Movie Box Office Gross Prediction using Machine Learning

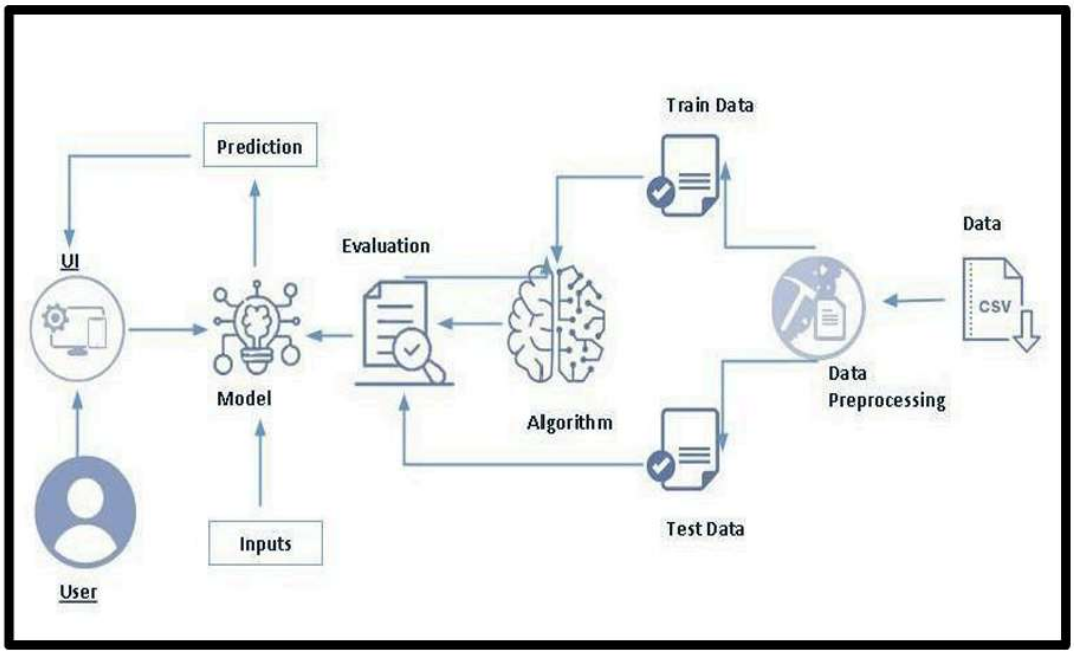
Movie Box Office Gross Prediction Using Machine Learning

The film industry invests heavily in movie production, marketing, and distribution. However, predicting a movie's box office success before release remains a challenge. Relying on intuition or past experience can often lead to poor financial decisions.

This project uses machine learning to predict a movie's box office gross based on various pre-release features such as genre, cast, director, budget, and release date. By analyzing historical movie data, the model identifies patterns that influence a film's commercial performance.

The goal is to help producers, investors, and marketers make informed decisions, minimize risk, and optimize resources to maximize returns.

Technical Architecture:



Project Flow:

● User interacts with the UI to enter the input.

● Entered input is analyzed by the model which is integrated.

● Once the model analyzes the input, the prediction is showcased on the UI.

To accomplish this, we have completed all the activities listed below:

1. Define Problem / Problem Understanding
    - Specify the business problem.
    - Define business requirements.
    - Conduct literature survey.
    - Describe social or business impact.
2. Data Collection & Preparation
    - Collect the dataset.
    - Prepare and clean the data.
3. Exploratory Data Analysis
    - Perform descriptive statistical analysis.
    - Conduct visual analysis (boxplots, heatmaps, correlations).
4. Model Building
    - Train the model using multiple regression algorithms (Linear Regression, Random Forest, XGBoost).
    - Test the model on validation data.
5. Performance Testing & Hyperparameter Tuning
    - Evaluate the model using multiple regression metrics:Mean Absolute Error (MAE), Root Mean Squared Error (RMSE) and R^2 Score
    - Compare model performance before and after applying hyperparameter tuning (optional for this project).
6. Model Deployment
    - Save the best performing model using Pickle.
    - Integrate the model with a web framework (Flask).
    - Build HTML templates to accept user inputs and show predictions.
7. Project Demonstration & Documentation
    - Record an explanation video for the end-to-end solution.
    - Prepare project documentation describing step-by-step development.

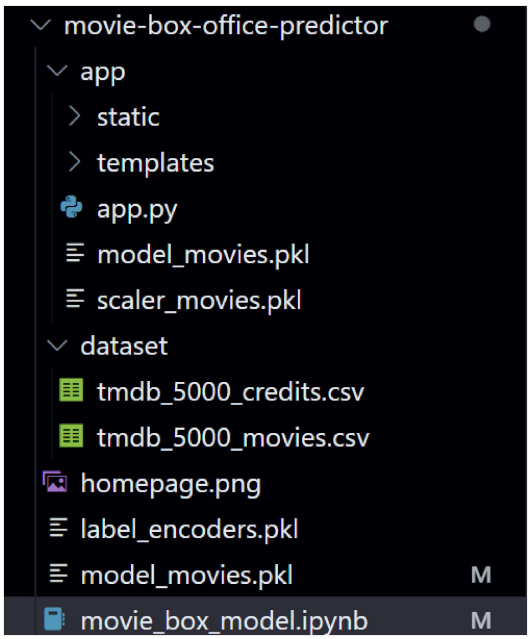
Prior Knowledge:

You must have prior knowledge of the following topics to complete this project:

- **ML Concepts:**
  - Supervised learning: https://www.javatpoint.com/supervised-machine-learning
  - Unsupervised learning: https://www.javatpoint.com/unsupervised-machine-learning
- **Regression Models:**
  - Linear Regression: https://scikit-learn.org/stable/modules/linear_model.html#ordinary-least-squares
  - Random Forest Regressor: https://www.javatpoint.com/machine-learning-random-forest-algorithm
  - XGBoost Regressor: https://www.analyticsvidhya.com/blog/2018/09/an-end-to-end-guide-to-understand-the-math-behind-xgboost/
- **Evaluation Metrics for Regression:**
  - MAE, RMSE, R^2 Score: https://www.analyticsvidhya.com/blog/2019/08/11-important-model-evaluation-error-metrics/
- **Flask Basics:**
  - How to build a web app with Flask: https://www.youtube.com/watch?v=lj4I_CvBnt0

Project Structure:

Create the Project folder which contains files as shown below



● We are building a Flask application which needs HTML pages stored in the templates folder and a Python script app.py for scripting.
● model_movies.pkl is our saved model. Further we will use this model for Flask integration.
● scaler_movies.pkl and label_encoders.pkl are saved preprocessing objects used during prediction.
● dataset folder contains the datasets used (tmdb_5000_movies.csv and tmdb_5000_credits.csv).
● The Notebook file contains procedure for building the model.

Milestone 1: Define Problem / Problem Understanding

Activity 1: Specify the Business Problem

The entertainment industry, particularly film production and distribution, involves significant financial investments with high risk. One major challenge faced by producers, investors, and distributors is predicting a movie's commercial success before release. Traditional decision-making based on intuition, past experience, or genre trends lacks the accuracy and reliability needed in today's data-driven environment.

Activity 2: Business Requirements

A box office prediction project has several key business requirements that ensure its effectiveness, reliability, and real-world applicability:

- **Accurate Predictions:** The model should produce accurate predictions of box office revenue, helping stakeholders make data-driven decisions regarding production and marketing.
- **Data Integration:** The system should integrate a wide range of features — including budget, genre, cast popularity, director's track record, release timing, competition, and promotion strategy — to improve prediction quality.
- **Scalability:** The model should handle a large number of movies and attributes, allowing for scalability as more data becomes available over time.
- **Actionable Insights:** Output should be interpretable and provide useful insights for stakeholders to take informed actions (e.g., optimize marketing budget, choose a better release window).
- **User-friendly Interface:** The system should include a dashboard or web-based interface that is accessible to producers, analysts, and studio executives with minimal technical knowledge.
- **Regulatory Compliance (if applicable):** If the system uses third-party data or publicly scraped content, it should comply with data privacy and copyright regulations.

Activity 3: Literature Survey

A literature survey for box office prediction using machine learning would include reviewing existing research papers, industry reports, and case studies on predictive models for film success. Topics of interest include:

- Regression and classification models used in prior studies (e.g., linear regression, decision trees, random forests, XGBoost, neural networks).
- Features contributing to prediction accuracy (e.g., IMDb ratings, trailer engagement, social media sentiment).
- Challenges identified in earlier works, such as data sparsity, release date effect, and dynamic consumer behavior.

- Evaluation metrics used to assess model performance (e.g., RMSE, MAE, R^2 score).

Activity 4: Social or Business Impact

Social Impact:
By helping filmmakers better understand audience preferences and market trends, box office prediction models can lead to the production of more relevant and engaging content, enhancing the overall movie-watching experience for viewers.

Business Impact:
Accurate revenue predictions enable producers and investors to make more informed financial decisions, minimizing losses and maximizing returns by backing movies with higher commercial potential.

Milestone 2: Data Collection & Preparation

ML depends heavily on data. It is the most crucial aspect that makes algorithm training possible. So, this section allows you to download the required dataset.

Activity 1: Collect the Dataset

There are many popular open sources for collecting the data. Eg: kaggle.com, UCI repository, etc. In this project we have used .csv data. This data is downloaded from kaggle.com. Please refer to the link given below to download the dataset.

Link: https://www.kaggle.com/datasets/tmdb/tmdb-movie-metadata

As the dataset is downloaded. Let us read and understand the data properly with the help of some visualisation techniques and some analysing techniques.

Note: There are a number of techniques for understanding the data. But here we have used some of it. In an additional way, you can use multiple techniques.

Activity 1.1: Importing the Libraries

To begin with, all essential libraries for data processing, visualization, and modeling are:

Step 1: Importing Libraries

```
import pandas as pd
import numpy as np
import seaborn as sns
import json
import matplotlib.pyplot as plt
import pickle
from wordcloud import WordCloud
from ast import literal_eval
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from sklearn.linear_model import LinearRegression
from collections import Counter as c
```

✓ 14.8s

Activity 1.2: Read the data set

Read the Dataset Our dataset format might be in .csv, excel files, .txt, .json, etc. We can read the dataset with the help of pandas.

In pandas we have a function called read_csv() to read the dataset. As a parameter we have to give the directory of the csv file.

```
credits = pd.read_csv(r"dataset/tmdb_5000_credits.csv")
movies_df = pd.read_csv(r"dataset/tmdb_5000_movies.csv")
```

✓ 0.4s

```
credits.head()
movies_df.head()
print(credits.columns)
print(movies_df.columns)
print(credits.shape)
print(movies_df.shape)
```

```
Index(['movie_id', 'title', 'cast', 'crew'], dtype='object')
Index(['budget', 'genres', 'homepage', 'id', 'keywords', 'original_language',
       'original_title', 'overview', 'popularity', 'production_companies',
       'production_countries', 'release_date', 'revenue', 'runtime',
       'spoken_languages', 'status', 'tagline', 'title', 'vote_average',
       'vote_count'],
      dtype='object')
(4803, 4)
(4803, 20)
```

Activity 2: Data Preparation

As we have understood how the data is, let's pre-process the collected data. The download data set is not suitable for training the machine learning model as it might have so much randomness so we need to clean the dataset properly in order to fetch good results. This activity includes the following steps

- Handling missing values
- Handling Outliers

Note: These are the general steps of pre-processing the data before using it for machine learning. Depending on the condition of your dataset, you may or may not have to go through all these steps

Activity 2.1: Handling missing values

For checking the null values, df.isna().any( ) function is used. To sum those null values we use .sum() function. From the below image we found that there are no null values present in our dataset. So we can skip handling the missing values step.

```
movies.isnull().any()
```

```
budget                 False
genres                 False
homepage                True
id                     False
keywords               False
original_language      False
original_title         False
overview                True
popularity             False
production_companies   False
production_countries   False
release_date            True
revenue                False
runtime                 True
spoken_languages       False
status                 False
tagline                 True
title_x                False
vote_average           False
vote_count             False
title_y                False
cast                   False
director               False
dtype: bool
```

```
movies.isnull().sum()
```

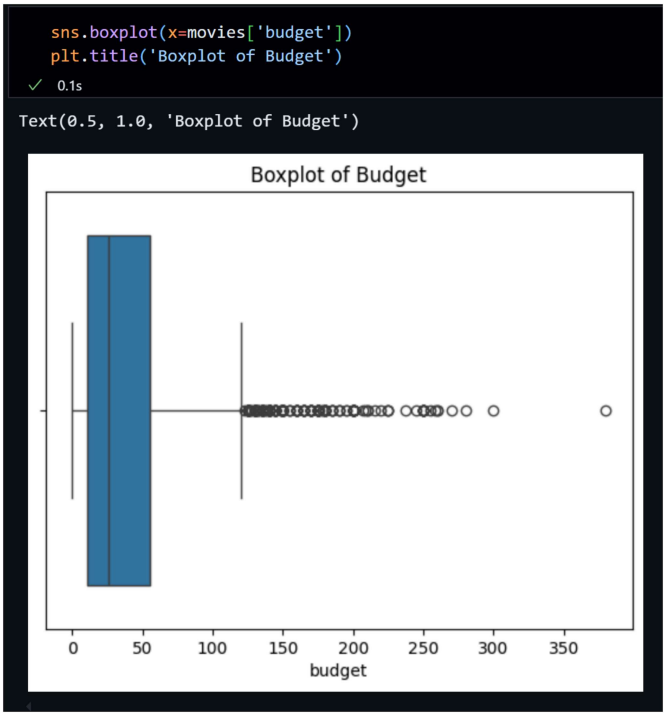
```
budget                    0
genres                    0
homepage               3091
id                        0
keywords                  0
original_language         0
original_title            0
overview                  3
popularity                0
production_companies      0
production_countries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

Activity 2.2: Handling Outliers

With the help of boxplots, outliers are visualized.
Here, we are going to find the upper bound and lower bound of different features (*budget*, *runtime*, and *revenue*) using a mathematical formula.

1.Budget

From the below diagram, we could visualize that the **budget** feature has several outliers (movies with very high budgets).



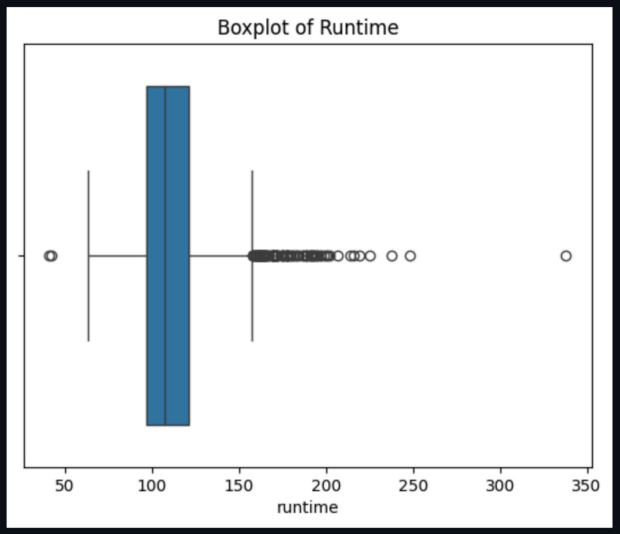
2. Runtime

From the below diagram, we could visualize that the **runtime** feature has a few outliers (movies much shorter or longer than the typical runtime).

```
sns.boxplot(x=movies['runtime'])
plt.title('Boxplot of Runtime')
```

✓ 0.1s

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

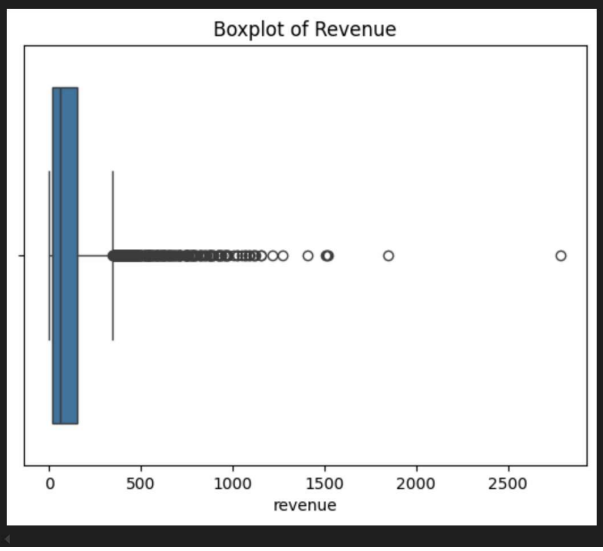


3. Revenue

From the below diagram, we could visualize that the **revenue** feature has significant outliers (very high-grossing movies).

```
sns.boxplot(x=movies['revenue'])
plt.title('Boxplot of Revenue')
```

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



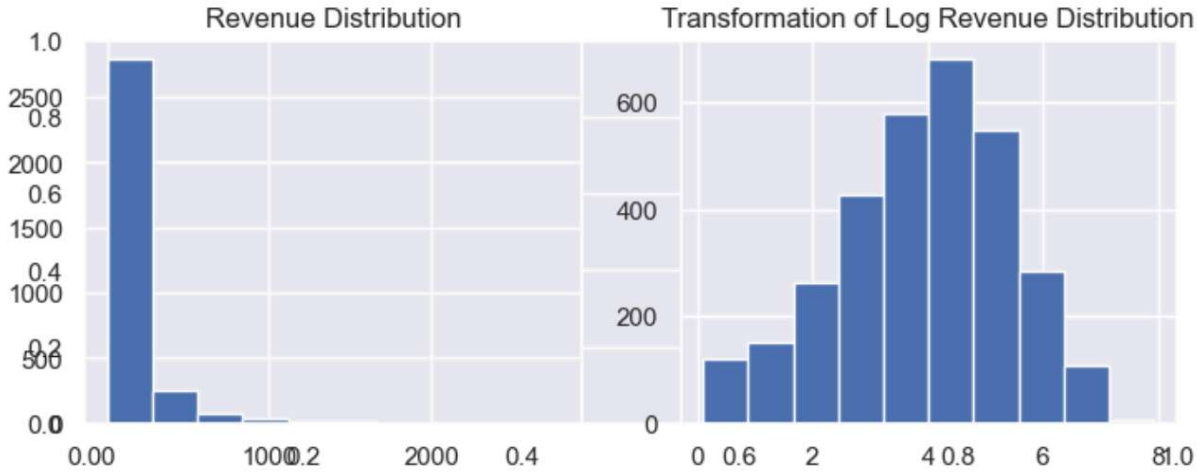
Milestone 3: Exploratory Data Analysis

Activity 1: Descriptive Statistics

1. **Summary statistics**
   - Use df.describe() to obtain count, mean, std, min, 25%, 50%, 75%, and max for all numeric features (budget, runtime, vote_average, revenue).
   - Inspect df.info() to confirm data types and non-null counts.
2. **Feature distributions**
   - Check skewness and kurtosis for key numeric features via df.skew() and df.kurtosis().
   - Identify any features requiring transformation (e.g., log-transforming revenue or budget).

Activity 2: Univariate Analysis

1. **Histograms & KDE plots**
   - Plot histograms for budget, runtime, popularity, and revenue using Matplotlib:
   - Overlay kernel density estimates (KDE) for smoother view of distribution tails.



- **Boxplots for outlier check**
   - Re-plot boxplots for budget, runtime, and revenue to visually confirm outlier thresholds.
   - Use .quantile() to compute IQR and flag extreme values.

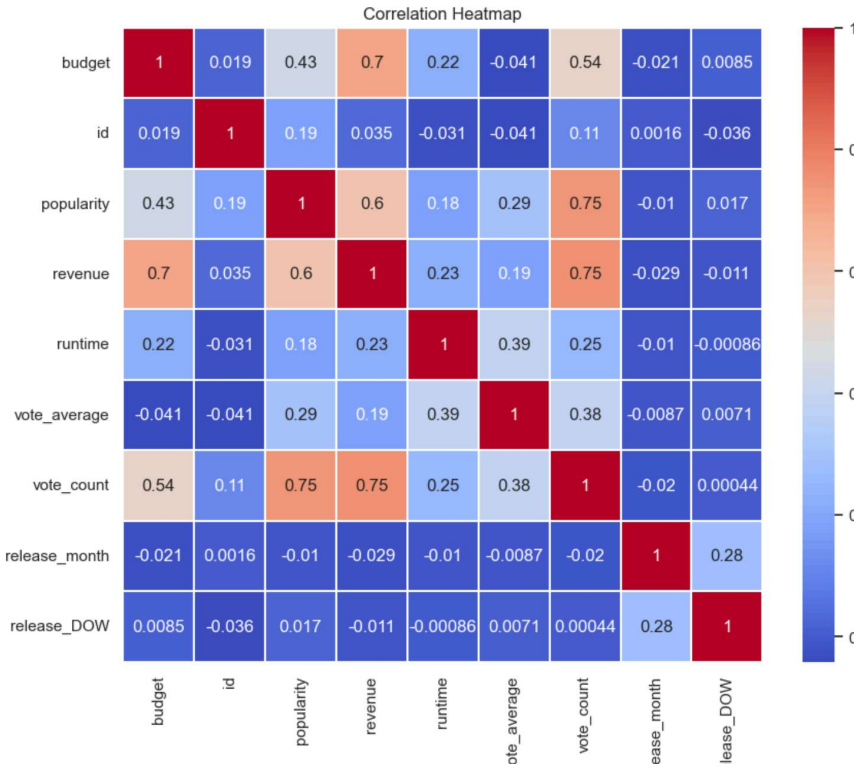
Activity 3: Categorical Feature Analysis

1. **Genre frequency**
   - Explode the genres column (list of genre names) and count-plot most common genres.
   - Identify top 5 genres by movie count.
2. **Release month effect**
   - Extract month from release_date (df['release_month'] = pd.to_datetime(df['release_date']).dt.month).
   - Bar plot average revenue per month to spot seasonal trends.
3. **Language and production country**
   - Count-plot of top spoken original languages.
   - Bar plot top 10 production countries by average revenue.

Activity 4: Correlation & Heatmap

1. **Compute correlation matrix**

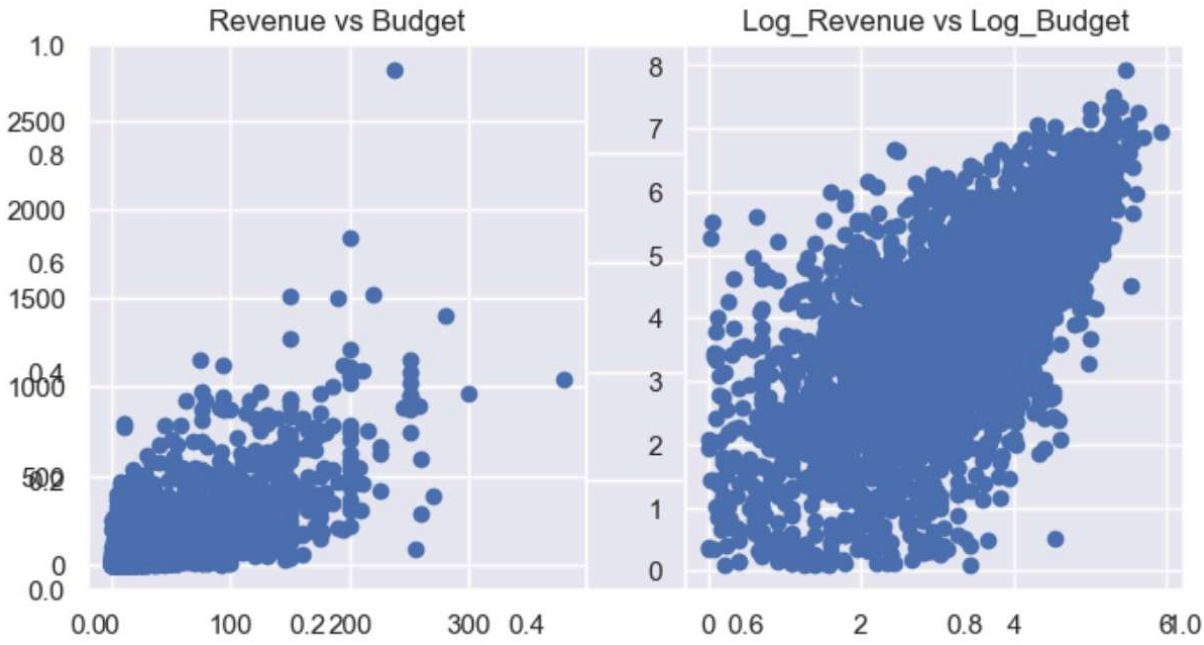
- **Heatmap visualization**
  - Use Matplotlib's imshow + colorbar to plot corr, annotating correlation coefficients.
  - Highlight strong positive (e.g., budget vs revenue) and negative relationships.



- **Pairwise scatterplots**
  - For the three strongest correlations, create scatterplots (e.g., budget vs revenue, popularity vs revenue, vote_count vs revenue) with trend lines using np.polyfit.

Activity 5: Feature Relationships & Trends

1. **Budget vs. Revenue Scatter**
   - Scatter plot with log scales on both axes to accommodate wide ranges.
   - Fit a simple regression line to visualize trend.



2. **Runtime vs. Revenue**
   - Scatter with binned boxplots: divide runtime into intervals (e.g., <90, 90–120, >120 minutes) and boxplot revenue per bin.
3. **Cast & Director Impact**
   - Extract top 10 most prolific directors and plot their average revenue.
   - Compute each actor's average movie revenue for top 20 billed cast members.

Activity 6: Textual & Sentiment Features

1. **Overview word count**
   - Create a new column overview_length = df['overview'].str.split().apply(len).
   - Plot distribution of overview lengths and correlate with revenue.
2. **Title sentiment**
   - Use a simple sentiment analyzer (e.g., VADER) to assign sentiment scores to title or overview.
   - Scatterplot sentiment vs. revenue to explore impact of “positive” wording.
3. **Word cloud of movie overviews**
   - Generate a word cloud from the concatenated overviews of the top 100 highest-grossing movies to surface prevalent themes.

Milestone 4: Model Building

Activity 1: Training the model using multiple regression algorithms

Now that our data is cleaned and preprocessed, we proceed to build predictive models. We train the data on different regression algorithms and compare their performance. For this project, we applied three regression algorithms. The best-performing model is selected based on its evaluation metrics.

Activity 1.1: Linear Regression Model

First, the Linear Regression model is imported from the sklearn.linear_model library. The LinearRegression() algorithm is initialized, and the training data is passed to the model using the .fit() function. Predictions on the test data are generated with the .predict() function. The performance of the model is evaluated using Mean Absolute Error (MAE), Root Mean Square Error (RMSE), and R^2 score. Linear Regression serves as the baseline for comparison.

```
lr_model = LinearRegression()
lr_model, lr_mae, lr_rmse, lr_r2 = evaluate_model(
    "Linear Regression",
    lr_model,
    x_train,
    x_test,
    y_train,
    y_test
)
```

✓ 0.6s

```
=== Linear Regression ===
MAE: $65.83 million
RMSE: $132.79 million
R² Score: 0.6677
```

Activity 1.2: Random Forest Regressor Model

The Random Forest Regressor is imported from the sklearn.ensemble library. The RandomForestRegressor() algorithm is initialized with 100 estimators and a fixed random seed for reproducibility. The training data is fit to the model using the .fit() function, and predictions are obtained using the .predict() function. The model's performance is measured with MAE, RMSE, and R^2 score to evaluate its improvement over the baseline Linear Regression model.

```
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)
rf_model, rf_mae, rf_rmse, rf_r2 = evaluate_model(
    "Random Forest Regressor",
    rf_model,
    x_train,
    x_test,
    y_train,
    y_test
)
```

✓ 2.0s

```
=== Random Forest Regressor ===
MAE: $62.17 million
RMSE: $126.85 million
R² Score: 0.6967
```

Activity 1.3: XGBoost Regressor Model

The XGBoost Regressor is imported from the xgboost library. The XGBRegressor() algorithm is initialized with 100 estimators and a fixed random seed. The training data is passed to the model with the .fit() function. Test data predictions are generated using the .predict() function. The model is evaluated using MAE, RMSE, and R^2 score. XGBoost provides a robust approach by optimizing gradient loss and improving predictive performance.

```
xgb_model = XGBRegressor(n_estimators=100, random_state=42)
xgb_model, xgb_mae, xgb_rmse, xgb_r2 = evaluate_model(
    "XGBoost Regressor",
    xgb_model,
    x_train,
    x_test,
    y_train,
    y_test
)
```

✓ 0.3s

```
=== XGBoost Regressor ===
MAE: $63.18 million
RMSE: $129.01 million
R² Score: 0.6863
```

Activity 2: Testing the model

Here, we have tested the trained regression model on sample input data. The .predict() function is used to predict the revenue for a new movie example.

```
try:
    predicted_revenue = predict_revenue(
        budget=50,
        genres_name="Action",
        popularity=20.0,
        runtime=120,
        vote_average=7.0,
        vote_count=1000,
        director_name="Unknown",
        release_month=7,
        release_DOW=5,
        has_homepage=1
    )
    print(f"\nExample Prediction:")
    print(f"Predicted Revenue: ${predicted_revenue:.2f} million")
except Exception as e:
    print(f"Prediction error: {e}")
    print("This might be due to unknown categories in the encoder")
```

✓ 0.0s

```
Example Prediction:
Predicted Revenue: $129.33 million
```

Milestone 5: Performance Testing & Hyperparameter Tuning

Activity 1: Testing model with multiple evaluation metrics

Multiple evaluation metrics means evaluating the model's performance on a test set using different performance measures. This provides a more comprehensive understanding of the model's strengths and weaknesses.

In our regression project, the following evaluation metrics were used:

- **Mean Absolute Error (MAE) –** measures average error magnitude
- **Root Mean Squared Error (RMSE) –** penalizes large errors
- **R² Score –** proportion of variance explained by the model

Activity 1.1: Compare the models

```
lr_model = LinearRegression()
lr_model, lr_mae, lr_rmse, lr_r2 = evaluate_model(
    "Linear Regression",
    lr_model,
    x_train,
    x_test,
    y_train,
    y_test
)
```

✓ 0.6s

```
=== Linear Regression ===
MAE: $65.83 million
RMSE: $132.79 million
R² Score: 0.6677
```

```
xgb_model = XGBRegressor(n_estimators=100, random_state=42)
xgb_model, xgb_mae, xgb_rmse, xgb_r2 = evaluate_model(
    "XGBoost Regressor",
    xgb_model,
    x_train,
    x_test,
    y_train,
    y_test
)
```

✓ 0.3s

```
=== XGBoost Regressor ===
MAE: $63.18 million
RMSE: $129.01 million
R² Score: 0.6863
```

```
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)
rf_model, rf_mae, rf_rmse, rf_r2 = evaluate_model(
    "Random Forest Regressor",
    rf_model,
    x_train,
    x_test,
    y_train,
    y_test
)
```

✓ 2.0s

```
=== Random Forest Regressor ===
MAE: $62.17 million
RMSE: $126.85 million
R² Score: 0.6967
```

- **Random Forest** achieved the best R² and lowest MAE.
- **XGBoost** performed comparably.
- **Linear Regression** served as a baseline.

Activity 2: Comparing model accuracy before & after applying hyperparameter tuning

Hyperparameter tuning involves searching for the best set of parameters to improve performance. Although optional, this process can significantly enhance model quality.

To evaluate consistency and avoid overfitting, cross_val_score() from Scikit-learn was used to compute **cross-validated R² scores**.

```
from sklearn.model_selection import cross_val_score

cv_scores = cross_val_score(
    RandomForestRegressor(n_estimators=200, random_state=42),
    x_train,
    y_train,
    cv=5,
    scoring='r2'
)

print(cv_scores)
print("Average R² Score:", np.mean(cv_scores))
```

✓ 10.4s

```
[0.69791379 0.62650419 0.73014302 0.73214803 0.7312428 ]
Average R² Score: 0.7035903688985853
```

Generate + Code

Milestone 6: Model Deployment

Activity 1: Save the Best Model

Saving the best regression model after comparing its performance using different evaluation metrics (such as MAE, RMSE, and R^2 score) helps ensure that the most accurate model is retained for predictions. This avoids retraining every time and enables easy reuse of the model in the future.In this project, the model is saved using the pickle library. The trained model, scaler, and label encoders are all saved as .pkl files so they can be reloaded during inference.

```
model = pickle.load(open('model_movies.pkl', 'rb'))
scaler = pickle.load(open('scaler_movies.pkl', 'rb'))
```

Activity 2: Integrate with Web Framework

In this section, we built a web application that integrates the trained regression model.
A user interface is provided where users can input values for prediction.
The entered values are processed and passed to the saved model, and the predicted revenue is displayed on the web page.

This section included the following tasks:

- Building HTML Pages
- Building server-side script
- Running the web application

Activity 2.1: Building HTML Pages

For this project, we created the following HTML files:

demo2.html: The main form for entering movie features

resultnew.html: The page displaying the predicted revenue

These files were saved in the **templates** folder in the Flask project directory.

Activity 2.2: Build Python Code

- **Import Libraries:**
  The necessary libraries (flask, pickle, numpy) were imported.
  The trained model and scaler were loaded using pickle.load().
- **Flask App Setup:**
  An object of the Flask class is created as the WSGI application.
  app = Flask(__name__)

```
from flask import Flask, render_template, request
import numpy as np
import pickle
import os

app = Flask(__name__)

model = pickle.load(open('model_movies.pkl', 'rb'))
scaler = pickle.load(open('scaler_movies.pkl', 'rb'))
```

- **Render HTML Page:**
  We created routes to render the HTML pages.
  The '/' URL endpoint renders the Demo2.html page containing the input form.

```
@app.route('/')
def home():
    return render_template('Demo2.html')
```

- **Retrieve Values from UI:**
  The /predict route retrieves input values using the POST method.
  Inputs are collected, processed, scaled, and passed to model.predict().
  The prediction result is displayed in resultnew.html.

```python
@app.route('/predict', methods=['POST'])
def predict():
    try:
        form_values = request.form
        genre_str = form_values['genres'].strip()
        director_str = form_values['director'].strip()
        genre_code = genre_map.get(genre_str, 0)
        director_code = director_map.get(director_str, 0)
        budget = float(form_values['budget'])
        popularity = float(form_values['popularity'])
        runtime = float(form_values['runtime'])
        vote_average = float(form_values['vote_average'])
        vote_count = float(form_values['vote_count'])
        release_month = int(form_values['release_month'])
        release_dow = int(form_values['release_DOW'])
        final_input = [[
            budget, genre_code, popularity, runtime,
            vote_average, vote_count, director_code,
            release_month, release_dow
        ]]
        print("Raw input:", final_input)
        scaled_input = scaler.transform(final_input)
        print("Scaled input:", scaled_input)
        log_prediction = model.predict(scaled_input)
        print("Log scale prediction:", log_prediction[0])
        try:
            revenue = log_prediction[0]
            if np.isinf(revenue) or revenue > 1e12:
                revenue = "Prediction too large or unrealistic input"
            else:
                revenue = f"${revenue:,.2f}"
        except Exception as e:
            revenue = f"Error in prediction: {str(e)}"
        return render_template('resultnew.html', prediction=revenue)
    except Exception as e:
        return f"An error occurred: {str(e)}"
```

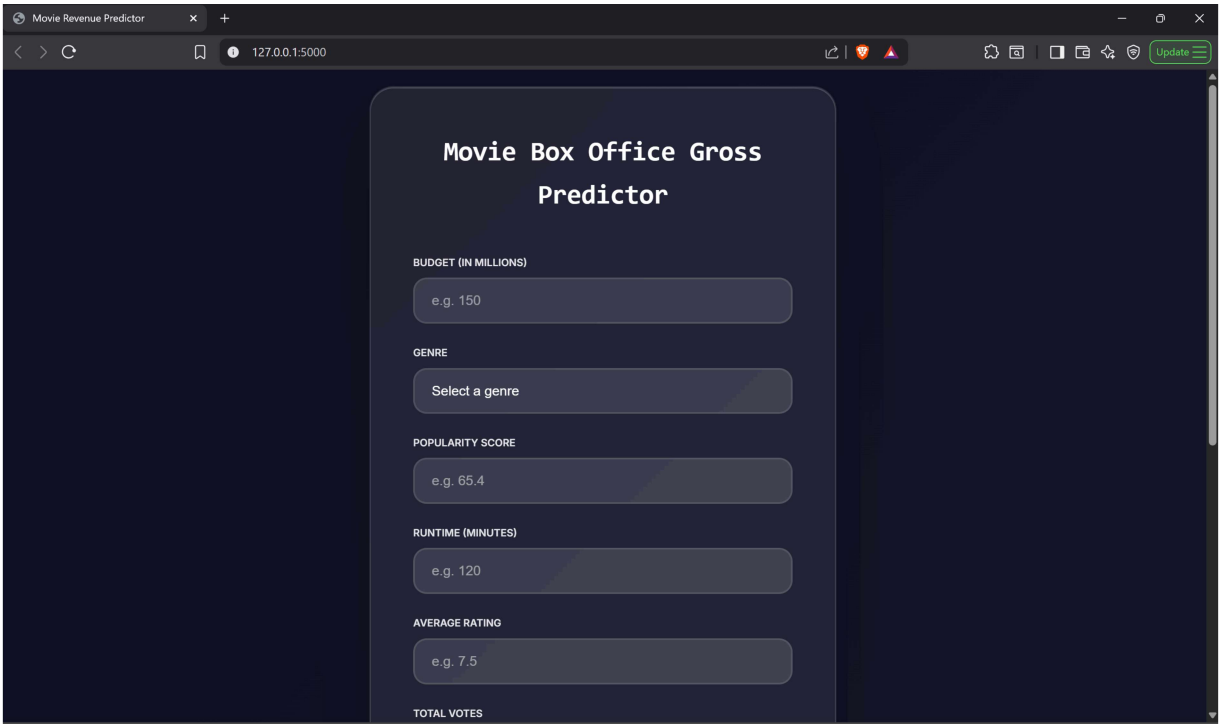
Activity 2.3: Run the Web Application

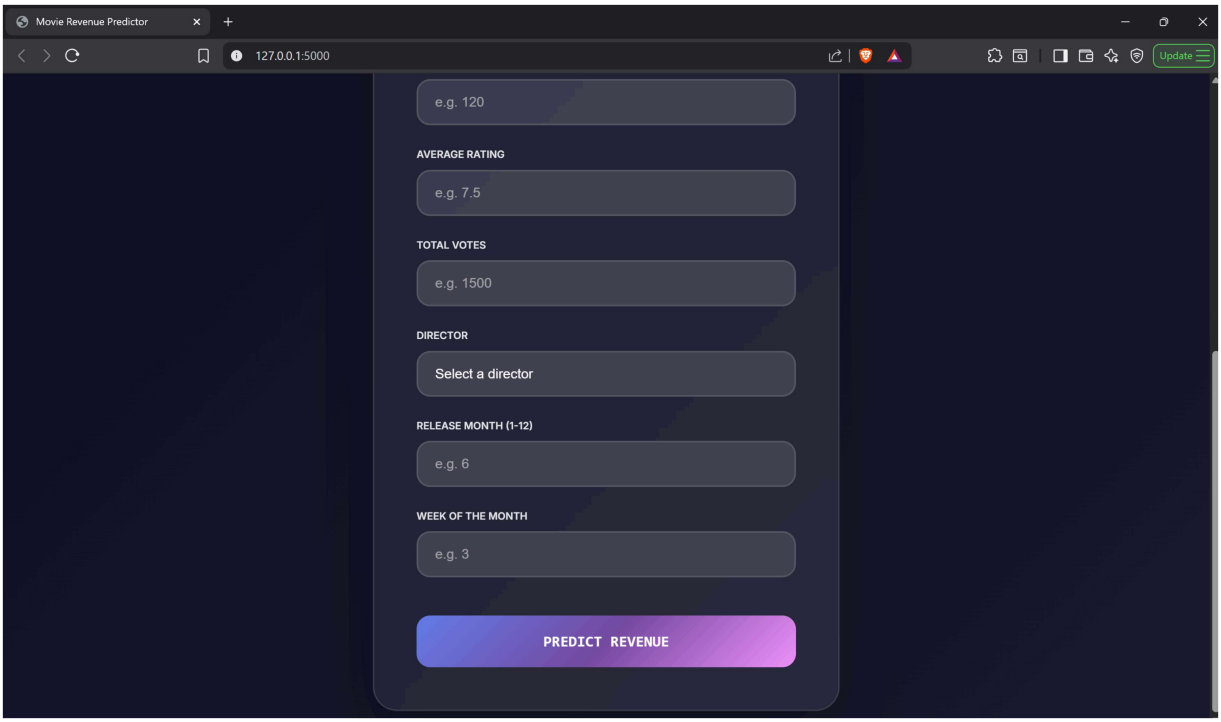
To launch the web app:

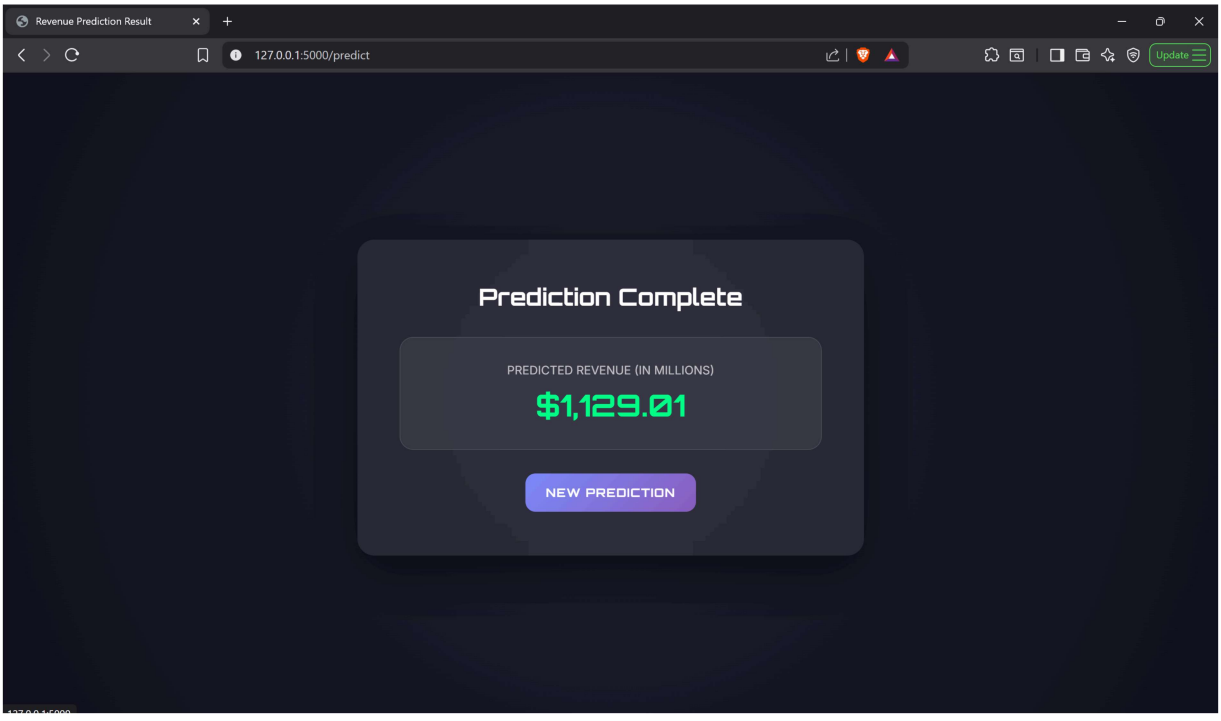
1. Open terminal
2. Navigate to app folder containing app.py.
3. Run the server using python app.py
4. Open the browser and visit: http://127.0.0.1:5000
5. Enter inputs in the form and click **Predict Revenue**.
6. View the predicted revenue displayed on the result page.

Sample Output:

```
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server
 * Running on http://127.0.0.1:5000
```




Movie Revenue Predictor
127.0.0.1:5000
Update
e.g. 120
AVERAGE RATING
e.g. 7.5
TOTAL VOTES
e.g. 1500
DIRECTOR
Select a director
RELEASE MONTH (1-12)
e.g. 6
WEEK OF THE MONTH
e.g. 3
PREDICT REVENUE


Revenue Prediction Result
127.0.0.1:5000/predict
Update
Prediction Complete
PREDICTED REVENUE (IN MILLIONS)
$1,129.01
NEW PREDICTION

Milestone 7- Project Demonstration & Documentation :

Below mentioned deliverables to be submitted along with other deliverables

Activity 1:- Record explanation Video for project end to end solution

https://drive.google.com/file/d/1mS6tHW0RRmu1VY2eU6i8xmCZUfHtPKYI/view?usp=sharing

Activity 2:- Project Documentation-Step by step project development procedure

https://github.com/nvshanmukh/movie-box-office-predictor

Activity 2:- Project Reports
https://drive.google.com/drive/folders/1rlNF271ccEylnJulKQMl4_R1Qp5hLVDps-?usp=drive_link