Infosys Internship 4.0 Project Documentation

Project Documentation:

TMDB BOX OFFICE PREDICTION

By
Shreyas Yaduvanshi
Infosys Springboard Project Intern

May - July 2024

CONTENTS

1. Introduction

1.1 Project Overview

This project endeavours to predict box office revenue for movies using data sourced from TMDB (The Movie Database). By meticulously collecting information such as budget, genres, release dates, cast, and crew, we preprocess this data and engineer features that could potentially influence a movie's financial success.

The culmination of this process is a robust predictive model capable of estimating box office revenue based on a multitude of factors. This model serves as a valuable tool for stakeholders in the film industry, enabling them to make informed decisions regarding resource allocation, marketing strategies, and investment opportunities. Moreover, it provides movie enthusiasts with insights into which films might achieve significant success, aiding them in their viewing choices.

1.2 Objectives

1. **Data Collection and Preprocessing:**
    - Gather comprehensive movie data from TMDB.
    - Implement preprocessing techniques to clean and structure the data.
2. **Predictive Model Development:**
    - Develop and train machine learning models to estimate box office revenue.
3. **Model Evaluation:**
    - Assess the performance of the predictive models using a variety of metrics.
4. **Model Deployment:**
    - Deploy the model to predict box office revenue for new and upcoming movies.
5. **Documentation and User Guide:**
    - Document the entire process and create a user guide for stakeholders to utilize the model effectively.

1.3 Significance

The capacity to predict box office revenue holds profound importance for a diverse array of stakeholders within the film industry:

- **Producers, Investors, and Marketers:**
    - These stakeholders can leverage the model to make well-informed decisions about resource allocation, marketing strategies, and investment opportunities, thereby optimizing their financial outcomes and strategic planning.
    - By understanding potential revenue, they can better negotiate distribution deals, set realistic financial goals, and allocate marketing budgets more effectively.
    - The predictive model can assist in identifying lucrative genres, optimal release windows, and target demographics, ensuring that resources are channelled towards the most promising projects.
- **Movie Enthusiasts:**
    - Enthusiasts can utilize the model's predictions to gauge the potential success of upcoming movies, thus enhancing their movie-watching decisions based on likely box office performance.
    - By predicting which movies are likely to be blockbusters, fans can prioritize their viewing schedules and make informed choices about which films to watch in theatres versus waiting for digital release.
    - Additionally, the model can foster a deeper understanding and appreciation of the factors that contribute to a movie's success, enriching the overall viewing experience.
- **Film Distributors and Exhibitors:**
    - Predictive insights can help distributors and theatre owners decide which movies to prioritize for wider releases and extended runs, thereby maximizing occupancy and revenue.
    - The model can inform marketing strategies and promotional campaigns, ensuring that the most promising films receive adequate exposure and support.

- **Academics and Researchers:**
    - Scholars studying film economics and market trends can use the model to conduct in-depth analyses of the factors influencing box office performance.
    - The model's predictions can serve as a valuable dataset for academic research, contributing to the broader understanding of the film industry's dynamics.
- **Content Creators and Scriptwriters:**
    - Content creators can gain insights into the types of stories and genres that are likely to resonate with audiences, guiding their creative processes.
    - Scriptwriters can use the model's outputs to refine their narratives and pitches, aligning their projects with market trends and audience preferences.
- **Streaming Services and OTT Platforms:**
    - As the lines between theatrical and digital releases blur, streaming services can use predictive models to identify which movies to acquire for their platforms.
    - The model can aid in programming decisions, ensuring that content with high potential viewership is prioritized for acquisition and promotion.

By harnessing machine learning techniques and comprehensive data from TMDB, this project aspires to deliver precise and actionable insights into the financial performance of movies, thereby supporting both industry stakeholders and the general public in their respective endeavours.

2. Project Scope

2.1 Included

- **Data Collection from TMDB API:**
    - Comprehensive retrieval of movie data, including attributes such as title, budget, genres, release date, cast, crew, and box office revenue.
- **Data Pre-Processing:**
    - Detailed cleaning and transformation of the collected data to ensure it is suitable for model training. This includes handling missing values, outliers, and categorical variables.
- **Development of Predictive Models:**
    - Creation and training of robust machine learning models aimed at predicting box office revenue, utilizing various algorithms and techniques to optimize performance.
- **Model Evaluation and Validation:**
    - Rigorous assessment of model accuracy and reliability through techniques such as cross-validation and evaluation metrics (e.g., MAE, MSE, R-squared).
- **Deployment of the Predictive Model:**
    - Implementation of the trained model within a web-based application using frameworks like Flask and Streamlit, ensuring accessibility for end-users.
- **Documentation and User Guide Preparation:**
    - Comprehensive documentation detailing the system architecture, user guides, technical specifications, and deployment instructions to facilitate easy understanding and use of the application.

2.2 Excluded

- **Real-Time Data Updates from TMDB:**
    - The project does not include automatic, real-time updates of movie data from TMDB API. Data updates will need to be manually initiated.

- **Integration with External Systems for Data Input/Output:**
    - There is no provision for integrating the application with external systems or databases for importing or exporting data.
- **Extensive User Interface Development:**
    - The focus is on a functional prediction input/output interface rather than an elaborate user interface. Advanced UI features and extensive design elements are beyond the project's scope.

2.3 Limitations

- **Variability in Predictive Accuracy:**
    - Predictive accuracy may vary across different genres, budgets, and release periods. The model might exhibit higher precision for certain types of movies, reflecting inherent biases in the training data.
- **Absence of Real-Time Data Updates:**
    - Automatic, real-time updates of movie data from the TMDB API were not incorporated into the project. Therefore, data updates must be manually initiated, which risks the use of outdated information for predictions.
- **Dynamic Market Influences:**
    - The model does not account for dynamic market variables such as competitor releases, promotional campaigns, or global events that could significantly influence box office performance.
- **Security and Privacy Measures:**
    - While basic security protocols were implemented to ensure data security and privacy, the project's scope and timeline did not allow for extensive security measures.
- **Data Quality and Completeness:**
    - The precision of our predictions is intrinsically tied to the quality and completeness of data sourced from TMDB and other repositories. Any discrepancies or gaps in this data could potentially undermine the model's efficacy.

3. Requirements

3.1 Functional Requirements

1. Data Collection

- **Objective:** Efficiently gather comprehensive movie data from TMDB (The Movie Database).
    - **Collect Essential Data:** Extract key attributes including movie title, budget, genres, release date, cast, crew, and box office revenue.
    - **Data Completeness:** Fetch additional movie details through API calls if any data is missing or inconsistent, ensuring all required information is available for analysis.

2. Data Pre-Processing

- **Objective:** Prepare raw data for effective model training through meticulous data cleaning and transformation.
    - **Handle Missing Data:** Identify and address missing values through data imputation techniques or by fetching missing information from external sources.
    - **Manage Outliers:** Detect and mitigate outliers to ensure data integrity and model accuracy.
    - **Data Transformation:** Convert categorical variables into numerical forms using methods such as one-hot encoding and label encoding to facilitate model processing.
    - **Exploratory Data Analysis (EDA):** Explore data distribution, visualize relationships among variables, and perform statistical analyses to uncover insights.

3. Model Development

- **Objective:** Construct a robust predictive model to estimate box office revenue based on various movie features.

- **Details:**
    - **Define Model Architecture:** Utilize PyCaret to evaluate and select the most effective regression model among options such as Linear Regression, Decision Tree Regression, or Neural Networks.
    - **Feature Selection:** Determine relevant input features and define the output target, which is the box office revenue.
    - **Model Training:** Train the chosen model on preprocessed data and perform hyperparameter tuning to optimize performance.
    - **Model Evaluation:** Assess the model's performance using metrics such as Mean Absolute Error (MAE), Mean Squared Error (MSE), and R-squared to ensure it meets accuracy and reliability standards.

4. Prediction & Cross-Validation

- **Objective:** Validate the model's performance and use it to make predictions on new data.
  - **Details:**
    - **Make Predictions:** Use the trained model to predict box office revenue for new or unseen movie data.
    - **Cross-Validation:** Implement cross-validation techniques such as k-fold cross-validation to evaluate the model's robustness and generalization capability across different datasets.

5. Deployment

- **Objective:** Deploy the model and provide a user interface for making predictions.
  - **Details:**
    - **Deployment Process:** Deploy the trained model using Flask and Streamlit to create a web-based application.

            - **Deployment Instructions:** Provide comprehensive instructions for deploying the application on cloud platforms (e.g., Heroku, AWS, Azure, GCP) and ensure the application's accessibility and functionality.

6. Documentation

- **Objective:** Develop thorough documentation for system architecture, user interaction, and technical details.
    - **Details:**
        - **System Architecture:** Document the overall architecture including high-level components and their interactions.
        - **User Guides:** Prepare detailed user guides for interacting with the application and making predictions.
        - **Technical Specifications:** Provide technical specifications including design decisions, code structure, and API details.

3.2 Non-Functional Requirements

1. Performance

- **Objective:** Ensure the application delivers high performance and is scalable to handle varying loads.
    - **Details:**
        - **Scalability:** Design the model and application to efficiently handle increasing data volumes and user interactions.
        - **Efficiency:** Optimize model training and prediction processes for quick execution and minimal resource consumption.

2. Security

- **Objective:** Protect data privacy and application security.
    - **Details:**

        - **Data Security:** Implement secure methods for data handling, storage, and transmission.
        - **Access Control:** Ensure that sensitive data and features are protected from unauthorized access.

3. Usability

- **Objective:** Provide a seamless and intuitive user experience.
    - **Details:**
        - **User Interface:** Design a user-friendly interface for stakeholders to input movie data and receive predictions.
        - **Accessibility:** Ensure the application is easy to navigate and understand, with clear instructions and feedback.

3.3 User Stories

1. As a Data Analyst

- **Story:** "I want to preprocess the movie data to prepare it for model training."
    - **Acceptance Criteria:** I can perform data cleaning tasks such as handling missing values and outliers and transforming data for model readiness.

2. As a Data Scientist

- **Story:** "I want to develop a predictive model to estimate box office revenue."
    - **Acceptance Criteria:** I can define and evaluate different model architectures, select the best model, and train it to predict movie revenue accurately.

3. As a Stakeholder

- **Story:** "I want to use the model to predict the potential success of new movies."
    - **Acceptance Criteria:** I can input movie details into the application and receive accurate predictions about the expected box office revenue.

4. Technical Stack

4.1 Programming Languages

- **Python**: Used for data manipulation, model development, and deployment.

4.2 Frameworks/Libraries

- **Pandas**: Data manipulation and analysis.
- **NumPy**: Numerical computations.
- **Scikit-learn**: Machine learning model development and evaluation.
- **Matplotlib**: Data visualization.
- **Seaborn**: Statistical data visualization.
- **Flask**: Web framework for deploying the model.
- **Streamlit**: Framework for creating a user interface.

4.3 Database

None: Data collected from TMDB API and processed in CSV format.

4.4 Tools/Platforms

- **Jupyter Notebook**: Interactive development environment.
- **Google Colab**: Cloud-based Jupyter Notebook environment.
- **GitHub**: Version control and code management.
- **Google Drive**: Storage for processed data and outputs.
- **Vscode**: UI code [Streamlit & Flask code] is written in this IDE.

5. Architecture/Design

5.1 System Architecture Overview

The TMDB Movie Revenue Prediction App is structured into several high-level components that work together to provide users with accurate movie revenue predictions. The architecture follows a modular design approach, emphasizing separation of concerns and scalability. The primary components of the system architecture are as follows:

1. Data Collection

Description:
The Data Collection component is responsible for fetching movie data from The Movie Database (TMDB) API. It retrieves essential movie details such as title, release date, budget, revenue, and other attributes required for revenue prediction.

Components:

- **TMDB API Integration**: The requests library is used to send HTTP requests to the TMDB API and receive movie data.
- **Data Fetching Module**: This module handles API requests, manages API keys, and processes the responses to extract relevant data.

2. Data Preprocessing

Description:
Data Preprocessing prepares the raw data for analysis and modeling by performing tasks such as data cleaning, feature extraction, and data transformation.

Components:

- **Data Cleaning**: Handles missing values, removes duplicates, and ensures data consistency.

- **Feature Engineering**: Extracts features like movie genre, production company, and release month. Converts categorical features into numerical format using techniques like label encoding.
- **Data Transformation**: Scales features to standardize data and prepare it for model training.

3. Model Training and Selection

Description:
This component is responsible for selecting, training, and evaluating the regression model used for predicting box office revenue.

Components:

- **Model Selection**: Uses PyCaret for model comparison and selection.
- **Model Training**: Trains the selected model (ElasticNet) on the preprocessed data.
- **Model Evaluation**: Assesses model performance using metrics like MAE, MSE, and R^2.
- **Model Serialization**: Saves the trained model as a .pkl file for future use.

4. Backend Service

Description:
The Backend Service manages HTTP requests from the user interface and interacts with the predictive model.

Components:

- **Flask API**: Handles incoming requests, processes input data, and sends requests to the model.
- **Model Integration**: Loads the trained model and uses it to generate predictions based on user inputs.

5. Frontend Interface

Description:
The Frontend Interface allows users to interact with the application and view predictions.

Components:

- **Streamlit Application**: Provides a user-friendly web interface for users to input movie details and receive revenue predictions.
- **User Interaction**: Collects user input, displays results, and provides feedback.

6. Deployment

Description:
The Deployment component manages the application's availability and accessibility.

Components:

- **Deployment Platform**: Hosted on cloud platforms like Heroku or AWS for public access.
- **Deployment Automation**: Uses deployment scripts to set up the environment and launch the application.

5.2 Design Decisions and Patterns

Design Pattern: MVC (Model-View-Controller)

Explanation: The architecture of the TMDB Movie Revenue Prediction App follows the MVC design pattern, which separates the application into three interconnected components:

- **Model**: Represents the data and the business logic. In this application, it includes the Model Training and Selection component.
- **View**: The user interface that displays data and interacts with users. Here, it includes the Frontend Interface component.
- **Controller**: Manages user input and updates the Model and View. In this app, the Backend Service acts as the controller, processing user requests and interacting with the Model.

Advantages:

- **Separation of Concerns**: Ensures that different aspects of the application are managed independently, making the system easier to develop and maintain.
- **Scalability**: Allows for easy updates and modifications to individual components without affecting others.
- **Code Maintainability**: Enhances code readability and maintainability by clearly defining roles for different components.

Design Decisions:

- **Choice of PyCaret for Model Selection**: PyCaret was chosen for its ease of use in model comparison and selection. It provides automated tools for feature selection and model evaluation, which streamlined the development process.
- **Selection of Flask for Backend Development**: Flask was selected for its lightweight nature and flexibility, making it suitable for creating the RESTful API required for the backend service.
- **Use of Streamlit for Frontend Development**: Streamlit was chosen for its simplicity in creating interactive web applications, which allowed for rapid development of a user-friendly interface.

5.3 Trade-offs and Alternatives:

- **Trade-off: One-Hot Encoding vs. Feature Selection**
  - **Alternative Considered**: One-hot encoding for the placement_companies column would have created a large number of features, making the model complex and less efficient.
  - **Decision**: Opted for a more manageable feature set by selecting a subset of features, which simplified the feature engineering process and improved model efficiency.
- **Trade-off: Manual Testing vs. Automated Testing**
  - **Alternative Considered**: Automated testing frameworks could have been used for more extensive and rigorous testing.

- **Decision**: Due to time constraints, manual testing was performed to ensure functionality and reliability, with a focus on critical features and common issues.

5.4 Design Diagrams

1.Flowchart

- Data Collection → Data Pre-processing → Model Training → Model Selection → Model Evaluation → Model Storage → Deployment

2. Component Diagram

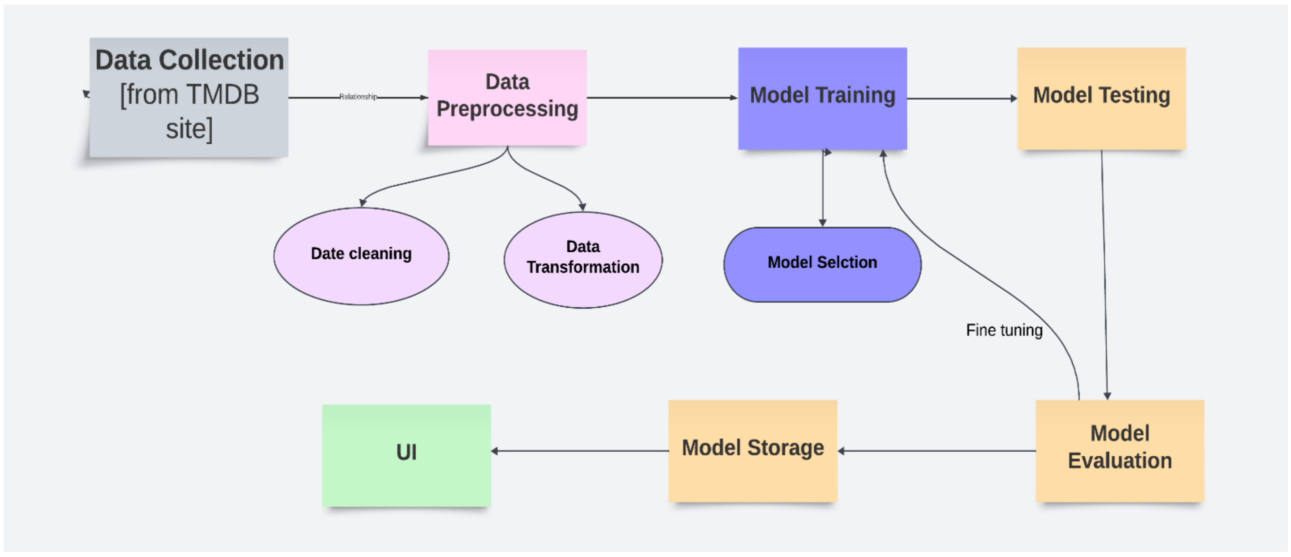


Figure 5.1: Detailed Component Diagram

The design and architecture of the TMDB Movie Revenue Prediction App were developed with a focus on modularity, maintainability, and scalability. By employing the MVC design pattern, utilizing robust technologies and frameworks, and making strategic design decisions, the project effectively met its objectives. The design decisions ensured a balance between functionality, performance, and user experience, while trade-offs and alternatives were carefully evaluated to achieve optimal results.

6. Development

6.1 Technologies and Frameworks

- **Python**

Python was selected as the primary programming language due to its robust ecosystem of libraries and frameworks for data science and machine learning. Its ease of use, extensive documentation, and active community support facilitate efficient data manipulation, model development, and application creation. Python's rich set of libraries such as Pandas for data manipulation, NumPy for numerical calculations, and Scikit-learn for machine learning, make it an ideal choice for the TMDB Box Office Prediction project.

- **Scikit-learn**

Scikit-learn was utilized for its comprehensive suite of machine learning algorithms and tools designed for predictive modeling. This library provided the necessary tools for model selection, feature engineering, and performance evaluation. It enabled us to implement various regression algorithms, perform hyperparameter tuning, and evaluate model performance through standardized metrics.

- **Flask and Streamlit**

Flask and Streamlit were employed to create a user-friendly web interface for interacting with the predictive model. Flask handled backend operations, including HTTP request management, while Streamlit was used to develop a straightforward, interactive frontend for users to input data and receive predictions.

- **Purpose of Flask and Streamlit:**
    - **Flask**: Developed the backend for handling user requests and interacting with the model, providing a simple yet flexible framework for building web applications.
    - **Streamlit**: Created an intuitive user interface that allows users to enter movie details and receive revenue predictions. Its simplicity and built-

in components facilitate rapid development of interactive web applications.

- **Merits of Flask & Streamlit**: Flask offers a lightweight and versatile web framework for creating APIs and handling server-side logic, while Streamlit enables quick and easy creation of interactive web applications focused on machine learning models.

6.2 Coding Standards and Best Practices

- **PEP 8 Coding Standards**

  The development process adhered to PEP 8, the style guide for Python code, which emphasizes writing readable and maintainable code.
- **Version Control with GitHub**

  GitHub was used for version control to manage code changes, collaborate with team members, and maintain project history. GitHub provided a robust platform for tracking revisions, handling code merges, and reviewing changes through pull requests.
- **Code Documentation**

  Code documentation was a key aspect of the development process, aimed at providing clear explanations of the codebase and its components. Documentation included comments, docstrings, and README files to support future maintenance and understanding of the project.

6.3 Challenges and Solutions

- **Data Quality**

Challenge: Ensuring high-quality data was a significant challenge due to the incomplete or inconsistent data received from the TMDB API. Inconsistent data could affect the accuracy of the predictive model.

Solution:

- **Data Validation and Retrieval**: Implemented validation checks to verify data completeness and accuracy. Used retry mechanisms to handle incomplete API responses and fetched missing data through additional API calls.
- **Cross-Referencing Data**: Utilized alternative data sources to cross-check and validate the data retrieved from TMDB, ensuring data integrity and completeness.
- **Data Imputation**: Applied data imputation techniques to handle missing values by replacing them with the mean or median values based on the context of the missing data.

❖ **Model Performance**

Challenge: Achieving optimal model performance required careful selection of algorithms and tuning of hyperparameters. Ensuring that the model was both accurate and efficient was a complex task.

Solution:

- **Feature Engineering**: Conducted extensive feature engineering to extract meaningful features from the data, improving the model's predictive capabilities.
- **Hyperparameter Tuning**: Used grid search and random search methods for hyperparameter optimization, which involved experimenting with different parameter settings to find the best model configuration.
- **Automated Model Selection**: Leveraged PyCaret for automated model selection and comparison, which provided insights into which algorithms performed best for predicting box office revenue.

❖ **Deployment**

Challenge: Ensuring a smooth and reliable deployment process for the web application presented several challenges. The deployment needed to be seamless, ensuring that the application was accessible to users.

Solution:

- **Environment Setup**: Created a requirements.txt file to manage dependencies, making it easy to replicate the environment across different machines.
- **Deployment Automation**: Developed a deployment script to automate the installation of dependencies and execution of the application. This script ensured that the environment was set up correctly and the application was launched without manual intervention.
- **Cloud Hosting**: Chose any cloud platform[Azure, AWS, etc] for hosting the application, benefiting of its ease of use and scalability. Ensured that the application was properly configured for the cloud environment and was accessible to users.

❖ **Feature Engineering Challenges**

Challenge: One-hot encoding the placement_companies column resulted in an unfeasible number of features due to the large number of unique companies, which would have led to high computational costs and complexity in feature selection.

Solution:

- **Feature Selection**: Instead of one-hot encoding, selected a subset of features to keep the dataframe manageable and efficient for the model.
- **Efficient Feature Selection**: Employed PyCaret’s automated feature selection capabilities to identify the most relevant features for the model, streamlining the feature engineering process and enhancing model performance.

6.4 Future Development Recommendations

To further enhance the TMDB Box Office Prediction App, several avenues for future development can be explored to improve its functionality, performance, and user experience.

- **Enhanced Data Collection**:
    - Expand the scope of data collection to include additional metrics such as marketing spend, audience reviews, and international box office

performance. This would provide a more comprehensive dataset for model training and lead to better predictive accuracy.

- **Advanced Model Algorithms**:
  - Experiment with advanced machine learning algorithms such as ensemble methods, deep learning techniques, or hybrid models. These approaches could potentially improve prediction accuracy and model performance.
- **Extended Feature Engineering**:
  - Develop and test new features based on additional data sources or advanced data processing techniques. This might include sentiment analysis of reviews or analysis of marketing strategies to uncover new predictors for box office success.
- **UI/UX Enhancements**:
  - Continue to refine the user interface and user experience based on user feedback. Improvements could involve adding new features, simplifying navigation, or enhancing visual design for better user interaction.
- **Automated Testing and CI/CD**:
  - Implement automated testing frameworks and CI/CD pipelines to streamline development processes and maintain code quality. Automated testing would ensure robustness, while CI/CD pipelines would facilitate continuous integration and deployment.

By addressing these recommendations, future iterations of the TMDB Box Office Prediction App can achieve greater functionality, performance, and user satisfaction.

7.Testing

7.1 General Steps involved in testing:

7.1.1 Testing Approach

1. **Unit Tests**: Tested individual functions for data processing and model predictions.
2. **Integration Tests**: Ensured that data flows correctly between modules.
3. **System Tests**: Validated the overall system functionality and performance.

7.1.2 Testing Results

- **Unit Tests**: Verified the correctness of data collection and pre-processing functions.
- **Integration Tests**: Ensured seamless data flow between modules.
- **System Tests**: Achieved satisfactory performance metrics: MAE, MSE, and R-squared.

7.2 Testing related to this project

Due to the project's scope and timeline, manual testing was performed to ensure the application's functionality and reliability. The following sections outline the approach to testing, the issues discovered, and the solutions implemented to ensure the application's effectiveness.

7.3 Testing Results

7.3.1. Issues Discovered:

- **Data Inconsistencies**:
  - **Issue**: During data collection and preprocessing, inconsistencies were found in the TMDB API responses, such as incomplete or corrupted data.
  - **Resolution**: Implemented data validation checks and retry mechanisms for API requests. Data consistency was ensured by cross-referencing

with additional sources and using error handling techniques to manage incomplete data.

- **Missing Values**:
    - **Issue**: Several missing values were identified in critical fields such as budget, revenue, and genre information.
    - **Resolution**: Addressed missing data through techniques such as data imputation and re-fetching missing details from the TMDB API. These techniques involved filling missing values with average data or using other relevant data points to maintain data integrity.
- **Feature Engineering Challenges**:
    - **Issue1**: Issues arose during feature engineering, particularly in converting categorical variables into numerical forms and extracting date-related features.
    - **Resolution1**: Applied one-hot encoding for categorical variables and separated release dates into distinct columns for day, month, and year. These transformations helped improve the model's ability to learn from the data.
    - **Issue2**: During feature engineering, a significant challenge was deciding how to handle the placement_companies column. One-hot encoding this column would have resulted in over 4000 columns, which was not feasible.
    - **Resolution**2: One-hot encoding is done wherever it is feasible like for genres, placement countries, etc. Let input of placement_companies in the format it is given. This will lead to keep the data frame manageable with 169 columns. This approach simplified feature selection and allowed for a more efficient model training process. PyCaret was then used to automate feature selection, improving model performance and making the process more manageable.

- **UI Bugs**:
    - **Issue**: Minor bugs and inconsistencies were found in the Streamlit user interface, such as layout issues and unresponsive buttons.
    - **Resolution**: Refined UI elements by adjusting layouts, fixing button functionalities, and ensuring a consistent and user-friendly interface.

- **Model Performance**:
  - **Issue**: Initial model performance was suboptimal with high error rates in revenue predictions.
  - **Resolution**: Conducted hyperparameter tuning and feature selection to improve the model’s performance. Leveraged PyCaret for automated model selection and validation to identify the most effective model and configurations.

7.3.2 Resolved Issues:

- **Error Handling and Data Consistency**:
  - **Solution**: Enhanced the robustness of the application by implementing error handling for API requests and performing data validation checks. Ensured that the data fetching process was reliable and that incomplete or erroneous data was managed effectively.
- **Handling Missing Data**:
  - **Solution**: Applied data imputation techniques and re-fetched missing information to complete the dataset. Missing values were addressed by filling gaps with average or median values and ensuring that the final dataset was comprehensive and accurate.

- **Feature Engineering Handling**:
  - One-hot encoding for the placement_companies column was considered but deemed impractical due to the resulting explosion of feature dimensions. With over 4000 unique companies, one-hot encoding would have created an excessively large feature space, making model training inefficient and slowing down the feature selection process. Instead, a more practical approach was adopted by selecting a smaller subset of features, which streamlined the feature selection process and enhanced the efficiency of the model training phase.

- **Improving UI/UX**:
  - **Solution**: Addressed UI inconsistencies by fixing layout issues and improving the user experience. Minor glitches were resolved to ensure that the interface was intuitive and easy to navigate for users.

- **Enhancing Model Accuracy**:
    - **Solution**: Performed hyperparameter tuning and feature selection to improve the model’s performance. Utilized advanced techniques and tools provided by PyCaret to refine the model and achieve better predictive accuracy.

7.4 Future Testing Recommendations

To further ensure the application's robustness and effectiveness, the following testing strategies are recommended for future enhancements:

- **Automated Testing**:
    - Implement automated tests for data fetching, preprocessing, and UI functionality to streamline the testing process and catch issues early.
- **Extended User Testing**:
    - Conduct user acceptance testing with a diverse group of users to gather feedback and identify potential improvements in the user experience.
- **Performance Testing**:
    - Perform load testing to assess the application's performance under different usage scenarios and ensure it can handle high traffic volumes.
- **Integration Testing**:
    - Test the integration of the application with various external APIs and services to ensure that all components work seamlessly together.

By addressing these future testing recommendations, the application can be further refined to meet the needs of users and stakeholders effectively.

8. Deployment

8.1 Deployment Process

The TMDB Movie Revenue Prediction App can be deployed efficiently using various platforms, including Streamlit Sharing and other cloud platforms. These platforms simplify the deployment process and ensure that the application is accessible online for users. Below is a detailed description of the deployment process, including environment setup, model deployment, and hosting options.

8.1.1 Environment Setup

To prepare the environment for deploying the TMDB Movie Revenue Prediction App, follow these steps:

- **Install Required Packages**: Ensure that all the necessary packages and dependencies are installed using the requirements.txt file. This file contains a list of all the libraries needed for the application, which can be installed in a single command.

  Steps:

  1. **Clone the Repository**:
     bash
     ```
     git clone https://github.com/username/tmdb-box-office-prediction.git
     ```

  2. **Navigate to the Project Directory**:
     bash
     ```
     cd tmdb-box-office-prediction
     ```

  3. **Install Packages**:
     bash
     ```
     pip install -r requirements.txt
     ```

8.1.2 Model Deployment

Deploy the trained model using Flask and Streamlit to create an interactive web application. Flask handles the backend API requests, while Streamlit provides the user interface for interacting with the model.

- **Deploy the Model**:
    - **Create app.py**: This script sets up the Flask API and Streamlit interface. It loads the trained model, defines endpoints for prediction, and displays the results in the web application.

    - **Run the Application**:
      bash
      ```
      streamlit run app.py
      ```

8.1.3 Hosting

After deploying the model and setting up the Streamlit interface, the application can be hosted on various cloud platforms. Each platform offers different features for hosting and managing web applications.

- **Streamlit Sharing**:
    - **Rapid Deployment**: Streamlit Sharing allows for quick deployment of Streamlit applications. By simply connecting your GitHub repository, Streamlit Sharing makes your application accessible online with minimal configuration.
    - **Steps**:
        1. **Push Code to GitHub**:
           bash
           ```
           git add .
           git commit -m "Deploy to Streamlit Sharing"
           git push origin main
           ```

        2. **Connect Repository**: Go to Streamlit Sharing, sign in, and connect your GitHub repository.
        3. **Deploy**: Deploy the application on any cloud platform.

8.2 Deployment Scripts

To automate the deployment process, you can use the following bash script to set up the environment and run the application:

Example Script:

```
bash
#!/bin/bash
# This script sets up the environment and runs the application.

# Install required packages
pip install -r requirements.txt

# Run the Streamlit application
streamlit run app.py
```

This script installs all necessary packages from requirements.txt and starts the Streamlit application, making it ready for users to interact with.

By following these deployment steps and utilizing the mentioned platforms, you can efficiently deploy and manage the TMDB Movie Revenue Prediction App, ensuring it is accessible and functional for all users.

9. User Guide

9.1 Setup Instructions

1. **Clone the Repository**:

   bash

   ```
   git clone https://github.com/username/tmdb-box-office-prediction.git
   ```

2. **Install Dependencies**:

   bash

   ```
   pip install -r requirements.txt
   ```

3. **Run the Application**:

   bash

   ```
   python app.py
   ```

9.2 Usage Instructions

1. **Input Movie Data**: Enter movie features (e.g., budget, genres, cast) in the user interface.
2. **Get Predictions**: Click the "Predict" button to get the estimated box office revenue.

9.3 Troubleshooting Tips

- **Common Issues**:
  - Ensure all dependencies are installed.
  - Verify the TMDB API key is correctly configured.
  - Check for correct data formats in input fields.

10. Conclusion

10.1 Outcomes

- **Predictive Model Development**: Successfully built and validated a predictive model for estimating box office revenue using multiple regression techniques. This model leverages data from TMDB, including budget, genres, release date, cast, and crew, demonstrating significant accuracy in revenue predictions.
- **User-Friendly Interface**: Developed an intuitive and accessible web-based interface using Flask and Streamlit, allowing users to input movie details and receive immediate revenue predictions. This interface is designed to cater to both industry stakeholders and casual movie enthusiasts.
- **Data-Driven Insights**: Generated actionable insights into the factors that most significantly influence a movie's box office success. These insights are valuable for film industry stakeholders in making informed decisions regarding production budgets, marketing strategies, and resource allocation.
- **Comprehensive Documentation**: Produced thorough documentation covering the entire project lifecycle, from data collection and preprocessing to model deployment and user guide. This ensures that the project is easily understandable and maintainable for future developers and users.
- **Robust Evaluation Metrics**: Utilized various evaluation metrics such as Mean Absolute Error (MAE), Mean Squared Error (MSE), and R-squared to assess model performance, ensuring the reliability and robustness of the predictive model.

10.2 Reflections

- **Lessons Learned**:
  - **Data Quality**: The accuracy and reliability of predictive models heavily depend on the quality of the input data. Significant effort was required to clean, preprocess, and transform the raw data into a usable format.
  - **Feature Engineering**: Effective feature engineering played a crucial role in enhancing the model's performance. Techniques such as one-hot

    encoding for categorical variables and normalization for numerical variables were essential in achieving better predictions.

    - **Model Selection**: The process of identifying the most suitable model involved extensive experimentation and comparison. PyCaret was instrumental in streamlining this process, ultimately leading to the selection of Elastic Net as the optimal model.
    - **Cross-Validation**: Implementing cross-validation techniques helped ensure that the model generalizes well to unseen data. This approach was vital in mitigating overfitting and improving the model's robustness.
    - **Deployment Challenges**: Deploying the model and creating a user-friendly interface presented its own set of challenges. Ensuring seamless integration and user experience required careful planning and execution.

10.3 Areas for Improvement

- **Incorporating Additional Data Sources**: Integrating more diverse data sources, such as social media trends and marketing data, could enhance the model's predictive power.
- **Enhancing Model Accuracy**: Exploring advanced machine learning techniques and fine-tuning hyperparameters may lead to further improvements in prediction accuracy.
- **Real-Time Data Updates**: Implementing real-time data updates can keep the model relevant and adaptive to changing trends.
- **User Interface Improvements**: Further refinement of the user interface to enhance usability and add more features, such as detailed prediction reports and visualizations, can provide greater value to users.

11. Appendices

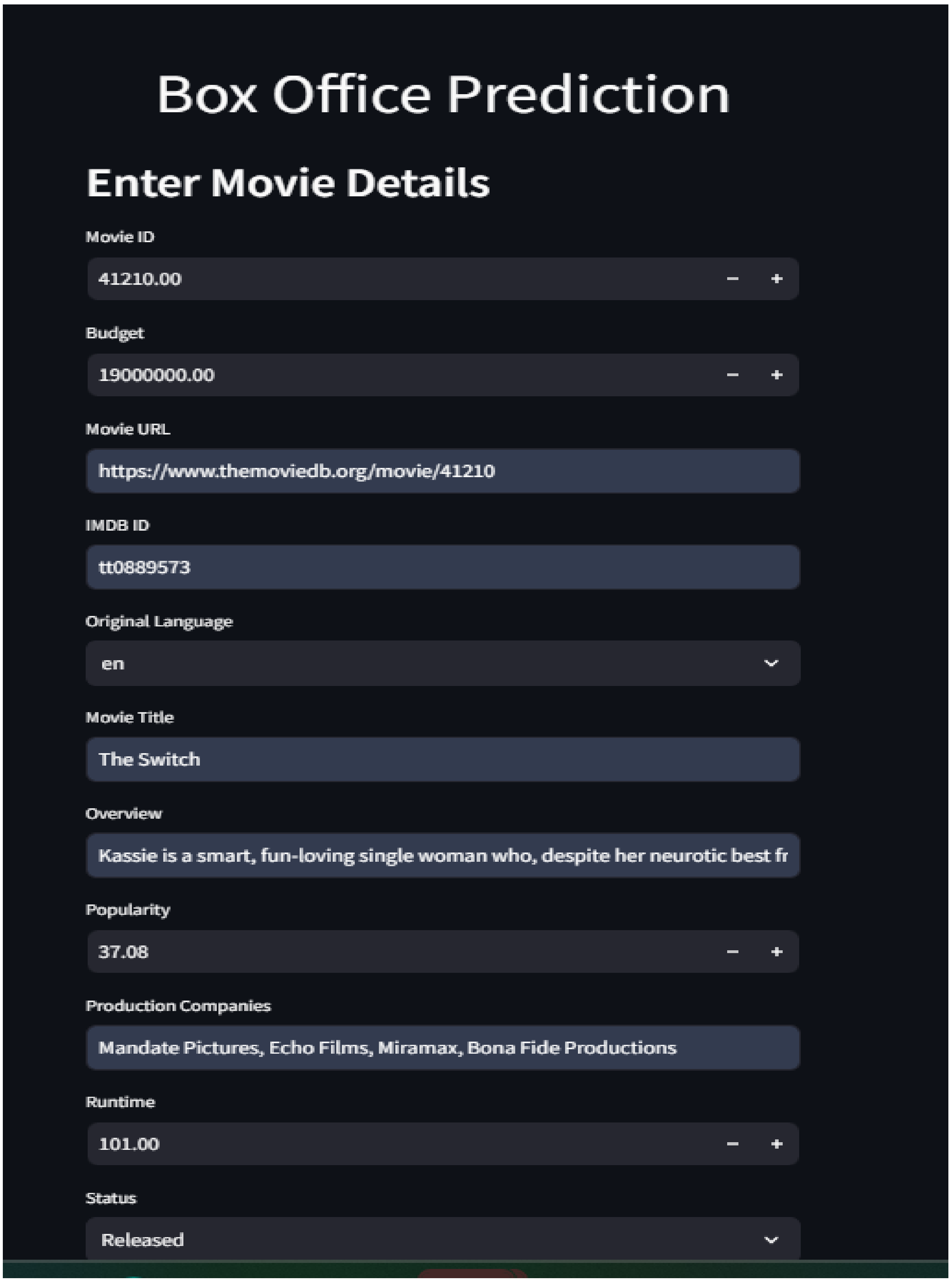
11.1 UI Code Snippets

```python
import streamlit as st
import pandas as pd
import joblib
from datetime import datetime, date
from pycaret.regression import load_model, predict_model

# Load the saved model
model_path = 'C:/Users/shreyas1bst/Desktop/INFY INTERN 2024
SUMMER/official/official submitted data sy/milestone 3 - week 5/code/3
t/ver 3/pkl/final_model.pkl'
loaded_model = joblib.load(model_path)

# Define the lists for production countries, unique languages, and
unique genres
unique_countries = [
    'Aruba', 'Australia', 'Austria', 'Bahamas', 'Belgium', 'Botswana',
'Brazil', 'Bulgaria',
    'Canada', 'Chile', 'China', 'Colombia', 'Cyprus', 'Czech Republic',
'Denmark',
    'Dominican Republic', 'Finland', 'France', 'Germany', 'Greece',
'Hong Kong',
    'Hungary', 'Iceland', 'India', 'Indonesia', 'Ireland', 'Israel',
'Italy',
    'Jamaica', 'Japan', 'Kenya', 'Kuwait', 'Lebanon', 'Libyan Arab
Jamahiriya',
    'Luxembourg', 'Malaysia', 'Malta', 'Mexico', 'Montenegro',
'Morocco', 'Netherlands',
    'New Zealand', 'Norway', 'Peru', 'Poland', 'Portugal', 'Puerto
Rico', 'Romania',
    'Russia', 'Saudi Arabia', 'Serbia', 'Singapore', 'Slovakia',
'Slovenia',
    'South Africa', 'South Korea', 'Soviet Union', 'Spain', 'Sweden',
'Switzerland',
    'Taiwan', 'Thailand', 'Turkey', 'Ukraine', 'United Arab Emirates',
    'United Kingdom', 'United States of America', 'Venezuela',
'Zimbabwe',

    'Unnamed: 90', 'ქართული', 'Pусский', '普通话', 'ਪੰਜਾਬੀ', 'සිංහල',
'български език', 'اردو', '广州话 / 廣州話', 'Український',
'한국어/조선말', 'العربية', '日本語', 'isiZulu', 'Tiếng Việt', 'Türkçe',
'Íslenska', 'shqip', 'euskera', 'suomi', 'தமிழ்',
    'ελληνικά', 'فارسی', 'Srpski', 'پښتو', 'עִבְרִית', 'Český', 'ภาษาไทย',
'svenska', 'বাংলা'
]

unique_languages = [
    'Afrikaans', 'Bahasa indonesia', 'Bahasa melayu', 'Bosanski',
'Català', 'Cymraeg',
    'Dansk', 'Deutsch', 'Eesti', 'English', 'Español', 'Esperanto',
'Français',
    'Fulfulde', 'Gaeilge', 'Hrvatski', 'Italiano', 'Kiswahili',
'Latin', 'Lietuvių',
    'Magyar', 'Nederlands', 'No Language', 'Norsk', 'Polski',
'Português', 'Română',
    'Slovenčina', 'Somali', 'हिन्दी'
]

unique_genres = [
    'Action', 'Adventure', 'Animation', 'Comedy', 'Crime',
'Documentary', 'Drama',
    'Family', 'Fantasy', 'History', 'Horror', 'Music', 'Mystery',
'Romance',
    'Science Fiction', 'TV Movie', 'Thriller', 'War', 'Western'
]

def transform_data(data):
    # Map original_language to integer
    original_language_mapping = {'en': 0, '': 1}
    data['original_language'] =
original_language_mapping.get(data.get('original_language', ''), 0)

    # Map status to integer
    status_mapping = {'Released': 3, 'Post Production': 2, 'Planned':
1, 'In Production': 0}
    data['status'] = status_mapping.get(data.get('status', ''), 0)

    # Initialize input_data with default values
    input_data = {
        'movie_id': 0,
        'budget': 0,
        'movie_url': '',
        'imdb_id': '',
        'original_language': 0,
        'movie_title': '',
        'overview': '',
        'popularity': 0.0,
        'production_companies': '',
        'poster_path': '',
        'runtime': 0,
        'status': 0,
        'tagline': '',
        # 'title': '',
        'keywords': '',
        'cast': '',
        'crew': '',
        'release_day': datetime.now().day,
        'release_month': datetime.now().month,
        'release_year': datetime.now().year
    }

    for genres in unique_genres:
        input_data[genres] = 0

    for spoken_languages in unique_languages:
        input_data[spoken_languages] = 0

    for production_countries in unique_countries:
        input_data[production_countries] = 0

    # Fill input_data with form values
    input_data.update({
        'movie_id': int(data.get('movieId', 0)),
        'budget': float(data.get('budget', 0)),
        'movie_url': data.get('movieUrl', ''),
        'imdb_id': data.get('imdbId', ''),
        'original_language': data['original_language'],
        'movie_title': data.get('movieTitle', ''),
        'overview': data.get('overview', ''),
        'popularity': float(data.get('popularity', 0)),
        'production_companies': data.get('production_companies', ''),
        'runtime': int(data.get('runtime', 0)),
        'status': data['status'],
        'tagline': data.get('tagline', ''),
        # 'title': data.get('title', ''),
        'keywords': data.get('keywords', ''),
        'cast': data.get('cast', ''),
        'crew': data.get('crew', ''),
        'release_day': int(data.get('releaseDate',
'0000/00/00').split('/')[2]),
        'release_month': int(data.get('releaseDate',
'0000/00/00').split('/')[1]),
        'release_year': int(data.get('releaseDate',
'0000/00/00').split('/')[0]),
    })

    # Set genres, spoken languages, and production countries to 1 if
present in data
```

```
    for genres in unique_genres:
        input_data[genres] = 1 if genres in data.get('genres', []) else
0
    for spoken_languages in unique_languages:
        input_data[spoken_languages] = 1 if spoken_languages in
data.get('spoken_languages', []) else 0
    for production_countries in unique_countries:
        input_data[production_countries] = 1 if production_countries in
data.get('production_countries', []) else 0

    return input_data

# Streamlit app
# Centering the title using HTML and CSS
st.markdown(
    """
    <style>
    .title {
        text-align: center;
        font-size: 3em;
    }
    </style>
    """,
    unsafe_allow_html=True
)

st.markdown('<div class="title">Box Office Prediction</div>',
unsafe_allow_html=True)

col1, col2, col3 = st.columns([1, 6, 1])

with col2:
    st.header('Enter Movie Details')
    movie_id = st.number_input('Movie ID')
    budget = st.number_input('Budget')
    movie_url = st.text_input('Movie URL')
    imdb_id = st.text_input('IMDB ID')
    original_language = st.selectbox('Original Language', ['en', ''])
    movie_title = st.text_input('Movie Title')
    overview = st.text_input('Overview')
    popularity = st.number_input('Popularity', min_value=0.0, step=0.1)
    production_companies = st.text_input('Production Companies')
    runtime = st.number_input('Runtime')
    status = st.selectbox('Status', ['Released', 'Post Production',
'Planned', 'In Production'])
    tagline = st.text_input('Tagline')
    # title = st.text_input('Title')
    keywords = st.text_input('Keywords')
```

```python
    cast = st.text_input('Cast')
    crew = st.text_input('Crew')
    release_date = st.date_input('Release Date', min_value=date(1900,
1, 1), max_value=date(2030, 12, 31))
    genres = st.multiselect('Genres', unique_genres)
    spoken_languages = st.multiselect('Spoken Languages',
unique_languages)
    production_countries = st.multiselect('Production Countries',
unique_countries)

    data = {
        'movieId': movie_id,
        'budget': budget,
        'movieUrl': movie_url,
        'imdbId': imdb_id,
        'original_language': original_language,
        'movieTitle': movie_title,
        'overview': overview,
        'popularity': popularity,
        'production_companies': production_companies,
        'runtime': runtime,
        'status': status,
        'tagline': tagline,
        # 'title': title,
        'keywords': keywords,
        'cast': cast,
        'crew': crew,
        'releaseDate': release_date.strftime('%Y/%m/%d'),
        'genres': genres,
        'spokenLanguages': spoken_languages,
        'productionCountries': production_countries
    }

    if st.button('Predict'):
        transformed_data = transform_data(data)

        # Convert the dictionary to a pandas DataFrame
        input_data_df = pd.DataFrame([transformed_data])

        # Ensure all expected columns are present in the DataFrame
        all_columns = [
            'movie_id', 'budget', 'movie_url', 'imdb_id',
'original_language', 'movie_title', 'overview',
            'popularity', 'production_companies', 'poster_path',
'runtime', 'status', 'tagline', 'title',
            'keywords', 'cast', 'crew', 'release_day', 'release_month',
'release_year'
        ] + unique_genres + unique_languages + production_countries
        for col in all_columns:
            if col not in input_data_df.columns:
                input_data_df[col] = 0

        input_data_df['original_language'] =
input_data_df['original_language'].astype('category')
        input_data_df['status'] =
input_data_df['status'].astype('category')


        # Make predictions using the loaded model
        predictions = predict_model(loaded_model, data=input_data_df)

        # Display predictions
        st.subheader('Prediction')
        st.write(f'Predicted Box Office Revenue:
{predictions["prediction_label"][0].round(1)}')

        # Add a title section with the same movie_title in the code
        st.write(f'Movie Title: {data.get("movieTitle", "")}')

        # Map status and original_language back to their string values
for display
        status_reverse_mapping = {3: 'Released', 2: 'Post Production',
1: 'Planned', 0: 'In Production'}
        original_language_reverse_mapping = {0 : 'en', 1 : ''}

        data['status'] = status_reverse_mapping.get(data['status'],
'Released')
        data['original_language'] =
original_language_reverse_mapping.get(data['original_language'], 'en')

        # Display entered movie details
        st.subheader('Entered Movie Details')
        st.table(pd.DataFrame([data]).T)
```

11.2 UI Output



Box Office Prediction

Enter Movie Details

Movie ID
41210.00

Budget
19000000.00

Movie URL
https://www.themoviedb.org/movie/41210

IMDB ID
tt0889573

Original Language
en

Movie Title
The Switch

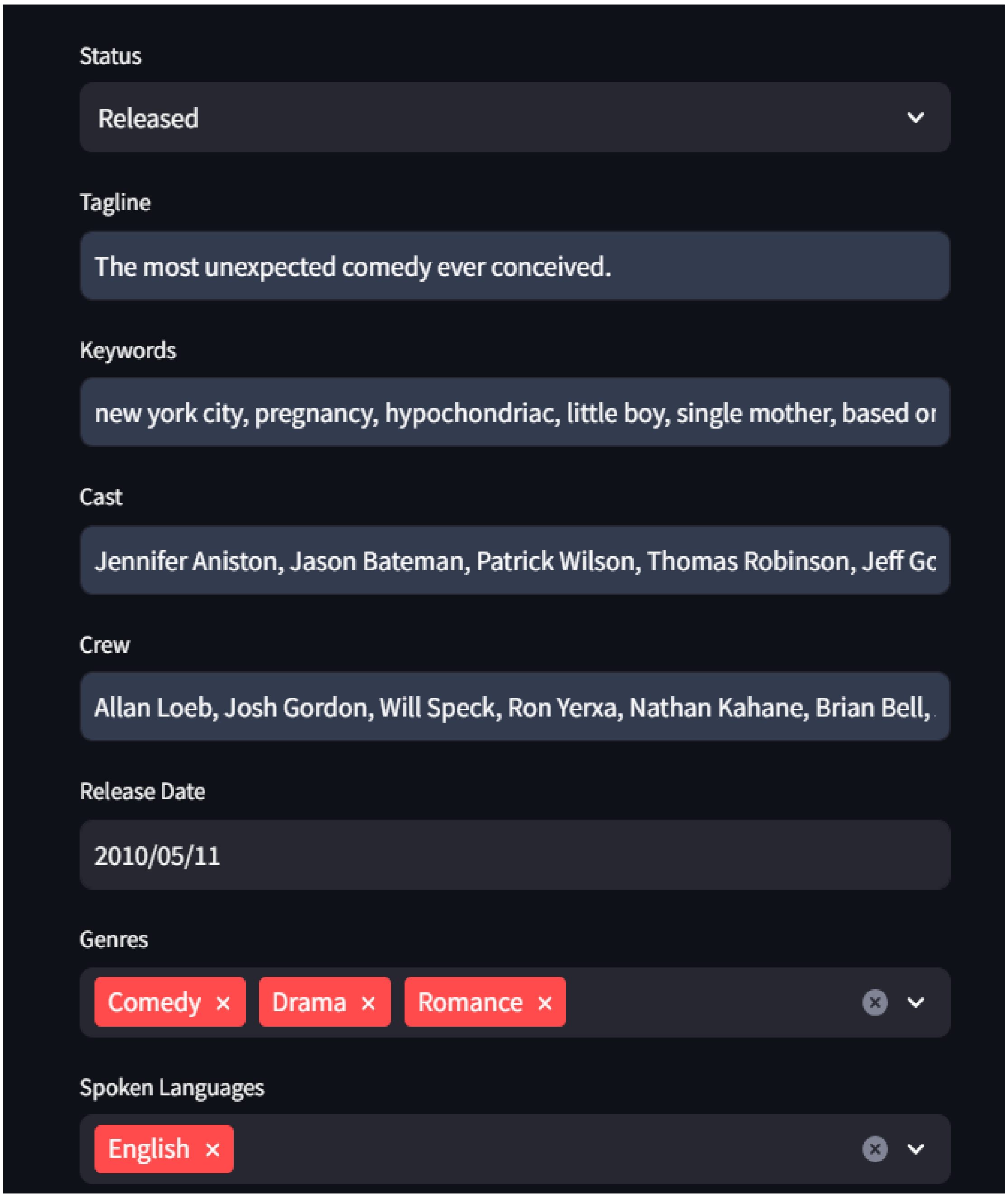
Overview
Kassie is a smart, fun-loving single woman who, despite her neurotic best fr

Popularity
37.08

Production Companies
Mandate Pictures, Echo Films, Miramax, Bona Fide Productions

Runtime
101.00

Status
Released



Status

Released

Tagline

The most unexpected comedy ever conceived.

Keywords

new york city, pregnancy, hypochondriac, little boy, single mother, based or

Cast

Jennifer Aniston, Jason Bateman, Patrick Wilson, Thomas Robinson, Jeff Gc

Crew

Allan Loeb, Josh Gordon, Will Speck, Ron Yerxa, Nathan Kahane, Brian Bell,

Release Date

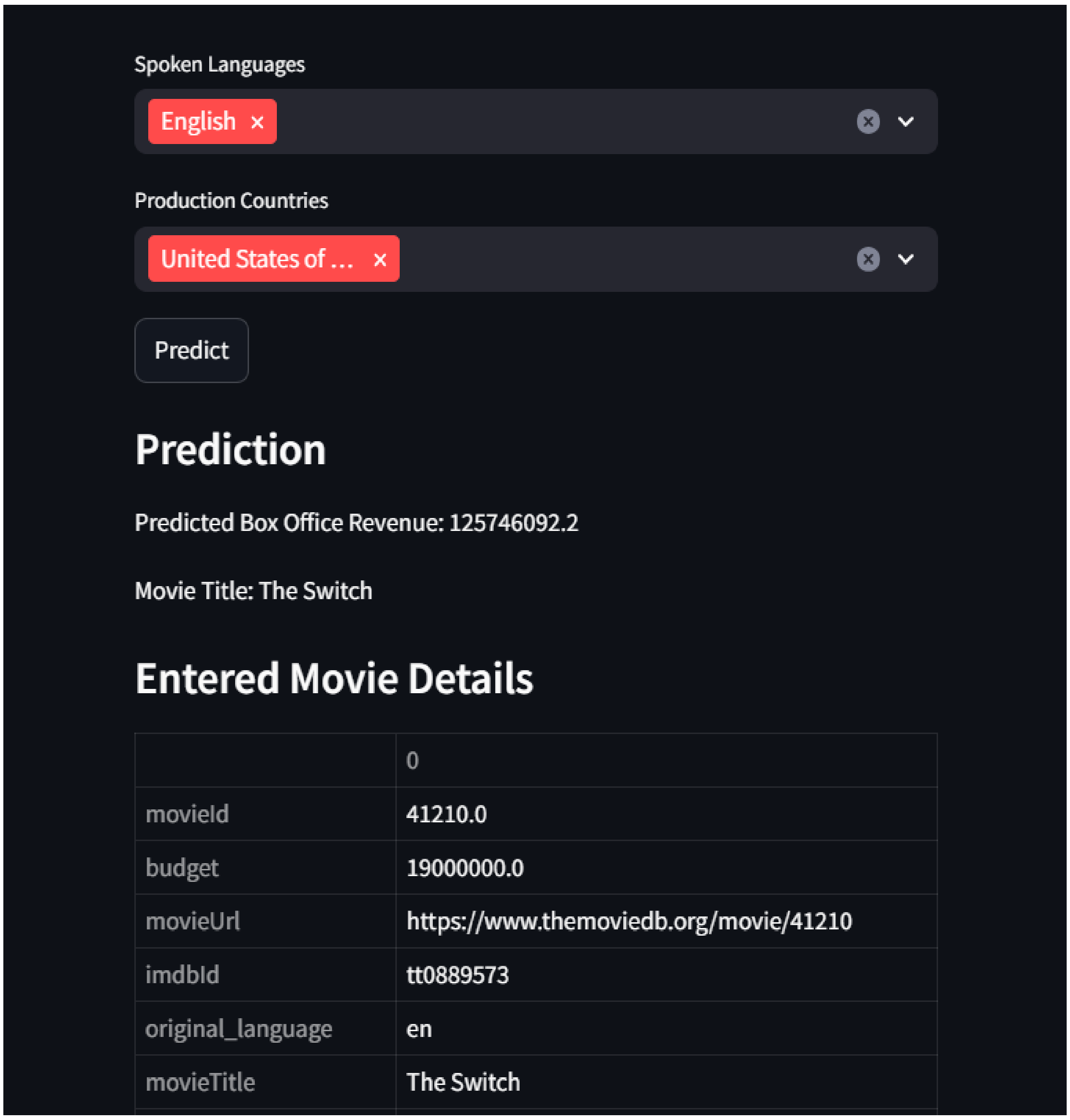
2010/05/11

Genres

Comedy × Drama × Romance ×

Spoken Languages

English ×



Prediction

Predicted Box Office Revenue: 125746092.2

Movie Title: The Switch

Entered Movie Details

	0
movieId	41210.0
budget	19000000.0
movieUrl	https://www.themoviedb.org/movie/41210
imdbId	tt0889573
original_language	en
movieTitle	The Switch

11.3 References

- **Python Tutorials**
  - https://docs.python.org/3/
- **Requests Tutorial**
  - https://realpython.com/python-requests/
- **PyCaret Docs**
  - PyCaret Documentation: https://pycaret.readthedocs.io/en/latest/
  - https://pycaret.readthedocs.io/en/latest/tutorials.html
- **Streamlit Docs**:
  - https://docs.streamlit.io/
  - https://docs.streamlit.io/develop/tutorials