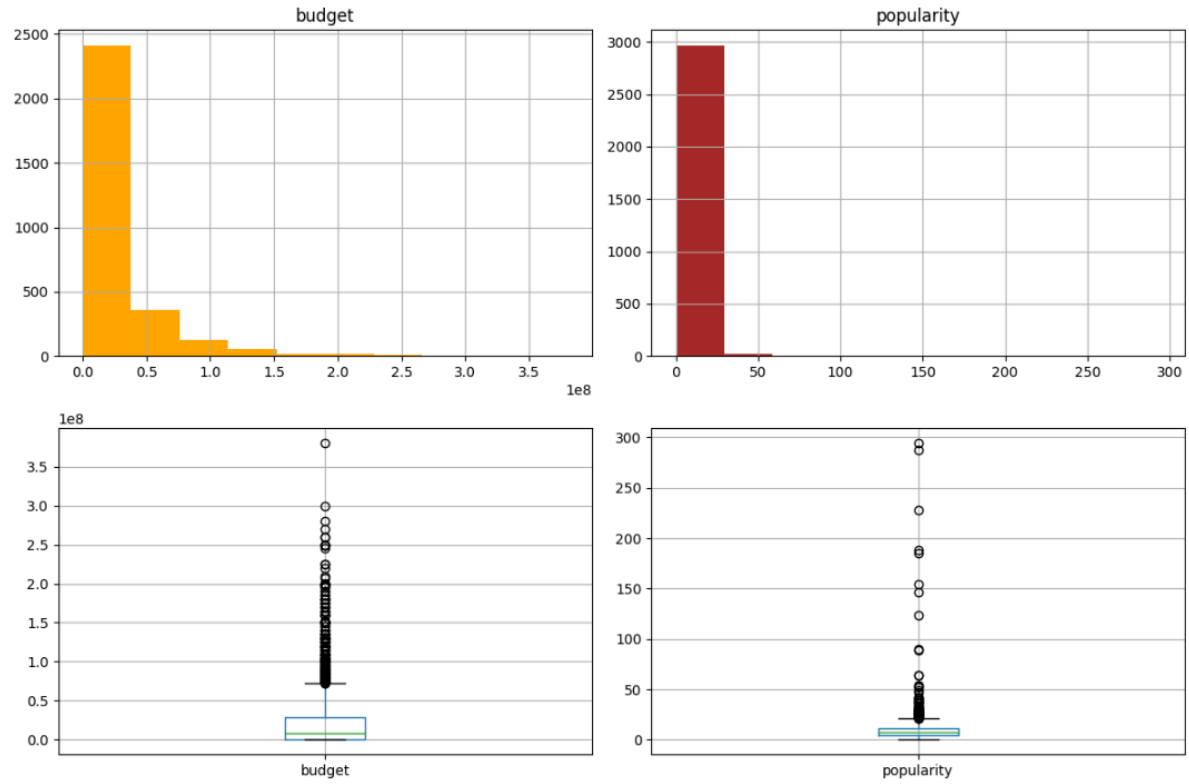
```
# Univariate Analysis: Revenue
train['log_revenue'] = np.log(train['revenue'])

fig, ax = plt.subplots(2, 2, figsize = (12, 8), tight_layout=True)
train.hist(column= ["revenue"], ax=ax[0][0], color='lightblue')
train.hist(column= ['log_revenue'], ax=ax[0][1], color='pink')
train.boxplot(column= ["revenue"], ax=ax[1][0])
train.boxplot(column= ['log_revenue'], ax=ax[1][1])
```

```
<Axes: >
```



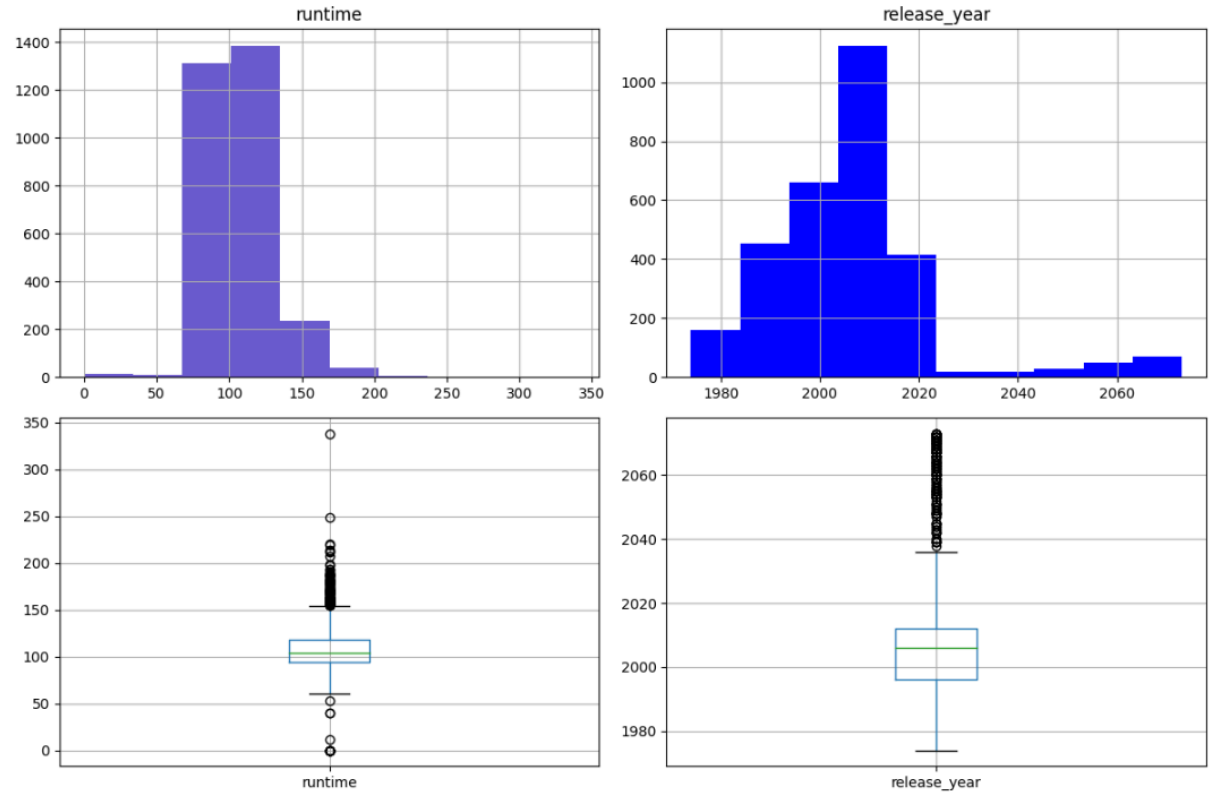
```
# Univariate Analysis: Budget & Popularity

fig, ax = plt.subplots(2, 2, figsize = (12, 8), tight_layout=True)
train.hist(column= ["budget"], ax=ax[0][0], color='orange')
train.hist(column= ['popularity'], ax=ax[0][1], color='brown')
train.boxplot(column= ['budget'], ax=ax[1][0])
train.boxplot(column= ['popularity'], ax=ax[1][1])
```

```
<Axes: >
```

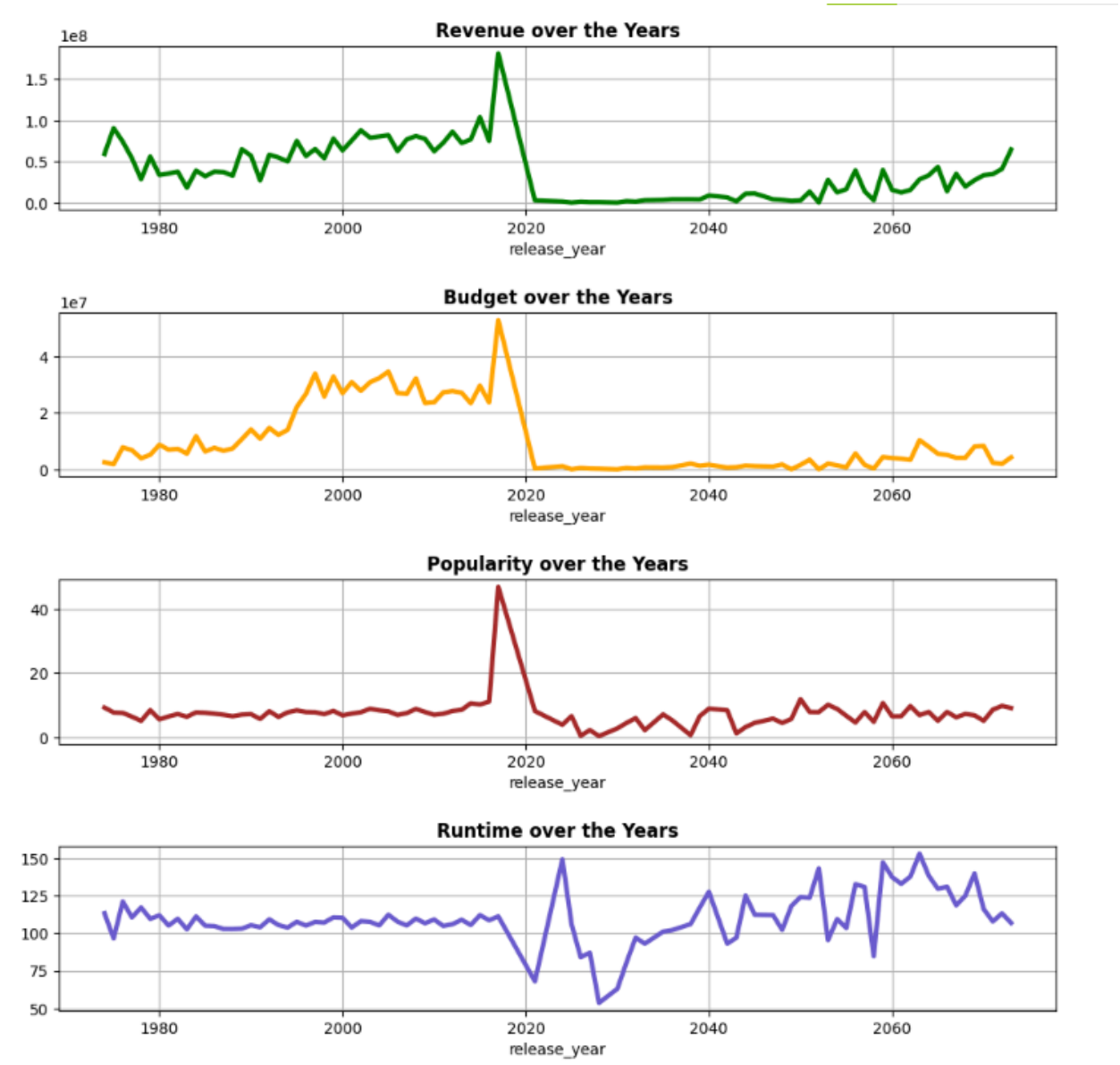


```
# Univariate Analysis: Runtime & Release Year

fig, ax = plt.subplots(2, 2, figsize = (12, 8), tight_layout=True)
train.hist(column= ["runtime"], ax=ax[0][0], color='slateblue')
train.hist(column= ['release_year'], ax=ax[0][1], color='blue')
train.boxplot(column= ['runtime'], ax=ax[1][0])
train.boxplot(column= ['release_year'], ax=ax[1][1])
```

```
<Axes: >
```



```
fig, ax = plt.subplots(4, 1, tight_layout=True)
plt.grid()

train.groupby('release_year')['revenue'].mean().plot(ax=ax[0], figsize=(10, 10), linewidth=3, color='green').set_title('Revenue over the Years', fontweight="bold")
ax[0].grid()

train.groupby('release_year')['budget'].mean().plot(ax=ax[1], figsize=(10, 10), linewidth=3, color='orange').set_title('Budget over the Years', fontweight="bold")
ax[1].grid()

train.groupby('release_year')['popularity'].mean().plot(ax=ax[2], figsize=(10, 10), linewidth=3, color='brown').set_title('Popularity over the Years', fontweight="bold")
ax[2].grid()

train.groupby('release_year')['runtime'].mean().plot(ax=ax[3], figsize=(10, 10), linewidth=3, color='slateblue').set_title('Runtime over the Years', fontweight="bold")
ax[3].grid()

fig.tight_layout(pad=2.0)
plt.show()
```



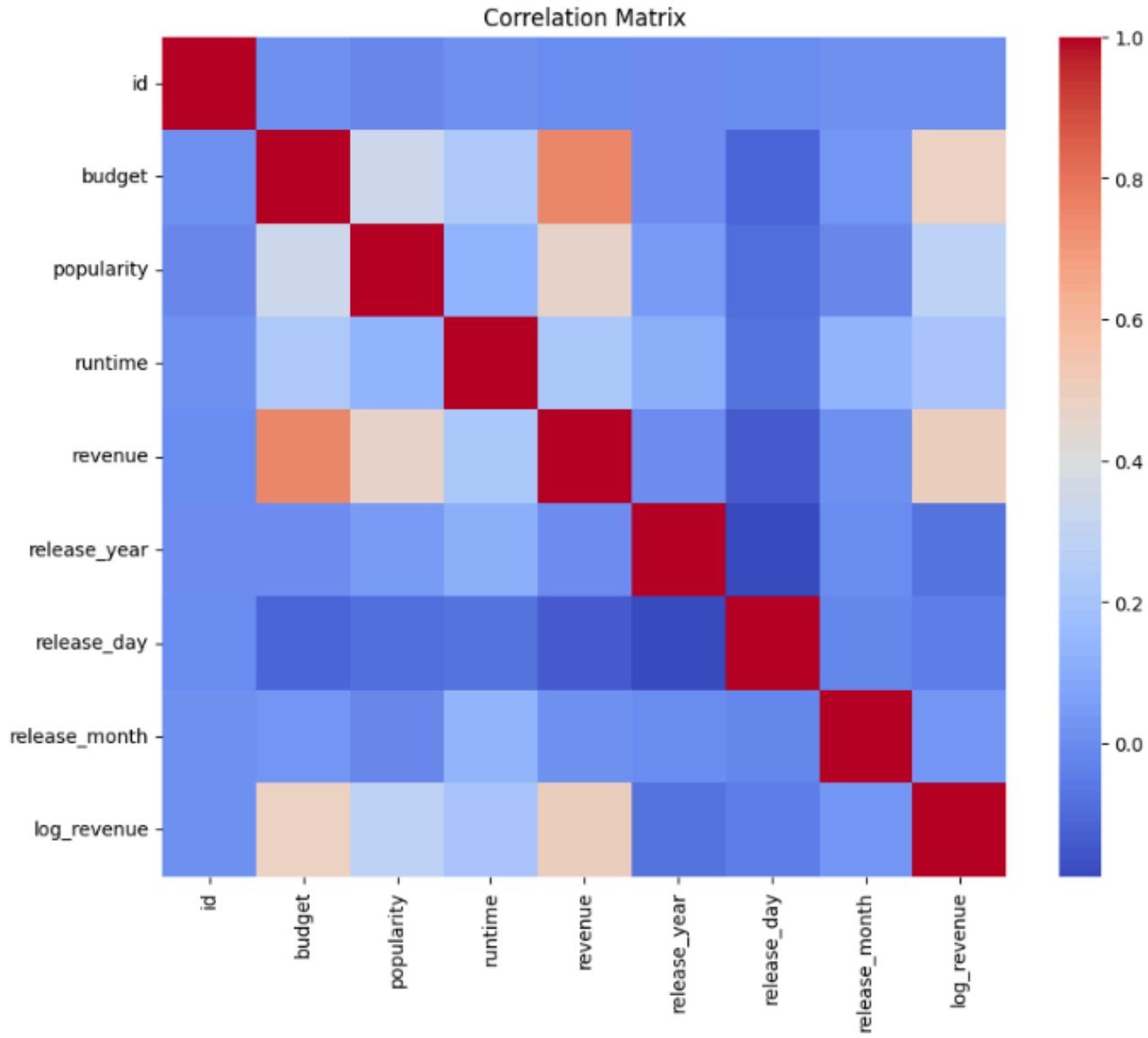
```
fig, ax = plt.subplots(2, 3, figsize=(15, 7), tight_layout=True)

train.plot(ax=ax[0][0], x='budget', y='revenue', style='o', ylabel= 'revenue', color='lightgreen').set_title('Revenue & Budget', fontweight="bold")
ax[0][0].grid()

train.plot(ax=ax[0][1], x='popularity', y='revenue', style='o', ylabel= 'revenue').set_title('Revenue & Popularity', fontweight="bold")
ax[0][1].grid()

train.plot(ax=ax[0][2], x='runtime', y='revenue', style='o', ylabel= 'revenue', color='slateblue').set_title('Revenue & Runtime', fontweight="bold")
ax[0][2].grid()

train.plot(ax=ax[1][0], x='budget', y='popularity', style='o', ylabel= 'popularity', color='salmon').set_title('Popularity & Budget', fontweight="bold")
ax[1][0].grid()

train.plot(ax=ax[1][1], x='runtime', y='budget', style='o', ylabel= 'budget', color='DarkBlue').set_title('Runtime & Budget', fontweight="bold")
ax[1][1].grid()

train.plot(ax=ax[1][2], x='runtime', y='popularity', style='o', ylabel= 'popularity', color='pink').set_title('Popularity & Runtime', fontweight="bold")
ax[1][2].grid()

fig.tight_layout(pad=4.0)
plt.show()
```

Statical Data Analysis

```
# Descriptive statistics
print(train.describe())
```

```
                id        budget   popularity      runtime       revenue  \
count  3000.000000  3.000000e+03  3000.000000  3000.000000  3.000000e+03
mean   1500.500000  2.253133e+07     8.463274   107.856571  6.672585e+07
std     866.169729  3.702609e+07    12.104000    22.079069  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

       release_year  release_day  release_month  log_revenue
count   3000.000000  3000.000000    3000.000000  3000.000000
mean    2005.879667     3.247333       6.775333    15.959894
std       16.765188     1.332949       3.409115     3.071323
min     1974.000000     0.000000       1.000000     0.000000
25%     1996.000000     2.000000       4.000000    14.682516
50%     2006.000000     4.000000       7.000000    16.637310
75%     2012.000000     4.000000      10.000000    18.048445
max     2073.000000     6.000000      12.000000    21.141685
```

```
# Correlation analysis
correlation_matrix = train.corr()
plt.figure(figsize=(10, 8))  # Adjust the size of the heatmap
sns.heatmap(correlation_matrix, cmap='coolwarm', annot=False)  # Remove annotations for simplicity
plt.title('Correlation Matrix')
plt.show()
```

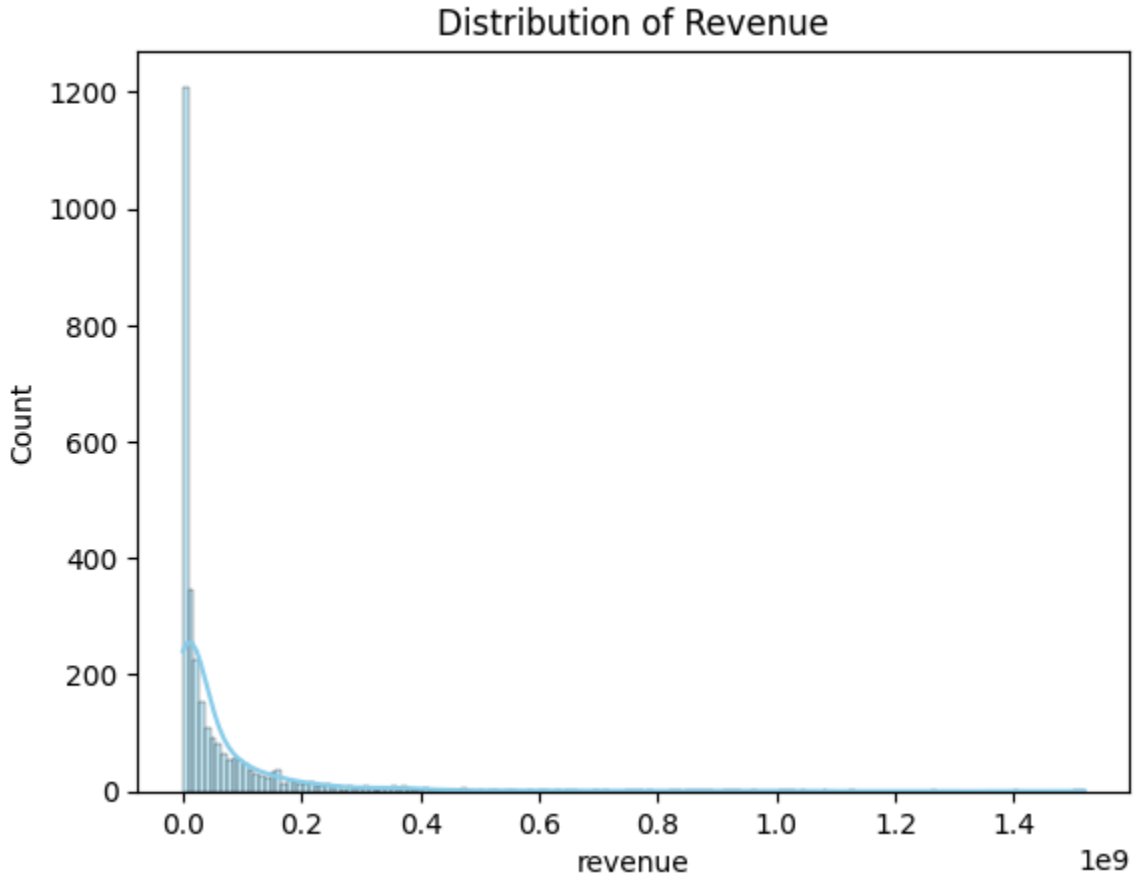



```
[75] # Hypothesis testing (Example: t-test)
     from scipy.stats import ttest_ind
     group1 = train[train['original_language'] == 'English']['revenue']
     group2 = train[train['original_language'] != 'English']['revenue']
     t_stat, p_value = ttest_ind(group1, group2)
     print("T-statistic:", t_stat)
     print("P-value:", p_value)
```

```
T-statistic: nan
P-value: nan
```

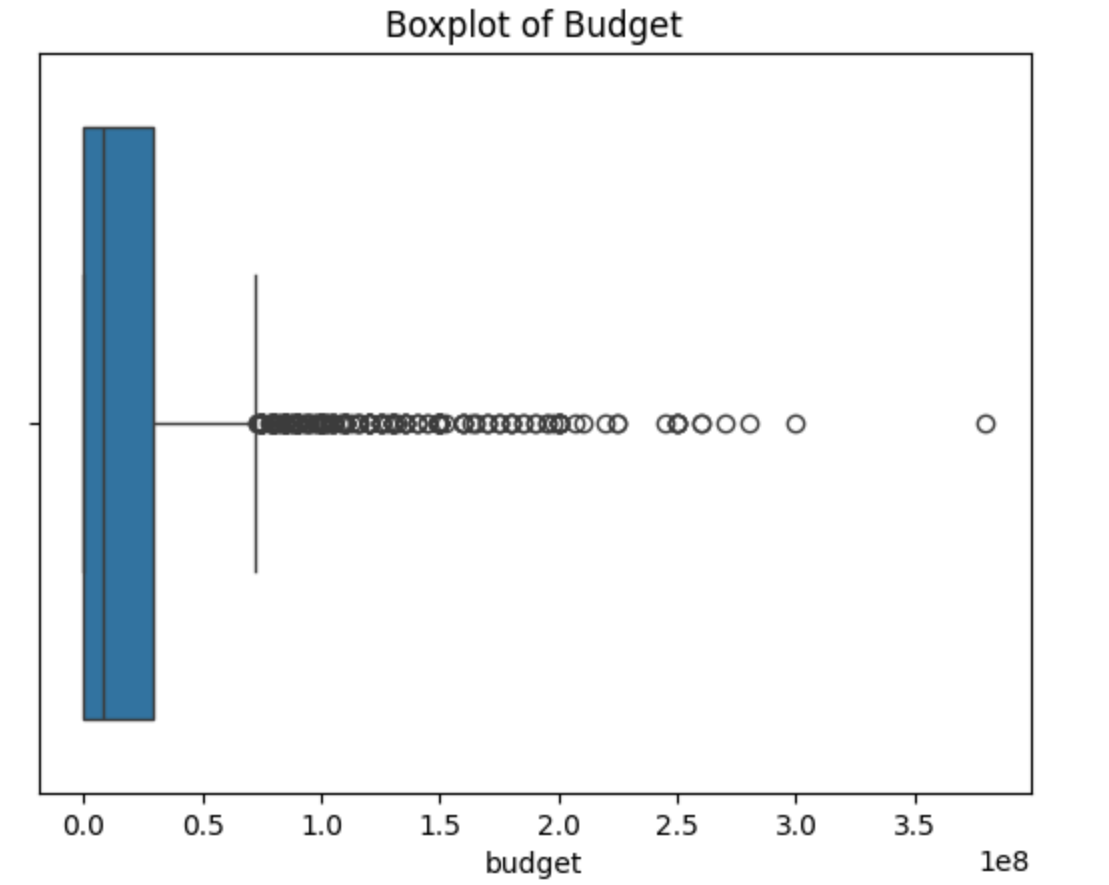
```
# Distribution analysis
sns.histplot(train['revenue'], kde=True, color='skyblue')
plt.title('Distribution of Revenue')
```

```
Text(0.5, 1.0, 'Distribution of Revenue')
```




```
# Outlier detection
sns.boxplot(x=train['budget'])
plt.title('Boxplot of Budget')
```

```
Text(0.5, 1.0, 'Boxplot of Budget')
```



Feature Engineering

```
# Many features are in json format.
for e in enumerate(test['genres'][:10]):
    print(e)
```

```
(0, "[{'id': 12, 'name': 'Adventure'}, {'id': 16, 'name': 'Animation'}, {'id': 10751, 'name': 'Family'}, {'id': 14, 'name': 'Fantasy'}]")
(1, "[{'id': 27, 'name': 'Horror'}, {'id': 878, 'name': 'Science Fiction'}]")
(2, "[{'id': 35, 'name': 'Comedy'}, {'id': 10749, 'name': 'Romance'}]")
(3, "[{'id': 18, 'name': 'Drama'}, {'id': 10752, 'name': 'War'}, {'id': 9648, 'name': 'Mystery'}]")
(4, "[{'id': 36, 'name': 'History'}, {'id': 99, 'name': 'Documentary'}]")
(5, "[{'id': 35, 'name': 'Comedy'}, {'id': 18, 'name': 'Drama'}]")
(6, "[{'id': 10749, 'name': 'Romance'}, {'id': 18, 'name': 'Drama'}, {'id': 35, 'name': 'Comedy'}]")
(7, "[{'id': 16, 'name': 'Animation'}, {'id': 10751, 'name': 'Family'}]")
(8, "[{'id': 18, 'name': 'Drama'}, {'id': 10749, 'name': 'Romance'}]")
(9, "[{'id': 16, 'name': 'Animation'}, {'id': 35, 'name': 'Comedy'}, {'id': 10751, 'name': 'Family'}]")
```

```
# First, I am converting the features in Json format to nominal format
def get_dictionary(s):
    try:
        d = eval(s)
    except:
        d = {}
    return d
```

```
[80] train.genres = train.genres.map(lambda x: sorted([d['name'] for d in get_dictionary(x)])).map(lambda x: ','.join(map(str, x)))
     train.spoken_languages = train.spoken_languages.map(lambda x: sorted([d['name'] for d in get_dictionary(x)])).map(lambda x: ','.join(map(str, x)))
     train.cast = train.cast.map(lambda x: sorted([d['name'] for d in get_dictionary(x)])).map(lambda x: ','.join(map(str, x)))
     train.crew = train.crew.map(lambda x: sorted([d['name'] for d in get_dictionary(x)])).map(lambda x: ','.join(map(str, x)))

     test.genres = test.genres.map(lambda x: sorted([d['name'] for d in get_dictionary(x)])).map(lambda x: ','.join(map(str, x)))
     test.spoken_languages = test.spoken_languages.map(lambda x: sorted([d['name'] for d in get_dictionary(x)])).map(lambda x: ','.join(map(str, x)))
     test.cast = test.cast.map(lambda x: sorted([d['name'] for d in get_dictionary(x)])).map(lambda x: ','.join(map(str, x)))
     test.crew = test.crew.map(lambda x: sorted([d['name'] for d in get_dictionary(x)])).map(lambda x: ','.join(map(str, x)))
```

```
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
Name: crew, dtype: object
```

```
[81] # Then, I am counting the occurances in those features
     # which I plan to use in the model, unless they are not redundant.
     # For instance, one might expect higher revenue from a movie
     # if that movie was produced in several spoken languages and/or had a more crowded crew.

     train['genres_count'] = train['genres'].str.count(',') + 1
     train['spoken_languages_count'] = train['spoken_languages'].str.count(',') + 1
     train['cast_count'] = train['cast'].str.count(',') + 1
     train['crew_count'] = train['crew'].str.count(',') + 1

     test['genres_count'] = test['genres'].str.count(',') + 1
     test['spoken_languages_count'] = test['spoken_languages'].str.count(',') + 1
     test['cast_count'] = test['cast'].str.count(',') + 1
     test['crew_count'] = test['crew'].str.count(',') + 1
     test['genres_count']
```

```
0       4
1       2
2       2
3       3
4       2
       ..
2995    3
2996    4
2997    3
2998    1
2999    2
Name: genres_count, Length: 3000, dtype: int64
```

```
# Converting nominal data to numberical data
train[['status',
       'original_language',
       'production_companies',
       'production_countries']] = train[['status',
                'original_language',
                'production_companies',
                'production_countries']].astype('category')

train['status'] = train['status'].cat.codes
train['original_language'] = train['original_language'].cat.codes
train['production_companies'] = train['production_companies'].cat.codes
train['production_countries'] = train['production_countries'].cat.codes

test[['status',
      'original_language',
      'production_companies',
      'production_countries']] = test[['status',
                'original_language',
                'production_companies',
                'production_countries']].astype('category')
```

```
test['status'] = test['status'].cat.codes
test['original_language'] = test['original_language'].cat.codes
test['production_companies'] = test['production_companies'].cat.codes
test['production_countries'] = test['production_countries'].cat.codes

train['production_countries']
```

```
0       316
1       316
2       316
3       210
4       236
       ...
2995    316
2996    111
2997    316
2998    316
2999    316
Name: production_countries, Length: 3000, dtype: int16
```

```
[83] # Budget has zero values for many movies including some high budget movies.
     # Additionally, it does not make sense to have movies with 0 runtimes.
     # I am imputing those zero values with mean.
     train['budget'] = train['budget'].replace(0, train['budget'].mean())
     train['runtime'] = train['runtime'].replace(0, train['runtime'].mean())

     test['budget'] = test['budget'].replace(0, test['budget'].mean())
     test['runtime'] = test['runtime'].replace(0, test['runtime'].mean())
```

```
[84] # Assigning the data corresponding to the target and predictor variables
     y = train['log_revenue']
     X = train.drop(['log_revenue', 'revenue'], axis=1)
```

Splitting Data:

spliting the data into training and validation sets.

```
X_train_full, X_valid_full, y_train, y_valid = train_test_split(X, y, train_size=0.8, test_size=0.2,random_state=0)
```

```
#Creating the list of features
feature_names = ['release_year', 'release_day', 'release_month', 'status', 'original_language',
                 'budget', 'popularity', 'genres_count', 'production_companies', 'production_countries',
                'spoken_languages_count', 'cast_count', 'crew_count', 'runtime']

# Assigning the data corresponding to features in feature_names
X_train_full = X_train_full[feature_names]

X_valid_full = X_valid_full[feature_names]

X_train_full.head()
```

	release_year	release_day	release_month	status	original_language	budget	popularity	genres_count	production_companies	production_countries	spoken_languages_count	cast_count	crew_count	runtime
2370	2012	1	3	0	7	150000000.0	7.739904	1	1012	121	1	18	31	99.0
1774	2001	4	10	0	7	35000000.0	7.790140	3	1997	80	3	26	65	122.0
731	2067	6	12	0	7	4000000.0	5.032469	3	1712	316	1	18	25	108.0
271	2006	4	8	0	7	72500000.0	6.936688	1	385	316	1	24	14	116.0
1077	1980	4	8	0	7	20000000.0	3.782547	3	2168	316	1	19	15	96.0

Random Forest Model

```
[87] # Defining the Random Forest Model
     rf_model = RandomForestRegressor(random_state=1)

     # Fitting the model
     rf_model.fit(X_train_full, y_train)
```

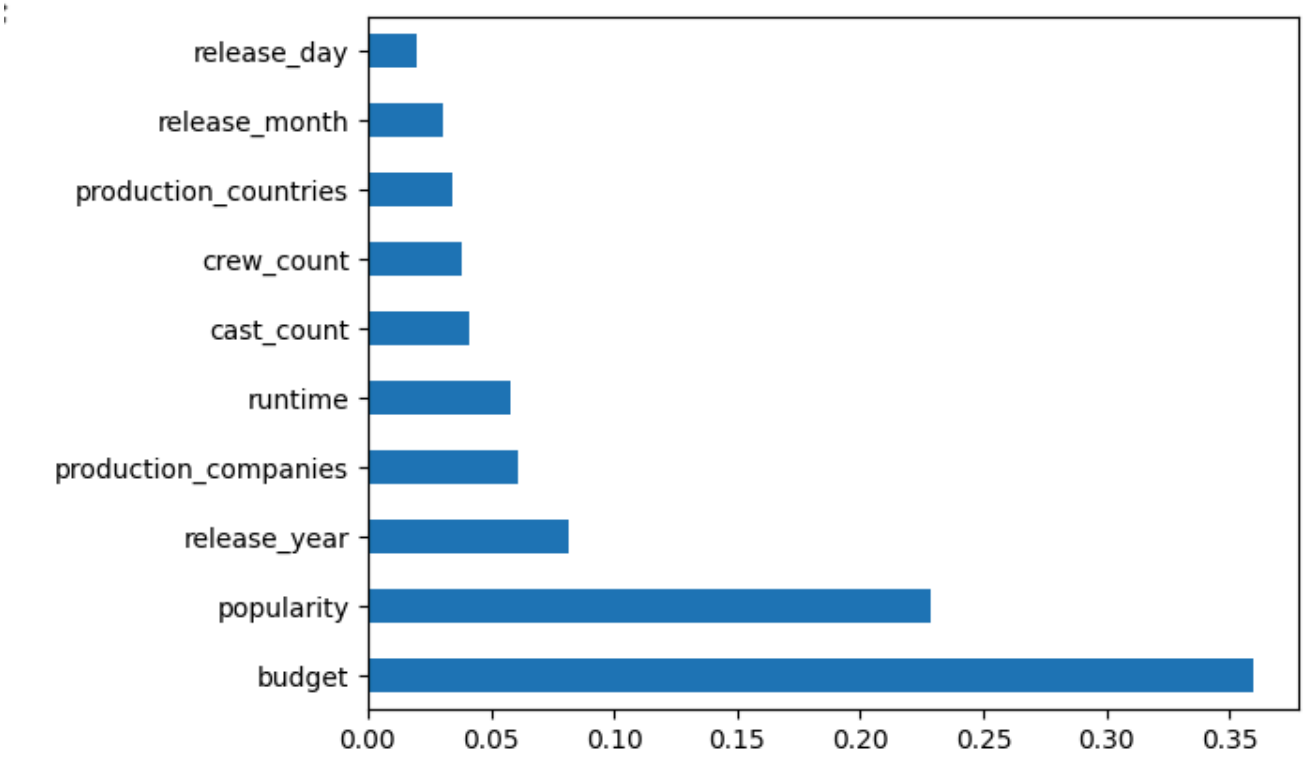
```
RandomForestRegressor
RandomForestRegressor(random_state=1)
```

```
# Prediction
y_pred_rf = rf_model.predict(X_valid_full)
```

```
# Calculate MAE
mae_rf = mean_absolute_error(y_pred_rf, y_valid)

print("Mean Absolute Error RF:" , mae_rf)
```

```
Mean Absolute Error RF: 1.418223817080765
```

```
# Calculating feature importance
feat_importances = pd.Series(rf_model.feature_importances_, index=X_train_full.columns)
feat_importances.nlargest(10).plot(kind='barh')
```



XGBoost Model

```
# Define the model
xgb_model = XGBRegressor() # Your code here

# Fit the model
xgb_model.fit(X_train_full, y_train) # Your code here
```

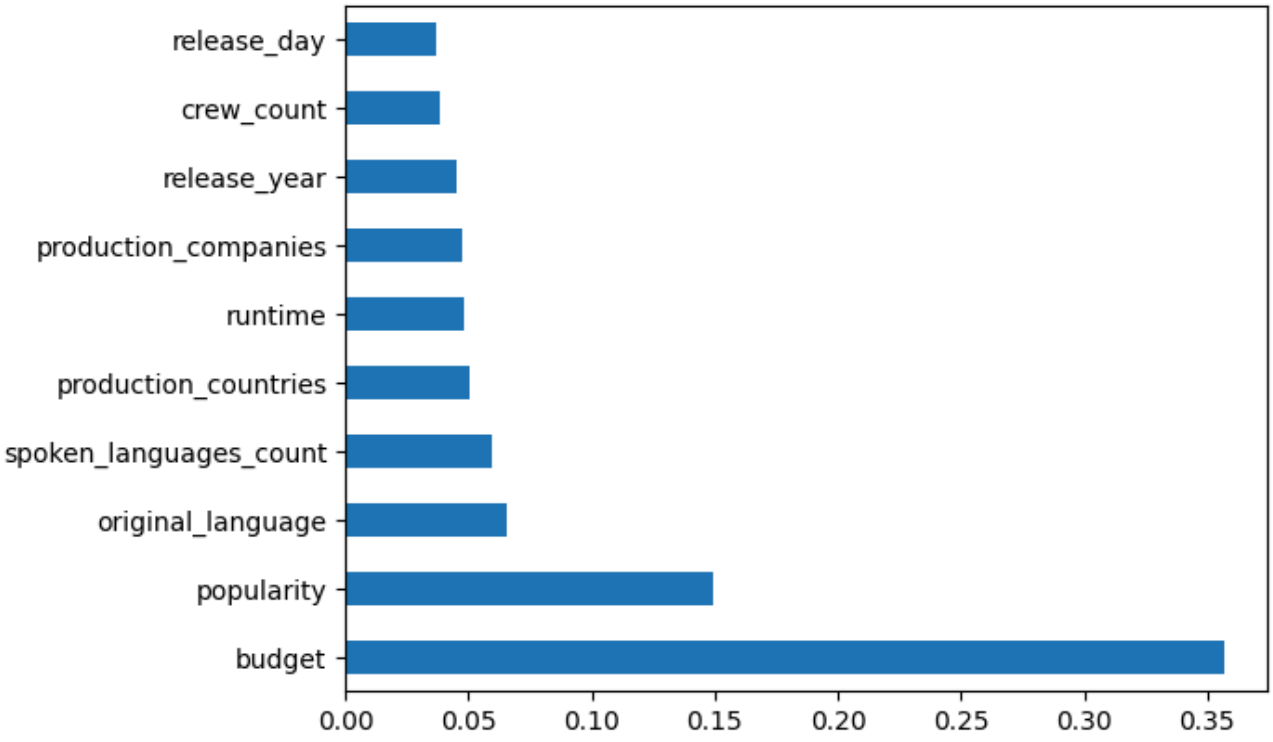
```
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, device=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             gamma=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=None, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=None, max_leaves=None,
             min_child_weight=None, missing=nan, monotone_constraints=None,
             multi_strategy=None, n_estimators=None, n_jobs=None,
             num_parallel_tree=None, random_state=None, ...)
```

```
# Prediction
y_pred_xgb = xgb_model.predict(X_valid_full)
```

```
# Calculate MAE
mae_xgb = mean_absolute_error(y_pred_xgb, y_valid)

print("Mean Absolute Error XGBOOST:" , mae_xgb)
```

```
Mean Absolute Error XGBOOST: 1.465860306740241
```

```
# Calculating feature importance for the XGBoost Model
feat_importances = pd.Series(xgb_model.feature_importances_, index=X_train_full.columns)
feat_importances.nlargest(10).plot(kind='barh')
```



Support Vector Machine (SVM)

```
[ ]  from sklearn.svm import SVR
```

```
# SVM Model
svm_model = SVR(kernel='rbf')  # You can choose different kernels based on your preference

# Fit the model
svm_model.fit(X_train_full, y_train)

# Prediction
y_pred_svm = svm_model.predict(X_valid_full)

# Calculate MAE
mae_svm = mean_absolute_error(y_pred_svm, y_valid)

print("Mean Absolute Error SVM:", mae_svm)
```

```
Mean Absolute Error SVM: 1.7559196697123385
```

Final Model

Based on the comparison of MAE results, the Random Forest model is the best fitted model.

```
# Based on the MAE results, the Random Forest Model is given better results than the XGBoost Model does.
# Therefore, the final model is defined using RF

X = train[feature_names]
X_test = test[feature_names]
```

```
# Defining the Final Model
final_model = RandomForestRegressor(random_state=1)

# Fitting the model
final_model.fit(X, y)
```

```
RandomForestRegressor(random_state=1)
```

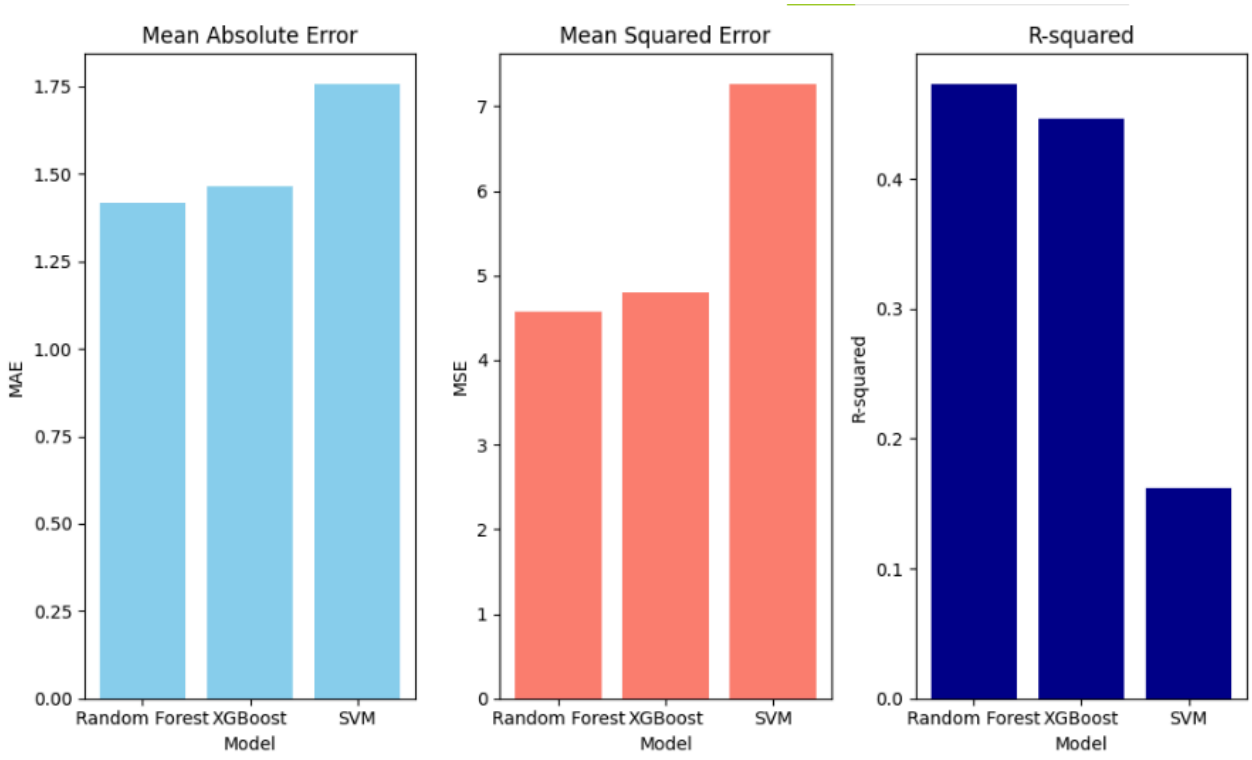
```
# Prediction
y_pred_final = final_model.predict(X_test)
pred = pd.DataFrame(y_pred_final)
```

```
submission = test[['id']].copy()
submission['revenue'] = pred
submission.to_csv('submission_1.csv', index=False)
```

```
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import matplotlib.pyplot as plt

# Calculate evaluation metrics for each ML model
evaluation_metrics = {}

# For Random Forest Model
y_pred_rf = rf_model.predict(X_valid_full)
mae_rf = mean_absolute_error(y_valid, y_pred_rf)
mse_rf = mean_squared_error(y_valid, y_pred_rf)
r2_rf = r2_score(y_valid, y_pred_rf)

evaluation_metrics['Random Forest'] = {'MAE': mae_rf, 'MSE': mse_rf, 'R-squared': r2_rf}

# For XGBoost Model
y_pred_xgb = xgb_model.predict(X_valid_full)
mae_xgb = mean_absolute_error(y_valid, y_pred_xgb)
mse_xgb = mean_squared_error(y_valid, y_pred_xgb)
r2_xgb = r2_score(y_valid, y_pred_xgb)
```

```
evaluation_metrics['XGBoost'] = {'MAE': mae_xgb, 'MSE': mse_xgb, 'R-squared': r2_xgb}

# For SVM Model
y_pred_svm = svm_model.predict(X_valid_full)
mae_svm = mean_absolute_error(y_valid, y_pred_svm)
mse_svm = mean_squared_error(y_valid, y_pred_svm)
r2_svm = r2_score(y_valid, y_pred_svm)

evaluation_metrics['SVM'] = {'MAE': mae_svm, 'MSE': mse_svm, 'R-squared': r2_svm}

# Visualize the results
# Create lists of metrics for each model
model_names = list(evaluation_metrics.keys())
mae_values = [evaluation_metrics[model]['MAE'] for model in model_names]
mse_values = [evaluation_metrics[model]['MSE'] for model in model_names]
r2_values = [evaluation_metrics[model]['R-squared'] for model in model_names]

# Plot the evaluation metrics
plt.figure(figsize=(10, 6))

plt.subplot(1, 3, 1)
plt.bar(model_names, mae_values, color='skyblue')
plt.title('Mean Absolute Error')
plt.xlabel('Model')
plt.ylabel('MAE')
```

```
plt.subplot(1, 3, 2)
plt.bar(model_names, mse_values, color='salmon')
plt.title('Mean Squared Error')
plt.xlabel('Model')
plt.ylabel('MSE')

plt.subplot(1, 3, 3)
plt.bar(model_names, r2_values, color='darkblue')
plt.title('R-squared')
plt.xlabel('Model')
plt.ylabel('R-squared')

plt.tight_layout()
plt.show()
```



Conclusion

```python
from sklearn.metrics import mean_squared_error, r2_score

# Assuming we have predictions for each model: y_pred_rf, y_pred_xgb, y_pred_svm
# And actual values for validation: y_valid

# Calculate MSE for each model
mse_rf = mean_squared_error(y_valid, y_pred_rf)
mse_xgb = mean_squared_error(y_valid, y_pred_xgb)
mse_svm = mean_squared_error(y_valid, y_pred_svm)

# Calculate R2 for each model
r2_rf = r2_score(y_valid, y_pred_rf)
r2_xgb = r2_score(y_valid, y_pred_xgb)
r2_svm = r2_score(y_valid, y_pred_svm)

# Print MSE for each model
print("Mean Squared Error for Random Forest:", mse_rf)
print("Mean Squared Error for XGBoost:", mse_xgb)
print("Mean Squared Error for SVM:", mse_svm)

# Print R2 for each model
print("\nR-squared for Random Forest:", r2_rf)
print("R-squared for XGBoost:", r2_xgb)
print("R-squared for SVM:", r2_svm)
```

```
Mean Squared Error for Random Forest: 4.569879018136001
Mean Squared Error for XGBoost: 4.793721819381448
Mean Squared Error for SVM: 7.258381967422898

R-squared for Random Forest: 0.4727053067584499
R-squared for XGBoost: 0.446877244188603
R-squared for SVM: 0.1624928629941349
```

We explored a variety of machine learning methods in this case study in order to forecast movie revenue depending on parameters like budget, genre, popularity, release date, and others. We used the Random Forest, XGBoost, and Support Vector Machine (SVM) algorithms for data preprocessing, feature engineering, and model training. The main conclusions and findings are as follows:

Preprocessing and Feature Engineering: We dealt with missing values, transformed numerical features from category forms, took extra features out of date columns, and treated textual data (cast lists and genres, for example).

Model Training: Using the training data, we trained three distinct machine learning models—Random Forest, XGBoost, and SVM—and assessed how well they performed using the validation set.
Model Evaluation: We used R-squared (R2), Mean Absolute Error (MAE), and Mean Squared Error (MSE) metrics to assess the models. These measurements showed that Random Forest performed better than the other models, with XGBoost and SVM following suit.

In terms of accuracy and goodness of fit, Random Forest outperformed XGBoost and SVM, showing the lowest Mean Squared Error (MSE) and the highest R-squared (R2).

Overall Performance: Compared to the other models, Random Forest performed better in forecasting the income from movies, indicating that it is able to better identify the underlying patterns in the data.

constraints and Future Work: Although the study produced encouraging results, it is still subject to certain constraints, including the difficulty of predicting movie income, the possibility of biases in the dataset, and the requirement for additional feature engineering.In the future, research may focus on developing more sophisticated methods, adding new data sources, and improving the models' accuracy.

The case study concludes by illustrating how machine learning techniques may be used to estimate movie revenue and by noting that Random Forest is the best model for this kind of work.

Questions

CO1: Understand different machine learning techniques and its applications with respect to your project/casestudy problem statement.
Ans
Linear Regression:
Application: Linear regression is the typical first step in the predictive analytics process – it is a basic linear model that can be used to predict continuous variables – like expected gross sales of a newly released movie. It stands for the linearity of the assumption that the features and the target variable have such an influence.
Usage: Basic linear regression may be engaged to form as a reference scenario which will explain the simple tie between parameters such as budget, popularity, and the date of release with box office revenue.

Decision Trees:
Application: Decision trees has various models and both numerical and categorical data can be treated. An important function of SVM, is partitioning the feature space into regions using some feature values and then predict the target variable for each region.
Usage: Hidden nonlinear connection between the features and movie revenue could be represented by the decision trees. They are useful in the sense, they help identifying main components and comprehending the decision making process.

Random Forest:
Application: RandomForest also a kind ensemblelearning method that includes numerous and decision trees trained on the data which were random samples. It is keeping away from the overfitting and getting better at generalization.
Usage: The Randoom Forest is extensible to forecast the film income by producing a baseline forecast from the aggregated forecasts coming from several decision trees. Multi-layered tree model has the advantage in, point of fact, compared to single one.

XGBoost (Extreme Gradient Boosting):
Application: XGBoost is a fairly advanced gradient boosting approach which consist of the construction of many base models iteratively to reduce prediction error at each step.
Usage: Being a very popular tool XGBoost is known for its superior results and scalability. It numerous times won data science competitions and it is used in real-world applications. It can cleverly cater to the complicated nature of data relationships and produce tremendously précise statistics of the gross movie revenue.

Support Vector Machines (SVM):
Application: SVM namely, is a supervised learning algorithm that can be employed for classification and regression. It therefore finds the best hyperplane that doesn't only distinguish the data points into different categories but also predicts the target variable.
Usage: SVM may be used to forecast movie earnings by representing its input characteristics in a new and higher dimensional space followed by marking the hyperplane which best splits the data points which belong to revenue. To match the data relationship complexity or hyperparameter tuning could be the necessary.

CO2: Understand the importance of simple linear regression in Predicting new observations with respect to your project/case study. Can Simple Linear Regression can be used for your problem? Yes/No Explain why?

Ans Movie revenue prediction by using simple linear regression can be effective to some extent.,but may not be the most reliable one.

Yes, Simple Linear Regression Can Be Used:
Establishing Baseline Prediction: Uncomplicated linear regression can be an initial model for the predictions. This enables us to have a direct proportion to extricate a variety of single input features (for instance, revenue, popularity) from the target variable. This integrated model will offer the initial comprehension of how materials impact the venture when all attributes are combined together.
Interpretability: Simple linear regression already gives understand interpretations of how the feature of the movie prices and audience opinions connects with the movie revenue. It provides us with the moving coefficient of the input feature, which is the change in revenue to be that amount for a unit change of the feature.

No, Simple Linear Regression May Not Be Ideal
Complexity of the Relationship: Different factors including budget, genre, cast and release date are the specific parts causing the movement in the accumulation of movie revenue. It follows that we can describe the relationship of the input feature and the target variable with the simple linear regression. On the other hand, the predictors of movie revenues are likely to be linear or involve interactions between characters that are not obvious to routine regression modelling.
Limited Predictive Power: One can point out that simple linear regression may not perform well in terms of predicting movie revenue due to having little predictive power. Complexity of the revenue reciprocation may be simplified, which in turn leads to poor predictions. The more complicated designs like ensemble methods such as Random Forest and XGBoost may be the more suitable model next in line since they can be able to represent the intricacies that the data may have.
Assumptions Violation: The simple linear regression model provides for underlying linearity, whether paired or unpaired (dependent or independent), constant variance of error terms and a normal distribution of errors. In such cases, existing assumptions, for instance with regards to uniform data sources or linear relationships, may not be supported at all, especially given the multifarious nature of data sources and nonlinearity of relationships.

To estimate movie revenue and draw broad conclusions, basic linear regression can be utilized as a first step, however there are certain notable exceptions where this approach may not be better. Indeed, it is more likely that the latter advancements, such as ensemble techniques and deep learning algorithms, will produce improved outcomes in terms of capturing the intricate relationships and interactions between

different variables. In order to meet our foreseeable forecast models for the movie's income, we should therefore analyze and evaluate a range of modeling techniques.

CO3: Understand the importance of Multiple linear regression in predicting new observations with respect to your project/case study. Can MultipleLinear Regression can be used for your problem? Yes/No Explain why?

Ans Yes, Multiple Linear Regression Can Be Used:

Incorporating Multiple Predictors: Multiple linear regression is the method to have features (variables) put into the model all at the same time. Among the factors which may influence takings for a film alleged to be marketing considerations, cast, genre, gong budget, and release date. Multiple linear regression components all these factors as a long run and moving target for revenue allowing to capture a collective result of the factors on revenue.

Capturing Complex Relationships: Revenue for movie business sometimes depends on the synergy of many different factors rather than on a single factor. These factors may have a complex relationship with each other. Multiple linear regression is a useful method which treats direct link between the predictors and response variables: linear summation of the predictors represents their complex relation. It enables us to determine which predictors have a poweful impact on market revenue and also shows how they work together.

Interpretability: Multiple linear regression analysis also provides linear coefficients for each predictor which are as easy to interpret as the coefficients from simple linear regression. These coefficients translate the change in the corresponding predictor, which is the change in revenues for one-unit increase in other predictors, while maintaining other predictors constantly. Consequently, humanizes the overall process of the importance of diverse factors such as marketing, content, production quality, and distribution in movie revenue.

Model Evaluation and Inference: Among statistical measurements of multiple linear regression includes R-squared, adjusted R-squared, and p-values which can be used to assess how well the model performs and identify which variables are significant. This makes it possible for modeling testing and inference, leading to the outcome of data interpretations and decisions being made in the correct direction.

No, Multiple Linear Regression May Have Limitations:
Assumption of Linearity: Multiple linear regression, naively says, that the relationship between the predictors and the goal variable is linear. The proposition though may be true for some predictors, hence it may not fully capture the situation of the presence of non-linear relationships and correlation among

predictors. Hence, rigid models like polynomial regression or machine learning pretty much will be feasible in such cases.

Potential Overfitting: If the number of predictors is high relative to the available observations in this high-dimensional model, there is a danger of the multiple linear regression being overfit. Overfitting is a case in which the training model doesn't capture and learn patterns but produces a lot of noise in the data leading to poor generalization on new data.

Conclusion:
Multiple linear regression is one of the best prediction models for the movie revenue as it explore the effect of numerous predictors on revenue rather making prediction just a single way. For cases like these, it plays a major role, but only specific features of the data count, and hidden patterns among predictors do not. Although multiple linear regression can get you started, nevertheless you should give a serious shot to sophisticated modelling techniques, which are capable of dealing with non-linear relationship or high dimensionality of data. In essence, multiple linear regression model is a very reliable part of a bigger collection of tools used by predictive models, to progressively come to the right answer.

CO4: For your project definition, explain the importance of parameter estimation. Check how it affects the overall results when you change the values of different parameters.

Ans Parameter estimation is crucial for building accurate and effective predictive models.

Model Selection: This parameter largely assists in determining the best model that can be used to portray a movie's earnings. Unlike other machine learning algorithms, these have parameters that require to be tunned for best results. On the other hand, the number of trees in Random Forest, maximum depth of a tree, minimum leaf samples and many more parameters are essential for tuning..

Model Complexity: Parameter estimation offers flexibility that allows us to regulate model simplicity. Selecting reasonable numbers of value for such parameters like regularization (for instance, alpha in Ridge regression), tree depth, or layers in neural network is a way to ensure they are not overfitting or underfitting. The term overfitting is described as a phenomenon when certain machine learning algorithms capture the noise present in the training data, with the consequence that generalization on unseen data gets worse. The model that is underfitted is too elementary, just fails to note the patterns of the data and so comes up with very high bias values and poor accuracy.

Generalization: The most important step for learning process is the precise determination of the model to measure the generalization capability. With the use of such techniques as cross-validation, grid search, or random search we may find parameter values that provide the best generalization ability on the dataset that was not used to optimize model parameters. Consequently, the model will be capable of generalizing and doing proficient work for the unknown data if the model is efficient to generalize to the new observations and predicting accurate outputs in real-world situations.

Interpretability: However, a set of machine learning model variables which have interpretability consequences is bound to exist. For instance, in linear regression one receives information for the coefficients which helps to show the link between movie revenue and each predictor variable. Parameter estimation of this sort helps not only to understand the factors standing behind the variations in revenue but also to derive clear steps to take.

Computational Efficiency: Parameter estimation is another variable that will affect the whole computing efficiency of the system training. Accurately setting up a parameter value can facilitate convergence speed and decrease complexity, ensuring efficient and scalable fitting, albeit for tremendous datasets too.

changing the values of the different parameters while performing parameter estimation can affect the results of the predictive model:

By adding more trees to Random Forest, we may lower variance and increase noise robustness, which will enhance the model's performance. Overcrowding, however, can result in lengthier training sessions and a higher chance of overfitting.
The training process's stability and rate of convergence in neural networks can both be impacted by changes in learning rate. While a low learning rate could lead to sluggish convergence, a high learning rate might cause the model to overshoot the ideal answer.
In Ridge Regression, the trade-off between fitting the training data and lowering model complexity can be managed by adjusting the regularization strength (alpha). Stronger regularization at higher alpha values can reduce the risk of overfitting but also increase bias.

After considering Everything, parameter estimation plays a crucial role in developing predictive models that forecast movie income because it has a direct impact on the model's functionality, interpretability, generalizability, and computing efficiency. Effective model parameter tuning can produce more precise and trustworthy forecasts, which will eventually increase the project's usefulness and success.

CO5: Compare different Machine Learning algorithms used for your project/case through evaluation metrices. Show which ML algorithm works best for your problem.

After calculating the values of Mean Squared Error (MSE) and R-squared (R2) we can conclude that:

Mean Squared Error (MSE):
XGBoost: 4.7937;
Random Forest: 4.5699;
SVM: 7.2584;
Lower MSE values suggest superior performance. As a result, Random Forest outperforms XGBoost and SVM in terms of minimizing the squared disparities between predicted and actual values, as seen by its lowest MSE.

R-squared (R2):
Random Forest: 0.4727
XGBoost: 0.4469
SVM: 0.1625
R-squared (R) measures how much percentage of the dependent variable (revenue) variance is predictable from the independent variables. Larger R2 values are indicative of good model-data fit. Hence, Random Forest is the model with the highest R2 among the three algorithms, which means that it is able to analyze the variance of revenue data more accurately, than XGBoost and SVM.

Overall, Random Forest seems to be the best-performing model out of the three for this task, based on both MSE and R2. Its superior quality of fit and accuracy over XGBoost and SVM are indicated by its lowest mean square error (MSE) and greatest R2.

References
https://www.kaggle.com/competitions/tmdb-box-office-prediction/data