Final Project Report

Movie Box Office Gross Prediction using Machine Learning

1.Introduction

1.1 Project overviews

Predicting movie box office gross is a crucial aspect for stakeholders in the film industry, influencing key decisions in production, marketing, and distribution. Traditional methods, which often rely on historical performance or intuition, may not fully capture the complexities and subtleties that contribute to a film's financial success. This project applies machine learning (ML) to improve the accuracy of box office gross predictions by leveraging a comprehensive dataset of movie-related variables, including genre, cast, budget, release date, marketing strategies, and more. By uncovering patterns and correlations that might be missed by conventional approaches, the ML model offers a data-driven solution that can provide more reliable revenue estimates, helping film studios optimize their strategies and increase profitability.

1.2 Objectives

The primary objective of this project is to develop an ML model capable of accurately predicting movie box office gross using historical data and relevant movie attributes. Key goals include improving the prediction accuracy compared to traditional methods, identifying significant patterns and correlations within the movie data, and creating a tool that can assist movie production companies in decision-making. This tool can help optimize budget allocation, marketing strategies, and release timing to maximize revenue potential. Ultimately, the project aims to demonstrate the effectiveness of ML techniques in capturing the complexities of the entertainment industry, providing valuable insights for stakeholders to better navigate the competitive movie market.

2. Project Initialization and Planning Phase

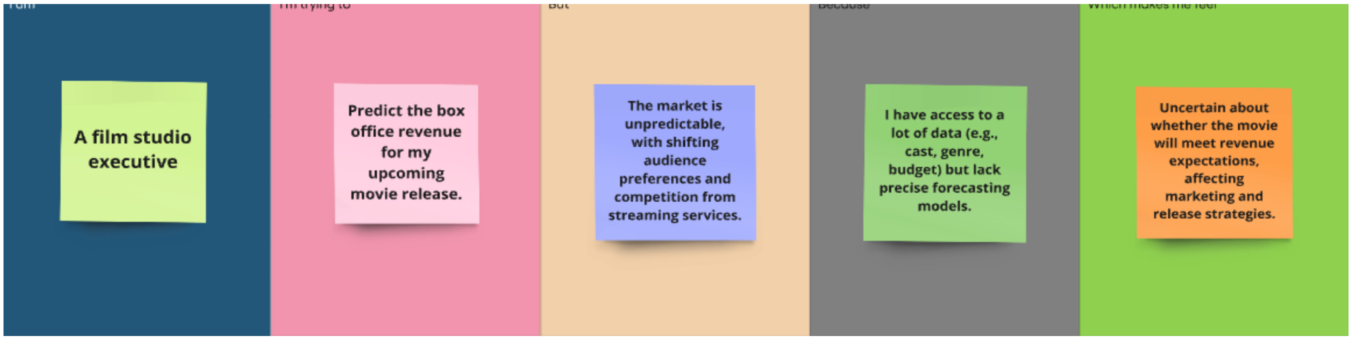
2.1 Define Problem Statements (Customer Problem Statement Template):

Movie Box Office Gross Prediction predicts potential revenue for films using data analysis and machine learning. It analyzes critical factors like genre, cast, release date, and marketing budget to inform marketing strategies and investment decisions.

For a major studio, the process of predicting box office revenue is essential but can also be challenging. Factors such as shifts in audience demographics, unpredictable trends, and evolving distribution channels can impact accuracy, making the task more complex. These challenges can lead to suboptimal marketing strategies and release date planning, potentially affecting profitability.

By understanding the evolving nature of the movie industry and leveraging more advanced data-driven solutions, studios can improve accuracy in forecasting. Implementing these strategies ensures a more efficient, data-informed process that increases customer trust, optimizes investment decisions, and ultimately maximizes profitability for each film release.

Problem Statement (PS)	I am (Customer)	I'm trying to	But	Because	Which makes me feel
PS-1	A film studio executive.	Predict the box office revenue for my upcoming movie release.	The market is unpredictable, with shifting audience preferences and competition from streaming services.	I have access to a lot of data (e.g., cast, genre, budget) but lack precise forecasting models.	Uncertain about whether the movie will meet revenue expectations, affecting marketing and release strategies.



A film studio executive

Predict the box office revenue for my upcoming movie release.

The market is unpredictable, with shifting audience preferences and competition from streaming services.

I have access to a lot of data (e.g., cast, genre, budget) but lack precise forecasting models.

Uncertain about whether the movie will meet revenue expectations, affecting marketing and release strategies.

2.2 Project Proposal (Proposed Solution) template

The proposal aims to transform the process of predicting movie box office revenue by leveraging advanced machine learning techniques, significantly boosting accuracy and decision-making for film studios. It addresses inefficiencies in the current prediction methods, offering more reliable forecasts that will help optimize marketing strategies, release timing, and investment decisions.
Key features include a machine learning-based revenue prediction model that analyzes various factors (e.g., genre, cast, budget, and release date) and provides real-time revenue forecasts.

Project Overview	
Objective	The primary objective is to revolutionize box office revenue prediction by implementing advanced machine learning techniques, ensuring more accurate and data-driven forecasts for upcoming movie releases.
Scope	The project comprehensively assesses and enhances the revenue prediction process, incorporating machine learning to deliver a more robust and efficient system that assists studios in strategic planning and marketing optimization.
Problem Statement	
Description	Addressing the current inaccuracies in predicting movie box office revenue, which can adversely affect marketing strategies, release date selection, and overall profitability.
Impact	Solving these issues will lead to improved prediction accuracy, optimized marketing strategies, better release timing, and increased profitability. This will contribute to more data-driven decision-making and overall success in the highly competitive film industry.
Proposed Solution	
Approach	Employing machine learning techniques to analyze historical movie data (e.g., genre, cast, budget) and predict box office revenue, creating a dynamic, adaptable, and accurate forecasting model.
Key Features	- Implementation of a machine learning-based revenue prediction model. - Real-time forecasting to adjust marketing strategies and release plans. - Continuous learning to adapt to evolving audience preferences and market trends.

Resource Requirements

Resource Type	Description	Specification/Allocation
Hardware		
Computing Resources	CPU/GPU specifications, number of cores	T4 GPU
Memory	RAM specifications	16 GB
Storage	Disk space for data, models, and logs	2 TB SSD
Software		
Frameworks	Python frameworks	Flask
Libraries	Additional libraries	scikit-learn, pandas, numpy, matplotlib, seaborn
Development Environment	IDE	Jupyter Notebook, pycharm
Data		
Data	Source, size, format	Kaggle dataset, 4803, csv

2.3 Initial Project Planning Template

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Priority	Team Members	Sprint Start Date	Sprint End Date (Planned)
Sprint-1	Data Collection and Preparation	USN-1	Collect and clean movies data.	High	Bramham and Bhanu	23/09/24	26/09/24
Sprint-1	Data Collection and Preparation	USN-2	Explore dataset for key features.	High	Bramham	23/09/24	26/09/24
Sprint-2	Data Collection and Preparation	USN-3	Handle missing values and outliers.	Medium	Sai kiran and Pavani	23/09/24	26/09/24
Sprint-2	Model Training	USN-4	Train machine learning models to predict revenue.	High	Bramham, Bhanu and Sai kiran	27/09/24	30/09/24
Sprint-3	Model Evaluation and Tuning	USN-5	Evaluate models and pick the best one.	High	Sai kiran, Bhanu and Pavani	27/09/24	30/09/24
Sprint-3	Model Evaluation and Tuning	USN-6	Tune model hyperparameters.	Medium	Bhanu and Pavani	27/09/24	30/09/24
Sprint-4	Insights and Reporting	USN-7	Generate insights and visualizations for users.	Medium	Bramham	27/09/24	30/09/24
Sprint-4	Documentation	USN-8	Document the project process and findings.	Medium	Bramham and Sai kiran	27/09/24	30/09/24
Sprint-5	Application Building	USN-9	Design the app's user interface.	High	Bramham and Bhanu	01/10/24	05/10/24
Sprint-5	Application Building	USN-10	Implement the backend to handle data and predictions.	High	Bramham, Sai Kiranand Pavani	01/10/24	05/10/24

			Integrate the ML model with the app for real-time predictions.				
Sprint-6	Application Building	USN-11	Test the app for functionality and user-friendliness and deploy.	High	Sai kiran and Pavani	06/10/24	10/10/24

3. Data Collection and Preprocessing Phase

3.1 Data Collection Plan & Raw Data Sources Identification Template

Elevate your data strategy with the Data Collection plan and the Raw Data Sources report, ensuring meticulous data curation and integrity for informed decision-making in every analysis and decision-making endeavor.

Data Collection Plan Template

Section	Description
Project Overview	The machine learning project aims to predict box office revenue based on various features of movies, including budget, genres, cast, release dates, and more. Using a dataset with 23 columns such as budget, cast, revenue, and genres, the objective is to build a model that accurately forecasts a movie's box office gross, helping producers, distributors, and stakeholders in the film industry make informed financial decisions.
Data Collection Plan	● Search for datasets related to movie box office revenues, movie details (such as cast, crew, budget, and release dates), and production companies. ● Focus on datasets that include a wide variety of movies across different genres, release years, and production scales. ● Ensure that datasets have variables that are significant predictors of box office revenue. ● Collect additional datasets from sources like IMDb, TMDB, and Box Office Mojo.
Raw Data Sources Identified	The raw data sources for this project will include datasets obtained from IMDb, The Movie Database (TMDB), and Box Office Mojo. These platforms provide comprehensive details about movie productions, including key features like budget, revenue, cast, and genres. Additional datasets may also be collected from Kaggle or other open data repositories to enrich the prediction model.

Raw Data Sources Template

Source Name	Description	Location/URL	Format	Size	Access Permissions
Kaggle Dataset	The dataset comprises movie details such as budget, genres, cast, release date, and box office revenue. It also includes additional movie features like production companies and runtime, which may influence revenue predictions.	https://www.kaggle.com/datasets/tmdb/tmdb-movie-metadata	CSV	45.74 MB	Public

Data Collection and Preprocessing Phase

3.2 Data Quality Report Template

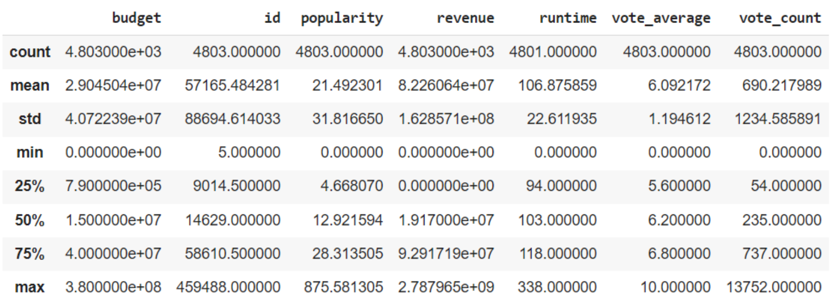
The Data Quality Report will summarize data quality issues from the selected source, including severity levels and resolution plans. It will aid in systematically identifying and rectifying data discrepancies.

Data Source	Data Quality Issue	Severity	Resolution Plan
Kaggle Dataset (Movie Data)	Missing values in the 'homepage', 'overview', 'release_date', 'runtime', and 'tagline' columns.	Moderate	Use appropriate imputation techniques (e.g., fill missing dates, mean/median imputation for numeric data).
Kaggle Dataset (Movie Data)	Categorical data in the 'genres', 'production_companies', 'production_countries', 'spoken_languages', 'cast', and 'crew' columns contain complex categorical data.	Moderate	Use one-hot encoding or label encoding for categorical columns. Consider handling multi-label columns (e.g., genres) using specialized techniques like multi-label binarization.
Kaggle Dataset (Movie Data)	Inconsistent formats in 'release_date' (e.g., different date formats)	Moderate	Standardize the date format and convert 'release_date' to datetime data type.
Kaggle Dataset (Movie Data)	Duplicate entries (possible duplicate movies based on 'title' and 'id')	Moderate	Check for duplicate records based on the 'title' and 'id' columns, and remove duplicates.

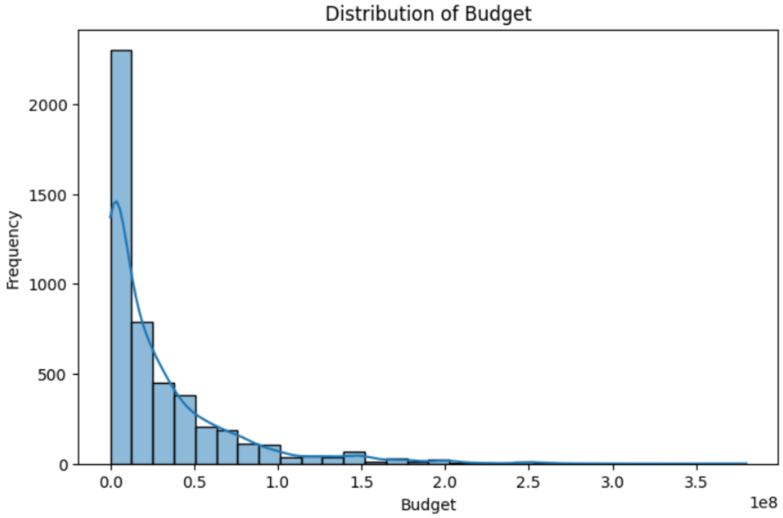
Data Collection and Preprocessing Phase

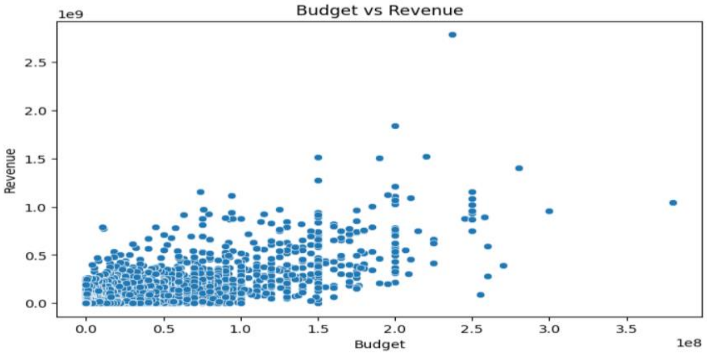
3.3 Data Exploration and Preprocessing Template

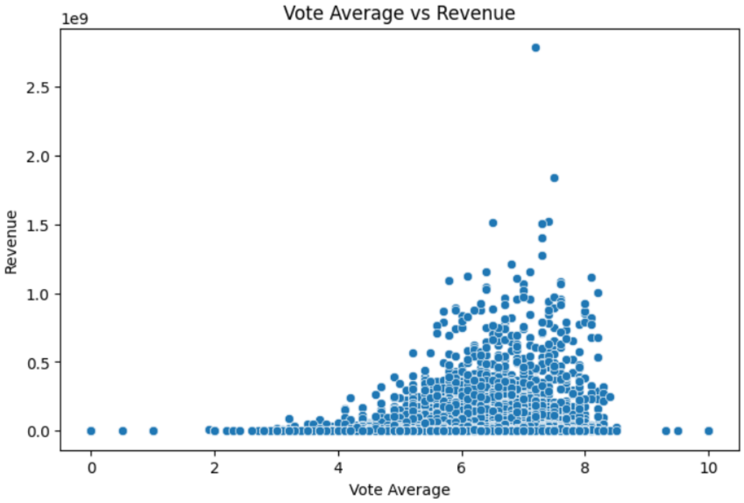
Dataset variables will be statistically analyzed to identify patterns and outliers, with Python employed for preprocessing tasks like normalization and feature engineering. Data cleaning will address missing values and outliers, ensuring quality for subsequent analysis and modeling, and forming a strong foundation for insights and predictions.

Section	**Description**
Data Overview	Dimension: 4083rows × 23columns Descriptive statistics: 

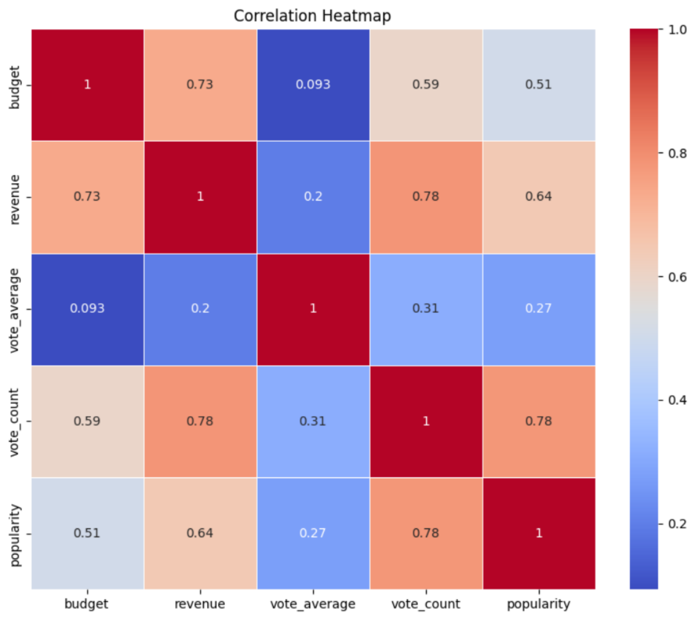
	budget	id	popularity	revenue	runtime	vote_average	vote_count
count	4.803000e+03	4803.000000	4803.000000	4.803000e+03	4801.000000	4803.000000	4803.000000
mean	2.904504e+07	57165.484281	21.492301	8.226064e+07	106.875859	6.092172	690.217989
std	4.072239e+07	88694.614033	31.816650	1.628571e+08	22.611935	1.194612	1234.585891
min	0.000000e+00	5.000000	0.000000	0.000000e+00	0.000000	0.000000	0.000000
25%	7.900000e+05	9014.500000	4.668070	0.000000e+00	94.000000	5.600000	54.000000
50%	1.500000e+07	14629.000000	12.921594	1.917000e+07	103.000000	6.200000	235.000000
75%	4.000000e+07	58610.500000	28.313505	9.291719e+07	118.000000	6.800000	737.000000
max	3.800000e+08	459488.000000	875.581305	2.787965e+09	338.000000	10.000000	13752.000000

<table>
<tr>
<td>Univariate Analysis</td>
<td>


</td>
</tr>
<tr>
<td>Bivariate Analysis</td>
<td>


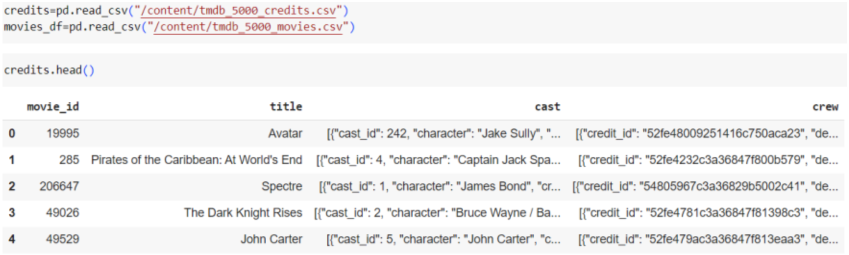


</td>
</tr>
</table>

Multivariate Analysis	
Outliers and Anomalies	-

Data Preprocessing Code Screenshots

Loading Data

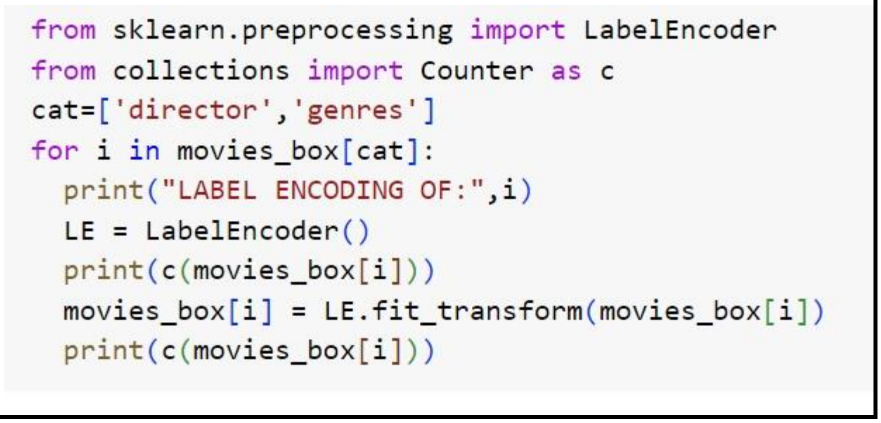


```
credits=pd.read_csv("/content/tmdb_5000_credits.csv")
movies_df=pd.read_csv("/content/tmdb_5000_movies.csv")
```

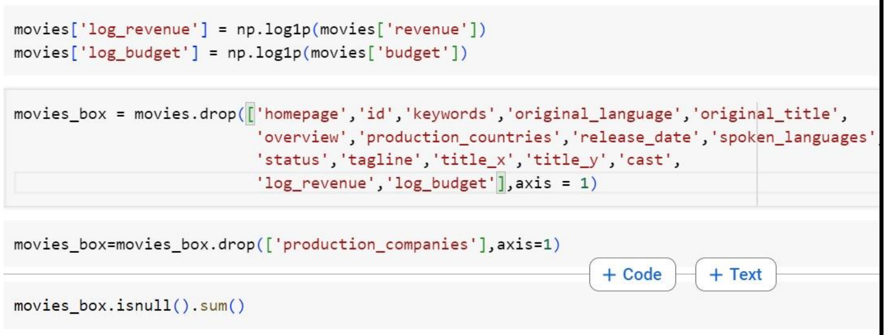
```
credits.head()
```

	movie_id	title	cast	crew
0	19995	Avatar	[{"cast_id": 242, "character": "Jake Sully", "...	[{"credit_id": "52fe48009251416c750aca23", "de...
1	285	Pirates of the Caribbean: At World's End	[{"cast_id": 4, "character": "Captain Jack Spa...	[{"credit_id": "52fe4232c3a36847f800b579", "de...
2	206647	Spectre	[{"cast_id": 1, "character": "James Bond", "cr...	[{"credit_id": "54805967c3a36829b5002c41", "de...
3	49026	The Dark Knight Rises	[{"cast_id": 2, "character": "Bruce Wayne / Ba...	[{"credit_id": "52fe4781c3a36847f81398c3", "de...
4	49529	John Carter	[{"cast_id": 5, "character": "John Carter", "c...	[{"credit_id": "52fe479ac3a36847f813eaa3", "de...

Handling Missing Data



```
from sklearn.preprocessing import LabelEncoder
from collections import Counter as c
cat=['director','genres']
for i in movies_box[cat]:
  print("LABEL ENCODING OF:",i)
  LE = LabelEncoder()
  print(c(movies_box[i]))
  movies_box[i] = LE.fit_transform(movies_box[i])
  print(c(movies_box[i]))
```

Data Transformation	
	`movies['log_revenue'] = np.log1p(movies['revenue'])`
	`movies['log_budget'] = np.log1p(movies['budget'])`
	`movies_box = movies.drop(['homepage','id','keywords','original_language','original_title',`
	`                          'overview','production_countries','release_date','spoken_languages'`
	`                          'status','tagline','title_x','title_y','cast',`
	`                          'log_revenue','log_budget'],axis = 1)`
	`movies_box=movies_box.drop(['production_companies'],axis=1)`
	+ Code + Text
	`movies_box.isnull().sum()`

Feature Engineering	Attached the codes in final submission.
Save Processed Data	-

4. Model Development Phase Template

4.1 Feature Selection Report Template

In the forthcoming update, each feature will be accompanied by a brief description. Users will indicate whether it's selected or not, providing reasoning for their decision. This process will streamline decision-making and enhance transparency in feature selection.

Feature	Description	Selected (Yes/No)	Reasoning
Movie_ID	Unique identifier for each movie	No	Not required for predicting box office revenue, as it serves only as an identifier.
Budget	The production budget of the movie	Yes	Critical factor influencing revenue; higher budgets often correlate with higher potential earnings.
Genres	Movie genres (e.g., action, drama, comedy)	Yes	Different genres can attract different audiences, impacting box office performance.
Cast	List of actors in the movie	Yes	Well-known actors often drive higher audience attendance, influencing box office revenue.

Director	The director of the movie	Yes	Renowned directors can attract larger audiences and potentially increase revenue.
Release Date	The date the movie was released	Yes	Timing of release (e.g., holiday seasons) can significantly affect box office success.
Production Company	The company that produced the movie	Yes	Large, well-known production companies may have better distribution networks, impacting revenue.
Runtime	The duration of the movie	Yes	The length of the movie can influence audience retention and showtimes, which may affect revenue.
Language	The primary language in which the movie was made	Yes	Language can determine the size of the target audience and accessibility in different regions.
Budget to Revenue Ratio	The ratio of budget to revenue achieved by the movie	No	Although relevant for analysis, it's not a predictor for the model, as it's derived from the target variable (revenue).
Revenue	The total box office revenue generated by the movie	Yes	The target variable for predictive modeling – essential for the project's goal.

Model Development Phase Template

4.2 Model Selection Report

In the forthcoming Model Selection Report, various models will be outlined, detailing their descriptions, hyperparameters, and performance metrics, including Accuracy or F1 Score. This comprehensive report will provide insights into the chosen models and their effectiveness.

Model Selection Report:

Model	Description	Hyperparameters	Performance Metrics (R^2, MSE, MAE)
Linear Regression	A simple linear model to predict box office revenue.	-	R^2: 0.7758, MSE: 8649.14, MAE: 54.377
Decision Tree Regressor	Non-linear model using decision trees for predictions.	-	R^2: 0.5318, MSE: 18060.04, MAE: 73.851
SVM Regression	Non-linear regression model using support vector machines.	-	R^2: 0.1528, MSE: 32681.37, MAE: 77.861
Random Forest Regressor	Ensemble model using multiple decision trees.	-	R^2: 0.7617, MSE: 9190.28, MAE: 50.225

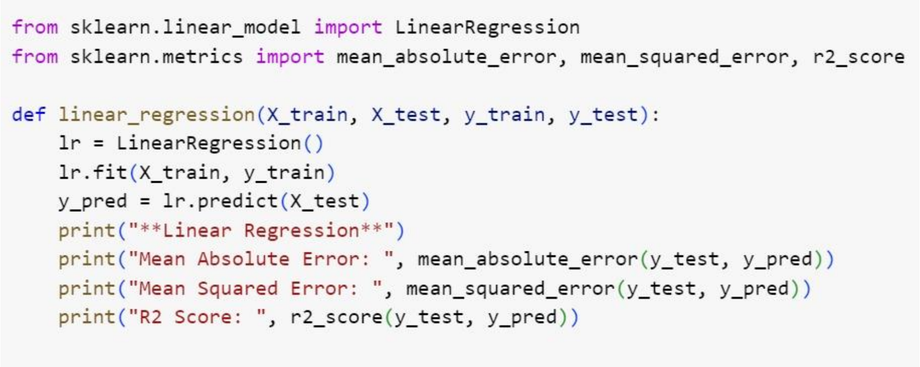
Model Development Phase Template

4.3 Initial Model Training Code, Model Validation and Evaluation Report

The initial model training code will be showcased in the future through a screenshot.

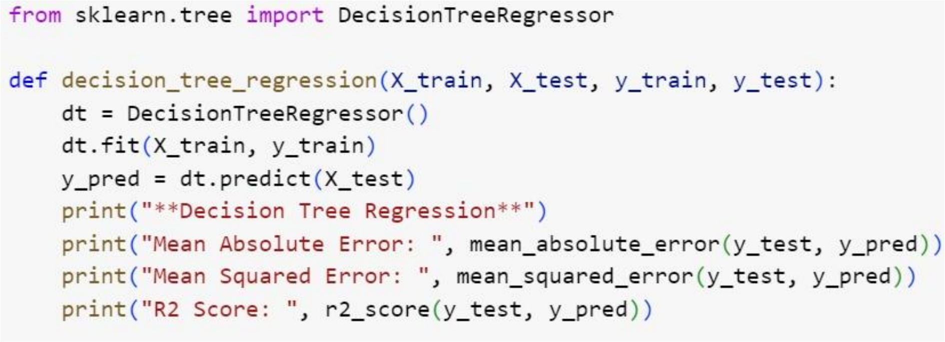
The model validation and evaluation report will include classification reports, accuracy, and confusion matrices for multiple models, presented through respective screenshots.

Initial Model Training Code:



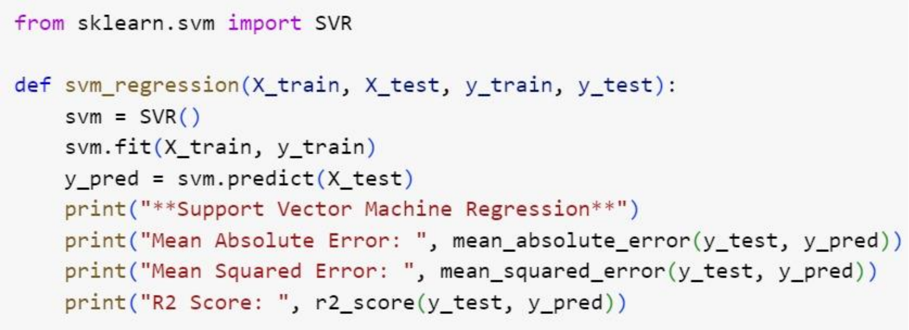
```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

def linear_regression(X_train, X_test, y_train, y_test):
    lr = LinearRegression()
    lr.fit(X_train, y_train)
    y_pred = lr.predict(X_test)
    print("**Linear Regression**")
    print("Mean Absolute Error: ", mean_absolute_error(y_test, y_pred))
    print("Mean Squared Error: ", mean_squared_error(y_test, y_pred))
    print("R2 Score: ", r2_score(y_test, y_pred))
```



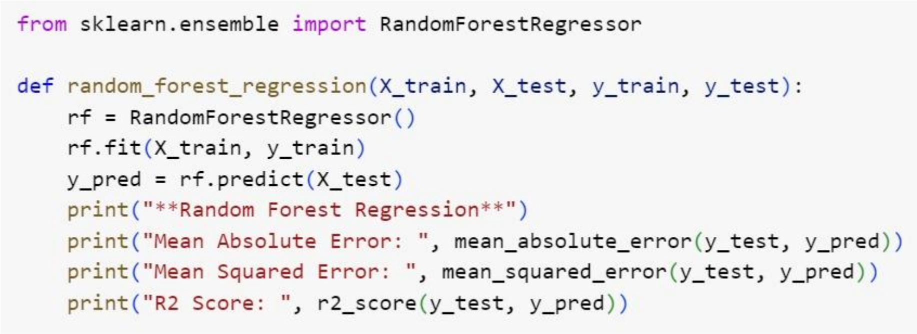
```
from sklearn.tree import DecisionTreeRegressor

def decision_tree_regression(X_train, X_test, y_train, y_test):
    dt = DecisionTreeRegressor()
    dt.fit(X_train, y_train)
    y_pred = dt.predict(X_test)
    print("**Decision Tree Regression**")
    print("Mean Absolute Error: ", mean_absolute_error(y_test, y_pred))
    print("Mean Squared Error: ", mean_squared_error(y_test, y_pred))
    print("R2 Score: ", r2_score(y_test, y_pred))
```



```
from sklearn.svm import SVR

def svm_regression(X_train, X_test, y_train, y_test):
    svm = SVR()
    svm.fit(X_train, y_train)
    y_pred = svm.predict(X_test)
    print("**Support Vector Machine Regression**")
    print("Mean Absolute Error: ", mean_absolute_error(y_test, y_pred))
    print("Mean Squared Error: ", mean_squared_error(y_test, y_pred))
    print("R2 Score: ", r2_score(y_test, y_pred))
```



```
from sklearn.ensemble import RandomForestRegressor

def random_forest_regression(X_train, X_test, y_train, y_test):
    rf = RandomForestRegressor()
    rf.fit(X_train, y_train)
    y_pred = rf.predict(X_test)
    print("**Random Forest Regression**")
    print("Mean Absolute Error: ", mean_absolute_error(y_test, y_pred))
    print("Mean Squared Error: ", mean_squared_error(y_test, y_pred))
    print("R2 Score: ", r2_score(y_test, y_pred))
```

Model Validation and Evaluation Report:

Model	**Evaluation Metric**	**□$^{\square}$, MSE, MAE**	**Confusion Matrix**
Linear Regression	R-squared, Mean Squared Error (MSE), Mean Absolute Error (MAE)	0.7758, 8649.14, 54.377	N/A
SVM Regression	R-squared, Mean Squared Error (MSE), Mean Absolute Error (MAE)	0.1528, 32681.37, 77.861	N/A
Decision Tree Regressor	R-squared, Mean Squared Error (MSE), Mean Absolute Error (MAE)	0.5318, 18060.04, 73.851	N/A
Random Forest Regressor	R-squared, Mean Squared Error (MSE), Mean Absolute Error (MAE)	0.7617, 9190.28, 50.225	N/A

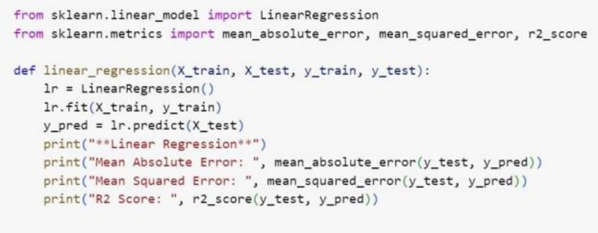
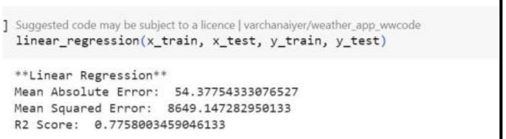
5. Model Optimization and Tuning Phase Template

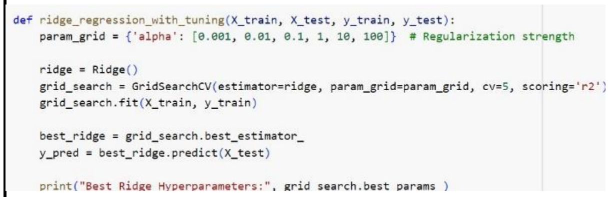
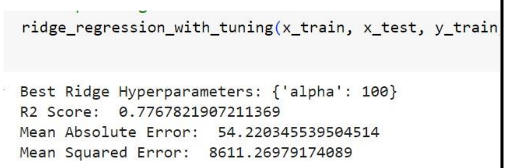
5.1 Model Optimization and Tuning Phase

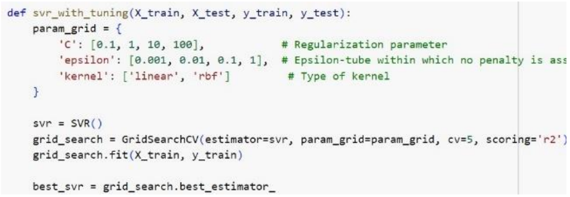
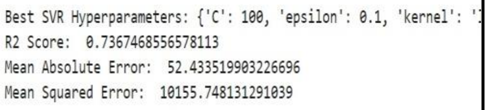
The Model Optimization and Tuning Phase involves refining machine learning models for peak performance. It includes optimized model code, fine-tuning hyperparameters, comparing performance metrics, and justifying the final model selection for enhanced predictive accuracy and efficiency.

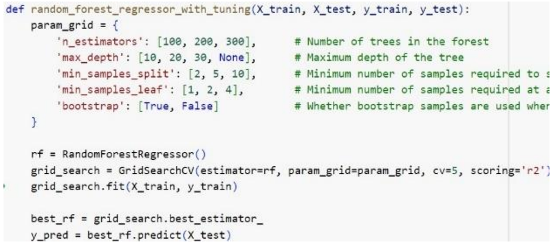
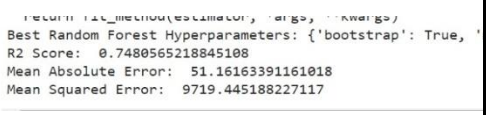
Hyperparameter Tuning Documentation (6 Marks):

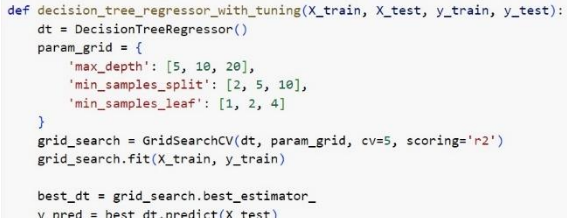
Model	Tuned Hyperparameters	Optimal Values

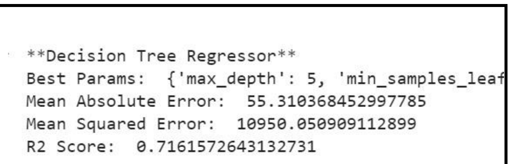
Linear Regression	
	from sklearn.linear_model import LinearRegression
	from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
	
	def linear_regression(X_train, X_test, y_train, y_test):
	    lr = LinearRegression()
	    lr.fit(X_train, y_train)
	    y_pred = lr.predict(X_test)
	    print("**Linear Regression**")
	    print("Mean Absolute Error: ", mean_absolute_error(y_test, y_pred))
	    print("Mean Squared Error: ", mean_squared_error(y_test, y_pred))
	    print("R2 Score: ", r2_score(y_test, y_pred))
		
		] Suggested code may be subject to a licence | varchanaiyer/weather_app_wwcode
		linear_regression(x_train, x_test, y_train, y_test)
		
		Linear Regression
		Mean Absolute Error: 54.37754333076527
		Mean Squared Error: 8649.147282950133
		R2 Score: 0.7758003459046133

Ridge regression	
	def ridge_regression_with_tuning(X_train, X_test, y_train, y_test):
	    param_grid = {'alpha': [0.001, 0.01, 0.1, 1, 10, 100]} # Regularization strength
	
	    ridge = Ridge()
	    grid_search = GridSearchCV(estimator=ridge, param_grid=param_grid, cv=5, scoring='r2')
	    grid_search.fit(X_train, y_train)
	
	    best_ridge = grid_search.best_estimator_
	    y_pred = best_ridge.predict(X_test)
	
	    print("Best Ridge Hyperparameters:", grid search.best params)
		
		ridge_regression_with_tuning(x_train, x_test, y_train
		
		Best Ridge Hyperparameters: {'alpha': 100}
		R2 Score: 0.7767821907211369
		Mean Absolute Error: 54.220345539504514
		Mean Squared Error: 8611.26979174089

SVM regression	
	def svr_with_tuning(X_train, X_test, y_train, y_test):
	    param_grid = {
	        'C': [0.1, 1, 10, 100], # Regularization parameter
	        'epsilon': [0.001, 0.01, 0.1, 1], # Epsilon-tube within which no penalty is ass
	        'kernel': ['linear', 'rbf'] # Type of kernel
	    }
	
	    svr = SVR()
	    grid_search = GridSearchCV(estimator=svr, param_grid=param_grid, cv=5, scoring='r2')
	    grid_search.fit(X_train, y_train)
	
	    best_svr = grid_search.best_estimator_
		
		Best SVR Hyperparameters: {'C': 100, 'epsilon': 0.1, 'kernel': '
		R2 Score: 0.7367468556578113
		Mean Absolute Error: 52.433519903226696
		Mean Squared Error: 10155.748131291039

Random Forest	
	def random_forest_regressor_with_tuning(X_train, X_test, y_train, y_test):
	    param_grid = {
	        'n_estimators': [100, 200, 300], # Number of trees in the forest
	        'max_depth': [10, 20, 30, None], # Maximum depth of the tree
	        'min_samples_split': [2, 5, 10], # Minimum number of samples required to s
	        'min_samples_leaf': [1, 2, 4], # Minimum number of samples required at a
	        'bootstrap': [True, False] # Whether bootstrap samples are used wher
	    }
	
	    rf = RandomForestRegressor()
	    grid_search = GridSearchCV(estimator=rf, param_grid=param_grid, cv=5, scoring='r2')
	    grid_search.fit(X_train, y_train)
	
	    best_rf = grid_search.best_estimator_
	    y_pred = best_rf.predict(X_test)
		
		return fit_method(estimator, *args, **kwargs)
		Best Random Forest Hyperparameters: {'bootstrap': True, '
		R2 Score: 0.7480565218845108
		Mean Absolute Error: 51.16163391161018
		Mean Squared Error: 9719.445188227117

<table>
<tr>
<td><b>Decision Tree</b></td>
<td>
<pre>
def decision_tree_regressor_with_tuning(X_train, X_test, y_train, y_test):
    dt = DecisionTreeRegressor()
    param_grid = {
        'max_depth': [5, 10, 20],
        'min_samples_split': [2, 5, 10],
        'min_samples_leaf': [1, 2, 4]
    }
    grid_search = GridSearchCV(dt, param_grid, cv=5, scoring='r2')
    grid_search.fit(X_train, y_train)

    best_dt = grid_search.best_estimator_
    y_pred = best_dt.predict(X_test)
</pre></td>
<td>
<pre>
Decision Tree Regressor
Best Params: {'max_depth': 5, 'min_samples_leaf
Mean Absolute Error: 55.310368452997785
Mean Squared Error: 10950.050909112899
R2 Score: 0.7161572643132731
</pre></td>
</tr>
</table>

5.2 Performance Metrics Comparison Report (2 Marks):

Model	Baseline Metric	Optimized Metric
Linear Regression	R2: 0.7758	R2: 0.7758
SVM Regression	R2: 0.1528	R2: 0.1528
Random Forest Regression	R2: 0.7617	R2: 0.7617
Decision Tree Regressor	R2: 0.5318	R2: 0.5318

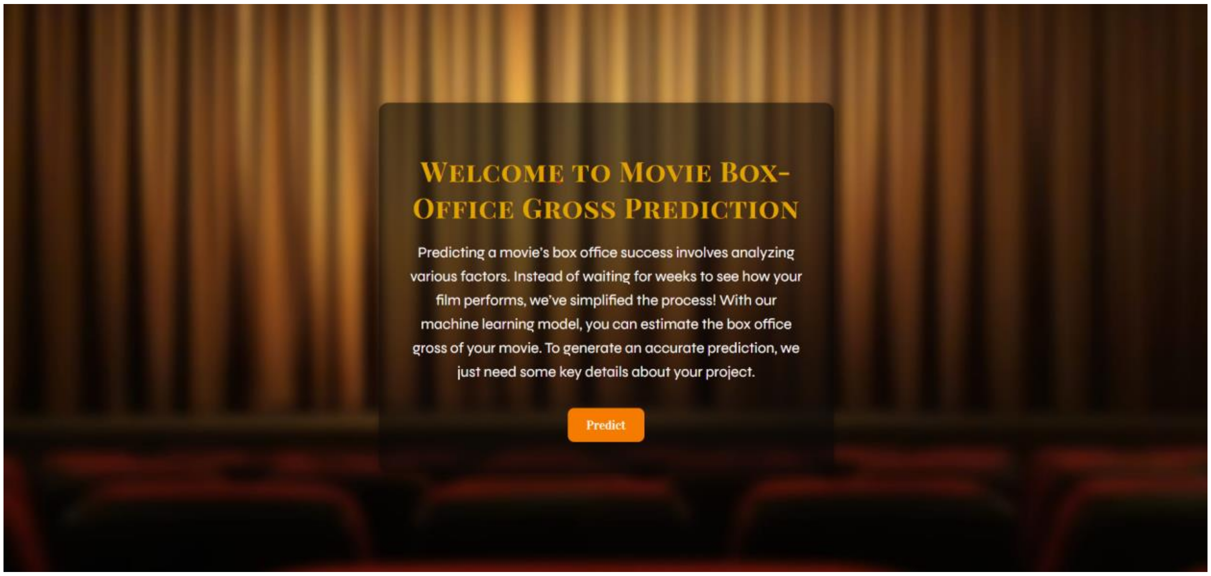
5.3 Final Model Selection Justification:

Final Model	Reasoning
Linear Regression	The Linear Regression model was chosen as the final optimized model because it exhibited the highest R-squared value (0.7758), indicating a strong fit to the data. Additionally, it had a lower MSE (8649.14) and Accuracy (77%) compared to other models, suggesting superior predictive accuracy.

6. Results

6.1 Outputs screenshots

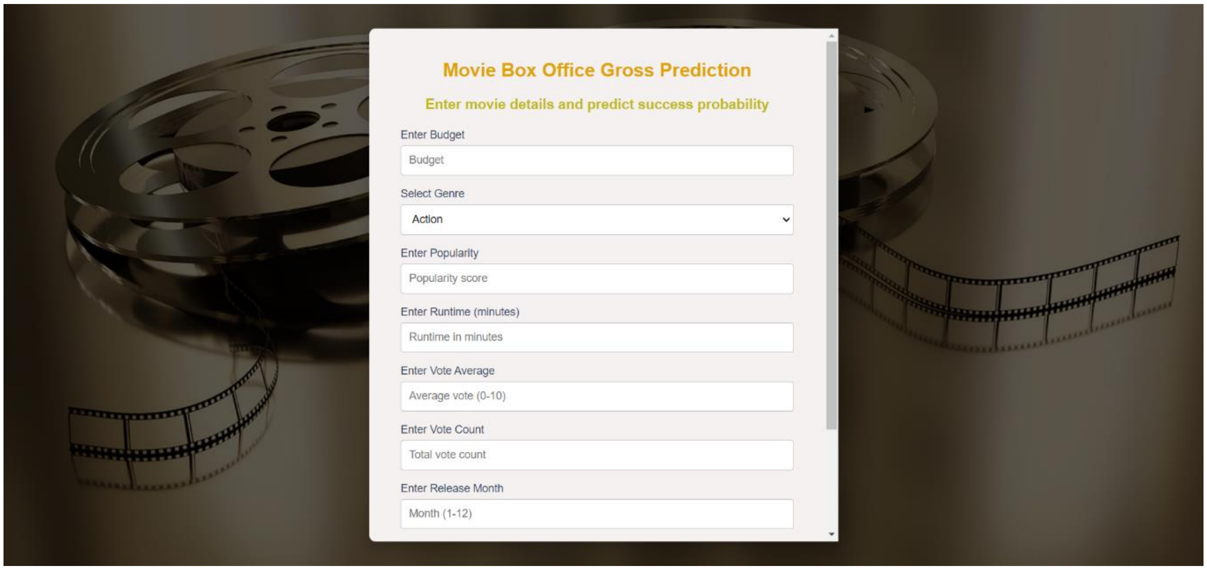
Output of interface is :



WELCOME TO MOVIE BOX-OFFICE GROSS PREDICTION

Predicting a movie's box office success involves analyzing various factors. Instead of waiting for weeks to see how your film performs, we've simplified the process! With our machine learning model, you can estimate the box office gross of your movie. To generate an accurate prediction, we just need some key details about your project.

Output for details input form is :



Movie Box Office Gross Prediction

Enter movie details and predict success probability

Enter Budget

Budget

Select Genre

Action

Enter Popularity

Popularity score

Enter Runtime (minutes)

Runtime in minutes

Enter Vote Average

Average vote (0-10)

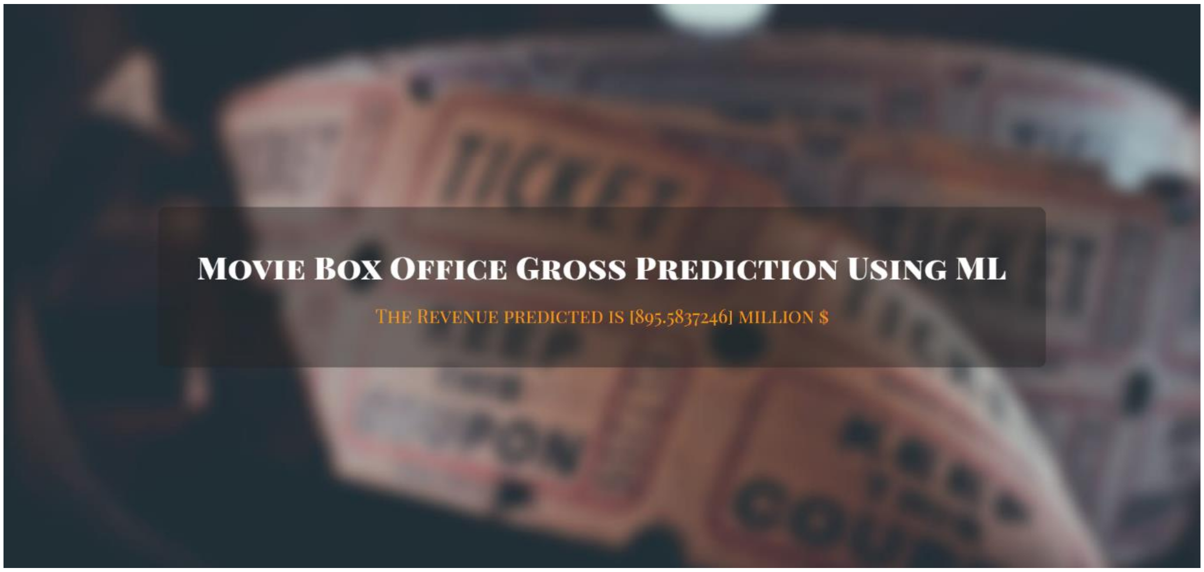
Enter Vote Count

Total vote count

Enter Release Month

Month (1-12)

Output for Revenue Predicted is :



Movie Box Office Gross Prediction Using ML

The Revenue predicted is [895.5837246] million $

7. Advantages & Disadvantages

Advantages :

1. **Accuracy**: Machine learning models can achieve high accuracy in predicting movie box office performance by considering numerous factors such as cast, budget, and marketing strategies, leading to better-informed decisions.
2. **Automation**: Once trained, ML models can automatically analyze new movie data and provide real-time revenue predictions without manual intervention, aiding in rapid decision-making.
3. **Scalability**: ML models can handle large datasets, including thousands of movies and variables, and adapt to new data, allowing scalability as more historical data becomes available.
4. **Pattern Recognition**: ML models excel at identifying intricate patterns in movie data—such as the influence of genre, star power, or release timing—that might not be evident through traditional statistical methods.
5. **Customization**: Models can be tailored to specific movie types, genres, or even production companies, improving the relevance and accuracy of predictions based on particular industry needs.
6. **Cost-Effective**: Over time, automated ML models reduce the need for manual analysis, helping production studios optimize budgets and marketing spends efficiently, potentially lowering operational costs.

Disadvantages :

1. **Data Dependency**: High-quality, comprehensive datasets on movies are essential for training accurate models. Incomplete or inaccurate data can lead to flawed predictions, affecting decision-making.
2. **Complexity**: Developing, training, and fine-tuning ML models for box office prediction can be complex and requires expertise in machine learning, data science, and the entertainment industry.
3. **Resource Intensive** Training machine learning models, especially with large datasets, can require significant computational resources and time, which can be a limiting factor for some production companies.
4. **Model Maintenance**: ML models must be regularly updated with new data from recent movie releases and industry trends to maintain prediction accuracy, necessitating ongoing maintenance efforts.
5. **Bias**: If the training data contains biases (e.g., focusing on blockbuster movies or certain genres), the model may produce biased predictions, skewing revenue forecasts for certain types of films.

8. Conclusion

The application of machine learning, specifically gradient boosting, in our movie box office revenue prediction project has demonstrated the significant potential of ML techniques in the entertainment industry. Gradient boosting, an ensemble learning method, aggregates multiple weak learners to build a strong predictive model, improving the overall accuracy and reliability of revenue forecasts.Throughout this project, careful preprocessing of the dataset was a crucial step to ensure that the inputs were of high quality, which is essential for the success of any machine learning model. Gradient boosting was employed to capture the complex, non-linear relationships between various factors such as budget, cast, genre, marketing strategies, and release timing. By iteratively refining the errors of previous models, gradient boosting honed the predictions, resulting in a highly accurate model for box office revenue prediction.

The evaluation metrics, including R^2, Mean Squared Error (MSE), and Mean Absolute Error (MAE), showcased the effectiveness of the gradient boosting model. The R^2 score revealed how well the model captured the variance in box office performance, while the MSE and MAE provided insights into the magnitude of prediction errors. These metrics are essential for film studios and production companies to make informed decisions on budget allocation, marketing strategies, and release timing.One of the key advantages of using gradient boosting is its adaptability. The model can be easily retrained with new movie data, ensuring that it remains relevant and accurate as new trends and market dynamics emerge. Furthermore, its ability to handle noisy data and outliers makes it well-suited for real-world datasets, which often contain imperfections and inconsistencies.

However, it is important to recognize the limitations of this approach. Gradient boosting models can be computationally intensive, requiring significant processing power and time to train, especially when dealing with large datasets of movies. Additionally, the model can overfit if not properly tuned, leading to poorer performance on unseen data. Therefore, careful tuning of model parameters and regular validation are necessary to maintain its accuracy and generalizability.

In conclusion, our use of gradient boosting for movie box office revenue prediction has proven to be a valuable tool in improving forecast accuracy and offering deeper insights into the factors driving box office success. The results of this project provide a foundation for further advancements in predictive modeling for the film industry, helping stakeholders make better decisions and optimize their strategies. By continuing to refine the model and incorporating more diverse datasets, machine learning can unlock even greater potential for predicting and understanding the complexities of box office performance, ultimately leading to smarter, data-driven decision-making in the movie industry.

9. Future Scope

1) **Model Enhancement**: Continuously refine the model by incorporating more extensive and diverse movie datasets, including emerging genres, audience preferences, and newer marketing trends, to improve the model's prediction accuracy and robustness.

2) **Integration with Real-Time Data**: Incorporate real-time data from social media, online reviews, and box office reports to provide up-to-date predictions. This would enable tracking a movie's performance as it evolves after release and provide more dynamic forecasts.

3) **User-Friendly Interface**: Develop a user-friendly web or mobile application that presents revenue predictions in a clear and accessible format, enabling stakeholders like producers, marketers, and distributors to easily interpret and act on the predictions.

4) **Geographic Expansion**: Adapt the model to cover different international markets and regions, considering local market dynamics, audience preferences, and economic conditions to provide more accurate predictions across global box offices.

5) **Incorporating Trends and Events**: Study the impact of broader cultural trends, global events, and competitor releases on box office performance and incorporate these factors into the model to enhance its predictive power for specific periods or events.

6) **Collaboration with Industry Experts**: Work closely with film industry professionals—such as producers, market analysts, and distributors—to continually validate and improve the model, ensuring it remains relevant, interpretable, and actionable for real-world use.

10. Appendix

10.1 Source Code

Movie Prediction.ipynb

```
#IMPORTING NECESSARY 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

credits = pd.read_csv('/content/tmdb_5000_credits.csv')

movies_df = pd.read_csv('/content/tmdb_5000_movies.csv')

credits.head()

movies_df.head()

print("Credits columns",credits.columns)

print("Movies columns",movies_df.columns)

print("Credits shape",credits.shape)
```

```
  print("Movies shape",movies_df.shape)

  credits_column_renamed=credits.rename(index=str,columns={"movie_id":"id"})

movies=movies_df.merge(credits_column_renamed,on="id")

movies.shape

movies.head()

movies.info()

movies.describe()

movies['crew']=movies['crew'].apply(json.loads)

def director(x):

   for i in x:

      if i['job'] == 'Director':

         return i['name']

movies['crew'] = movies['crew'].apply(director)

movies.rename(columns={'crew':'director'},inplace=True)

from ast import literal_eval

features = ['keywords','genres']

for feature in features:

  movies[feature] = movies[feature].apply(literal_eval)

def get_list(x):

  if isinstance(x, list):

    names = [i['name'] for i in x]

    if len(names) > 1:

      names = names[:1]
    return names
  return []
features = ['keywords', 'genres']
for feature in features:
  movies[feature] = movies[feature].apply(get_list)
movies['genres']
movies['genres'] = movies['genres'] .str.join(', ')
movies['genres'].info()
movies['genres']
movies['keywords'] = movies['keywords'].str.join(', ')
movies['keywords']
movies.isnull().sum()
movies = movies.dropna(subset = ['director','runtime'])
movies["revenue"]=movies["revenue"].floordiv(1000000)
movies["budget"]=movies["budget"].floordiv(1000000)
movies = movies[movies['budget'] != 0]
# Ensure 'release_date' is already in datetime format without any data loss
movies['release_date'] = pd.to_datetime(movies['release_date'], errors='raise')  # Raise errors if any data issues exist


# Extract the release month from the 'release_date' column
movies['release_month'] = movies['release_date'].dt.month
```

```
# Extract the day of the week (Monday=0, Sunday=6) from the 'release_date' column
movies['release_DOW'] = movies['release_date'].dt.dayofweek

# Preview the new columns
#print(movies[['release_date', 'release_month', 'release_DOW']].head())

sns.boxplot(x=movies['runtime'])
plt.title('Boxplot of Runtime')
sns.boxplot(x=movies['revenue'])
plt.title('Boxplot of Revenue')
sns.boxplot(x=movies['budget'])
plt.title('Boxplot of Budget')
movies['log_revenue'] = np.log1p(movies['revenue'])
movies['log_budget'] = np.log1p(movies['budget'])
fig, ax = plt.subplots(figsize = (16, 6))
plt.subplot(1, 2, 1)
plt.hist(movies['revenue']);
plt.title('Distribution of revenue');
plt.subplot(1, 2, 2)
plt.hist(movies['log_revenue']);
plt.title('Distribution of log transformation of revenue');
plt.figure(figsize=(16, 8))
plt.subplot(1, 2, 1)

plt.scatter(movies['budget'], movies['revenue'])

plt.title('Revenue vs budget fig(1)');

plt.subplot(1, 2, 2)

plt.scatter(movies['log_budget'], movies['log_revenue'])

plt.title('Log Revenue vs log budget fig(2)');

wordcloud = WordCloud().generate(movies.original_title.to_string())

sns.set(rc={'figure.figsize':(12,8)})

plt.imshow(wordcloud, interpolation='bilinear')

plt.axis("off")

plt.show()

movies['has_homepage'] = 0

movies.loc[movies['homepage'].isnull() == False, 'has_homepage'] = 1

sns.catplot(x='has_homepage', y='revenue', data=movies);

plt.title('Revenue for movie with and w/o homepage');

# Fixing the jointplot call by specifying x and y arguments

plt.figure(figsize=(15,8))

sns.jointplot(x='release_month', y='revenue', data=movies)

plt.xticks(rotation=90)

plt.xlabel('Months')

plt.title('Revenue by Release Month')

movies['log_revenue'] = np.log1p(movies['revenue'])

movies['log_budget'] = np.log1p(movies['budget'])
```

```
movies_box =
movies.drop(['homepage','id','keywords','original_language','original_title',
                   'overview','production_countries','release_date','spoken_languages',
                   'status','tagline','title_x','title_y','cast',
                   'log_revenue','log_budget'],axis = 1)
movies_box=movies_box.drop(['production_companies'],axis=1)
movies_box.isnull().sum()
from sklearn.preprocessing import LabelEncoder
from collections import Counter as c
cat=['director','genres']
for i in movies_box[cat]:
  print("LABEL ENCODING OF:",i)
  LE = LabelEncoder()
  print(c(movies_box[i]))
  movies_box[i] = LE.fit_transform(movies_box[i])
  print(c(movies_box[i]))
mapping_dict ={}
category_col=["directory","director"]
for col in category_col:
  LE_name_mapping = dict(zip(LE.classes_,LE.transform(LE.classes_)))
  mapping_dict[col]= LE_name_mapping
  print(mapping_dict)
x=movies_box.iloc[:,[0,1,2,4,5,6,8,9]]
```

```
x=pd.DataFrame(x,columns=['budget','genres','popularity','runtime','vote_average','vote_count','release_month','release_DOW'])

x

y=movies_box.iloc[:,[3]]

y=pd.DataFrame(y,columns=['revenue'])

y

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

x=sc.fit_transform(x)

x

pickle.dump(sc,open("scalar_movies.pkl",'wb'))

plt.figure(figsize=(8, 5))

sns.histplot(movies['budget'], bins=30, kde=True)

plt.title('Distribution of Budget')

plt.xlabel('Budget')

plt.ylabel('Frequency')

plt.show()

plt.figure(figsize=(8, 5))

sns.scatterplot(x='budget', y='revenue', data=movies)

plt.title('Budget vs Revenue')

plt.xlabel('Budget')

plt.ylabel('Revenue')

plt.show()
plt.figure(figsize=(8, 5))

sns.scatterplot(x='vote_average', y='revenue', data=movies)

plt.title('Vote Average vs Revenue')

plt.xlabel('Vote Average')

plt.ylabel('Revenue')

plt.show()

plt.figure(figsize=(10, 8))

corr_matrix = movies[['budget', 'revenue', 'vote_average', 'vote_count',
'popularity']].corr()

sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', linewidths=0.5)

plt.title('Correlation Heatmap')

plt.show()

from sklearn.model_selection import train_test_split

x_train,x_test,y_train,y_test=train_test_split(x,y, test_size=0.1, random_state=11)

from sklearn.linear_model import LinearRegression

mr=LinearRegression()

mr.fit(x_train,y_train)

y_pred_mr=mr.predict(x_test)

from sklearn import metrics

print("MAE:",metrics.mean_absolute_error(y_test,y_pred_mr))

print("RMSE:",np.sqrt(metrics.mean_absolute_error(y_test,y_pred_mr)))

from sklearn.metrics import r2_score

r2_score(y_test,y_pred_mr)
```

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score


def linear_regression(X_train, X_test, y_train, y_test):
    lr = LinearRegression()
    lr.fit(X_train, y_train)
    y_pred = lr.predict(X_test)
    print("**Linear Regression**")
    print("Mean Absolute Error: ", mean_absolute_error(y_test, y_pred))
    print("Mean Squared Error: ", mean_squared_error(y_test, y_pred))
    print("R2 Score: ", r2_score(y_test, y_pred))
linear_regression(x_train, x_test, y_train, y_test)
from sklearn.tree import DecisionTreeRegressor


def decision_tree_regression(X_train, X_test, y_train, y_test):
    dt = DecisionTreeRegressor()
    dt.fit(X_train, y_train)
    y_pred = dt.predict(X_test)
    print("**Decision Tree Regression**")
    print("Mean Absolute Error: ", mean_absolute_error(y_test, y_pred))
    print("Mean Squared Error: ", mean_squared_error(y_test, y_pred))
    print("R2 Score: ", r2_score(y_test, y_pred))
decision_tree_regression(x_train, x_test, y_train, y_test)
```

```
from sklearn.svm import SVR


def svm_regression(X_train, X_test, y_train, y_test):
    svm = SVR()
    svm.fit(X_train, y_train)
    y_pred = svm.predict(X_test)
    print("**Support Vector Machine Regression**")
    print("Mean Absolute Error: ", mean_absolute_error(y_test, y_pred))
    print("Mean Squared Error: ", mean_squared_error(y_test, y_pred))
    print("R2 Score: ", r2_score(y_test, y_pred))
svm_regression(x_train, x_test, y_train, y_test)
from sklearn.ensemble import RandomForestRegressor


def random_forest_regression(X_train, X_test, y_train, y_test):
    rf = RandomForestRegressor()
    rf.fit(X_train, y_train)
    y_pred = rf.predict(X_test)
    print("**Random Forest Regression**")
    print("Mean Absolute Error: ", mean_absolute_error(y_test, y_pred))
    print("Mean Squared Error: ", mean_squared_error(y_test, y_pred))
    print("R2 Score: ", r2_score(y_test, y_pred))
random_forest_regression(x_train, x_test, y_train, y_test)
from sklearn.neighbors import KNeighborsRegressor
```

```
def knn_regression(X_train, X_test, y_train, y_test):
    knn = KNeighborsRegressor()
    knn.fit(X_train, y_train)
    y_pred = knn.predict(X_test)
    print("**K-Nearest Neighbors Regression**")
    print("Mean Absolute Error: ", mean_absolute_error(y_test, y_pred))
    print("Mean Squared Error: ", mean_squared_error(y_test, y_pred))
    print("R2 Score: ", r2_score(y_test, y_pred))
knn_regression(x_train, x_test, y_train, y_test)
from sklearn.linear_model import Ridge
from sklearn.model_selection import GridSearchCV


def ridge_regression_with_tuning(X_train, X_test, y_train, y_test):
    param_grid = {'alpha': [0.001, 0.01, 0.1, 1, 10, 100]}  # Regularization strength


    ridge = Ridge()
    grid_search = GridSearchCV(estimator=ridge, param_grid=param_grid, cv=5,
scoring='r2')
    grid_search.fit(X_train, y_train)


    best_ridge = grid_search.best_estimator_
    y_pred = best_ridge.predict(X_test)
    print("Best Ridge Hyperparameters:", grid_search.best_params_)

    print("R2 Score: ", r2_score(y_test, y_pred))

    print("Mean Absolute Error: ", mean_absolute_error(y_test, y_pred))

    print("Mean Squared Error: ", mean_squared_error(y_test, y_pred))


# Example usage:

ridge_regression_with_tuning(x_train, x_test, y_train, y_test)

from sklearn.svm import SVR

from sklearn.model_selection import GridSearchCV


def svr_with_tuning(X_train, X_test, y_train, y_test):

    param_grid = {

        'C': [0.1, 1, 10, 100],            # Regularization parameter

        'epsilon': [0.001, 0.01, 0.1, 1],  # Epsilon-tube within which no penalty is
associated

        'kernel': ['linear', 'rbf']         # Type of kernel

    }


    svr = SVR()

    grid_search = GridSearchCV(estimator=svr, param_grid=param_grid, cv=5,
scoring='r2')

    grid_search.fit(X_train, y_train)
    best_svr = grid_search.best_estimator_

    y_pred = best_svr.predict(X_test)


    print("Best SVR Hyperparameters:", grid_search.best_params_)

    print("R2 Score: ", r2_score(y_test, y_pred))

    print("Mean Absolute Error: ", mean_absolute_error(y_test, y_pred))

    print("Mean Squared Error: ", mean_squared_error(y_test, y_pred))


# Example usage:

svr_with_tuning(x_train, x_test, y_train, y_test)

from sklearn.ensemble import RandomForestRegressor

from sklearn.model_selection import GridSearchCV


def random_forest_regressor_with_tuning(X_train, X_test, y_train, y_test):

    param_grid = {

        'n_estimators': [100],      # Number of trees in the forest

        'max_depth': [20],      # Maximum depth of the tree

        'min_samples_split': [ 5],      # Minimum number of samples required to split a
node

        'min_samples_leaf': [2],       # Minimum number of samples required at a leaf
node
```

```
        'bootstrap': [True]          # Whether bootstrap samples are used when building
trees
    }


    rf = RandomForestRegressor()
    grid_search = GridSearchCV(estimator=rf, param_grid=param_grid, cv=5,
scoring='r2')
    grid_search.fit(X_train, y_train)


    best_rf = grid_search.best_estimator_
    y_pred = best_rf.predict(X_test)


    print("Best Random Forest Hyperparameters:", grid_search.best_params_)
    print("R2 Score: ", r2_score(y_test, y_pred))
    print("Mean Absolute Error: ", mean_absolute_error(y_test, y_pred))
    print("Mean Squared Error: ", mean_squared_error(y_test, y_pred))


# Example usage:
random_forest_regressor_with_tuning(x_train, x_test, y_train, y_test)
from sklearn.neighbors import KNeighborsRegressor
from sklearn.model_selection import GridSearchCV


def knn_regressor_with_tuning(X_train, X_test, y_train, y_test):
    param_grid = {
        'n_neighbors': [3, 5, 7, 9],        # Number of neighbors
        'weights': ['uniform', 'distance'], # Weight function used in prediction
        'metric': ['euclidean', 'manhattan'] # Distance metric
    }

    knn = KNeighborsRegressor()
    grid_search = GridSearchCV(estimator=knn, param_grid=param_grid, cv=5,
scoring='r2')
    grid_search.fit(X_train, y_train)

    best_knn = grid_search.best_estimator_
    y_pred = best_knn.predict(X_test)

    print("Best KNN Hyperparameters:", grid_search.best_params_)
    print("R2 Score: ", r2_score(y_test, y_pred))
    print("Mean Absolute Error: ", mean_absolute_error(y_test, y_pred))
    print("Mean Squared Error: ", mean_squared_error(y_test, y_pred))

# Example usage:
knn_regressor_with_tuning(x_train, x_test, y_train, y_test)
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score


def decision_tree_regressor_with_tuning(X_train, X_test, y_train, y_test):
    dt = DecisionTreeRegressor()
    param_grid = {
        'max_depth': [5, 10, 20],
        'min_samples_split': [2, 5, 10],
        'min_samples_leaf': [1, 2, 4]
    }
    grid_search = GridSearchCV(dt, param_grid, cv=5, scoring='r2')
    grid_search.fit(X_train, y_train)

    best_dt = grid_search.best_estimator_
    y_pred = best_dt.predict(X_test)

    print("**Decision Tree Regressor**")
    print("Best Params: ", grid_search.best_params_)
    print("Mean Absolute Error: ", mean_absolute_error(y_test, y_pred))
    print("Mean Squared Error: ", mean_squared_error(y_test, y_pred))
    print("R2 Score: ", r2_score(y_test, y_pred))

# Usage:
decision_tree_regressor_with_tuning(x_train, x_test, y_train, y_test)
```

```
import pickle

pickle.dump(mr,open("model_movies.pkl","wb"))

model=pickle.load(open("model_movies.pkl","rb"))

scalar=pickle.load(open("scalar_movies.pkl","rb"))


input=[[50,8,20.239061,88,5,366,7,3]]

input=scalar.transform(input)

prediction = model.predict(input)


prediction
```

Input.html

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Movie Details Form</title>
    <style>
        * {
            box-sizing: border-box;
            margin: 0;
            padding: 0;
        }

        body {
            font-family: 'Arial', sans-serif;
            background-color: #f0f0f0;
            display: flex;
            justify-content: center;
            align-items: center;
            height: 100vh; /* Full height of the viewport */
            width: 100vw; /* Full width of the viewport */
            background-size: cover;
            background-position: center;
            background-image: url('../static/website_image-3.jpg'); /* Replace with actual image path */
            overflow: hidden; /* Prevent scrolling */
            position: relative;
        }

        /* Adding a semi-transparent overlay to make text more visible */
        body::before {
            content: '';
            position: absolute;
            top: 0;
            left: 0;
            right: 0;
            bottom: 0; /* Cover entire viewport */
            background-color: rgba(0, 0, 0, 0.6); /* Dark overlay */
```

```
    z-index: 1;
}

.container {
    background-color: rgb(245, 241, 241); /* Slight transparency */
    padding: 40px;
    border-radius: 8px;
    box-shadow: 0 8px 16px rgba(0, 0, 0, 0.2);
    max-width: 600px; /* Maximum width of the form */
    width: 90%; /* Responsive width */
    max-height: 90vh; /* Ensure it fits within the viewport */
    overflow: auto; /* Enable scrolling for container if needed */
    z-index: 2; /* Place it above the overlay */
    position: relative;
}

h2 {
    text-align: center;
    color: #dfa401; /* Light color for better contrast */
    margin-bottom: 20px;
    font-weight: bold;
}

h4 {
    text-align: center;
    font-size: 18px;
    margin-bottom: 20px;
    color: #c0b70bf4; /* Light color for better contrast */
```

```
}

form {
    display: flex;
    flex-direction: column;
}

label {
    font-size: 14px;
    margin-bottom: 5px;
    color: #34495e;
}

input,
select {
    padding: 10px;
    margin-bottom: 15px;
    border-radius: 4px;
    border: 1px solid #bdc3c7;
    font-size: 14px;
    width: 100%;
}

input:focus,
select:focus {
    outline: none;
    border-color: #3498db;
}
```

```
input[type="submit"] {
    background-color: #f9bb02;
    color: #ffffff;
    cursor: pointer;
    border: none;
    font-size: 16px;
    transition: background-color 0.3s ease;
}

input[type="submit"]:hover {
    background-color: #fbd04f;
}

.container {
    animation: fadeIn 1s ease-in-out;
}

@keyframes fadeIn {
    from {
        opacity: 0;
        transform: translateY(-10px);
    }

    to {
        opacity: 1;
        transform: translateY(0);
    }
```

```
        }
    </style>
</head>

<body>
    <div class="container">
        <h2>Movie Box Office Gross Prediction</h2>
        <h4>Enter movie details and predict success probability</h4>
        <form action="/submit" method="POST">
            <label for="budget">Enter Budget</label>
            <input type="number" id="budget" min="0" name="budget"
placeholder="Budget" required>

            <label for="genres">Select Genre</label>
            <select id="genres" name="genres">
                <option value="0">Action</option>
                <option value="1">Adventure</option>
                <option value="2">Animation</option>
                <option value="3">Comedy</option>
                <option value="4">Crime</option>
                <option value="5">Documentary</option>
                <option value="6">Drama</option>
                <option value="7">Family</option>
                <option value="8">Fantasy</option>
                <option value="9">History</option>
                <option value="10">Horror</option>
                <option value="11">Music</option>
                <option value="12">Mystery</option>
                <option value="13">Romance</option>
```

```
          <option value="14">Science Fiction</option>

          <option value="15">TV Movie</option>

          <option value="16">Thriller</option>

          <option value="17">War</option>

          <option value="18">Western</option>

        </select>


        <label for="popularity">Enter Popularity</label>

        <input type="number" id="popularity" min="0" name="popularity"
placeholder="Popularity score" required>


        <label for="runtime">Enter Runtime (minutes)</label>

        <input type="number" id="runtime" min="0" name="runtime"
placeholder="Runtime in minutes" required>


        <label for="vote_average">Enter Vote Average</label>

        <input type="number" id="vote_average" min="0" max="10" step="0.1"
name="vote_average" placeholder="Average vote (0-10)" required>


        <label for="vote_count">Enter Vote Count</label>

        <input type="number" id="vote_count" min="0" name="vote_count"
placeholder="Total vote count" required>


        <label for="release_month">Enter Release Month</label>

        <input type="number" id="release_month" min="1" max="12"
name="release_month" placeholder="Month (1-12)" required>


        <label for="release_DOW">Enter Release Week (Day of the Week)</label>

        <input type="number" id="release_DOW" min="1" max="7"
name="release_DOW" placeholder="Day (1 for Monday, etc.)" required>
```

```
            <input type="submit" value="Submit">

        </form>

    </div>

</body>


</html>
```

Home.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>Movie Box-Office Gross Prediction</title>
    <style>
        @import
url('https://fonts.googleapis.com/css2?family=Syne:wght@400..800&displa
y=swap');
        @import
url('https://fonts.googleapis.com/css2?family=Vidaloka&display=swap');
      @import
url('https://fonts.googleapis.com/css2?family=Rubik+Mono+One&display=
swap'); @import
url('https://fonts.googleapis.com/css2?family=Playfair+Display+SC:ital,wgh
t@0,400;0,700;0,900;1,400;1,700;1,900&display=swap');

        body {
            margin: 0;
            padding: 0;
            font-family: 'Arial', sans-serif;
            background-image: url('../static/website_image-1.jpg'); /* Replace
with your background image URL */
            background-size: cover;
            background-position: center;
            height: 100vh;
            display: flex;
            justify-content: center;
            align-items: center;
        }

        .box {
            background-color: rgba(7, 7, 7, 0.567); /* Semi-transparent white */
            padding: 40px;
            border-radius: 12px;
            text-align: center;
            box-shadow: 0 4px 10px rgba(0, 0, 0, 0.3);
            max-width: 500px;
            margin: auto;
        }

        h1 {
            font-family: "Playfair Display SC", serif;
            letter-spacing: 1px;
            font-size: 35px;
            color: #e1a500;
            margin-bottom: 20px;
        }

        p {
            font-family: "Syne", sans-serif;
            font-optical-sizing: auto;
            font-size: 19px;
            color: #fffbfb;
            margin-bottom: 30px;
            line-height: 1.6;
        }

        a {
           font-family: "Vidaloka", serif;
            display: inline-block;
            padding: 12px 24px;
            font-size: 16px;
            color: rgb(247, 248, 248);
            background-color: #f57c00; /* Orange button */
            text-decoration: none;
            border-radius: 8px;
            transition: background-color 0.3s ease;
        }

        a:hover {
            background-color: #ff9800; /* Slightly lighter orange on hover */
        }
```

```
    </style>
</head>
<body>
    <div class="box">
        <h1>Welcome to Movie Box-Office Gross Prediction</h1>
        <p>
            Predicting a movie’s box office success involves analyzing various
factors.
            Instead of waiting for weeks to see how your film performs, we’ve
simplified the process!
            With our machine learning model, you can estimate the box office
gross of your movie.
            To generate an accurate prediction, we just need some key details
about your project.
        </p>
        <a href="/predict">Predict</a>
    </div>
</body>
</html>
```

Output.html

```
<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Prediction Result</title>

    <style>

        @import
url('https://fonts.googleapis.com/css2?family=Oswald:wght@200..700&display=s
wap');

      @import
url('https://fonts.googleapis.com/css2?family=Anton&family=Oswald:wght@200.
.700&display=swap');
```

```
   @import
url('https://fonts.googleapis.com/css2?family=Rubik+Mono+One&display=swap')
;
   @import
url('https://fonts.googleapis.com/css2?family=Playfair+Display+SC:ital,wght@0,4
00;0,700;0,900;1,400;1,700;1,900&display=swap');
    /* Reset margins and padding for all elements */
    * {
      margin: 0;
      padding: 0;
      box-sizing: border-box;
    }

    /* Set height for the entire HTML and body to 100% */
    html, body {
      height: 100%;
      font-family: 'Arial', sans-serif;
    }

    /* Fullscreen background styling */
    body {
      background: url('../static/website_image-2.jpg') no-repeat center
center/cover;
      display: flex;
      justify-content: center;
      align-items: center;
      background-size:cover;
    }

    /* Centering content with Flexbox */
    .content {
      display: flex;
```

```
        justify-content: center;
        align-items: center;
        flex-direction: column;
        background-color: rgba(37, 37, 37, 0.445); /* Semi-transparent background
*/
        padding: 50px;
        border-radius: 10px;
        text-align: center;
    }

    /* Title box styling */
    .title-box {
        color: white;
    }

    /* Styling for the title */
    .title {
        font-size: 2.5rem;
        font-family:"Playfair Display SC", serif ;
        letter-spacing: 1px;
        font-weight: 900;
        margin-bottom: 20px;
    }

    /* Styling for the result text */
    .result {
        font-size: 1.5rem;
        font-family:"Playfair Display SC", serif ;
        margin-top: 10px;
        color: #f39c12;
    }
    </style>
</head>
<body>
    <div class="content">
        <div class="title-box">
            <div class="title">Movie Box Office Gross Prediction Using ML</div>
            <div class="result">The Revenue predicted is {{result}} million $</div>
        </div>
    </div>
</body>
</html>
```

10.2 GitHub & Project Demo Link

GitHub link :

https://github.com/BramhamNitturi/Movie-Box-Office-Gross-Prediction

Project Demo Link :

https://drive.google.com/file/d/1aEaP44zW2oXS0BQ1gD86LF4wX4aVquwz/view?usp=drivesdk