In the second part of the notebook, we will focus on constructing a regression model.

Table of Contents


```
pip install scikeras
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-whe
Collecting scikeras
  Downloading scikeras-0.10.0-py3-none-any.whl (27 kB)
Requirement already satisfied: packaging>=0.21 in /usr/local/lib/python3.10/dist
Requirement already satisfied: scikit-learn>=1.0.0 in /usr/local/lib/python3.10/
Requirement already satisfied: numpy>=1.17.3 in /usr/local/lib/python3.10/dist-p
Requirement already satisfied: scipy>=1.3.2 in /usr/local/lib/python3.10/dist-pa
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-p
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10
Installing collected packages: scikeras
Successfully installed scikeras-0.10.0
```

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import plotly.express as px
import plotly.graph_objects as go
# Plotly
```

```
import plotly.graph_objects as go
# Sklearn
import seaborn as sns
import sklearn
from tqdm.notebook import tqdm
from sklearn.ensemble import RandomForestRegressor
import statsmodels.api as sm
from sklearn.feature_selection import SelectKBest, f_regression
from sklearn.metrics import mean_squared_error
from sklearn.ensemble import AdaBoostRegressor
from xgboost import XGBRegressor
from sklearn.model_selection import GridSearchCV
from sklearn.neural_network import MLPRegressor
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV

from sklearn.preprocessing import OneHotEncoder
from sklearn.pipeline import Pipeline
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.compose import ColumnTransformer

# Deep learning
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.utils import to_categorical
from keras.callbacks import ReduceLROnPlateau, EarlyStopping
from scikeras.wrappers import KerasRegressor
from keras.models import load_model
import warnings

# Suppress all warnings
warnings.filterwarnings("ignore")
```

Gathering data:

In this notebook, we have two data frames. One includes movie ID, title, cast, and crew, while the second has additional features like genres, budget, and original language. We will merge these data frames since they both contain useful information for predicting movie popularity. By combining the data, we can utilize a broader set of predictors to enhance the accuracy of our popularity predictions.

```
# Uploading and viewing the data
tmdb_5000_cred = pd.read_csv(r'tmdb_5000_credits.csv', index_col=False)
tmdb_5000_cred.head()
```

	movie_id	title	cast
0	19995	Avatar	[{"cast_id": 242, "character": "Jake Sully", "...
1	285	Pirates of the Caribbean: At World's End	[{"cast_id": 4, "character": "Captain Jack Spa...
2	206647	Spectre	[{"cast_id": 1, "character": "James Bond", "cr...
3	49026	The Dark Knight Rises	[{"cast_id": 2, "character": "Bruce Wayne / Ba...
4	49529	John Carter	[{"cast_id": 5, "character": "John Carter", "c...

```
# Uploading and viewing the data
tmdb_5000_cred.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4803 entries, 0 to 4802
Data columns (total 4 columns):
 #   Column    Non-Null Count  Dtype 
---  ------    --------------  ----- 
 0   movie_id  4803 non-null   int64 
 1   title     4803 non-null   object
 2   cast      4803 non-null   object
 3   crew      4803 non-null   object
dtypes: int64(1), object(3)
memory usage: 150.2+ KB
```

```
# Uploading and viewing the data
tmdb_5000_mov = pd.read_csv(r'tmdb_5000_movies.csv')
tmdb_5000_mov.head()
```

	budget	genres	homepage	id	keywords	orig
0	237000000	[{"id": 28, "name": "Action"}, {"id": 12, "nam...	http://www.avatarmovie.com/	19995	[{"id": 1463, "name": "culture clash"}, {"id":...	
1	300000000	[{"id": 12, "name": "Adventure"}, {"id": 14, "...	http://disney.go.com/disneypictures/pirates/	285	[{"id": 270, "name": "ocean"}, {"id": 726, "na...	
2	245000000	[{"id": 28, "name": "Action"}, {"id": 12, "nam...	http://www.sonypictures.com/movies/spectre/	206647	[{"id": 470, "name": "spy"}, {"id": 818, "name...	
3	250000000	[{"id": 28, "name": "Action"}, {"id": 80, "nam...	http://www.thedarkknightrises.com/	49026	[{"id": 849, "name": "dc comics"}, {"id": 853,...	
		[{"id": 28,			[{"id": 818,	

```
# Checking the details of the data
tmdb_5000_mov.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4803 entries, 0 to 4802
Data columns (total 20 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   budget                4803 non-null   int64  
 1   genres                4803 non-null   object 
 2   homepage              1712 non-null   object 
 3   id                    4803 non-null   int64  
 4   keywords              4803 non-null   object 
 5   original_language     4803 non-null   object 
 6   original_title        4803 non-null   object 
 7   overview              4800 non-null   object 
 8   popularity            4803 non-null   float64
 9   production_companies  4803 non-null   object 
 10  production_countries  4803 non-null   object 
 11  release_date          4802 non-null   object 
 12  revenue               4803 non-null   int64  
 13  runtime               4801 non-null   float64
```

```
 14  spoken_languages      4803 non-null   object
 15  status                4803 non-null   object
 16  tagline               3959 non-null   object
 17  title                 4803 non-null   object
 18  vote_average          4803 non-null   float64
 19  vote_count            4803 non-null   int64
dtypes: float64(3), int64(4), object(13)
memory usage: 750.6+ KB
```

Merging the data

```
# Merging the two data sets
tmdb_5000_cred.columns = ['id','tittle','cast','crew']
tmdb_5000_mov = tmdb_5000_mov.merge(tmdb_5000_cred,on='id')
```

```
# View more details
tmdb_5000_mov.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 4803 entries, 0 to 4802
Data columns (total 23 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   budget                4803 non-null   int64
 1   genres                4803 non-null   object
 2   homepage              1712 non-null   object
 3   id                    4803 non-null   int64
 4   keywords              4803 non-null   object
 5   original_language     4803 non-null   object
 6   original_title        4803 non-null   object
 7   overview              4800 non-null   object
 8   popularity            4803 non-null   float64
 9   production_companies  4803 non-null   object
 10  production_countries  4803 non-null   object
 11  release_date          4802 non-null   object
 12  revenue               4803 non-null   int64
 13  runtime               4801 non-null   float64
 14  spoken_languages      4803 non-null   object
 15  status                4803 non-null   object
 16  tagline               3959 non-null   object
 17  title                 4803 non-null   object
 18  vote_average          4803 non-null   float64
 19  vote_count            4803 non-null   int64
 20  tittle                4803 non-null   object
 21  cast                  4803 non-null   object
 22  crew                  4803 non-null   object
dtypes: float64(3), int64(4), object(16)
memory usage: 900.6+ KB
```

The value count in the "tagline" and "homepage" columns shows null values which we will handle next.

Cleaning Data

Handling null values

```
# Count null values in each column
null_counts = tmdb_5000_mov.isnull().sum()

# Print the null value counts for each column
print(null_counts)
```

```
budget                     0
genres                     0
homepage                3091
id                         0
keywords                   0
original_language          0
original_title             0
overview                   3
popularity                 0
production_companies       0
production_countries       0
release_date               1
revenue                    0
runtime                    2
spoken_languages           0
status                     0
tagline                  844
title                      0
vote_average               0
vote_count                 0
tittle                     0
cast                       0
crew                       0
dtype: int64
```

This is how we will handle the null values:

- homepage' has 3091 null values. - we will remove the column
- 'overview' has 3 null values. - we will remove the null
- 'release_date' has 1 null value. - we will remove the null values
- 'runtime' has 2 null values. - we will remove the null
- 'tagline' has 844 null values. - We will remove the column

```
# Dropping columns with high null values (homepage and tagline)
tmdb_5000_mov.drop(['homepage', 'tagline'], axis=1, inplace=True)
```

```
# Removing the remaining rows with null values in the other columns
tmdb_5000_mov.dropna(inplace=True)
```

Handling duplicates

We are checking for and removing any duplicated rows in the dataset to ensure data accuracy and reliability.

```
# Checking the number of duplicated rows
num_duplicates = tmdb_5000_mov.duplicated().sum()
print(f"Number of duplicated rows: {num_duplicates}")

# Dropping the duplicated rows
data = tmdb_5000_mov.drop_duplicates()

# Verifing the number of rows after dropping duplicates
num_rows = len(data)
print(f"Number of rows after dropping duplicates: {num_rows}")
```

```
Number of duplicated rows: 0
Number of rows after dropping duplicates: 4799
```

```
# Checking the shape
tmdb_5000_mov.shape
```

```
(4799, 21)
```

We are transposing the data frame. This is easier to examine the data closely and see what further action should be takes to clean the data.

```
# Display the first two rows of the dataset

tmdb_5000_mov[:2].T
```

	0	
budget	237000000	
genres	[{"id": 28, "name": "Action"}, {"id": 12, "nam...	[{"id": 12, "name": "Adv
id	19995	
keywords	[{"id": 1463, "name": "culture clash"}, {"id":...	[{"id": 270, "name": "oce
original_language	en	
original_title	Avatar	Pirates of the Carib
overview	In the 22nd century, a paraplegic Marine is di...	Captain Barbossa, long belie
popularity	150.437577	
production_companies	[{"name": "Ingenious Film Partners", "id": 289...	[{"name": "Walt Disney
production_countries	[{"iso_3166_1": "US", "name": "United States o...	[{"iso_3166_1": "US", "nam
release_date	2009-12-10	
revenue	2787965087	
runtime	162.0	
spoken_languages	[{"iso_639_1": "en", "name": "English"}, {"iso...	[{"iso_639_1": "er
status	Released	
title	Avatar	Pirates of the Carib
vote_average	7.2	
vote_count	11800	
tittle	Avatar	Pirates of the Carib
cast	[{"cast_id": 242, "character": "Jake Sully", "...	[{"cast_id": 4, "character"
crew	[{"credit_id": "52fe48009251416c750aca23", "de...	[{"credit_id": "52fe4232c3a3

Popularity IMDbPro uses proprietary algorithms that take into account several measures of popularity for people, titles and companies. The primary measure is who and what people are looking at on IMDb. The rankings are updated on a weekly basis, typically by the end of Monday.

This line filters the dataset to exclude rows where 'revenue' = 0. Those rows represent movies that did not generate any revenue, and therefore do not provide relevant information for our analysis.

Let's count the number of movies with a budget equal to 0 and the number of movies with revenue equal to 0.

```
# Counting the number of rows with revenue equal to 0
zero_revenue_count = len(tmdb_5000_mov[tmdb_5000_mov['revenue'] == 0])

# Printing the counts
print("Number of movies with revenue = 0:", zero_revenue_count)
```

```
Number of movies with revenue = 0: 1423
```

```
# Filtering data where revenue is not 0
tmdb_5000_mov = tmdb_5000_mov[tmdb_5000_mov['revenue']!=0]
```

```
# Checking shape
tmdb_5000_mov.shape
```

```
(3376, 21)
```

Preparing the data for modeling

- Split the data into dependent and independent variables.
- Performed a train-test split on the data.
- Scale and encoding the data separately using a pipeline to avoid leakage.

```
# List the variables
list(tmdb_5000_mov)
```

```
['budget',
 'genres',
 'id',
 'keywords',
 'original_language',
 'original_title',
 'overview',
 'popularity',
 'production_companies',
 'production_countries',
 'release_date',
 'revenue',
 'runtime',
 'spoken_languages',
 'status',
 'title',
 'vote_average',
 'vote_count',
 'tittle',
```

```
       'cast',
       'crew']
```

```
# Create a copy for the data
df = tmdb_5000_mov.copy()
```

```
# Selecting only the important variables that are relevant for our analysis.
imp_cols = ['budget', 'genres','popularity','original_language',
            'runtime','vote_average','vote_count','release_date']
```

```
# Creating a dataframe with all the important columns
df = df[imp_cols]
```

```
# Vewing dataframe
df.head()
```

	budget	genres	popularity	original_language	ru
0	237000000	[{"id": 28, "name": "Action"}, {"id": 12, "nam...	150.437577	en	
1	300000000	[{"id": 12, "name": "Adventure"}, {"id": 14, "...	139.082615	en	
2	245000000	[{"id": 28, "name": "Action"}, {"id": 12, "nam...	107.376788	en	
3	250000000	[{"id": 28, "name": "Action"}, {"id": 80, "nam...	112.312950	en	
4	260000000	[{"id": 28, "name": "Action"}, {"id": 12, "nam...	43.926995	en	

We want to inclide the release month on our model, we will extract the month from the 'release_date' column.

```
# Converting release_date to datetime
# any invalid values to a NaT (Not a Time)
df['release_date'] = pd.to_datetime(df['release_date'],
                                                errors='coerce')

# Extracting the month from the release_date column
df['release_date_month'] = df['release_date'].dt.month
```

```
# Dropping release date since we wont need it any more
df.drop('release_date',axis=1,inplace=True)
```

```
# Dropping null values
df.dropna(inplace=True)
```

```
#  Changing the data type to integer
# df['release_date_month'] = df['release_date_month'].astype('int')
```

```
#  Changing the data type to object
df['release_date_month'] = df['release_date_month'].astype('object')
```

Handling Genres Column

```
# View numerical columns
```

- The 'genres' column is stored as dictionaries.
- We use the a lambda function to convert the string into actual dictionaries lists using the eval().
- This allows us to access the genre names within the dictionaries. We then drop any rows with null values using dropna().
- Finally, we build a function called get_val() to extract the genre names from the dictionaries.

```
# Checking the data
df.head()
```

	budget	genres	popularity	original_language	ru
0	237000000	[{"id": 28, "name": "Action"}, {"id": 12, "nam...	150.437577	en	
1	300000000	[{"id": 12, "name": "Adventure"}, {"id": 14, "...	139.082615	en	
2	245000000	[{"id": 28, "name": "Action"}, {"id": 12, "nam...	107.376788	en	
3	250000000	[{"id": 28, "name": "Action"}, {"id": 80, "nam...	112.312950	en	
4	260000000	[{"id": 28, "name": "Action"}, {"id": 12, "nam...	43.926995	en	

```
# Dropping null values
df.dropna(inplace=True)
```

```
# Running the function
def get_val(dictionary_list):
    val = [d['name'] for d in eval(dictionary_list)]
    return val

# Running a lmabda function for the genres column
from tqdm.notebook import tqdm
tqdm.pandas()

# Apply the get_val function to extract the genre namesn

df['genres'] = df['genres'].progress_apply(get_val)
```

```
100%                                                3376/3376 [00:00<00:00, 19276.11it/s]
```

```
# Viewing df
df.head()
```

	budget	genres	popularity	original_language	ru
0	237000000	[Action, Adventure, Fantasy, Science Fiction]	150.437577	en	
1	300000000	[Adventure, Fantasy, Action]	139.082615	en	
2	245000000	[Action, Adventure, Crime]	107.376788	en	
3	250000000	[Action, Crime, Drama, Thriller]	112.312950	en	
4	260000000	[Action, Adventure, Science Fiction]	43.926995	en	

Splitting Variables

```
# Splitting the data into independent and dependent Variables
X = df.drop('popularity',axis=1)
y = df['popularity']
```

```
# Splitting the data into train and test
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size=0.20, random_state=42)
```

```
# Viewing Training data
X_train
```

	budget	genres	original_language	runtime	vote
3206	0	[Drama]	fr	127.0	
826	55000000	[Drama, Thriller, Crime, Mystery, Romance]	en	123.0	
3966	2500000	[Action, Crime, Drama, Thriller]	en	92.0	
1317	38000000	[Action, Drama]	en	129.0	
2854	0	[Comedy, Romance]	en	93.0	
...	...	...	...	...	
1164	40000000	[Horror, Mystery]	en	116.0	
1199	40000000	[Action, Adventure, Drama, Thriller]	en	139.0	
1400	35000000	[Comedy, Romance]	en	106.0	
902	50000000	[Comedy, Drama, Romance]	en	139.0	
4190	500000	[Drama, Horror, Thriller, Romance]	en	93.0	

2700 rows × 7 columns

We need to use PassThroughTransformer in order to transform the 'genres' column into multiple binary columns representing each unuique genre.

It iterates over the unique genres, creating a new binary column for each genre, and populates it with 1 if the movie belongs to that genre, and 0 otherwise.

```
# Storing genres
passthrough_features = ['genres']

# Creating a custom transformer class for passthrough features

class PassthroughTransformer(BaseEstimator):
    def fit(self, X, y = None):
```

```
        self.cols = X.columns
        return self

    def transform(self, X, y = None):

        #Creating a copy
        X_ = X.copy()

        # Getting all unique genres from the 'genres' column
        self.all_genre = set(sum(df['genres'],[]))


        # Iterating over each genre and create a new binary column for it
        for gen in tqdm(self.all_genre):
            X_[gen] = X_['genres'].apply(lambda x: 1 if gen in x else 0)

        # Dropping the original 'genres' column
        X_.drop('genres',axis=1,inplace=True)
        return X_

    def get_feature_names(self):
        return list(self.all_genre)

# enc = ColumnTransformer([('pass' , PassthroughTransformer(),
# passthrough_features)])

# https://www.appsloveworld.com/scikit-learn/7/adding-get-feature-names-
# to-columntransformer-
# https://gist.github.com/tdpetrou/6a97304dd4452a53be98e4f4e93196e6
```

```
set(sum(df['genres'],[]))
```

```
{'Action',
 'Adventure',
 'Animation',
 'Comedy',
 'Crime',
 'Documentary',
 'Drama',
 'Family',
 'Fantasy',
 'Foreign',
 'History',
 'Horror',
 'Music',
 'Mystery',
 'Romance',
 'Science Fiction',
 'Thriller',
 'War',
 'Western'}
```

Confirming Column Type

```
# Remind us of the columns we are deaing with

def get_column_types(data):

   # Initializing empty lists for categorical and numerical
    categorical_cols = []
    numerical_cols = []
# Iterating over each column
    for col in data.columns:
      # Check the data type of the column
        if data[col].dtype == 'object':
          # If it's an object type assign it to categorical column
            categorical_cols.append(col)
        else:
          # Else it is a numerical column
            numerical_cols.append(col)

    return categorical_cols, numerical_cols
```

```
# Passing data to get column type
categorical_cols, numerical_cols = get_column_types(df)

print("Categorical columns:")
print(categorical_cols)
print()
print("Numerical columns:")
print(numerical_cols)
```

```
Categorical columns:
['genres', 'original_language', 'release_date_month']

Numerical columns:
['budget', 'popularity', 'runtime', 'vote_average', 'vote_count']
```

Separating numerical and categorical

We will be working with two types of data: numerical and categorical. For the pipeline - we need to separate these data types for conducting scaling. Numerical data will be scaled or normalized, while categorical data will undergo encoding techniques.

```

```

```
# Storing numerical columns
numeric_cols = ['budget','runtime','vote_average','vote_count']
```

Pipeline

We use the OneHotEncoder to handle categorical features, the StandardScaler to scale numeric features, and the ColumnTransformer to combine these preprocessing steps.

```
# Create an instance of the OneHotEncoder

ohe = OneHotEncoder(handle_unknown='ignore')

# Storing one hot encoder in a pipeline
categorical_processing = Pipeline(steps=[('ohe', ohe)])
scaling_processing = Pipeline(steps=[('scale', StandardScaler())])

# we need to bring together the preprocessing steps for both cat. and num.
# We create a ColumnTransformer object for preprocessing steps.
# We apply the categorical_processing step to the 'original_language' and
# 'release_date_month' columns,
# and the scaling_processing step to the numeric columns.

preprocessing = ColumnTransformer(transformers=[
                ('categorical', categorical_processing, ['original_language',
                                                         'release_date_month']),
        ('numeric', scaling_processing, numeric_cols),
         ('pass' , PassthroughTransformer(), passthrough_features)
                                              ],
                                 )
```

```
# Checking shape for the training data
X_train.shape
```

```
(2700, 7)
```

```
# Checking shape for the testing data
X_test.shape
```

```
(676, 7)
```

By applying fit_transform on the training data and transform on the test data, we make sure that the data is processes for both and we avoid any leakage of information from the test data into the training process.

```
# # Scale and hot encode the train and test data
X_train = preprocessing.fit_transform(X_train)
```

```
X_test = preprocessing.transform(X_test)
```

```
100%                                                  19/19 [00:00<00:00, 348.45it/s]
100%                                                  19/19 [00:00<00:00, 654.88it/s]
```

```
# pip show scikit-learn
```

```
# pip install -U scikit-learn
```

```
'''genre_label = preprocessing.named_transformers_['pass'].get_feature_names()
enc_cat_features = preprocessing.named_transformers_['categorical']['ohe'].get_feature
other_col = ['runtime', 'vote_average', 'vote_count', 'budget']
passthrough_features = ['passthrough1', 'passthrough2', 'passthrough3']  # Define the
labels = np.concatenate([genre_label, enc_cat_features, other_col, passthrough_feature
'''
```

```
'genre_label = preprocessing.named_transformers_['pass'].get_feature_names()\nen
= ['runtime', 'vote_average', 'vote_count', 'budget']\npassthrough_features = ['
atenate([genre_label, enc_cat_features, other_col, passthrough_features])\n'
```

We retrieve the feature names for the passthrough transformer and one-hot encoder transformer used in the pipeline. We also define the names of passthrough features and other columns/features.

```
# Retrieving the feature names for the passthrough transformer
genre_label = preprocessing.named_transformers_['pass'].get_feature_names()

# Retrieving the feature names for the one-hot encoder transformer
enc_cat_features = (preprocessing.named_transformers_['categorical']
          ['ohe'].get_feature_names_out()
                               )
# Defining the names of passthrough features
passthrough_features = ['passthrough1', 'passthrough2', 'passthrough3']

# Define the names of other features

other_col = ['runtime','vote_average','vote_count','budget']

# Concatenating all the feature names together

labels = np.concatenate([genre_label, enc_cat_features,other_col])
```

```
# https://www.youtube.com/watch?v=NxLfpcfGzns
```

```
X_train = pd.DataFrame(X_train,columns=labels)
X_test = pd.DataFrame(X_test,columns=labels)
```

```
X_train.info()
```

```
      3   Foreign                2700 non-null   float64
      4   Romance                2700 non-null   float64
      5   Adventure              2700 non-null   float64
      6   Horror                 2700 non-null   float64
      7   Science Fiction        2700 non-null   float64
      8   Family                 2700 non-null   float64
      9   Drama                  2700 non-null   float64
      10  Documentary            2700 non-null   float64
      11  Action                 2700 non-null   float64
      12  War                    2700 non-null   float64
      13  History                2700 non-null   float64
      14  Fantasy                2700 non-null   float64
      15  Comedy                 2700 non-null   float64
      16  Thriller               2700 non-null   float64
      17  Music                  2700 non-null   float64
      18  Mystery                2700 non-null   float64
      19  original_language_cn   2700 non-null   float64
      20  original_language_da   2700 non-null   float64
      21  original_language_de   2700 non-null   float64
      22  original_language_el   2700 non-null   float64
      23  original_language_en   2700 non-null   float64
      24  original_language_es   2700 non-null   float64
      25  original_language_fa   2700 non-null   float64
      26  original_language_fr   2700 non-null   float64
      27  original_language_he   2700 non-null   float64
      28  original_language_hi   2700 non-null   float64
      29  original_language_id   2700 non-null   float64
      30  original_language_is   2700 non-null   float64
      31  original_language_it   2700 non-null   float64
      32  original_language_ja   2700 non-null   float64
      33  original_language_ko   2700 non-null   float64
      34  original_language_nb   2700 non-null   float64
      35  original_language_nl   2700 non-null   float64
      36  original_language_no   2700 non-null   float64
      37  original_language_ro   2700 non-null   float64
      38  original_language_ru   2700 non-null   float64
      39  original_language_te   2700 non-null   float64
      40  original_language_th   2700 non-null   float64
```

```
 49  release_date_month_7   2700 non-null   float64
 50  release_date_month_8   2700 non-null   float64
 51  release_date_month_9   2700 non-null   float64
 52  release_date_month_10  2700 non-null   float64
 53  release_date_month_11  2700 non-null   float64
 54  release_date_month_12  2700 non-null   float64
 55  runtime                2700 non-null   float64
 56  vote_average           2700 non-null   float64
 57  vote_count             2700 non-null   float64
 58  budget                 2700 non-null   float64
dtypes: float64(59)
memory usage: 1.2 MB
```

```
# X_train = X_train.toarray()
# X_test = X_test.toarray()
```

Steps for Regression Modeling Approach:

- We will start with a baseline model using ordinary least squares (OLS) regression
- We will select the best features based on their significance
- Then we will implement three machine learning models: Random Forest, XGBoost, and AdaBoost
- We will follow by training a general MLP (Multi-Layer Perceptron) model
- Ultimately, we will evaluate the performance of each model and compare their results

Regression

```
# Checking shape
print('Test data shape:', X_test.shape,'Train data shape:', X_train.shape)
```

```
Test data shape: (676, 59) Train data shape: (2700, 59)
```

```
# Resetting index
y_train.reset_index(drop=True,inplace=True)
```

```
# Running the OLS regression model.
X_train # Using the best features for the model
X_train_int = sm.add_constant(X_train)  # Adding a constant
model_3 = sm.OLS(y_train, X_train).fit() # Fitting the training data
model_3.summary()
```

OLS Regression Results

Dep. Variable:	popularity	**R-squared:**	0.550
Model:	OLS	**Adj. R-squared:**	0.541
Method:	Least Squares	**F-statistic:**	56.75
Date:	Sun, 21 May 2023	**Prob (F-statistic):**	0.00
Time:	17:49:57	**Log-Likelihood:**	-12369.
No. Observations:	2700	**AIC:**	2.485e+04
Df Residuals:	2642	**BIC:**	2.520e+04
Df Model:	57		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Western	10.0677	9.227	1.091	0.275	-8.026	28.161
Crime	11.7103	13.653	0.858	0.391	-15.061	38.482
Animation	9.8599	8.144	1.211	0.226	-6.108	25.828
Foreign	20.9702	23.395	0.896	0.370	-24.904	66.844
Romance	10.4536	2.440	4.285	0.000	5.670	15.237
Adventure	11.9390	7.409	1.611	0.107	-2.589	26.467
Horror	5.7775	23.493	0.246	0.806	-40.289	51.844
Science Fiction	8.0596	5.621	1.434	0.152	-2.963	19.082
Family	-3.2455	23.648	-0.137	0.891	-49.615	43.124
Drama	2.3431	9.189	0.255	0.799	-15.674	20.361
Documentary	15.6877	16.726	0.938	0.348	-17.109	48.485
Action	2.0370	23.423	0.087	0.931	-43.892	47.966
War	21.3283	9.968	2.140	0.032	1.782	40.875
History	10.4099	7.641	1.362	0.173	-4.574	25.394
Fantasy	10.6718	9.841	1.084	0.278	-8.624	29.968
Comedy	1.4063	23.457	0.060	0.952	-44.589	47.402
Thriller	2.7312	16.779	0.163	0.871	-30.170	35.632
Music	3.8033	23.456	0.162	0.871	-42.191	49.797
Mystery	6.2069	23.375	0.266	0.791	-39.628	52.042
original_language_cn	3.3651	9.152	0.368	0.713	-14.580	21.311
original_language_da	11.3148	23.585	0.480	0.631	-34.932	57.562
original_language_de	5.7040	16.660	0.342	0.732	-26.963	38.371
original_language_el	3.6358	23.437	0.155	0.877	-42.321	49.592
original_language_en	8.7181	7.226	1.207	0.228	-5.451	22.887
original_language_es	17.8932	3.082	5.805	0.000	11.849	23.938
original_language_fa	15.5608	2.989	5.206	0.000	9.700	21.421
original_language_fr	15.2868	3.033	5.040	0.000	9.339	21.235
original_language_he	13.9259	3.028	4.599	0.000	7.989	19.863
original_language_hi	15.0933	2.994	5.041	0.000	9.223	20.964
original_language_id	19.1222	2.924	6.539	0.000	13.388	24.857
original_language_is	14.8590	2.909	5.108	0.000	9.155	20.563
original_language_it	16.7937	2.837	5.920	0.000	11.231	22.356
original_language_ja	15.4336	2.840	5.435	0.000	9.865	21.002
original_language_ko	15.2381	2.935	5.191	0.000	9.482	20.994
original_language_nb	19.1748	3.032	6.323	0.000	13.229	25.121

Selecting best features

The SelectKBest model evaluates the statistical significance of each feature's relationship with the target variable using F-value. Based on the scores, SelectKBest selects the top k features with the highest scores, which are the features that are most relevant for predicting the target variable.

```
# Selecting the best features for training using SelectKBest
# Selecting the top 5 features
selector = SelectKBest(score_func=f_regression, k=7) # Selecing using select k best
ch = selector.fit(X_train, y_train) # Fitting
X_train_selectk = ch.transform(X_train) # Transforming
```

```
 # fit and transform using selector
X_train_selected = selector.fit_transform(X_train, y_train)

# Get the selected feature indices
selected_features = selector.get_support(indices=True)
# Print the selected feature names
col_sel = X_train.columns[selected_features]
col_sel = list(col_sel)
col_sel
```

```
['original_language_no',
 'original_language_ro',
 'original_language_ru',
 'original_language_te',
 'release_date_month_3',
 'release_date_month_5',
 'release_date_month_9']
```

Notes:

```
X_train_selectk = pd.DataFrame(X_train_selectk,columns=col_sel)
X_train_selectk.head()
```

```
# X_train.columns[selected_features]
```

```
# Selecting for test as well
X_test_selectk = selector.transform(X_test)
```

```
# Checking training data
X_train_selectk.shape
```

```
(2700, 7)
```

```
# Checking testing data
X_test_selectk.shape
```

```
(676, 7)
```

```
# See the scaling of X_train

# Resetting index

y_train.reset_index(drop=True,inplace=True)

# Adding a constant column to the selected features

X_train_selectk = X_train_selectk.copy()

X_train_int = sm.add_constant(X_train_selectk)

# Creating the OLS model
model = sm.OLS(y_train, X_train_int)

# Fitting the model
OLS_SF = model.fit()

# Printing the summary
OLS_SF.summary()
```

OLS Regression Results

Dep. Variable:	popularity	**R-squared:**	0.543
Model:	OLS	**Adj. R-squared:**	0.542
Method:	Least Squares	**F-statistic:**	457.8
Date:	Sun, 21 May 2023	**Prob (F-statistic):**	0.00
Time:	17:49:57	**Log-Likelihood:**	-12390.
No. Observations:	2700	**AIC:**	2.480e+04
Df Residuals:	2692	**BIC:**	2.484e+04
Df Model:	7		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	27.4192	0.583	47.046	0.000	26.276	28.562
original_language_no	1.8555	0.618	3.003	0.003	0.644	3.067
original_language_ro	-0.4238	0.514	-0.824	0.410	-1.432	0.584
original_language_ru	0.9562	0.559	1.710	0.087	-0.140	2.053
original_language_te	24.3467	0.624	38.988	0.000	23.122	25.571
release_date_month_3	4.3053	1.332	3.233	0.001	1.694	6.916

Interpretation of SlectK OLS

Omnibus:	5611.716	**Durbin-Watson:**	2.052

- The model explains approximately 54.3% of the variability in the popularity.

original_language_no and release_date_month_1, have significant effects on popularity (low p-values).

Evaluating Results

We build a function to calculate and return the mean squared error (MSE) and root mean squared error (RMSE) score. It also adds the results to a pandas dataframe for later use. includes a conditional if the model is from statsmodels - it addes a constant.

```
# Creating an empty dataframe
results_df_ml = pd.DataFrame([],columns=['Model','MSE','RMSE'])
# Initialize an empty string for the name of the model
mname = ''

# Function
def  get_results(reg,x_test):
    mname = ''
    # If the reg model is from statmodels
    if 'statsmodels' in str(type(reg)):
        x_test = sm.add_constant(x_test)
        star = '*'
        mname = 'OLS - SelectK'
```

```
    # Getting predicted values
    y_pred = reg.predict(x_test)
    # Calculate mean squared error
    mse = mean_squared_error(y_test, y_pred)

    # Calculate root mean squared error
    rmse = mean_squared_error(y_test, y_pred,squared=False)

    print('mse',mse)
    print('rmse',rmse)

    # Obtaining the name for the reg model
    if not mname:
        mname = type(reg).__name__

    # Store the results in the DataFrame
    results_df_ml.loc[len(results_df_ml)] = [mname,mse,rmse]
    return results_df_ml
# https://datascience.stackexchange.com/questions/26555/valueerror-shapes-1-10-
# and-2-not-aligned-10-dim-1-2-dim-0
# https://stackoverflow.com/questions/54003129/valueerror-shapes-993-228-and-1
# -228-not-aligned-228-dim-1-1-dim-0
```

```
# Checking shape
X_test_selectk.shape
```

```
(676, 7)
```

```
# Run the function

# X_test_selectk = X_test.copy()
get_results(OLS_SF, X_test_selectk)
```

```
mse 470.6600243850183
rmse 21.694700375552973
```

	Model	MSE	RMSE
0	OLS - SelectK	470.660024	21.6947

Machine Learning Models

Random Forest Random Forest is an algorithm that combines multiple decision trees to make predictions.

It hands complex datasets. Random Forest aggregates the predictions of multiple trees, and that way it reduces overfitting and improves prediction accuracy.

XGBoost

XGBoost is a gradient boosting algorithm. It is used by training a series of weak learners of decision trees, and adding them to the ensemble. It focuses on the mistakes made by the previous learners, allowing the model to improve its predictions. It employs gradient boosting, where the next models are trained to minimize the errors of the previous models.

AdaBoost

AdaBoost, similar to XGBoost, also learns from weak learners, but it differs in the way it combines their predictions. While XGBoost uses gradient boosting to optimize the overall model, AdaBoost assigns weights to the weak learners based on their performance and focuses on samples with higher error.

Machine learning function

Initializing the models (Random Forest, XGBoost, and AdaBoost)

- Creating an empty DataFrame to store the results.
- Iterating through each model and performs the following:
    1. Fitting the model using the training data.
    2. Obtaining predictions on the test data.
    3. Calculating the (MSE) and (RMSE) between the predicted and actual values.
    4. Adding the model name, MSE, and RMSE to the results DataFrame.
- Returning the results DataFrame containing the model names, MSE, and RMSE for each model.

```
def run_regression_models(X_train, y_train, X_test, y_test):
    # Initialize the models
    models = {
        'Random Forest': RandomForestRegressor(),
        'XGBoost': XGBRegressor(),
        'AdaBoost': AdaBoostRegressor()

    }

    # Initialize the DataFrame to store results
    # Results_df_ml = pd.DataFrame(columns=['Model', 'MSE', 'RMSE'])

    # Loop through each model
    for model_name, model in models.items():
        # Fit the model
        model.fit(X_train, y_train)
```

```
        # Make predictions
        y_pred = model.predict(X_test)

        # Calculate MSE and RMSE
        mse = mean_squared_error(y_test, y_pred)
        rmse = mean_squared_error(y_test, y_pred, squared=False)

#         results_df_ml
        # Add results to the DataFrame

        results_df_ml.loc[len(results_df_ml)] = [model_name,mse,rmse]
#         results_df_ml = results_df_ml.append({
#             'Model': model_name,
#             'MSE': mse,
#             'RMSE': rmse
#         }, ignore_index=True)

    return results_df_ml
```

```
# Viewing resutls of the ML models
results_df_ml = run_regression_models(X_train, y_train, X_test, y_test)
results_df_ml
```

	Model	MSE	RMSE
0	OLS - SelectK	470.660024	21.694700
1	Random Forest	432.597364	20.798975
2	XGBoost	979.035084	31.289536
3	AdaBoost	1002.931148	31.669088

OLS Select performed better than the othermodels in predicting movie popularity. We will explore the potential of a General Multilayer Perceptron (MLP) model, a deep learning approach, to further improve our predictions.

Multilayer Perceptron MLP

MLP is a type of neural network that consists of 3 or more layers of neurons. MLP, short for MLP, is a type of neural network that is composed of multiple layers of nodes, with each node being a simple computational unit that performs a mathematical operation. The MLP takes input data, processes it through the layers of nodes, and produces output predictions.

During training, the weights between nodes are adjusted through a process called backpropagation. The weights are updated to minimize the difference between the predicted outputs and the actual outputs.

The input shape reflects the dimensions of the data that will be fed into the mode

```
# Setting the input shape
input_shape = (X_train.shape[1],)
print(f'Feature shape: {input_shape}')
```

```
Feature shape: (59,)
```

We utilized two callbacks:

1. ReduceLROnPlateau adjusts the learning rate based on the validation loss.
2. EarlyStopping stops training if the mean squared error improvement is below a certain threshold.

```
# Initializing callback
callbacks = [ ReduceLROnPlateau(monitor='val_loss', patience=5, cooldown=0),
              EarlyStopping(monitor='mean_squared_error',
                            min_delta=1e-4,
                            patience=5)
                             ]
```

```
# Create the model
mlp_model = Sequential()

# Adding the input layer with 16 neurons and ReLU activation
mlp_model.add(Dense(16, input_shape=input_shape, activation='relu'))

# Adding a hidden layer with 8 neurons and ReLU activation
mlp_model.add(Dense(8, activation='relu'))

# Adding the output layer with 1 neuron and linear activation
mlp_model.add(Dense(1, activation='linear'))

# Configure the model and start training
mlp_model.compile(loss='mean_squared_error', optimizer='adam',
                  metrics=['mean_squared_error'])

# Train the model on the training
mlp_model.fit(X_train, y_train, epochs=5, batch_size=32,
          verbose=1, validation_split=0.2,callbacks=callbacks)
```

```
Epoch 1/5
68/68 [==============================] - 2s 7ms/step - loss: 2134.9712 - mean_squ
Epoch 2/5
68/68 [==============================] - 0s 4ms/step - loss: 1957.2526 - mean_squ
Epoch 3/5
68/68 [==============================] - 0s 4ms/step - loss: 1479.1876 - mean_squ
Epoch 4/5
68/68 [==============================] - 0s 4ms/step - loss: 1078.5012 - mean_squ
Epoch 5/5
68/68 [==============================] - 0s 4ms/step - loss: 938.9202 - mean_squa
<keras.callbacks.History at 0x7febbfd07100>
```

```
'''# True vs predicted
out = pd.DataFrame({
    'y_true':y_test[:10],
    'y_pred':mlp_model.predict(X_test[:10]).ravel()
}
).T

out '''
```

```
'# True vs predicted\nout = pd.DataFrame({\n    'y_true':y_test[:10],\n    'y_pr
```

```
# Making predictions on the test set
y_pred_mlp = mlp_model.predict(X_test)
```

```
22/22 [==============================] - 0s 2ms/step
```

```
# Calculating mean squared error
mlp_mse = mean_squared_error(y_test, y_pred_mlp)
print('Mean Squared Error:', mlp_mse)
```

```
Mean Squared Error: 801.5169342213096
```

```
# Seeing the shape
X_train.shape
```

```
(2700, 59)
```

```
# Calculating root mean squared error
mlp_rmse = mean_squared_error(y_test, y_pred_mlp,squared=False)
print('Root Mean Squared Error:', round(mlp_rmse,3))
```

```
Root Mean Squared Error: 28.311
```

```
# Adding results for mlp model
mlp_results = {'Model': 'MLP', 'MSE': mlp_mse, 'RMSE': mlp_rmse}
```

```
results_df_ml = results_df_ml.append(mlp_results, ignore_index=True)
results_df_ml
```

	Model	MSE	RMSE
0	OLS - SelectK	470.660024	21.694700
1	Random Forest	432.597364	20.798975
2	XGBoost	979.035084	31.289536
3	AdaBoost	1002.931148	31.669088
4	MLP	801.516934	28.311074

```
# Sorting by RMSE in descending order
results_df_ml_sorted = results_df_ml.sort_values(by='RMSE',
                                                 ascending=False)

# Creating a bar plot with RMSE values in descending order
fig = go.Figure(data=go.Bar(x=results_df_ml_sorted['Model'],
                            y=results_df_ml_sorted['RMSE'],
                            marker_color='lightskyblue'))

# Find the index of the lowest RMSE value
lowest_rmse_index = results_df_ml_sorted['RMSE'].idxmin()

# Add a marker for the lowest RMSE value
fig.add_trace(go.Scatter(x=[results_df_ml_sorted['Model'][lowest_rmse_index]],
                         y=[results_df_ml_sorted['RMSE'][lowest_rmse_index]],
                         mode='markers', marker=dict(color='red', size=10),
                         name='Lowest RMSE'))

# Customize the layout
fig.update_layout(title='RMSE Comparison',
                  xaxis_title='Model',
                  yaxis_title='RMSE',
                  showlegend=True)

# Show the plot
fig.show()
```

Random Forest was the top performer. While MLP did not perform as well, we will explore hyperparameter tuning for MLP to see if we can enhance its performance.

Hypertune Random Forest

```
# Seeing result
results_df_ml
```

	Model	MSE	RMSE
0	OLS - SelectK	470.660024	21.694700
1	Random Forest	432.597364	20.798975
2	XGBoost	979.035084	31.289536
3	AdaBoost	1002.931148	31.669088
4	MLP	801.516934	28.311074

We will explore different combinations of these parameters, we aim to find the optimal cmbinations that can improve the model's performance.

```
# https://www.kaggle.com/code/sociopath00/random-forest-using-gridsearchcv
# Setting up parameters
parameters = {
    'n_estimators': [100, 150, 200, 250, 300,400],
```

```
    'max_depth': [1,2,3,4,5,7,9],
}
```

```
# Creating the Random Forest regressor
rf = RandomForestRegressor(n_jobs=-1) #
```

We will use 5-fold cross-validation to evaluate the performance of each parameter combination.

```
# Performing randomized search with progress bar
gird_search_rf = GridSearchCV(estimator=rf,
                                      param_grid=parameters,
                                      verbose=1,n_jobs=-1,
                                      cv=5)
```

```
# Fitting grid search
rf_model = gird_search_rf.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 42 candidates, totalling 210 fits
```

```
# Getting best estimator
grid_rf = rf_model.best_estimator_
grid_rf
```

```
▾ RandomForestRegressor
RandomForestRegressor(max_depth=2, n_jobs=-1)
```

```
# Fitting grid
grid_rf.fit(X_train, y_train)
```

```
▾ RandomForestRegressor
RandomForestRegressor(max_depth=2, n_jobs=-1)
```

```
# Predicting
y_pred_rf_tuned = grid_rf.predict(X_test)
```

```
# Get the best hyperparameters and model
# best_params_rf = random_search_rf.best_params_
# best_model_rf = random_search_rf.best_estimator_
```

```
# Calculating RMSE
tuned_rf_rmse = mean_squared_error(y_test, y_pred_rf_tuned,squared=False)
```

```
print('Root Mean Squared Error:', round(tuned_rf_rmse,3))
```

```
Root Mean Squared Error: 21.584
```

```
# Calculating MSE
tuned_rf_mse = mean_squared_error(y_test, y_pred_rf_tuned,squared=True)
print('Mean Squared Error:', round(tuned_rf_mse,3))
```

```
Mean Squared Error: 465.89
```

```
# Adding results for MLP model
grid_rf_results = {'Model': 'RF_Tuned', 'MSE': tuned_rf_mse,
                   'RMSE': tuned_rf_rmse }

results_df_ml = results_df_ml.append(grid_rf_results,
                                     ignore_index=True)
# Sorting it out
results_df_ml.sort_values('RMSE', inplace =True)
results_df_ml
```

	Model	MSE	RMSE
1	Random Forest	432.597364	20.798975
5	RF_Tuned	465.889631	21.584477
0	OLS - SelectK	470.660024	21.694700
4	MLP	801.516934	28.311074
2	XGBoost	979.035084	31.289536
3	AdaBoost	1002.931148	31.669088

Hypertune MLP

```
# Create the MLP regressor
mlp = mlp_model
```

```
# Installign scikeras
# pip install scikeras
```

We define a parameter grid of different options for optimizers, epochs, and batch sizes. Then we aplly GridSearchCV with the specified parameter grid to find the best combination of hyperparameters.

```
# Keras model using MLP architecture
Kmodel = KerasRegressor(build_fn=mlp)

# Hyperparameter options for grid search
optimizers = ['rmsprop', 'adam']

epochs = np.array([50, 100])
batches = np.array([10, 20])
param_grid = dict(optimizer=optimizers, epochs=epochs, batch_size=batches)

# Grid search using cross-validation
grid_search_mlp = GridSearchCV(estimator=Kmodel, param_grid=param_grid,
                               n_jobs=-1,cv=3,verbose=2)


#Source: https://www.kaggle.com/code/shujunge/gridsearchcv-with-keras
      # https://stackoverflow.com/questions/60350049/tensorflow-fit-
      # gives-typeerror-cannot-clone-object-error
```

```
# Returns the valid parameters for the Kmodel
Kmodel.get_params().keys()
```

```
dict_keys(['model', 'build_fn', 'warm_start', 'random_state', 'optimizer',
'loss', 'metrics', 'batch_size', 'validation_batch_size', 'verbose',
'callbacks', 'validation_split', 'shuffle', 'run_eagerly', 'epochs'])
```

```
# Fitting the mdoel
grid_search_mlp.fit(X_train, y_train)
```

```
WARNING:absl:Found untraced functions such as _update_step_xla while saving (sho
Fitting 3 folds for each of 8 candidates, totalling 24 fits
WARNING:absl:Found untraced functions such as _update_step_xla while saving (sho
WARNING:absl:Found untraced functions such as _update_step_xla while saving (sho
WARNING:absl:Found untraced functions such as _update_step_xla while saving (sho
WARNING:absl:Found untraced functions such as _update_step_xla while saving (sho
WARNING:absl:Found untraced functions such as _update_step_xla while saving (sho
WARNING:absl:Found untraced functions such as _update_step_xla while saving (sho
WARNING:absl:Found untraced functions such as _update_step_xla while saving (sho
WARNING:absl:Found untraced functions such as _update_step_xla while saving (sho
WARNING:absl:Found untraced functions such as _update_step_xla while saving (sho
WARNING:absl:Found untraced functions such as _update_step_xla while saving (sho
WARNING:absl:Found untraced functions such as _update_step_xla while saving (sho
WARNING:absl:Found untraced functions such as _update_step_xla while saving (sho
WARNING:absl:Found untraced functions such as _update_step_xla while saving (sho
WARNING:absl:Found untraced functions such as _update_step_xla while saving (sho
WARNING:absl:Found untraced functions such as _update_step_xla while saving (sho
WARNING:tensorflow:Detecting that an object or model or tf.train.Checkpoint is b
WARNING:tensorflow:Value in checkpoint could not be found in the restored object
WARNING:tensorflow:Value in checkpoint could not be found in the restored object
WARNING:tensorflow:Value in checkpoint could not be found in the restored object
WARNING:tensorflow:Value in checkpoint could not be found in the restored object
WARNING:tensorflow:Detecting that an object or model or tf.train.Checkpoint is b
WARNING:tensorflow:Value in checkpoint could not be found in the restored object
WARNING:tensorflow:Value in checkpoint could not be found in the restored object
WARNING:tensorflow:Value in checkpoint could not be found in the restored object
WARNING:tensorflow:Value in checkpoint could not be found in the restored object
WARNING:absl:Found untraced functions such as _update_step_xla while saving (sho
WARNING:absl:Found untraced functions such as _update_step_xla while saving (sho
WARNING:absl:Found untraced functions such as _update_step_xla while saving (sho
WARNING:absl:Found untraced functions such as _update_step_xla while saving (sho
WARNING:tensorflow:Detecting that an object or model or tf.train.Checkpoint is b
WARNING:tensorflow:Value in checkpoint could not be found in the restored object
WARNING:tensorflow:Value in checkpoint could not be found in the restored object
WARNING:tensorflow:Value in checkpoint could not be found in the restored object
WARNING:tensorflow:Value in checkpoint could not be found in the restored object
WARNING:tensorflow:Detecting that an object or model or tf.train.Checkpoint is b
WARNING:tensorflow:Value in checkpoint could not be found in the restored object
WARNING:tensorflow:Value in checkpoint could not be found in the restored object
WARNING:tensorflow:Value in checkpoint could not be found in the restored object
WARNING:tensorflow:Value in checkpoint could not be found in the restored object
WARNING:tensorflow:Detecting that an object or model or tf.train.Checkpoint is b
WARNING:tensorflow:Value in checkpoint could not be found in the restored object
WARNING:tensorflow:Value in checkpoint could not be found in the restored object
WARNING:tensorflow:Value in checkpoint could not be found in the restored object
WARNING:tensorflow:Value in checkpoint could not be found in the restored object
WARNING:tensorflow:Detecting that an object or model or tf.train.Checkpoint is b
WARNING:tensorflow:Value in checkpoint could not be found in the restored object
WARNING:tensorflow:Value in checkpoint could not be found in the restored object
WARNING:tensorflow:Value in checkpoint could not be found in the restored object
WARNING:tensorflow:Value in checkpoint could not be found in the restored object
WARNING:tensorflow:Detecting that an object or model or tf.train.Checkpoint is b
WARNING:tensorflow:Value in checkpoint could not be found in the restored object
WARNING:tensorflow:Value in checkpoint could not be found in the restored object
WARNING:tensorflow:Value in checkpoint could not be found in the restored object
WARNING:tensorflow:Value in checkpoint could not be found in the restored object
```

```
WARNING:tensorflow:Detecting that an object or model or tf.train.Checkpoint is be
```

```
# Get the best hyperparameters and model
best_params_mlp = grid_search_mlp.best_params_
best_model_mlp = grid_search_mlp.best_estimator_
```

```
WARNING:tensorflow:Detecting that an object or model or tf.train.Checkpoint is be
```

```
# Evaluate the best model
test_loss_mlp = mean_squared_error(y_test, best_model_mlp.predict(X_test))
test_loss_mlp
```

```
68/68 [==============================] - 0s 1ms/step
528.4815112712278
```

```
WARNING:tensorflow:Value in checkpoint could not be found in the restored object
```

```
# Getting predicted values
y_pred_mlp_tuned = best_model_mlp.predict(X_test)
```

```
68/68 [==============================] - 0s 1ms/step
```

```
WARNING:tensorflow:Value in checkpoint could not be found in the restored object
```

```
# Calculating RSME
test_loss_mlp = mean_squared_error(y_test, y_pred_mlp_tuned,squared= False)
print('Root Mean Squared Error:', round(test_loss_mlp,3))
```

```
Root Mean Squared Error: 22.989
```

```
WARNING:tensorflow:Unresolved object in checkpoint: (root).layer_with_weights-0.
```

```
results_all = get_results(best_model_mlp, X_test)
results_all.sort_values('RMSE', inplace=True)
```

```
68/68 [==============================] - 0s 1ms/step
mse 528.4815112712278
rmse 22.988725742659764
```

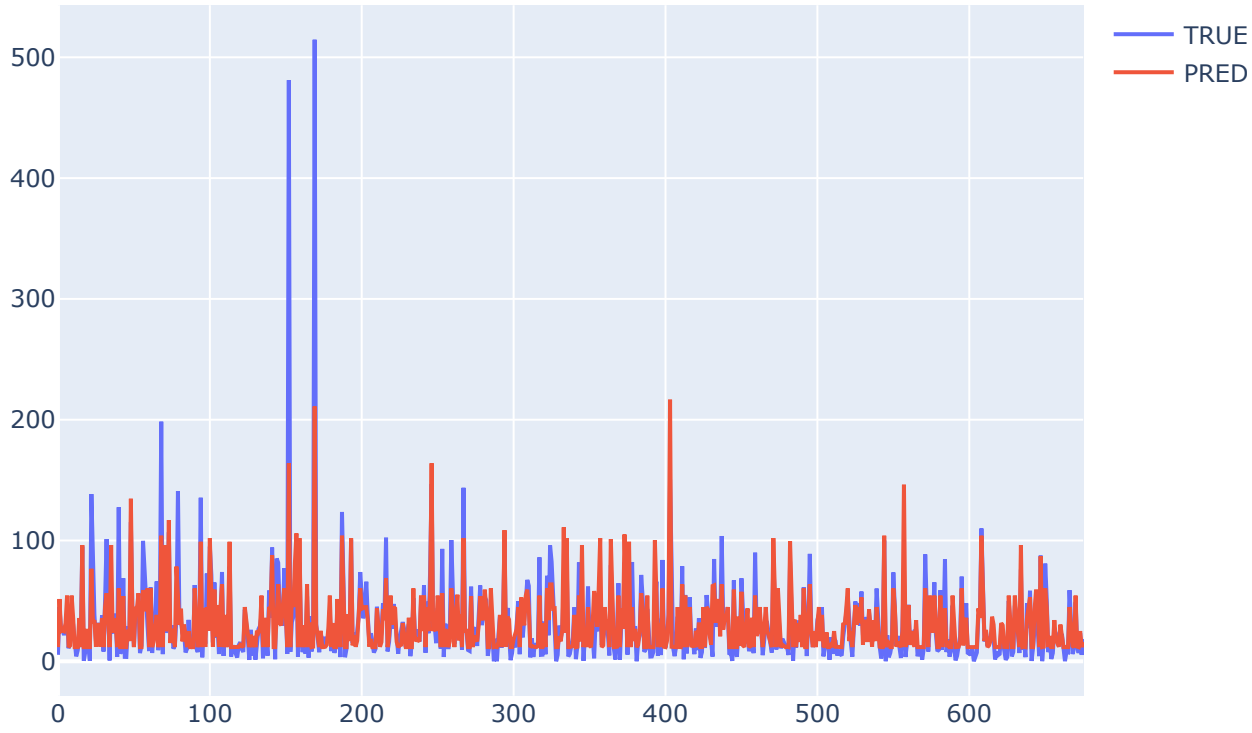
```
WARNING:tensorflow:Detecting that an object or model or tf.train.Checkpoint is be
```

```
results_all
```

	Model	MSE	RMSE
1	Random Forest	432.597364	20.798975
5	RF_Tuned	465.889631	21.584477
0	OLS - SelectK	470.660024	21.694700
6	KerasRegressor	528.481511	22.988726
4	MLP	801.516934	28.311074
2	XGBoost	979.035084	31.289536
3	AdaBoost	1002.931148	31.669088

```
Epoch 5/50
```

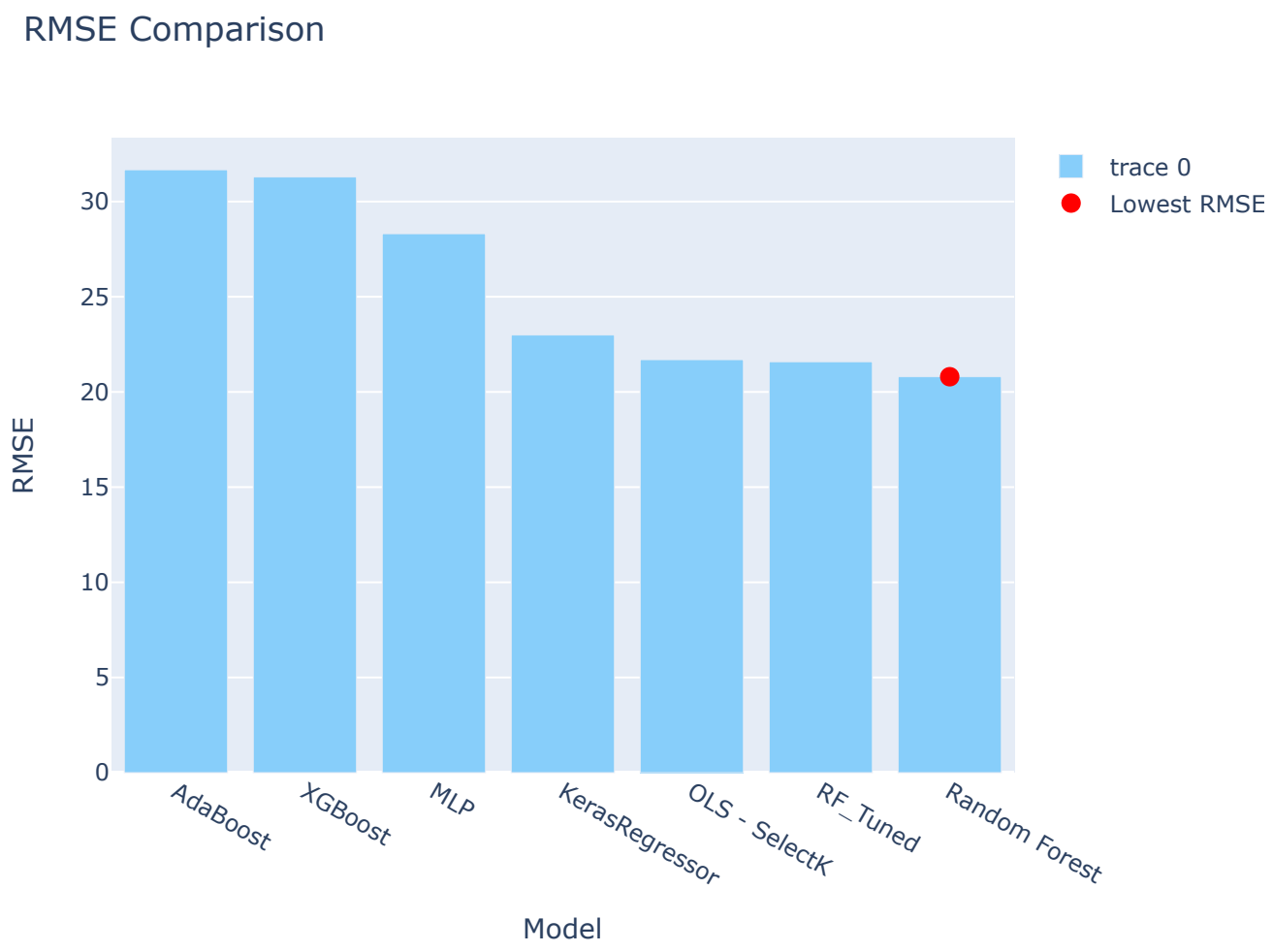
```
fig = go.Figure()
fig.add_trace(go.Scatter(x=list(range(0, len(y_pred_rf_tuned))),
                         y=y_test,
                         name='TRUE'))
fig.add_trace(go.Scatter(x=list(range(0, len(y_pred_rf_tuned))),
                         y=y_pred_rf_tuned,
                         name='PRED'))

fig.show()
```




```
270/270 [==============================] - 1s 2ms/step - loss: 533.2509 - mean_s
```

```
import plotly.graph_objects as go

# Sorting by RMSE in descending order
results_df_ml_sorted = results_df_ml.sort_values(by='RMSE',
                                                 ascending=False)

# Creating a bar plot with RMSE values in descending order
fig = go.Figure(data=go.Bar(x=results_df_ml_sorted['Model'],
                            y=results_df_ml_sorted['RMSE'],
                            marker_color='lightskyblue'))

# Find the index of the lowest RMSE value
```

```
lowest_rmse_index = results_df_ml_sorted['RMSE'].idxmin()

# Add a marker for the lowest RMSE value
fig.add_trace(go.Scatter(x=[results_df_ml_sorted['Model'][lowest_rmse_index]],
                         y=[results_df_ml_sorted['RMSE'][lowest_rmse_index]],
                         mode='markers', marker=dict(color='red', size=10),
                         name='Lowest RMSE'))

# Customize the layout
fig.update_layout(title='RMSE Comparison',
                  xaxis_title='Model',
                  yaxis_title='RMSE',
                  showlegend=True)

# Show the plot
fig.show()
```



Features of Importance

Now we will analyze the importance of different features.

```
feature_names = list(X_train)
feature_names
```

```
['Western',
 'Crime',
 'Animation',
 'Foreign',
 'Romance',
 'Adventure',
 'Horror',
 'Science Fiction',
 'Family',
 'Drama',
 'Documentary',
 'Action',
 'War',
 'History',
 'Fantasy',
 'Comedy',
 'Thriller',
 'Music',
 'Mystery',
 'original_language_cn',
 'original_language_da',
 'original_language_de',
 'original_language_el',
 'original_language_en',
 'original_language_es',
 'original_language_fa',
 'original_language_fr',
 'original_language_he',
 'original_language_hi',
 'original_language_id',
 'original_language_is',
 'original_language_it',
 'original_language_ja',
 'original_language_ko',
 'original_language_nb',
 'original_language_nl',
 'original_language_no',
 'original_language_ro',
 'original_language_ru',
 'original_language_te',
 'original_language_th',
 'original_language_vi',
 'original_language_zh',
 'release_date_month_1',
 'release_date_month_2',
 'release_date_month_3',
 'release_date_month_4',
 'release_date_month_5',
 'release_date_month_6',
 'release_date_month_7',
 'release_date_month_8',
 'release_date_month_9',
```

```
        'release_date_month_10',
        'release_date_month_11',
        'release_date_month_12',
        'runtime',
        'vote_average',
        'vote count',
```

```
# Define Random Forest Tuned model
forest = grid_rf
```

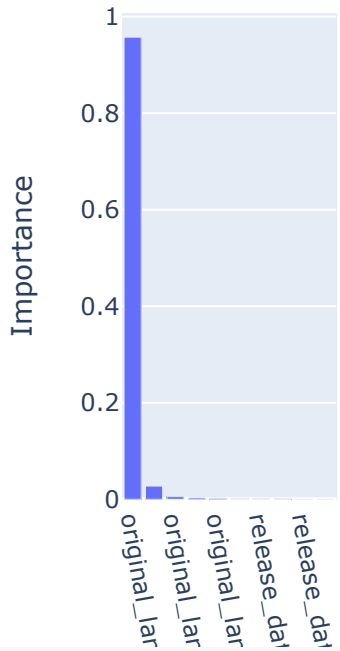
```
# Calculating the feature importances

importances = forest.feature_importances_

std = np.std([tree.feature_importances_ for tree in forest.estimators_], axis=0)
```

```
# Creating a pandas series for feature importances

forest_importances = pd.Series(importances, index=feature_names)
```

```
# Sort the features by top 10 most important
forest_importances = forest_importances.sort_values(ascending=False)[:10]
forest_importances
```

```
original_language_te    0.957424
original_language_ro    0.027401
original_language_nb    0.005521
original_language_id    0.003202
original_language_ru    0.002416
original_language_no    0.001600
release_date_month_9    0.001518
original_language_is    0.000919
release_date_month_1    0.000000
original_language_ja    0.000000
dtype: float64
```

```
#Plot teh features

fig = go.Figure(data=go.Bar(x=forest_importances.index, y=forest_importances.value
fig.update_layout(title="Feature Importances", xaxis_title="Features", yaxis_title
fig.update_xaxes(tickangle=80)
fig.show()
```




```
### Conclusion

RMSE quantifies the average distance
between the predicted values and the
actual values, and indicates how well
the model's predictions match the true
values. A lower RMSE value indicates
that the model's predictions are closer
to the actual values.

Random Forest (RF) model with
hyperparameter tuning and the OLS with
SelectKBest features model have achieved
the best results with the least error.
These models show lower MSE and RMSE
values compared to the other models,
indicating better accuracy in predicting
the target variable.
```

Conclusion

RMSE quantifies the average distance between the predicted values and the actual values, and indicates how well the model's predictions match the true values. A lower RMSE value indicates that the model's predictions are closer to the actual values.

Random Forest (RF) model with hyperparameter tuning and the OLS with SelectKBest features model have achieved the best results with the least error. These models show lower MSE and RMSE values compared to the other models, indicating better accuracy in predicting the target variable.

Save and Load the Model for the Demo

```
from joblib import Parallel, delayed
import joblib

# Save the model as a pickle in a file
joblib.dump(gird_search_rf,'gird_search_rf.pkl')

joblib.dump(preprocessing, 'preprocessing.pkl')

# Load the model from the file
rf_from_joblib = joblib.load('gird_search_rf.pkl')

cleaner_from_joblib = joblib.load('preprocessing.pkl')

# Use the loaded model to make predictions
y_pred_rf = rf_from_joblib.predict(X_test)

#https://www.geeksforgeeks.org/saving-a-machine-learning-model/
```

We will test to see if joblib works by adding a new input and see if it gives the prediciton based on our model.

```
# Seeing df
df.head(1)
```

	budget	genres	popularity	original_language	ru
0	237000000	[Action, Adventure, Fantasy, Science Fiction]	150.437577	en	

```
# Testing with new input
X_new = [237000,['Action', 'Adventure'],'en','162.0','7.2','11800','12']
```

```
# Transpose the data
df_new = pd.DataFrame(X_new).T
df_new
```

	0	1	2	3	4	5	6
0	237000	[Action, Adventure]	en	162.0	7.2	11800	12

```
# Adding the columns
df_new.columns = ['budget',
 'genres',
 'original_language',
 'runtime',
 'vote_average',
 'vote_count',
 'release_date_month']

# Seeing new df
df_new
```

	budget	genres	original_language	runtime	vote_average	vote_count	r
0	237000	[Action, Adventure]	en	162.0	7.2	11800	

```
# Transforming df new
df_new = cleaner_from_joblib.transform(df_new)
```

```
100%                                                19/19 [00:00<00:00, 563.79it/s]
```

```
# Getting pred values
y_pred_rf = rf_from_joblib.predict(df_new)
```

```
# Predicted values
y_pred_rf[0]
```

```
222.08995402224926
```

✓ 0s completed at 2:00 PM