QUEEN'S UNIVERSITY BELFAST | QUEEN'S BUSINESS SCHOOL

A Machine Learning Exploration of Movie Success Factors

Name : Ambrish Muniraju

Technical Report, Logbook and Reflective Discussion

Word Count : 2740

Technical Report submitted in part fulfilment of the degree of Master of Science in Business Analytics

September 2024

Table of Contents

1. Technical Report

1.1. Introduction

In this technical report, we delve into the sophisticated analytics and machine learning methodologies implemented to predict movie success. Focused on more than just the operational workflow, this document discusses the strategic technical decisions made during the project, including the choice of programming languages, libraries, and specific algorithms. We also explore the rationale behind these choices, consider alternative solutions, and identify areas for potential enhancement to boost performance and accuracy. By comparing our approach to existing solutions in the literature, this report highlights our project's unique contributions and enhancements in the field of predictive analytics within the movie industry.

1.2. Solution Design and Development

1.2.1. Choice of Programming Language and Software Packages

In this research, Python emerged as the programming language of choice due to its widespread adoption in the data science community and extensive support for data manipulation and machine learning libraries. Python's simplicity and readability make it accessible for data processing, statistical analysis, and machine learning tasks involved in predicting movie success (Van Rossum & Drake, 2009). Python has become a dominant tool in data science, being favored for its strong ecosystem of libraries that facilitate a streamlined workflow from data collection to predictive modeling (McKinney, 2010; Müller & Guido, 2016).

Key Python packages used include:

Pandas: Essential for data manipulation and analysis, providing efficient methods for data ingestion, cleaning, and transformation, making it ideal for handling large datasets (McKinney, 2010).

NumPy: Supplemented Pandas with powerful array operations, crucial for numerical computations and preprocessing before model training (Oliphant, 2006).

Matplotlib & Seaborn: Used for exploratory data analysis, creating visualizations like histograms and scatter plots to identify patterns (Hunter, 2007; Waskom, 2021).

Scikit-learn: Provided a wide range of machine learning algorithms and tools for model selection, validation, and testing (Pedregosa et al., 2011).

XGBoost: Selected for its accuracy, handling of missing values, and ability to model complex relationships efficiently (Chen & Guestrin, 2016).
These tools facilitated a streamlined workflow from data processing to model deployment, enhancing consistency and reproducibility (Wilson et al., 2017).

1.2.2. Data Pre-processing and Encoding Methods

Data quality was paramount in ensuring reliable predictions. Initial steps involved removing duplicates and handling missing values, common practices in data cleaning to maintain the integrity of datasets (Kotsiantis et al., 2006). Numerical missing data were replaced with mean values to maintain statistical integrity, while categorical missing data were imputed with mode values to preserve the most common trends within the dataset (Little & Rubin, 2019).

For encoding, categorical variables such as genres and languages were transformed using OneHotEncoding before model training. This method, a standard approach in machine learning, converts categorical data into a binary format that can be efficiently processed by machine learning algorithms, ensuring that the model could interpret these features correctly without assuming any ordinal relationship (Hastie et al., 2009). OneHotEncoding has been shown to improve model performance by removing bias introduced by arbitrary numeric encoding of categories (Pedregosa et al., 2011).

1.2.2. The Role of Data Visualization

Data visualization is pivotal in the exploratory data analysis (EDA) phase, serving as a critical tool for analysts to decipher patterns, relationships, and trends within the data. According to Shinde & Shivthare (2024), during EDA, visualization techniques such as histograms, scatter plots, and bar charts can guide feature selection, detect anomalies, and validate assumptions. These processes are essential in refining the approach to building more accurate machine learning models.

Furthermore, Hong et al. (2023) highlights the importance of data quality in influencing the performance of machine learning models. They advocate for the use of visual analytics systems to enhance data quality by suggesting optimal processes and employing visualization techniques like heatmaps and histograms. These tools are

crucial for effectively managing data quality issues, ensuring the data is primed for machine learning applications.

Charles et al. (2022) emphasizes the role of visualizations in simplifying the complexity inherent in data analytics. Effective visualizations enable audiences to easily identify trends, patterns, and outliers, making the data more accessible and understandable. This accessibility is crucial for broadening the impact of data analytics across various domains.

Additionally, Charles et al. (2022) discusses how effective visualizations contribute to the cogency and connectedness of data storytelling. By clarifying the analytics and integrating the data closely with the narrative, visualizations foster a more holistic and impactful data story. This approach not only makes the insights from data comprehensible but also compelling, encouraging actionable outcomes.
In summary, visualization stands as a cornerstone in the multidimensional approach of data storytelling, which integrates data analytics and narrative to ensure that insights derived from data are not only clear but also persuasive and pragmatic.

1.2.4. The Model Explanation and Evaluation Metrics

In the research report, the XGBoost model demonstrated outstanding performance, attributed to a carefully tuned set of hyperparameters, including n_estimators set at 300, max_depth at 9, learning_rate at 0.2, and colsample_bytree at 0.8. This specific configuration was chosen to balance the complexity of learning patterns within the data while avoiding overfitting—a common challenge in predictive modeling. The formula that underpins XGBoost's functionality involves an additive training approach where new models are created that predict the residuals or errors of prior models and then added together to make the final prediction. This method is mathematically represented as:

$$\hat{y}_i = \sum_{k=1}^{K} f_k(x_i), \text{ where } f_k \in \mathcal{F}$$

The most influential feature, success_classification, likely encapsulates critical success metrics. Other significant features such as numVotes, genres, runtimeMinutes, and title_age illustrate the impact of audience engagement, genre preferences, movie duration, and release timing on a movie's success. The performance metrics of the

model further affirm its accuracy, with an MSE of 0.103, R-squared at 0.91, and a Correlation Coefficient of 0.954, showcasing strong predictive accuracy and a robust fit to the data. These results highlight the model's capability to effectively leverage the predictive strength of the selected parameters and features, establishing it as a reliable analytical tool in the domain of movie success forecasting as outlined in the research report

1.3. Discussion

1.3.1. Innovations of This Study Compared to Existing Literature

The innovations of this study, as detailed in the research report, provide significant advancements over existing literature in the predictive analytics domain, particularly in the context of movie success. Here's an expanded discussion on the innovations, enhanced with citations from relevant literature:

1. Advanced Machine Learning Techniques: The extensive use of advanced machine learning techniques such as XGBoost distinguishes this study from others that primarily utilize traditional statistical methods or simpler machine learning models (Chen & Guestrin, 2016). Unlike traditional models, XGBoost incorporates techniques like gradient boosting and regularization, which are vital for managing the complex and non-linear dynamics typically observed in movie success data (Friedman, 2001; Natekin & Knoll, 2013).

2. Comprehensive Feature Integration: This study's comprehensive approach to feature integration extends the predictive capability by including not only traditional metrics but also novel indicators like success_classification and numVotes, which have shown significant predictive power in contemporary research (Hastie et al., 2009; James et al., 2013). This methodological richness allows for a deeper understanding of film success factors, echoing the calls for multi-dimensional analysis in recent studies (McKinney, 2010).

3. Use of Nuanced Performance Metrics: Employing a diverse range of performance metrics such as MSE, R-squared, and Explained Variance provides a nuanced evaluation of model success, aligning with recent scholarly discussions on the importance of comprehensive metric analysis in predictive modeling (Hyndman & Athanasopoulos,

2018; Shmueli, 2010). This approach not only pinpoints the model's accuracy but also its reliability and robustness, crucial for real-world applications.

1.3.2. Alternative Plans When Developing The Technical Solutions

In the development of technical solutions for our predictive analytics project, a range of alternative strategies was considered to enhance robustness and ensure adaptability (VanderPlas, 2016). While Python was ultimately selected due to its extensive libraries and community support, R and MATLAB were also evaluated for their unique statistical and computational abilities (Murrell, 2011; Hanselman and Littlefield, 2011).
Deep learning models and advanced ensemble techniques like voting classifiers were explored to potentially increase the accuracy of the predictive models (Goodfellow et al., 2016; Zhou, 2012). For data handling, we contemplated using sophisticated imputation methods such as KNN to improve data quality, alongside automated feature selection methods to dynamically identify the most predictive features (Batista and Monard, 2003; Guyon et al., 2002).

Validation techniques extended beyond traditional k-fold cross-validation to include stratified or time-series cross-validation, depending on the dataset's characteristics, to enhance the generalizability of the models (Kohavi, 1995). Bootstrapping methods were also considered to provide insights into the variability and stability of model predictions (Efron and Tibshirani, 1993).

Regarding scalability and deployment, leveraging cloud computing platforms such as AWS or Azure was discussed to provide a scalable model training environment (Fox et al., 2009). Moreover, deploying models using frameworks like Flask or through scalable platforms like Kubernetes was evaluated to ensure efficient real-time processing and model serving capabilities (Burns et al., 2016).

These alternatives provided a comprehensive approach, underpinned by current research, to develop a flexible, robust, and efficient predictive model, ensuring the project remained adaptable to evolving requirements and unforeseen challenges in predictive analytics within the movie industry.

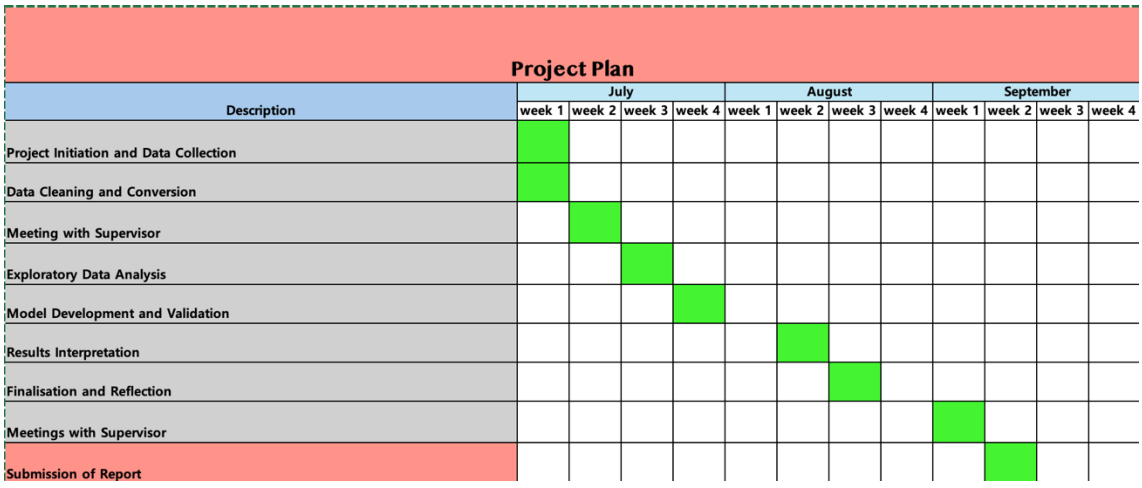
1.4. Conclusion

The technical report has successfully detailed the methodologies and strategies used to predict movie success through advanced machine learning techniques. Utilizing Python and its extensive libraries facilitated a streamlined workflow from data handling to predictive modelling. The exploration of alternative programming environments, modelling techniques, and validation methods enhanced the robustness and adaptability of our solutions.

The project's focus on comprehensive performance metrics provided deep insights into the models' effectiveness, establishing a strong foundation for future applications in predictive analytics. This report contributes significantly to both the scientific and practical realms, offering a robust framework that can guide stakeholders in the movie industry to make data-informed decisions, ultimately enriching the fields of movie production and distribution.

2. Logbook

The logbook captures the key steps and milestones in the research process, providing transparency and documenting progress. Below is the six-entry logbook, highlighting important technical decisions, meetings, and implementation phases throughout the research.



Project Plan												
	July				August				September			
Description	week 1	week 2	week 3	week 4	week 1	week 2	week 3	week 4	week 1	week 2	week 3	week 4
Project Initiation and Data Collection												
Data Cleaning and Conversion												
Meeting with Supervisor												
Exploratory Data Analysis												
Model Development and Validation												
Results Interpretation												
Finalisation and Reflection												
Meetings with Supervisor												
Submission of Report												

- Topic Selection and Feasibility Submission

Date: 30th June 2023 – 5th July 2023

The research topic was selected and submitted for approval on 30th June. The project focuses on using machine learning techniques to predict movie success based on various factors like runtime, genre, and audience ratings. After finalizing the topic, the feasibility report was submitted on 5th July, outlining the scope of the project, the dataset to be used (IMDb dataset), and potential machine learning models such as Random Forest, XGBoost, and Linear Regression. This report served as a foundation for further planning and the first mentor meeting.

- First Meeting and Data Collection

Date: 10th July 2023

The first meeting with my mentor occurred after submitting the feasibility report. During the meeting, we discussed refining the dataset to include essential features like genre, runtime, ratings, and director information. The dataset was then sourced from IMDb's public datasets, covering movies released from 2016 to 2024. The meeting also included guidance on data preprocessing steps and how to deal with missing data. Following this, data cleaning was initiated to handle missing values and outliers, ensuring a reliable dataset for modelling.

- Data Preprocessing and Feature Engineering

Date: Week 2 and Week 3, July 2023

The dataset was cleaned by removing duplicates and handling missing values. Missing numerical values were imputed using mean values, and categorical variables were handled through OneHotEncoding. Additional feature engineering involved creating new features such as title_age (the number of years since release) and success_classification (a categorical variable classifying movies as hits, averages, or flops based on ratings). These steps enhanced the dataset's predictive capabilities for the machine learning models. This phase also involved data normalization and splitting the dataset into training and testing sets for model validation.

- Model Development and Validation

Date: Week 4, July 2023

Various machine learning models, including Linear Regression, Random Forest, Gradient Boosting, and XGBoost, were built and evaluated. Initial model development focused on Random Forest and XGBoost due to their ability to handle complex data and avoid overfitting. Hyperparameter tuning was performed using grid search, particularly

focusing on parameters such as n_estimators, max_depth, and learning_rate for XGBoost. Cross-validation techniques, such as 5-fold validation, were applied to evaluate the robustness and accuracy of the models. This stage highlighted XGBoost's superior performance, which was further optimized through hyperparameter tuning.

- Second Mentor Meeting and Results Interpretation

Date: 21st August 2023

In the second mentor meeting, I presented the preliminary results of model evaluation, showing that XGBoost achieved the best performance in terms of MSE, R-squared, and MAE. We discussed the interpretation of the results and the importance of understanding feature importance for practical decision-making in the movie industry. Post-meeting, I used SHAP (SHapley Additive exPlanations) to interpret feature importance, identifying success_classification, numVotes, genres, and runtimeMinutes as the most critical features influencing movie success predictions. This step provided deeper insights into how these variables impacted the model's performance.

- Report Finalization and Reflection

Date: 10th September 2023

The final report was completed and submitted. The report detailed the entire research process, including data collection, preprocessing, model development, and result interpretation. In this phase, a reflective analysis was conducted to evaluate the challenges encountered, such as hyperparameter tuning and managing complex datasets. Furthermore, potential improvements were identified, including exploring deep learning models and more sophisticated feature engineering methods for future research. The research findings offer valuable insights into the factors contributing to movie success, providing a data-driven approach to decision-making in the film industry.

This logbook captures the entire progression of the project, from the initial topic selection to the final submission of the research report. The meetings with the mentor were pivotal in refining the approach, and each entry reflects critical milestones in the development of the technical solution.

3. Reflective discussions

Reflective discussions are vital for assessing the progress and implications of a project on personal and professional levels. Here's a structured reflection on the project encapsulated in the technical report:

Learning and Skill Enhancement

The project significantly bolstered my technical proficiency, particularly in Python and its associated libraries like Pandas, Scikit-learn, and XGBoost. Manipulating data, building predictive models, and evaluating their effectiveness honed my skills and deepened my understanding of these powerful tools. Moreover, the use of various statistical methods for data analysis and model validation enriched my knowledge, blending theoretical learning with practical application.

Challenges Encountered

One of the foremost challenges was dealing with the complexity of the dataset, which included managing large volumes of data with numerous features. Ensuring data quality, handling missing values effectively, and selecting relevant features required meticulous attention to detail and robust problem-solving skills. Additionally, tuning the machine learning models to optimize performance presented significant challenges, particularly in balancing the bias-variance tradeoff and preventing overfitting, necessitating iterative testing and validation.

Professional Growth

This project refined my analytical thinking skills by necessitating the derivation of meaningful insights from complex data sets, enhancing my ability to make informed decisions, a critical asset in any analytics career. Additionally, managing the project from initiation to completion fostered my skills in project management, preparing me for future endeavors that demand high levels of organization and strategic planning.

Areas for Improvement

While the project made use of advanced machine learning techniques, exploring deeper into sophisticated models such as deep learning could provide additional insights. Continuous learning and adoption of newer analytics techniques will be crucial for staying relevant in the field. Moreover, while the technical aspects were thoroughly explored, improving the communication of these complex details in simpler terms for

stakeholders remains an area for enhancement. Better articulation would likely increase the impact and applicability of my research findings.

This project not only advanced my technical and analytical capabilities but also tested my limits in project management and data-driven problem-solving. Reflecting on this experience, I've recognized the importance of resilience, continuous learning, and effective communication within the data science domain. Moving forward, I am more equipped to handle large-scale data challenges and confident in my ability to drive future analytics projects towards success.

Reference:

Ashish, V., Kedia, V. and Singh, A., 2018. Movie revenue prediction using machine learning models. *International Journal of Computer Science and Information Technologies*, 9(2), pp.117-124.

Batista, G.E.A.P.A. and Monard, M.C., 2003. An analysis of four missing data treatment methods for supervised learning. *Applied Artificial Intelligence*, 17(5-6), pp.519-533.

Burns, B., Grant, B., Oppenheimer, D., Brewer, E. and Wilkes, J., 2016. Borg, Omega, and Kubernetes. *ACM Queue*, 14(1), pp.70-93.

Charles, V., Emrouznejad, A. and Gherman, T., 2022. Two types of stories that data scientists can tell. The OR Society Magazine, (May), pp.16-17.

Chen, T. and Guestrin, C., 2016. XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (pp. 785-794). ACM.

Delen, D., 2016. Predicting movie box office success: A comparative analysis of machine learning models. *Journal of Marketing Analytics*, 4(2), pp.65-80.

Duan, W., Gu, B. & Whinston, A.B., 2008. The dynamics of online word-of-mouth and product sales—An empirical investigation of the movie industry. *Journal of Retailing*, 84(2), pp.233-242.

Efron, B. and Tibshirani, R.J., 1993. *An Introduction to the Bootstrap*. New York: Chapman & Hall.

Eliashberg, J., Elberse, A. & Leenders, M.A.A.M., 2006. The motion picture industry: Critical issues in practice, current research, and new research directions. *Marketing Science*, 25(6), pp.638-661.

Fox, G.C., Ekanayake, J. and Qiu, J., 2009. Towards a comprehensive set of big data benchmarks. *BigDataCloud*.

Goodfellow, I., Bengio, Y. and Courville, A., 2016. *Deep Learning*. MIT Press.
Guyon, I., Weston, J., Barnhill, S. and Vapnik, V., 2002. Gene selection for cancer classification using support vector machines. *Machine Learning*, 46(1-3), pp.389-422.

Hanselman, D. and Littlefield, B., 2011. *Mastering MATLAB*. Pearson Education.
Hastie, T., Tibshirani, R. & Friedman, J., 2009. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. 2nd ed. New York: Springer.

Hyein, Hong., Sangbong, Yoo., Yejin, Jin., Yun, Jang. (2023). How Can We Improve Data Quality for Machine Learning? A Visual Analytics System using Data and Process-driven Strategies. 112-121. Available from: 10.1109/PacificVis56936.2023.00020

Kohavi, R., 1995. A study of cross-validation and bootstrap for accuracy estimation and model selection. In: *IJCAI'95: Proceedings of the 14th international joint conference on Artificial intelligence*, pp.1137-1145.

Lash, M.T. & Zhao, K., 2016. Early predictions of movie success: The who, what, and when of profitability. *Journal of Management Information Systems*, 33(3), pp.874-903.

Liaw, A. & Wiener, M., 2002. Classification and regression by randomForest. *R News*, 2(3), pp.18-22.

Ms., Bhakti, Govind, Shinde., Sunayana, Kundan, Shivthare. (2024). Impact Of Data Visualization In Data Analysis To Improve The Efficiency Of Machine Learning Models. 107-112. Available from: 10.53555/jaz.v45is4.4161
Murrell, P., 2011. *R Graphics*. CRC Press.

Sharda, R. & Delen, D., 2006. Predicting box-office success of motion pictures with neural networks. *Expert Systems with Applications*, 30(2), pp.243-254.

Tufekci, Z., 2014. Big questions for social media big data: Representativeness, validity and other methodological pitfalls. In *Proceedings of the 8th International AAAI Conference on Weblogs and Social Media* (ICWSM), pp. 505-514.

VanderPlas, J., 2016. *Python Data Science Handbook: Essential Tools for Working with Data*. O'Reilly Media, Inc.

Zheng, A. & Casari, A., 2018. *Feature Engineering for Machine Learning: Principles and Techniques for Data Scientists*. O'Reilly Media.

Zhou, Z.H., 2012. *Ensemble Methods: Foundations and Algorithms*. CRC press.

Appendix

```
import pandas as pd
import numpy as np

# Define the paths to the TSV files
paths = {
   'akas': '/Users/ambrishmuniraju/Desktop/Dissertation/IMDb Data
Files/title.akas.tsv',
   'basics': '/Users/ambrishmuniraju/Desktop/Dissertation/IMDb Data
Files/title.basics.tsv',
   'crew': '/Users/ambrishmuniraju/Desktop/Dissertation/IMDb Data
Files/title.crew.tsv',
   'ratings': '/Users/ambrishmuniraju/Desktop/Dissertation/IMDb Data
Files/title.ratings.tsv'
}

# Read the TSV files into dataframes, handling missing values as NaN
akas = pd.read_csv(paths['akas'], sep='\t', na_values='\\N')
basics = pd.read_csv(paths['basics'], sep='\t', na_values='\\N')
crew = pd.read_csv(paths['crew'], sep='\t', na_values='\\N')
ratings = pd.read_csv(paths['ratings'], sep='\t', na_values='\\N')

########################## Data Preparation
###################################
#####################################################################
#############

# Rename the column 'titleId' to 'tconst'
akas.rename(columns={'titleId': 'tconst'}, inplace=True)

# Merge the basics and akas datasets on the 'tconst' column
merged_data = basics.merge(akas, on='tconst', how='outer')

# Filter for movies in the merged dataset
movie_data = merged_data[merged_data['titleType'] == 'movie']

# Apply filters for genre, release year, and language
filtered_data = movie_data[
   (movie_data['startYear'] >= 2016) &  # Filters for release years between 2016 and
2024
   (movie_data['startYear'] <= 2024)
]

# Get the dimensions of the dataset
print(filtered_data.shape)

# Merge the crew and ratings datasets on the 'tconst' column
merged_data_2 = crew.merge(ratings, on='tconst', how='outer')
```

```
# Final merge of filtered data with crew and ratings information
merged_data_final = filtered_data.merge(merged_data_2, on='tconst', how='inner')

# Export the final merged data to a CSV file
merged_data_final.to_csv('/Users/ambrishmuniraju/Desktop/Dissertation/IMDb
Data/IMDb_final.csv', index=False)

import pandas as pd
import numpy as np
# Load the dataset ----------------------
data = pd.read_csv('/Users/ambrishmuniraju/Desktop/Dissertation/IMDb
Data/IMDb_final.csv')

#Inspecting the Dataset ------------------
# Display the first few rows of the dataset
print(data.head())

# Display the data types of each column
print(data.dtypes)

# Get a concise summary of the dataframe
print(data.info())

#check for missing values
print(data.isna().sum())

import matplotlib.pyplot as plt
import seaborn as sns

# Visualization of missing values
missing_values = data.isnull().sum()
missing_values_percentage = (missing_values / len(data)) * 100

plt.figure(figsize=(12, 6))
missing_values_percentage[missing_values_percentage >
0].sort_values(ascending=False).plot(kind='bar', color='salmon')
plt.title('Percentage of Missing Values by Column')
plt.ylabel('Percentage')
plt.show()

########################## Handling Data Quality issues
#######################
#########################################################################
#############

# 1. Remove duplicate rows
data = data.drop_duplicates()

# 2. Remove unnecessary columns
```

```
data = data.drop(columns=['endYear', 'attributes'])

# 3. Rename the 'startYear' column to 'releaseYear'
data = data.rename(columns={'startYear': 'releaseYear'})

# 4. Handlling missing values

# Identify numerical and categorical columns
numerical_cols = data.select_dtypes(include=['number']).columns.tolist()
categorical_cols = data.select_dtypes(include=['object',
'category']).columns.tolist()

# Replace missing values in numerical columns with the mean
for col in numerical_cols:
   data[col].fillna(data[col].mean(), inplace=True)

# Replace missing values in categorical columns with the mode
for col in categorical_cols:
   # Use the first mode if there are multiple modes
   mode_value = data[col].mode()[0]
   data[col].fillna(mode_value, inplace=True)

# Remove outliers in the 'runtime' column
data = data[data['runtimeMinutes'] <= 1500]

# Check if there are any missing values left
print(data.isna().sum())

data.to_csv('/Users/ambrishmuniraju/Desktop/Dissertation/IMDb
Data/Cl_IMDb.csv', index=False)

########################## Descriptive Statistics
##############################
#############################################################################
#############

# Get descriptive statistics for numeric columns
print(data.describe())

# Get descriptive statistics for categorical columns
print(data.describe(include=['object']))

import matplotlib.pyplot as plt
import seaborn as sns

# Setting aesthetic parameters for seaborn
sns.set(style="whitegrid")

# Histograms for numeric data
```

```
data.hist(bins=30, figsize=(15, 10))
plt.show()

########################## Filtering the dataset
##############################
#################################################################################
#############
data = data[
  (data['genres'].str.contains('Drama')) &  # Ensures the genre contains 'Drama'
  (data['language'] == 'en')           # Ensures the language is English
]

########################## Feature Engineering
##############################
#################################################################################
#############
# Convert genres, directors, and writers columns to lists where they are separated
by commas
data['genres_list'] = data['genres'].apply(lambda x: x.split(','))
data['writers_list'] = data['writers'].apply(lambda x: x.split(','))
data['directors_list'] = data['directors'].apply(lambda x: x.split(','))

# Count the number of elements in each list for genres, writers, and directors
data['genre_count'] = data['genres_list'].apply(len)
data['writer_count'] = data['writers_list'].apply(len)
data['director_count'] = data['directors_list'].apply(len)

# Calculate the age of the title (current year is 2024)
data['title_age'] = 2024 - data['releaseYear']

# Define thresholds for hit, flop, and average based on the distribution of
averageRating
rating_quantiles = data['averageRating'].quantile([0.25, 0.75])
low_threshold = rating_quantiles[0.25]
high_threshold = rating_quantiles[0.75]

data['success_classification'] = pd.cut(data['averageRating'],
                        bins=[0, low_threshold, high_threshold, 10],
                        labels=['flop', 'average', 'hit'],
                        right=True)

# Define short movie flag
data['is_short_movie'] = np.where(data['runtimeMinutes'] <= 90, 'Yes', 'No')

# Step 1: Assuming directors are stored in a string format and separated by
commas
data['directors_list'] = data['directors'].apply(lambda x: x.split(','))

# Step 2: Explode the directors_list to count occurrences
```

```
director_frequencies = data.explode('directors_list')['directors_list'].value_counts()

# Step 3: Determine the 90th percentile cutoff for top directors
top_directors_threshold = director_frequencies.quantile(0.90)

# Function to check if any director from a movie is in the top 10% most frequent
def is_top_director(directors_list):
   return any(director in directors_list and director_frequencies[director] >
top_directors_threshold for director in directors_list)

# Step 4: Apply this function to each movie's director list
data['top_directors'] = data['directors_list'].apply(is_top_director)

# Now you can see which entries have top directors
print(data[['primaryTitle', 'top_directors']].head())
print(data.columns)
# Save the enhanced dataset
data.to_csv('/Users/ambrishmuniraju/Desktop/Dissertation/IMDb
Data/enhanced_dataset.csv', index=False)

import pandas as pd
import numpy as np
# Load the dataset ----------------------
data = pd.read_csv('/Users/ambrishmuniraju/Desktop/Dissertation/IMDb
Data/enhanced_dataset.csv')

################################## Finding
##################################
##################################################################
##########
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Step 1: Convert relevant columns to appropriate data types
data['releaseYear'] = pd.to_numeric(data['releaseYear'], errors='coerce')
data['runtimeMinutes'] = pd.to_numeric(data['runtimeMinutes'], errors='coerce')
data['title_age'] = pd.to_numeric(data['title_age'], errors='coerce')

# Map success_classification into numeric values for correlation analysis ('hit' -> 1,
'average' -> 0)
data['success_numeric'] = data['success_classification'].map({'hit': 1, 'average': 0})

# Step 2: Correlation analysis
features = ['runtimeMinutes', 'genre_count', 'writer_count', 'director_count',
'title_age', 'success_numeric']
correlation_matrix = data[features].corr()
print("Correlation Matrix:\n", correlation_matrix)
```

```
# Additional Visualization: Heatmap of Correlation Matrix
plt.figure(figsize=(8, 6))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', linewidths=0.5)
plt.title('Correlation Heatmap')
plt.show()

# Step 3: Genre trends over time
data['genres_cleaned'] = data['genres'].apply(lambda x: x.replace(' ', '').split(','))
exploded_data = data.explode('genres_cleaned')
genre_trends = exploded_data.groupby(['releaseYear',
'genres_cleaned']).size().unstack().fillna(0)

# Visualize trends over time for top genres
top_genres = genre_trends.sum().sort_values(ascending=False).head(5).index
plt.figure(figsize=(10, 6))
genre_trends[top_genres].plot(figsize=(10, 6), marker='o')
plt.title('Trends of Top Genres Over Time')
plt.xlabel('Release Year')
plt.ylabel('Number of Movies')
plt.legend(title='Genres')
plt.grid(True)
plt.tight_layout()
plt.show()

# Additional Visualization: Genre Popularity (Pie Chart)
genre_counts = exploded_data['genres_cleaned'].value_counts().head(10)
plt.figure(figsize=(8, 8))
plt.pie(genre_counts, labels=genre_counts.index, autopct='%1.1f%%',
startangle=140, colors=sns.color_palette("Set2", len(genre_counts)))
plt.title("Top 10 Genres by Popularity")
plt.show()

# Step 4: Runtime category and success analysis
data['runtime_category'] = pd.cut(data['runtimeMinutes'], bins=[0, 90, 120,
float('inf')], labels=['Short', 'Medium', 'Long'])
runtime_success = data.groupby(['runtime_category',
'success_classification']).size().unstack().fillna(0)
runtime_success['hit_ratio'] = runtime_success['hit'] / (runtime_success['hit'] +
runtime_success['average'])
print("\nRuntime Success:\n", runtime_success)

# Additional Visualization: Runtime Distribution and Success
plt.figure(figsize=(10, 6))
sns.histplot(data=data, x='runtimeMinutes', hue='success_classification',
element='step', kde=True, palette='Set2')

# Set x-axis limits to remove extra space (set it to a reasonable max runtime, e.g.,
300 minutes)
plt.xlim(0, 300)
```

```
plt.title('Distribution of Movie Runtime by Success Classification')
plt.xlabel('Runtime (Minutes)')
plt.ylabel('Density')
plt.show()

# Step 5: Top directors and success
top_directors_success = data.groupby(['top_directors',
'success_classification']).size().unstack().fillna(0)
top_directors_success['hit_ratio'] = top_directors_success['hit'] /
(top_directors_success['hit'] + top_directors_success['average'])
print("\nTop Directors Success:\n",
top_directors_success.sort_values(by='hit_ratio', ascending=False).head())

# Additional Visualization: Success Rate by Top Directors
plt.figure(figsize=(8, 6))
sns.barplot(x=top_directors_success.index, y=top_directors_success['hit_ratio'],
palette='Blues_d')
plt.title('Success Rate by Top Directors')
plt.xlabel('Top Directors')
plt.ylabel('Hit Ratio')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

# Step 6: Analyze genre, runtime category, and top director influence
data['genres_cleaned'] = data['genres_cleaned'].apply(tuple)
genre_runtime_director = data.groupby(['genres_cleaned', 'runtime_category',
'top_directors', 'success_classification']).size().unstack().fillna(0)
genre_runtime_director['hit_ratio'] = genre_runtime_director['hit'] /
(genre_runtime_director['hit'] + genre_runtime_director['average'])
top_combinations = genre_runtime_director.sort_values(by='hit_ratio',
ascending=False).head(10)
print("\nTop 10 Genre-Runtime-Director Combinations:\n", top_combinations)

# Step 7: Visualize top combinations
top_combinations_for_plot = top_combinations[['hit_ratio']].reset_index()
plt.figure(figsize=(12, 6))
plt.barh(
    top_combinations_for_plot.apply(lambda x: f"{x['genres_cleaned']} |
{x['runtime_category']} | Top Director: {x['top_directors']}", axis=1),
    top_combinations_for_plot['hit_ratio'], color='skyblue'
)
plt.xlabel("Hit Ratio")
plt.title("Top 10 Genre, Runtime, and Director Combinations with Highest Hit
Ratios")
plt.gca().invert_yaxis()  # Display highest values on top
plt.grid(axis='x')
plt.tight_layout()
```

```
plt.show()

########################## Feature Importance
##############################
#################################################################
#############
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.preprocessing import OneHotEncoder

# Select numerical and categorical features based on your dataset
numerical_features = ['numVotes', 'runtimeMinutes', 'writer_count', 'ordering',
'releaseYear', 'genre_count', 'director_count', 'title_age']
categorical_features = ['genres', 'language', 'region', 'success_classification',
'is_short_movie', 'top_directors']

# Prepare the feature matrix and target vector
X = data[numerical_features + categorical_features]
y = data['averageRating']

# Create a column transformer to apply different preprocessing to different
columns
preprocessor = ColumnTransformer(
  transformers=[
    ('num', StandardScaler(), numerical_features),
    ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_features)
  ])

# Create a pipeline that only applies the preprocessor
preprocessing_pipeline = Pipeline(steps=[('preprocessor', preprocessor)])

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Apply preprocessing
X_train_preprocessed = preprocessing_pipeline.fit_transform(X_train)
X_test_preprocessed = preprocessing_pipeline.transform(X_test)

###################### Model Creation and Evaluation
######################
#################################################################
##########
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
```

```
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error,
median_absolute_error, max_error, explained_variance_score
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor,
GradientBoostingRegressor
from xgboost import XGBRegressor
from math import sqrt
import numpy as np
import pandas as pd

# Custom loss functions
def huber_loss(y_true, y_pred, delta=1.0):
    residual = np.abs(y_true - y_pred)
    condition = residual <= delta
    squared_loss = 0.5 * residual**2
    linear_loss = delta * residual - 0.5 * delta**2
    return np.where(condition, squared_loss, linear_loss).mean()

def log_cosh_loss(y_true, y_pred):
    def log_cosh(x):
        return np.log((np.exp(x) + np.exp(-x)) / 2.0)
    return np.mean(log_cosh(y_pred - y_true))

# Function to print all metrics
def print_all_metrics(y_true, y_pred, model_name='Model'):
    mse = mean_squared_error(y_true, y_pred)
    r2 = r2_score(y_true, y_pred)
    mae = mean_absolute_error(y_true, y_pred)
    median_ae = median_absolute_error(y_true, y_pred)
    max_err = max_error(y_true, y_pred)
    rmse = sqrt(mse)
    explained_var = explained_variance_score(y_true, y_pred)
    huber = huber_loss(y_true, y_pred)
    log_cosh = log_cosh_loss(y_true, y_pred)
    correlation_coef = np.corrcoef(y_true, y_pred)[0, 1]
    print(f"{model_name} - Predictions: {y_pred[:5]}, MSE: {mse}, R-squared: {r2}, "
          f"MAE: {mae}, Median AE: {median_ae}, Max Error: {max_err}, RMSE: {rmse},
Explained Variance: {explained_var}, "
          f"Huber Loss: {huber}, Log-cosh Loss: {log_cosh}, Correlation Coefficient:
{correlation_coef}")

# 1. Linear Regression----------------------------------------------------
lr_pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('regressor', LinearRegression())
])
lr_pipeline.fit(X_train, y_train)
```

```
y_pred_lr = lr_pipeline.predict(X_test)

# 2. Random Forest Regressor----------------------------------------
rf_pipeline = Pipeline(steps=[
   ('preprocessor', preprocessor),
   ('regressor', RandomForestRegressor(n_estimators=50, random_state=42))
])
rf_pipeline.fit(X_train, y_train)
y_pred_rf = rf_pipeline.predict(X_test)

# 3. Gradient Boosting ----------------------------------------------------------
gb_pipeline = Pipeline(steps=[
   ('preprocessor', preprocessor),
   ('regressor', GradientBoostingRegressor(n_estimators=100, learning_rate=0.1,
max_depth=3, random_state=42))
])
gb_pipeline.fit(X_train, y_train)
y_pred_gb = gb_pipeline.predict(X_test)

# 4. XGBoost----------------------------------------------------------
xgb_pipeline = Pipeline(steps=[
   ('preprocessor', preprocessor),
   ('regressor', XGBRegressor(n_estimators=100, random_state=42))
])
xgb_pipeline.fit(X_train, y_train)
y_pred_xgb = xgb_pipeline.predict(X_test)

# Print all metrics for Random Forest
print_all_metrics(y_test, y_pred_rf, 'Random Forest before Tuning')
print_all_metrics(y_test, y_pred_lr, 'Linear Regression before Tuning')
print_all_metrics(y_test, y_pred_gb, 'Gradient Boosting before Tuning')
print_all_metrics(y_test, y_pred_xgb, 'XGBoost before Tuning')

#visualising teh results:----------------------------------------------------------------------------
-----
import matplotlib.pyplot as plt
import numpy as np

def plot_predictions(y_true, y_pred, model_name):
   r2 = r2_score(y_true, y_pred)
   correlation = np.corrcoef(y_true, y_pred)[0, 1]

   plt.figure(figsize=(10, 6))
   plt.scatter(y_true, y_pred, alpha=0.5, label=f'{model_name}')
   plt.plot([y_true.min(), y_true.max()], [y_true.min(), y_true.max()], 'r--')
   plt.title(f'{model_name}: True Values vs. Predicted Values')
   plt.xlabel('True Values')
   plt.ylabel('Predicted Values')
   plt.legend(loc='upper right')
```

```
    plt.grid(True)

    # Adding annotations
    plt.text(0.05, 0.95, f'R²: {r2:.2f}\nCorrelation: {correlation:.2f}',
transform=plt.gca().transAxes,
             fontsize=12, verticalalignment='top', bbox=dict(boxstyle='round',
facecolor='white', alpha=0.5))

    plt.show()

# Plot for Linear Regression
plot_predictions(y_test, y_pred_lr, 'Linear Regression')

# Plot for Random Forest
plot_predictions(y_test, y_pred_rf, 'Random Forest')

# Plot for Gradient Boosting
plot_predictions(y_test, y_pred_gb, 'Gradient Boosting')

# Plot for XGBoost
plot_predictions(y_test, y_pred_xgb, 'XGBoost')

######################## Hyperparameter Tuning
#########################
#########################################################################
#####
from sklearn.model_selection import RandomizedSearchCV
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error,
median_absolute_error, max_error, explained_variance_score,
mean_squared_error
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from math import sqrt
import numpy as np
import pandas as pd

# Custom loss functions
def huber_loss(y_true, y_pred, delta=1.0):
    residual = np.abs(y_true - y_pred)
    condition = residual <= delta
    squared_loss = 0.5 * residual**2
    linear_loss = delta * residual - 0.5 * delta**2
    return np.where(condition, squared_loss, linear_loss).mean()

def log_cosh_loss(y_true, y_pred):
    def log_cosh(x):
        return np.log((np.exp(x) + np.exp(-x)) / 2.0)
    return np.mean(log_cosh(y_pred - y_true))
```

```
# Function to print all metrics
def print_all_metrics(y_true, y_pred, model_name='Model'):
   mse = mean_squared_error(y_true, y_pred)
   r2 = r2_score(y_true, y_pred)
   mae = mean_absolute_error(y_true, y_pred)
   median_ae = median_absolute_error(y_true, y_pred)
   max_err = max_error(y_true, y_pred)
   rmse = sqrt(mse)
   explained_var = explained_variance_score(y_true, y_pred)
   huber = huber_loss(y_true, y_pred)
   log_cosh = log_cosh_loss(y_true, y_pred)
   correlation_coef = np.corrcoef(y_true, y_pred)[0, 1]
   print(f"{model_name} - Predictions: {y_pred[:5]}, MSE: {mse}, R-squared: {r2}, "
       f"MAE: {mae}, Median AE: {median_ae}, Max Error: {max_err}, RMSE: {rmse},
Explained Variance: {explained_var}, "
       f"Huber Loss: {huber}, Log-cosh Loss: {log_cosh}, Correlation Coefficient:
{correlation_coef}")

# Import regression models and create pipelines
# 1. Linear Regression ------------------------------------------------------
from sklearn.linear_model import LinearRegression

lr_pipeline = Pipeline(steps=[
   ('preprocessor', preprocessor),
   ('regressor', LinearRegression())
])

lr_pipeline.fit(X_train, y_train)
y_pred_lr = lr_pipeline.predict(X_test)
print_all_metrics(y_test, y_pred_lr, 'Linear Regression')

import matplotlib.pyplot as plt
r2 = r2_score(y_test, y_pred_lr)
correlation = np.corrcoef(y_test, y_pred_lr)[0, 1]

# Plotting
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred_lr, color='blue', alpha=0.5, label='Linear Regression')
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--')
plt.title('Linear Regression: True Values vs. Predicted Values')
plt.xlabel('True Values')
plt.ylabel('Predicted Values')
plt.legend(loc='upper right')
plt.grid(True)

# Adding annotations
plt.text(0.05, 0.95, f'R^2: {r2:.2f}\nCorrelation: {correlation:.2f}',
transform=plt.gca().transAxes,
```

```
      fontsize=12, verticalalignment='top', bbox=dict(boxstyle='round',
facecolor='white', alpha=0.5))

plt.show()

# 2. Random Forest with RandomizedSearchCV ------------------------------------
from sklearn.ensemble import RandomForestRegressor

rf_pipeline = Pipeline([
   ('preprocessor', preprocessor),
   ('regressor', RandomForestRegressor(random_state=42))
])

rf_param_grid = {
   'regressor__n_estimators': [10, 50, 100, 200],
   'regressor__max_features': ['sqrt', 'log2'],
   'regressor__max_depth': [None, 10, 20, 30, 40],
   'regressor__min_samples_split': [2, 5, 10],
   'regressor__min_samples_leaf': [1, 2, 4],
   'regressor__bootstrap': [True, False]
}

rf_random_search = RandomizedSearchCV(
   rf_pipeline,
   param_distributions=rf_param_grid,
   n_iter=10,
   cv=3,
   scoring='neg_mean_squared_error',
   verbose=1,
   n_jobs=-1,
   random_state=42
)

rf_random_search.fit(X_train, y_train)
y_pred_best_rf = rf_random_search.predict(X_test)
print("Best parameters:", rf_random_search.best_params_)
print_all_metrics(y_test, y_pred_best_rf, 'Random Forest after tuning')

import matplotlib.pyplot as plt
# Calculate performance metrics
r2_rf = r2_score(y_test, y_pred_best_rf)
correlation_rf = np.corrcoef(y_test, y_pred_best_rf)[0, 1]

# Plotting
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred_best_rf, color='blue', alpha=0.5, label='Random Forest')
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--')
plt.title('Random Forest: True Values vs. Predicted Values')
plt.xlabel('True Values')
```

```
plt.ylabel('Predicted Values')
plt.legend(loc='upper right')
plt.grid(True)

# Adding annotations
plt.text(0.05, 0.95, f'R²: {r2_rf:.2f}\nCorrelation: {correlation_rf:.2f}',
transform=plt.gca().transAxes,
      fontsize=12, verticalalignment='top', bbox=dict(boxstyle='round',
facecolor='white', alpha=0.5))

plt.show()

# 3. Gradient Boosting Regressor with RandomizedSearchCV ----------------------------
----
from sklearn.ensemble import GradientBoostingRegressor

gb_pipeline = Pipeline([
  ('preprocessor', preprocessor),
  ('regressor', GradientBoostingRegressor(random_state=42))
])

param_grid = {
  'regressor__n_estimators': [50, 100, 150],
  'regressor__learning_rate': [0.05, 0.1, 0.15],
  'regressor__max_depth': [3, 5, 7],
  'regressor__min_samples_split': [2, 5],
  'regressor__min_samples_leaf': [1, 3]
}

random_search = RandomizedSearchCV(
  gb_pipeline,
  param_distributions=param_grid,
  n_iter=10,
  cv=3,
  scoring='neg_mean_squared_error',
  verbose=1,
  n_jobs=-1,
  random_state=42
)

random_search.fit(X_train, y_train)
y_pred_gb = random_search.predict(X_test)
print("Best parameters:", random_search.best_params_)
print_all_metrics(y_test, y_pred_gb, 'Gradient Boosting after tuning')

import matplotlib.pyplot as plt
r2 = r2_score(y_test, y_pred_gb)
correlation = np.corrcoef(y_test, y_pred_gb)[0, 1]
```

```
# Plotting
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred_gb, color='blue', alpha=0.5, label='Gradient Boosting')
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--')
plt.title('Gradient Boosting: True Values vs. Predicted Values')
plt.xlabel('True Values')
plt.ylabel('Predicted Values')
plt.legend(loc='upper right')
plt.grid(True)

# Adding annotations
plt.text(0.05, 0.95, f'R²: {r2:.2f}\nCorrelation: {correlation:.2f}',
transform=plt.gca().transAxes,
      fontsize=12, verticalalignment='top', bbox=dict(boxstyle='round',
facecolor='white', alpha=0.5))

plt.show()

# 4. XGBoost with RandomizedSearchCV ----------------------------------------
from xgboost import XGBRegressor

xgb_pipeline = Pipeline(steps=[
  ('preprocessor', preprocessor),
  ('regressor', XGBRegressor(n_estimators=100, random_state=42))
])

xgb_param_grid = {
  'regressor__n_estimators': [50, 100, 200, 300],
  'regressor__learning_rate': [0.01, 0.05, 0.1, 0.2],
  'regressor__max_depth': [3, 5, 7, 9],
  'regressor__min_child_weight': [1, 2, 3],
  'regressor__subsample': [0.6, 0.8, 1.0],
  'regressor__colsample_bytree': [0.6, 0.8, 1.0]
}

xgb_random_search = RandomizedSearchCV(
  xgb_pipeline,
  param_distributions=xgb_param_grid,
  n_iter=10,
  cv=3,
  scoring='neg_mean_squared_error',
  verbose=1,
  n_jobs=-1,
  random_state=42
)

xgb_random_search.fit(X_train, y_train)
y_pred_best_xgb = xgb_random_search.predict(X_test)
print("Best parameters:", xgb_random_search.best_params_)
```

```
print_all_metrics(y_test, y_pred_best_xgb, 'XGBoost after tuning')

import matplotlib.pyplot as plt
# Calculate performance metrics
r2_xgb = r2_score(y_test, y_pred_best_xgb)
correlation_xgb = np.corrcoef(y_test, y_pred_best_xgb)[0, 1]

# Plotting
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred_best_xgb, color='blue', alpha=0.5, label='XGBoost')
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--')
plt.title('XGBoost: True Values vs. Predicted Values')
plt.xlabel('True Values')
plt.ylabel('Predicted Values')
plt.legend(loc='upper right')
plt.grid(True)

# Adding annotations
plt.text(0.05, 0.95, f'R²: {r2_xgb:.2f}\nCorrelation: {correlation_xgb:.2f}',
transform=plt.gca().transAxes,
      fontsize=12, verticalalignment='top', bbox=dict(boxstyle='round',
facecolor='white', alpha=0.5))

plt.show()

#Feature Importance----------------------------------------------------
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

# Get the fitted random forest model from the pipeline
rf_best_model = rf_random_search.best_estimator_.named_steps['regressor']

# Get the feature importance from the best RandomForest model
feature_importances = rf_best_model.feature_importances_

# Get feature names (numerical + encoded categorical)
num_features = numerical_features
cat_features =
rf_random_search.best_estimator_.named_steps['preprocessor'].named_transfor
mers_['cat'].get_feature_names_out(categorical_features)
all_features = np.concatenate([num_features, cat_features])

# Create a DataFrame for feature importances
feature_importance_df = pd.DataFrame({
   'Feature': all_features,
   'Importance': feature_importances
})
```

```
# Aggregate the importance for categorical features by their original feature
def aggregate_feature_importance(df, num_features, cat_features):
    # Initialize a dictionary to store the aggregate importances
    aggregated_importances = {}

    # Add numerical feature importances directly
    for feature in num_features:
        aggregated_importances[feature] = df[df['Feature'] ==
feature]['Importance'].sum()

    # Aggregate categorical features by their original feature
    for cat_feature in cat_features:
        # Find all the one-hot encoded features related to this categorical feature
        related_features = [col for col in df['Feature'] if col.startswith(cat_feature)]
        # Sum the importances of these related features
        aggregated_importances[cat_feature] =
df[df['Feature'].isin(related_features)]['Importance'].sum()

    return pd.DataFrame(list(aggregated_importances.items()), columns=['Feature',
'Importance'])

# Aggregate feature importances
aggregated_feature_importance_df =
aggregate_feature_importance(feature_importance_df, numerical_features,
categorical_features)

# Sort the features by importance
top_10_aggregated_features =
aggregated_feature_importance_df.sort_values(by='Importance',
ascending=False).head(10)

# Plot the top 10 important features after aggregation
plt.figure(figsize=(10, 6))
plt.barh(top_10_aggregated_features['Feature'],
top_10_aggregated_features['Importance'], color='skyblue')
plt.xlabel('Importance')
plt.title('Top 10 Important Features (Aggregated) from Random Forest')
plt.gca().invert_yaxis()  # Invert y-axis to have the highest at the top
plt.tight_layout()
plt.show()

# Comparing the Models  Before Tuning----------------------------------------------------
----------
import matplotlib.pyplot as plt

# Data for each model and metric
models = ['Linear Regression', 'Random Forest', 'Gradient Boosting', 'XGBoost']
mse = [0.2710665687697467, 0.03834502162991697, 0.23524440093162938,
0.17645180248466186]  # Mean Squared Error
```

```
r_squared = [0.7638863997266039, 0.9665994181772687, 0.7950894398368005,
0.846300963824228]  # R-squared
mae = [0.37525006633496233, 0.0728182117330136, 0.3344735113319631,
0.28371716151019927]  # Mean Absolute Error
median_ae = [0.2850329309691064, 1.1253220577600587e-12,
0.26035236448284493, 0.20684327719936135]  # Median Absolute Error
max_error = [3.9506479341410774, 4.090452380952381, 4.044598620703611,
3.6791024208068848]  # Maximum Error
explained_variance = [0.7638865504365535, 0.9665995742259247,
0.7950897449112317, 0.846302141396944]  # Explained Variance

# Create subplots
fig, axs = plt.subplots(2, 3, figsize=(18, 10))

# Function to add value labels
def add_labels(bars, axis):
    for bar in bars:
        height = bar.get_height()
        axis.annotate(f'{height:.2f}',
                      xy=(bar.get_x() + bar.get_width() / 2, height),
                      xytext=(0, 3),  # 3 points vertical offset
                      textcoords="offset points",
                      ha='center', va='bottom')

# Plotting and adding value labels
for i, ax in enumerate(axs.flat):
    color = 'lightcoral'
    if i == 0:
        bars = ax.bar(models, mse, color=color)
        ax.set_title('Mean Squared Error (MSE)')
        ax.set_ylabel('MSE')
    elif i == 1:
        bars = ax.bar(models, mae, color=color)
        ax.set_title('Mean Absolute Error (MAE)')
        ax.set_ylabel('MAE')
    elif i == 2:
        bars = ax.bar(models, median_ae, color=color)
        ax.set_title('Median Absolute Error')
        ax.set_ylabel('Median AE')
    elif i == 3:
        bars = ax.bar(models, max_error, color=color)
        ax.set_title('Maximum Error')
        ax.set_ylabel('Max Error')
    elif i == 4:
        bars = ax.bar(models, explained_variance, color=color)
        ax.set_title('Explained Variance')
        ax.set_ylabel('Explained Variance')
    else:
        ax.axis('off')
```

```
    add_labels(bars, ax)
    ax.set_xticklabels(models, rotation=45, ha='right')

# Add main title
fig.suptitle('Comparison of the Models Before Tuning', fontsize=16,
fontweight='bold')

# Adjust layout
plt.tight_layout()

# Display the plot
plt.show()

# Comparing the Models after Tuning----------------------------------------------------------
------------------
import matplotlib.pyplot as plt
# Data for each model and metric after tuning
models = ['Linear Regression', 'Random Forest', 'Gradient Boosting', 'XGBoost']
mse = [0.2710665687697467, 0.14127502008252243, 0.18143471000547312,
0.10336777314265935]  # Mean Squared Error
mae = [0.37525006633496233, 0.24717794164560178, 0.2888185046518539,
0.20359720469809]  # Mean Absolute Error
median_ae = [0.2850329309691064, 0.1714707124128365,
0.21478541654941719, 0.126236534118652]  # Median Absolute Error
max_error = [3.9506479341410774, 3.8843630367226636, 4.038041175164341,
3.6104869842529297]  # Maximum Error
explained_variance = [0.7638865504365535, 0.8769418498961854,
0.8419613934662463, 0.909961564850776]  # Explained Variance

# Create subplots
fig, axs = plt.subplots(2, 3, figsize=(18, 10))

# Function to add value labels
def add_labels(bars, axis):
    for bar in bars:
        height = bar.get_height()
        axis.annotate(f'{height:.2f}',
                      xy=(bar.get_x() + bar.get_width() / 2, height),
                      xytext=(0, 3),  # 3 points vertical offset
                      textcoords="offset points",
                      ha='center', va='bottom')

# Plotting and adding value labels
for i, ax in enumerate(axs.flat):
    if i == 0:
        bars = ax.bar(models, mse, color='red')
        ax.set_title('Mean Squared Error (MSE)')
        ax.set_ylabel('MSE')
```

```
    elif i == 1:
        bars = ax.bar(models, mae, color='red')
        ax.set_title('Mean Absolute Error (MAE)')
        ax.set_ylabel('MAE')
    elif i == 2:
        bars = ax.bar(models, median_ae, color='red')
        ax.set_title('Median Absolute Error')
        ax.set_ylabel('Median AE')
    elif i == 3:
        bars = ax.bar(models, max_error, color='red')
        ax.set_title('Maximum Error')
        ax.set_ylabel('Max Error')
    elif i == 4:
        bars = ax.bar(models, explained_variance, color='red')
        ax.set_title('Explained Variance')
        ax.set_ylabel('Explained Variance')
    else:
        ax.axis('off')

    add_labels(bars, ax)
    ax.set_xticklabels(models, rotation=45, ha='right')

# Add main title
fig.suptitle('Comparison of the Models After Tuning', fontsize=16,
fontweight='bold')

# Adjust layout
plt.tight_layout()

# Display the plot
plt.show()

#Error Metrics and R² Score Comparison before Tuning ----------------------------------
-------------------------
import matplotlib.pyplot as plt

# Data setup
models = ['Linear Regression', 'Random Forest', 'Gradient Boosting', 'XGBoost']
rmse = [0.5206405370020152, 0.1958188490159131, 0.48502000054804895,
0.4200616650977114]
huber_loss = [0.12136211199915771, 0.017936209819826937,
0.10640086509342019, 0.08225067424770932]
log_cosh_loss = [0.1110809587483981, 0.016433679132679648,
0.09745010332265062, 0.07554176268918415]
r_squared = [0.7638863997266039, 0.9665994181772687, 0.7950894398368005,
0.846300963824228]

# Create the plot
fig, ax1 = plt.subplots(figsize=(14, 7))
```

```
# Bar plots with light colors
width = 0.2
x = range(len(models))
bars1 = ax1.bar(x, rmse, width, label='RMSE', color='lightblue')
bars2 = ax1.bar([p + width for p in x], huber_loss, width, label='Huber Loss',
color='peachpuff')
bars3 = ax1.bar([p + width * 2 for p in x], log_cosh_loss, width, label='Log-Cosh
Loss', color='lightgreen')
ax1.set_ylabel('Error Metrics')
ax1.set_title('Error Metrics and R^2 Score Comparison of Models before Tuning')
ax1.set_xticks([p + width for p in x])
ax1.set_xticklabels(models)
ax1.legend(loc='upper right')

# Add labels on the bars
def add_labels(bars):
  for bar in bars:
    height = bar.get_height()
    ax1.annotate(f'{height:.2f}',
          xy=(bar.get_x() + bar.get_width() / 2, height),
          xytext=(0, 3),  # 3 points vertical offset
          textcoords="offset points",
          ha='center', va='bottom')

add_labels(bars1)
add_labels(bars2)
add_labels(bars3)

# Line plot for R-squared
ax2 = ax1.twinx()
ax2.plot(models, r_squared, 'r-o', label='R^2 Score')
ax2.set_ylabel('R^2 Score')
ax2.legend(loc='upper right', bbox_to_anchor=(1, 0.85))

# Show plot
plt.show()

#Error Metrics and R^2 Score Comparison After Tuning ----------------------------------
------------------------
import matplotlib.pyplot as plt

# Data setup
models = ['Linear Regression', 'Random Forest', 'Gradient Boosting', 'XGBoost']
rmse = [0.5206, 0.3759, 0.4260, 0.3215]
huber_loss = [0.12136211199915771, 0.06629939535459012,
0.08442972032604971, 0.04909373292032188]
log_cosh_loss = [0.1110809587483981, 0.060992820100191254,
0.0775328580009213, 0.045441521677047715]
```

```
r_squared = [0.7639, 0.8769, 0.8420, 0.9100]

# Create the plot
fig, ax1 = plt.subplots(figsize=(14, 7))

# Bar plots with dark colors
width = 0.2
x = range(len(models))
bars1 = ax1.bar(x, rmse, width, label='RMSE', color='navy')
bars2 = ax1.bar([p + width for p in x], huber_loss, width, label='Huber Loss',
color='lightcoral')
bars3 = ax1.bar([p + width * 2 for p in x], log_cosh_loss, width, label='Log-Cosh
Loss', color='green')
ax1.set_ylabel('Error Metrics')
ax1.set_title('Error Metrics and R^2 Score Comparison of Models after Tuning')
ax1.set_xticks([p + width for p in x])
ax1.set_xticklabels(models)
ax1.legend(loc='upper right')

# Add labels on the bars
def add_labels(bars):
   for bar in bars:
      height = bar.get_height()
      ax1.annotate(f'{height:.2f}',
                  xy=(bar.get_x() + bar.get_width() / 2, height),
                  xytext=(0, 3),  # 3 points vertical offset
                  textcoords="offset points",
                  ha='center', va='bottom')

add_labels(bars1)
add_labels(bars2)
add_labels(bars3)

# Line plot for R-squared
ax2 = ax1.twinx()
ax2.plot(models, r_squared, 'r-o', label='R^2 Score')
ax2.set_ylabel('R^2 Score')
ax2.legend(loc='upper right', bbox_to_anchor=(1, 0.85))

# Show plot
plt.show()
```