**PROJECT REPORT ON**

“Predicting Movie Box Office Revenue

**TABLE OF CONTENTS**

|  |  |  |
| --- | --- | --- |
| **S.No.** | **Title** | **Page No.** |
| 01 | INTRODUCTION | 02 |
| 02 | OBJECTIVE | 03 |
| 03 | BACKGROUND | 04-08 |
| 04 | HARDWARE AND SOFTWARE REQUIREMENTS | 09 |
| 06 | CODING | 10-18 |
| 07 | OUTPUT SCREENSHOTS | 19-25 |
| 08 | FUTURE SCOPE | 26 |
| 09 | CONCLUSION | 27 |
| 10 | REFERENCES | 28 |

**INTRODUCTION**

Predicting Movie Box office Revenue is built using the Machine Learning in python. For GUI tkinter is used where user will be able to interact with the GUI. User will be able to see the revenue generated by a particular movie by selecting Movie Name from the dropdown which is inside GUI window. User can also select the Lead actor through which user will be able to see how many movies are done by the lead actor and how much revenue generated and a graph visualization will be shown where the feature will be lead actor and target will be revenue. Graph of feature and target where feature will be (Budget(INR), Movie name, genre) target will be Revenue(INR) which is generated in the Movie Box office. Predicting the box office revenue of a movie is a challenging yet crucial task for producers, distributors, and investors. Accurate revenue predictions can inform decision-making processes, such as marketing strategies and budget allocations. This project aims to leverage machine learning techniques to predict the revenue generated by the movie which is inside a particular Movie dataset. By selecting Number of Screens from the dropdown and after selecting clicking on the Button Predict Revenue the user will see the messagebox where actual and predicted revenue will be shown.

**OBJECTIVE**

The main objectives of this project include:

* Developing a machine learning model to predict movie box office revenue based on various features.
* Creating an interactive and user-friendly GUI using Tkinter to allow users to input movie details and receive revenue predictions.
* Evaluating and fine-tuning the model for optimal accuracy.
* Understanding the concept of various Machine Learning Model which can be used in the project.
* Knowing other modules in the python which is used in this project.
* Understanding the plotting of various types of Graph using matplotlib and embedding in GUI tkinter window

**BACKGROUND**

The project involves implementation of Predicting Movie Box Office Revenue using Machine Learning in Python, specifically leveraging the ML model. This model provides a suite of functionality to provide the Revenue.

MACHINE LEARNING MODEL USED:

* **Random Forest Regressor Overview:**
* **Why to Use RandomForestRegressor for Predicting Movie Box Office Revenue:**
* Handling Non-linearity: Movie box office revenue prediction often involves non-linear relationships between various features (e.g., budget, cast size, critical reviews) and the revenue outcome. Random Forest can capture these non-linearities effectively
* Random Forest is an ensemble learning algorithm that operates by constructing a multitude of decision trees during training and outputs the mean prediction of the individual trees for regression tasks. It belongs to the family of bagging algorithms, and its power lies in combining the predictive strength of multiple decision trees to create a more robust and accurate model.
* key concepts related to the RandomForestRegressor:
* Decision Trees: The basic building blocks of a Random Forest are decision trees. Decision trees split the data into subsets based on feature conditions, creating a tree-like structure. Each leaf node represents a prediction.
* Ensemble Averaging: The final prediction of the Random Forest Regressor is the average (or mean) of the predictions made by individual trees. This ensemble averaging helps in reducing overfitting and improving generalization.
* Variable Importance: Random Forest provides a measure of feature importance. By assessing the average decrease in impurity (e.g., mean squared error) caused by each feature, you can understand which features contribute the most to the predictive performance.

DATASET USED

* Name of Dataset : movie\_dataset.csv
* Source of Dataset: Kaggle
* About the Dataset:
* movie\_dataset.csv is containing 1698 Top Bollywood movies from 2005 to 2017.
* All the movie of dataset is in hindi language.
* This dataset includes 14 columns and 1698 rows.
* Column Name: Movie Name, Release Period, Whether Remake, Whether Franchise, Genre, New Actor ,New Director, New Music Director, Lead Star, Director, Music Director, Number of Screens, Revenue(INR), Budget(INR)
* Revenue(INR) will be the target of this project bcz the project is basically the predicting movie box office revenue.
* Datatype of the Columns are:
* Movie Name: String
* Release Period: String
* Whether Remake: String
* Whether Franchise: String
* Genre: String
* New Actor: String
* New Director: String
* New Music Director: String
* Lead Star: String
* Director: String
* Music Director: String
* Number of Screens: Integer
* Revenue(INR): Integer
* Budget(INR): Integer

MODULE USED:

* Tkinter: Tkinter is the standard GUI (Graphical User Interface) toolkit for Python. Tkinter provides a simple way to create windows, dialogs, buttons, textboxes, and various other GUI elements for desktop applications
* Python: Python is a high-level, interpreted, and general-purpose programming language.
* **Importing Libraries:**
  + **import numpy as np**: Imports the NumPy library and aliases it as np for convenience.
  + **import pandas as pd**: Imports the Pandas library and aliases it as pd.
  + **import matplotlib.pyplot as plt**: Imports the Pyplot module from the Matplotlib library and aliases it as plt.
  + **from tkinter import \***: Imports all classes and functions from the Tkinter library.
  + **from tkinter import ttk, messagebox:** Imports specific classes (ttk for themed widgets, messagebox for displaying message boxes) from Tkinter.
  + **from tabulate import tabulate:** Imports the tabulate function from the Tabulate library for formatting tables.
  + **from sklearn.compose import ColumnTransformer**: Imports the ColumnTransformer class from scikit-learn for column transformations.
  + **from sklearn.pipeline import Pipeline:** Imports the Pipeline class from scikit-learn for constructing a sequence of data processing steps.
  + **from matplotlib.backends.backend\_tkagg import FigureCanvasTkAgg**: Imports the backend for embedding Matplotlib figures in Tkinter applications.
  + **from sklearn.model\_selection import train\_test\_split**: Imports the train\_test\_split function for splitting datasets into training and testing sets.
  + **from sklearn.ensemble import RandomForestRegressor**: Imports the Random Forest Regressor from scikit-learn.
  + **from sklearn.metrics import mean\_squared\_error, r2\_score**: Imports evaluation metrics for regression models from scikit-learn.
  + **from sklearn.preprocessing import OneHotEncoder**: Imports the One-Hot Encoder for categorical variable encoding from scikit-learn.

CLASS

* MovieRevenuePredictionApp: CLASS name
* **Machine Learning Model Loading:**
  + Assigns the pre-trained machine learning model (model) to the class attribute self.model.
* **Analyze Lead Star Method:**
  + Retrieves the selected lead star from the dropdown.
  + Queries the dataset to get the number of movies done by the selected lead star.
  + Displays a message box with information about the lead star's movie count.
  + Calls the plot\_lead\_star\_graph method to visualize lead star vs. revenue.
* **Plot Lead Star Graph Method:** Creates a scatter plot using Matplotlib to visualize revenue vs. movie name for the selected lead star.
* **Calculate Revenue Method:**
  + Retrieves the selected movie from the dropdown.
  + Checks if the selected movie exists in the dataset.
  + Displays a message box with information about the selected movie's revenue, genre, and number of screens.
* **Revenue Method:**
* Converts the selected number of screens to an integer.
* Checks if the selected number of screens exists in the dataset.
* Retrieves actual and predicted revenue based on the selected number of screens.
* Displays a message box with details about the actual and predicted revenue, as well as the names of movies associated with the selected number of screens.
* **Display Graph Window Method:**
  + Creates a new Tkinter window (graph\_window) for displaying the revenue graph.
  + Sets the window title and background color.
  + Embeds the Matplotlib figure (fig) in the new window using FigureCanvasTkAgg.

To import all the modules and libraries one should install it first through the terminal

* + pip install numpy
* pip install pandas
* pip install matplotlib
* pip install tk
* pip install tabulate
* pip install -U scikit-learn
* pip install jupyter in the terminal of vscode

**HARDWARE AND SOFTWARE REQUIREMENTS**

**HARDWARE REQUIREMENTS:**

|  |  |
| --- | --- |
| **Hardware Tools** | **Minimum Requirements** |
| OS Name | Microsoft Windows 10 |
| System Type | x64 based PC |
| Processor | Intel(R) Core i5 |
| RAM | 8GB |
| Hardware | 10GB |

**­**

**SOFTWARE REQUIREMENTS:**

|  |  |
| --- | --- |
| **SOFTWARE TOOLS** | **MINIMUM REQUIREMENTS** |
| Platform | Windows, MacOs or Linux |
| Operating System | Windows, MacOs or Linux |
| Technology | Python |
| Technology version | Python 3.12.0 |
| IDE | VsCode |
| Data Analysis & Visualization | Jupyter Notebook |

**CODING**

predictingMovieBoxOfficeRevenue.ipynb

# importing the modules and library in python

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from tkinter import \*

from tkinter import ttk, messagebox

from tabulate import tabulate

from sklearn.compose import ColumnTransformer

from sklearn.pipeline import Pipeline

from matplotlib.backends.backend\_tkagg import FigureCanvasTkAgg

from sklearn.model\_selection import train\_test\_split

from sklearn.ensemble import RandomForestRegressor

from sklearn.metrics import mean\_squared\_error, r2\_score

from sklearn.preprocessing import OneHotEncoder

# Loading movie dataset

# Dataset loading

df = pd.read\_csv('movie\_dataset.csv')

# Data Preprocessing

# Data Preprocessing

selected\_features = ['Movie Name','Release Period','Whether Remake','Whether Franchise','Genre','New Actor','New Director',

                     'New Music Director','Lead Star','Director','Music Director','Number of Screens','Revenue(INR)',

                     'Budget(INR)']

# Remove leading/trailing spaces from column names

df.columns = df.columns.str.strip()

# Check for missing values in the selected columns

print(df[selected\_features].isnull().sum())

# Select the specified columns

df = df[selected\_features]

# Handling missing values (if any)

df.dropna(inplace=True)

# Shape of Movie Dataset

# Print the shape of the entire dataset

# Print the number of rows and columns in the dataset

print("Number of rows:", df.shape[0])

print("Number of columns:", df.shape[1])

# Print the first 10 rows of the dataset

# Print the first 10 rows of the dataset in a tabular format

print(tabulate(df.head(10), headers='keys', tablefmt='pretty'))

# Printing information of dataset such as Datatype and null/missing values

# Print data types and non-null counts for each column

print(df.info())

# Check for null values in the DataFrame

print(df.isnull().sum())

# Splitting the dataset into training and testing

# Train-Test Split

X = df.drop('Revenue(INR)', axis=1)

y = df['Revenue(INR)']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

# Shape of training and testing dataset of movie dataset

# Print the shape of the training and testing data

print("Total Data - ", X.shape)

print("Training Data - X shape:", X\_train.shape)

print("Testing Data - X shape:", X\_test.shape)

# Training the model RandomForestRegressor and plotting the graph between Actual and Predicted revenue

# Convert all column names to strings

X\_train.columns = X\_train.columns.astype(str)

X\_test.columns = X\_test.columns.astype(str)

# Create a column transformer for one-hot encoding

categorical\_columns = ['Movie Name', 'Release Period', 'Whether Remake', 'Whether Franchise', 'Genre', 'New Actor', 'New Director',

                         'New Music Director', 'Lead Star', 'Director', 'Music Director']

preprocessor = ColumnTransformer(

    transformers=[

        ('cat', OneHotEncoder(drop='first', sparse=False, handle\_unknown='ignore'), categorical\_columns)

    ],

    remainder='passthrough'

)

# Create a pipeline with the column transformer and random forest regressor

model = Pipeline([

    ('preprocessor', preprocessor),

    ('regressor', RandomForestRegressor(random\_state=42))

])

# Fit the model

model.fit(X\_train, y\_train)

# Predictions

y\_train\_pred = model.predict(X\_train)

y\_test\_pred = model.predict(X\_test)

# Plotting Actual vs. Predicted Revenue

plt.figure(figsize=(10, 5))  # Set the figure size

plt.subplot(1, 2, 1)  # Create a subplot (1 row, 2 columns, position 1)

plt.title("Actual Revenue(INR) vs Predicted Revenue(INR)")

plt.plot(y\_test.index, y\_test.values, label='Actual Revenue(INR)', marker='o', color='green')

plt.plot(y\_test.index, y\_test\_pred, label='Predicted Revenue(INR)', marker='o', color='magenta')

plt.xlabel("Index")

plt.ylabel("Revenue(INR)")

plt.legend()

# Assuming X\_test is the testing data and 'Budget(INR)' is a feature in X\_test

budget\_values = X\_test['Budget(INR)']

# Plotting Revenue vs Budget

plt.subplot(1, 2, 2)  # Create a subplot (1 row, 2 columns, position 2)

plt.title("Revenue(INR) vs Budget(INR)")

plt.plot(y\_test.index, y\_test\_pred, label='Revenue(INR)', marker='o', color='red')

plt.plot(y\_test.index, budget\_values, label='Budget(INR)', marker='o', color='blue')

plt.xlabel("Index")

plt.ylabel("Revenue(INR)")

plt.legend()

plt.tight\_layout()  # Adjust layout for better spacing

plt.show()

# Actual Revenue: This line represents the actual revenue values from your testing dataset.

# Predicted Revenue: This line represents the revenue values predicted by your model for the testing dataset.

# Budget: This line represents the 'Budget(INR)' values from your testing dataset.

# Index: In this context, the index refers to the row number or index of the data points in your testing dataset. It's used to plot the values against the corresponding indices.

# Accuracy Calculation

# Assuming y\_test and y\_test\_pred are your actual and predicted values

r2 = r2\_score(y\_test, y\_test\_pred)

# Convert R² to percentage

accuracy\_percentage = r2 \* 100

print(f"R-squared (Accuracy) in Percentage: {accuracy\_percentage:.2f}%")

# Graph of Genre vs Revenue(INR)

# Group by Genre and calculate the total revenue for each genre

genre\_revenue\_sum = df.groupby('Genre')['Revenue(INR)'].sum().sort\_values(ascending=False)

# Choose darker colors for the pie chart

colors = plt.cm.Dark2.colors

# Plotting the pie chart

plt.figure(figsize=(12, 8))

patches, texts, autotexts = plt.pie(genre\_revenue\_sum, labels=genre\_revenue\_sum.index, autopct='%1.1f%%', startangle=140, colors=colors)

# Create legend entries with genre and revenue information

legend\_labels = [f"{genre}: {revenue:,.2f} INR" for genre, revenue in zip(genre\_revenue\_sum.index, genre\_revenue\_sum)]

# Add legend with custom legend labels

plt.legend(legend\_labels, loc="center left", bbox\_to\_anchor=(1, 0, 0.5, 1))

plt.title('Distribution of Revenue(INR) by Genre')

plt.show()

# GUI Window

class MovieRevenuePredictionApp:

    def \_\_init\_\_(self, root):

        self.root = root

        self.root.title("Movie Revenue Prediction")

        self.root.configure(bg="#092635")

        # Create a frame at the top

        title\_frame = Frame(root, bg="#092635")

        title\_frame.grid(row=0, column=0, columnspan=3, sticky="ew")

        # Add the title to the frame

        title\_label = Label(title\_frame, text="Movie Revenue Prediciton", font=("Times New Roman", 20,"bold"), fg="white", bg="#092635")

        title\_label.pack(pady=10)

        # GUI component

        self.lead\_star\_var = StringVar()

        self.selected\_movie\_var = StringVar()

        self.num\_screens\_var = StringVar()

        # Load the machine learning model

        self.model = model  # Use the previously defined model

        # Lead Star Analysis Section

        Label(root, text="Select Lead Star:", font=("Times New Roman",15,"bold"), width=20,bg="#1B4242",fg="white").grid(row=1, column=0, padx=10, pady=10)

        self.lead\_star\_dropdown = ttk.Combobox(root, textvariable=self.lead\_star\_var,

                                               values=df['Lead Star'].unique().tolist(), width=30)

        self.lead\_star\_dropdown.grid(row=1, column=1, padx=10, pady=10)

        Button(root, text="Analyze Lead Star", command=self.analyze\_lead\_star, width=20,font=("Times New Roman",15,"bold"), bg="#1B4242",fg="white").grid(row=1, column=2, padx=10, pady=10)

        # Movie Prediction Section

        Label(root, text="Select Movie:",font=("Times New Roman",15,"bold"),width=20, bg="#5C8374",fg="black").grid(row=2, column=0, padx=10, pady=10)

        self.movie\_dropdown = ttk.Combobox(root, textvariable=self.selected\_movie\_var,

                                           values=df['Movie Name'].unique().tolist(),width=30)

        self.movie\_dropdown.grid(row=2, column=1, padx=10, pady=10)

        Button(root, text="Calculate Revenue", command=self.calculate\_revenue, width=20,font=("Times New Roman",15,"bold"),bg="#5C8374",fg="black").grid(row=2, column=2, padx=10, pady=10)

        # Number of Screens Section

        Label(root, text="Number of Screens:", font=("Times New Roman",15,"bold"), width=20,bg="#9EC8B9",fg="black").grid(row=3, column=0, padx=10, pady=10)

        self.num\_screens\_dropdown = ttk.Combobox(root, textvariable=self.num\_screens\_var,

                                                  values=df['Number of Screens'].unique().tolist(),width=30)

        self.num\_screens\_dropdown.grid(row=3, column=1, padx=10, pady=10)

        Button(root, text="Predict Revenue", command=self.predict\_revenue,width=20,font=("Times New Roman",15,"bold"), bg="#9EC8B9",fg="black").grid(row=3, column=2, padx=10, pady=10)

    def analyze\_lead\_star(self):

        selected\_lead\_star = self.lead\_star\_var.get()

        # Query your dataset to get the number of movies done by the selected lead star

        num\_movies = df[df['Lead Star'] == selected\_lead\_star].shape[0]

        messagebox.showinfo("Lead Star Analysis", f"{selected\_lead\_star} has done {num\_movies} movies.")

        # Plot the lead star vs. revenue scatter plot

        self.plot\_lead\_star\_graph(selected\_lead\_star)

    def plot\_lead\_star\_graph(self, lead\_star):

        # Filter dataset for the selected lead star

        lead\_star\_df = df[df['Lead Star'] == lead\_star]

        # Plot the graph

        fig, ax = plt.subplots(figsize=(10, 6))

        # Plot the scatter plot for Revenue vs. Lead Star

        scatter = ax.scatter(lead\_star\_df['Movie Name'], lead\_star\_df['Revenue(INR)'], label='Revenue', color='green')

        # Annotate each marker with the actual revenue value on the right side

        for i, (x, y) in enumerate(zip(lead\_star\_df['Movie Name'], lead\_star\_df['Revenue(INR)'])):

            ax.annotate(f'{y:,.2f}', (x, y), textcoords="offset points", xytext=(5, 0), ha='left', color='black',

                        fontsize=10)

        ax.set\_xlabel('Movie Name')

        ax.set\_ylabel('Revenue(INR)')

        ax.set\_title(f'Revenue Generated by {lead\_star}')

        ax.legend()

        # Embed the plot in Tkinter window

        self.display\_graph\_window(fig)

    def calculate\_revenue(self):

        selected\_movie = self.selected\_movie\_var.get()

        # Check if the selected movie exists in the dataset

        if selected\_movie in df['Movie Name'].values:

            # Query your dataset to get revenue and genre of the selected movie

            movie\_data = df[df['Movie Name'] == selected\_movie].iloc[0]

            revenue = movie\_data['Revenue(INR)']

            genre = movie\_data['Genre']

            screenNumber = movie\_data['Number of Screens']

            messagebox.showinfo("Revenue Prediction", f"Movie Name: {selected\_movie}\nRevenue Generated: {revenue:,.2f} INR \nGenre: {genre}\nNumber of Screens: {screenNumber}")

        else:

            messagebox.showwarning("Movie Not Found", f"The selected movie '{selected\_movie}' was not found in the dataset.")

    def predict\_revenue(self):

        selected\_num\_screens = int(self.num\_screens\_var.get())  # Convert to int

        # Check if the selected number of screens exists in the dataset

        if selected\_num\_screens in df['Number of Screens'].values:

            # Query your dataset to get the actual and predicted revenue based on the selected number of screens

            revenue\_data = df[df['Number of Screens'] == selected\_num\_screens]

            actual\_revenue = revenue\_data['Revenue(INR)'].sum()

            # Prepare features for prediction

            features\_for\_prediction = revenue\_data.drop('Revenue(INR)', axis=1)

            # Use the pre-trained model to make predictions

            predicted\_revenue = self.model.predict(features\_for\_prediction)

            # Get the names of movies associated with the selected number of screens

            movie\_names = revenue\_data['Movie Name'].tolist()

            # Display messagebox with details

            messagebox.showinfo("Revenue Calculation",f"Actual Revenue: {actual\_revenue:,.2f} INR\nPredicted Revenue: {predicted\_revenue.sum():,.2f} INR\nMovies: {', '.join(movie\_names)}")

        else:

            messagebox.showwarning("Number of Screens Not Found",f"The selected number of screens '{selected\_num\_screens}' was not found in the dataset.")

    def display\_graph\_window(self, fig):

        # Create a new window for the graph

        graph\_window = Toplevel(self.root)

        graph\_window.title("Revenue Graph")

        graph\_window.configure(bg="lightpink")

        # Embed the plot in the new window

        canvas = FigureCanvasTkAgg(fig, master=graph\_window)

        canvas\_widget = canvas.get\_tk\_widget()

        canvas\_widget.pack()

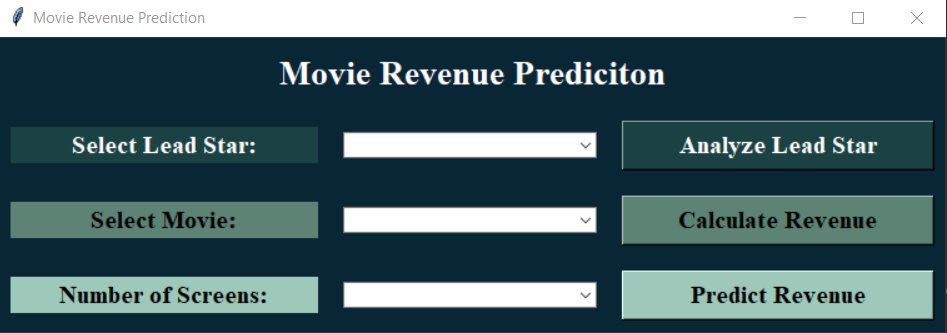
root = Tk()

movieApp = MovieRevenuePredictionApp(root)

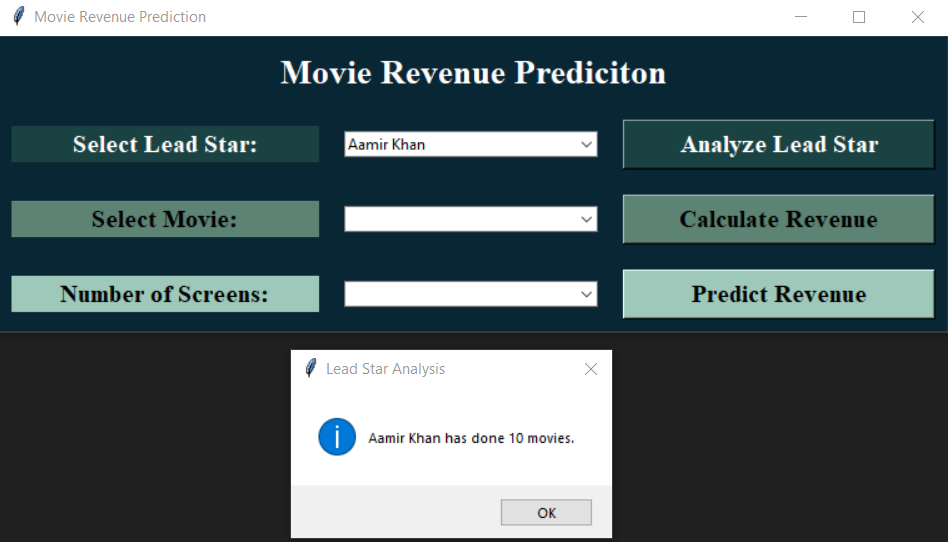
root.mainloop()

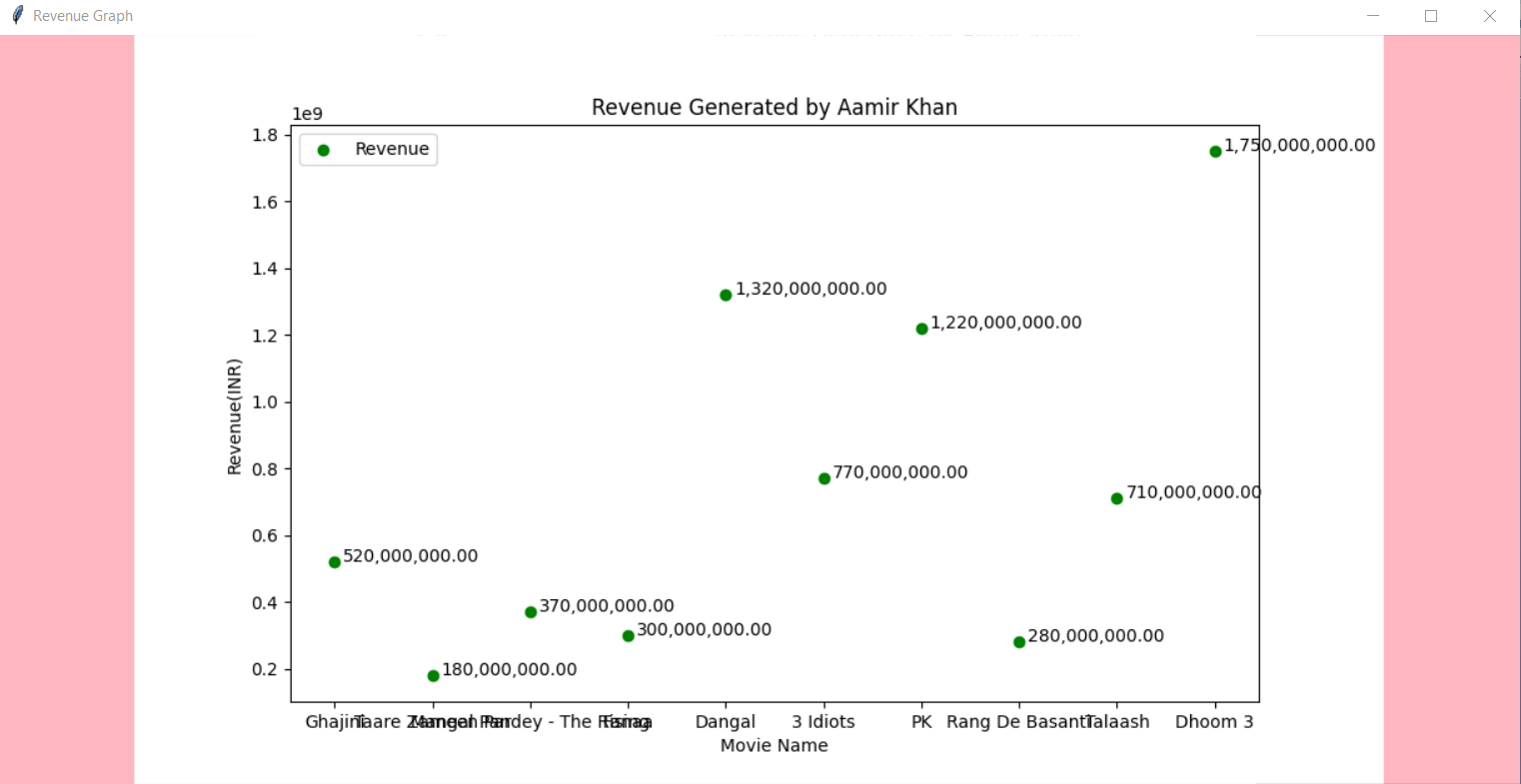
**OUTPUT SCREENSHOTS**

* Main window of GUI whose title is Movie Revenue Prediction.

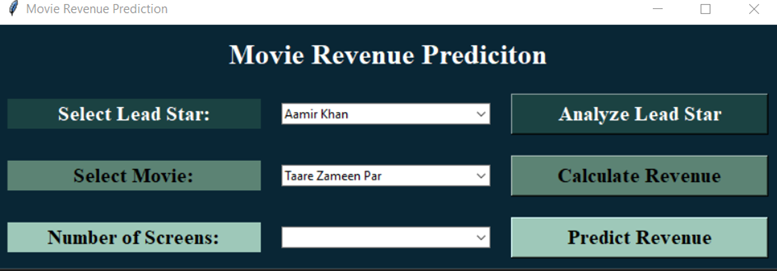
****

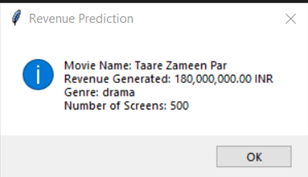
* User will select the lead star from the dropdown and after clicking on Analyze Lead Star button a messagebox will appear that the Lead Star has done this number of movies and after clicking on OK button of message box user will see the Graph between Revenue(INR) and the movie done by the Lead star.

****

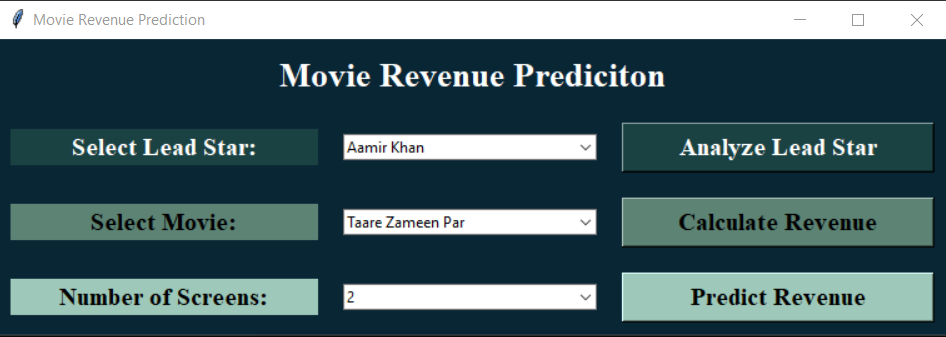
****

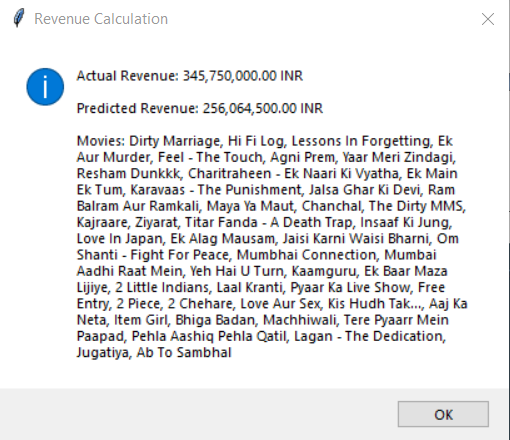
* User will select the movie name from the dropdown and after selecting user will click on the Calculate Revenue button, a messagebox will appear where the Movie Name, Revenue Generated by the movie, Genre of the movie as well as Number of Screens of the movie will be shown.

****

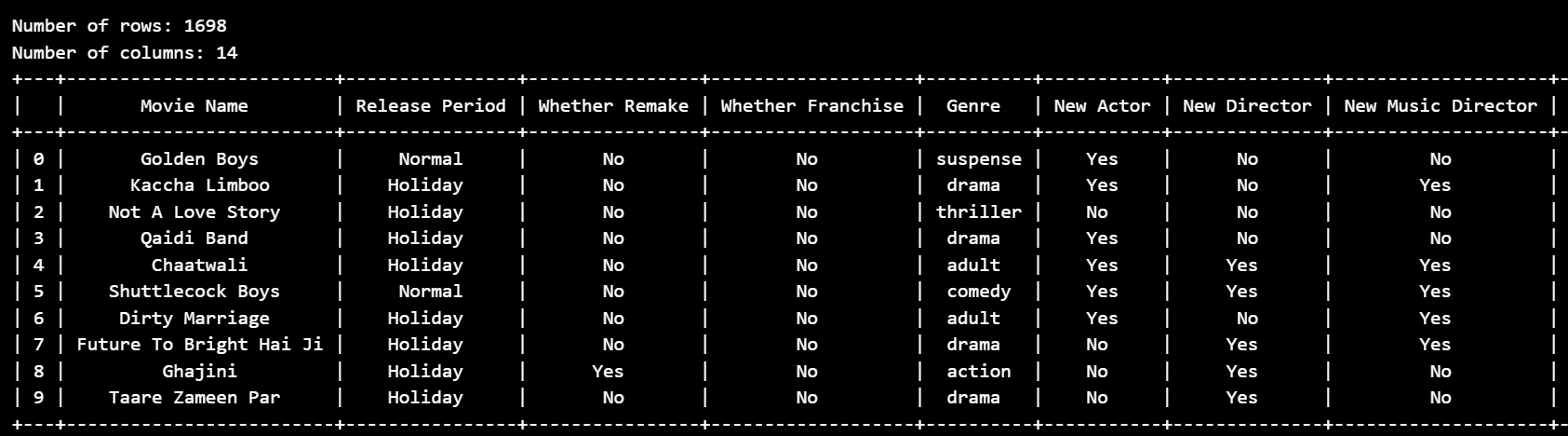
****

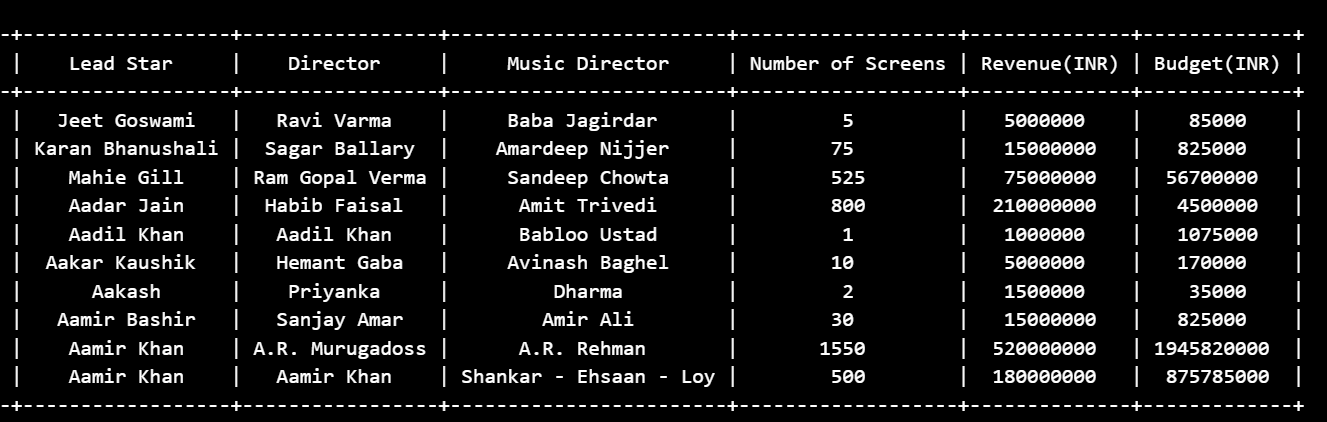
* User will select the Number of Screens from the dropdown then after clicking on Predict Revenue button the user will be able to see the Actual Revenue of movie as well Predicted Revenue of movie and A list of movies which is having the same Number of Screens. The model RandomForestRegressor is used to predict the revenue of the movie which is trained on the movie dataset.

****

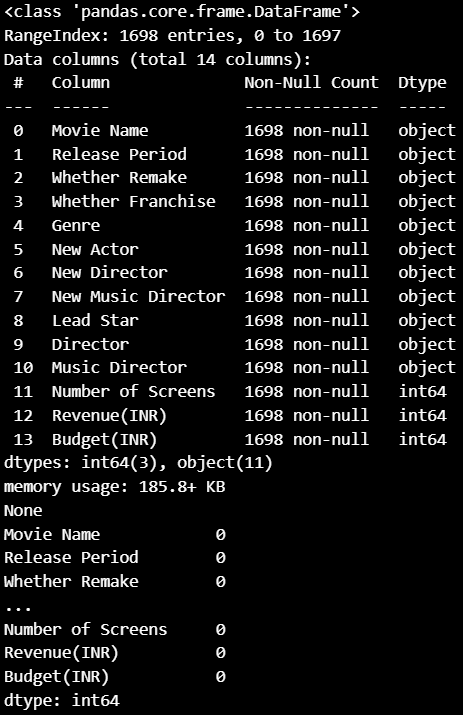
****

* JUPYTER NOTEBOOK OUTPUT SCREENSHOTS
* Shape of movie dataset and showing 10 rows of movie dataset

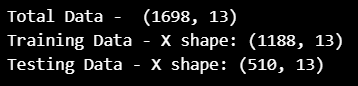
****

****

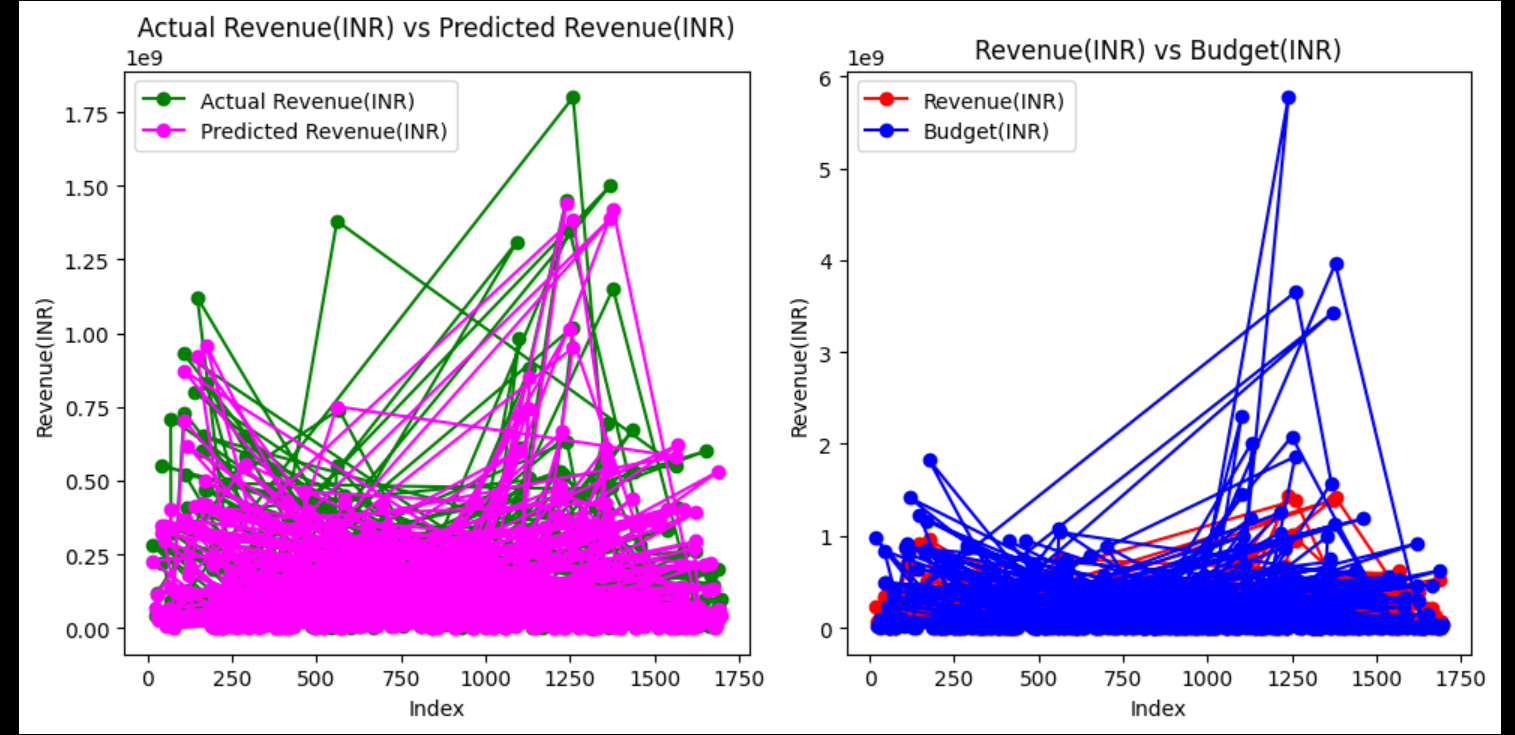
* Printing information of dataset such as Datatype and null/missing values

****

* Shape(Rows and Columns) of training, testing as well as total dataset.

****

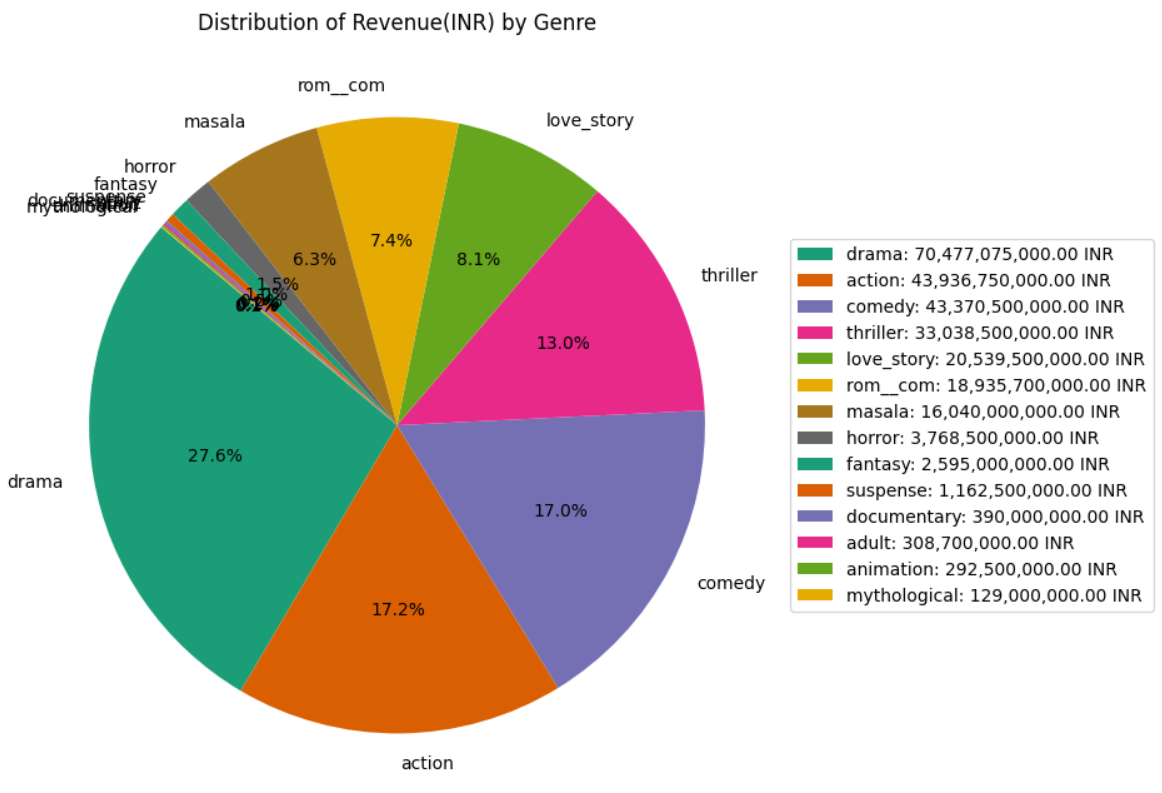
* Graph between Actual and Predicted Revenue

****

* R-squared Accuracy Calculation

****

* Graph of Genre vs Revenue(INR)

****

**FUTURE SCOPE**

The project's future scope includes:

* Incorporating real-time data for more accurate predictions.
* Expanding the model to consider additional features and external factors.
* Integrating the application with online databases for live updates on movie-related information.
* Collaborating with industry professionals to enhance the model's accuracy and relevance.

**CONCLUSION**

Predicting movie box office revenue is a multidimensional challenge that requires a combination of datasets, machine learning, and user interface design. The RandomForestRegressor model, integrated with a Tkinter GUI, provides a practical and accessible solution for stakeholders in the film industry. Continuous refinement, feedback incorporation, and adaptation to changing industry dynamics will contribute to the model's ongoing relevance and effectiveness. Understanding the concepts of ML Model, tkinter GUI and how to develop it in PYTHON . The application allows users to select a lead star, movie, or number of screens to analyze or predict revenue. Lead star analysis includes a scatter plot that visualizes the relationship between the lead star and movie revenue. The application uses a pre-trained machine learning model (presumably for regression) to predict movie revenue based on selected features. Features for prediction are extracted from the dataset, and the model is used to make revenue predictions.

**REFERENCES AND BIBLIOGRAPHY**

1. Python Documentation: (<https://www.python.org/downloads/>)
2. Vscode: (<https://code.visualstudio.com/download>)
3. Tkinter library in Python: (<https://docs.python.org/3/library/tkinter.html>)
4. Jupyter Notebook: (<https://jupyter.org/>)
5. Dataset Link: (<https://www.kaggle.com/datasets/rishidamarla/bollywood-movies-dataset>)
6. SklearnRandomForestRegressor:(https://scikitlearn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html)
7. Numpy: (<https://numpy.org/>)
8. Pandas: (<https://pandas.pydata.org/>)
9. Matplotlib.pyplot: (<https://matplotlib.org/3.5.3/api/_as_gen/matplotlib.pyplot.html>)
10. Tabulate: (<https://pypi.org/project/tabulate/>)
11. ColumnTransformer:(<https://scikitlearn.org/stable/modules/generated/sklearn.compose.ColumnTransformer.html>)
12. Pipeline:(<https://scikit-learn.org/stable/modules/generated/sklearn.pipeline.Pipeline.html>)
13. Train test split:(<https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html>)
14. R2 score:(<https://scikit-learn.org/stable/modules/generated/sklearn.metrics.r2_score.html>)
15. Onehotencoder: (<https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OneHotEncoder.html>)