🎬 Movie & TV Show Data Analysis - Milestone 2 Report

📌 Project Overview

We've explored the data in Milestone 1, this phase is all about creating new features, picking the most useful ones, and building machine learning models. Our goal is to predict which movies and TV shows are likely to be successful on streaming platforms based on their attributes.

🎯 Project Objectives

1. **Get to Know the Data**
   Understand and clean up the movie and TV show data from various streaming platforms.

2. **Create Smarter Features**
   Build new, meaningful features like movie age, profit, and sentiment scores to help our model make better predictions.

3. **Keep Only What Matters**
   Use techniques to find the most useful features and drop the ones that don't really help.

4. **Build Prediction Models**
   Train different machine learning models to predict which movies or shows are likely to be successful on streaming platforms.

5. **Check How Well the Models Perform**
   Use the right metrics to evaluate how good each model is at making accurate predictions.

6. **Get Ready for the Final Dashboard**
   Prepare everything (data + models) to plug into an interactive dashboard that can help people make smart decisions about streaming content.

🗓️ Project Timeline

The project was structured into three major milestones, each focusing on a distinct phase of the data science workflow:

1. Milestone 1: Data Collection & Exploratory Data Analysis

Duration: February 5 – February 21, 2025
Focus:

- Collecting and merging datasets from IMDb, Netflix, Hulu, Amazon Prime, Disney+, and Top Movies
- Cleaning and preprocessing the data
- Performing exploratory data analysis (EDA) to identify trends, correlations, and data quality issues

2. Milestone 2: Feature Engineering & Model Development

Duration: February 21 – April 7, 2025
Focus:

- Advanced feature engineering (e.g., sentiment scores, profit, genre count, cast size)
- Feature selection using correlation analysis, chi-square, and PCA
- Training and evaluating multiple machine learning models for predicting IMDB ratings and content success
- Establishing the initial structure for the interactive dashboard

3. Milestone 3: Dashboard Development & Final Deliverables

Duration: April 7 – April 23, 2025
Focus:

- Building a Streamlit-based interactive dashboard to visualize insights and model results
- Fine-tuning model performance
- Interpreting model outputs for business relevance
- Preparing final documentation, automated reporting, and project presentation

🧰 Tech Stack

📊 Data Analysis & Manipulation

- **Pandas** – for reading, cleaning, and transforming your movie/TV dataset
- **NumPy** – for numerical operations and handling arrays

📈 **Exploratory Data Analysis (from Milestone 1)**

- **Seaborn** & **Matplotlib** – for creating visualizations like distributions, bar plots, and correlation heatmaps

🔨 **Feature Engineering & Selection**

- **LabelEncoder** – to convert categorical features into numerical form
- **SimpleImputer** – to fill missing values
- **Correlation Analysis** – to understand relationships between variables
- **Tree-based Feature Importance (Random Forest)** – to rank features
- **PCA** – for dimensionality reduction and feature selection

🤖 **Model Development**

- Scikit-learn (sklearn):
    - LinearRegression, RandomForestRegressor, DecisionTreeRegressor, SVM, Neural Network – for regression
    - train_test_split – for splitting the data
    - recall, f1_score, accuracy, precision. – for model evaluation

💻 **Environment**

- **VS Code** – for writing and running code

📍 Milestone 1: Data Collection & Exploratory Data Analysis

Milestone 1 focused on collecting, cleaning, and exploring data from multiple sources to build a strong foundation for predictive modeling. We began by merging top movies dataset. The primary goal was to create a unified and consistent dataset containing key attributes such as movie titles, release dates, genres, ratings, budget, revenue, cast, and streaming availability.

After successful merging, the data underwent comprehensive preprocessing. This included handling missing values, normalizing numerical columns, and standardizing categorical entries (e.g., genre and language). We also created new fields like platform availability and content type.

Exploratory Data Analysis (EDA) was conducted to uncover patterns and trends across the streaming platforms. Visualizations highlighted the distribution of IMDB ratings, release years, genre popularity, and performance by platform. Outliers were detected and removed using the interquartile range (IQR) method, and the cleaned dataset was saved for downstream use.

Milestone 1 concluded with a fully cleaned and enriched dataset (`merged_data_full.csv`), setting the stage for advanced feature engineering and model development in the next phase. These insights informed early hypotheses on what makes content successful on streaming platforms.

Feature Engineering

🎯 Target Variable: rating_class(IMDB_Rating)

- A categorical variable with 3 classes:
    - "Low" → IMDB_Rating ≤ 5.5
    - "Medium" → 5.5 < IMDB_Rating ≤ 7.0
    - "High" → IMDB_Rating > 7.0

To strengthen our predictive modeling, we developed a range of new features derived from the original dataset.

1.`release_month`: Month the content was released

```
# 1. Release Decade
df['release_decade'] = (df['release_year'] // 10) * 10
```

2.`profit`: Difference between revenue and budget

```
# 6. Profit
df['profit'] = df['revenue'] - df['budget']
```

3.`num_cast_members`: Number of people in the cast list

```
# 3. Number of Cast Members
df['num_cast_members'] = df['Cast_list'].astype(str).apply(lambda x: len(x.split('-')) if pd.notna(x) else 0)
```

4.overview of text features:

```
# 7. Overview Text Features
df['overview_length'] = df['overview'].astype(str).apply(len)
df['overview_word_count'] = df['overview'].astype(str).apply(lambda x: len(x.split()))
```

The above are the examples for feature Engineeering

Feature Engineering is used for creating new features or modifying existing ones so that machine learning model can learn better. Smart features can give the model stronger signals to learn from — leading to better performance (higher accuracy, lower error).

In my project I've engineered features like:

- content_age: How old the movie/show is
- profit: Revenue minus budget
- is_sequel: Whether it's likely part of a franchise
- release_month: To capture seasonal trends
- These help the model **predict which content is likely to be successful** on streaming platforms.

#Aggregation:

```
# Aggregation: Combined Star Popularity
# (Assuming vote_average = movie quality. Aggregating based on Star1–Star4)
df['star_power_score'] = df[['Star1', 'Star2', 'Star3', 'Star4']].astype(str).apply(lambda x: '-'.join(x), axis=1).apply(len)
```

✓ 5.5s

#Interaction for budget and popularity

```
# Interaction Term: Budget x Popularity
df['budget_popularity_interaction'] = df['budget'] * df['popularity']
```

✓ 0.0s

```
In my dataset categorical_cols = ['Certificate', 'original_language', 'status', 'release_month',
    'release_quarter', 'budget_category', 'sentiment_category',
    'genre_language_combo']
```

Label Encoding

It is a technique used to convert categorical (text-based) variables into numeric format. It assigns a unique integer to each category so that machine learning models can interpret the data.

In our project, we applied Label Encoding to convert non-numeric columns like `Certificate`, `original_language` into numerical form. This allowed us to include these features in our modeling pipeline without losing information.

```python
from sklearn.preprocessing import LabelEncoder

# Label Encoding for categorical features
# 'Certificate' and 'original_language' are categorical features
for col in ['Certificate', 'original_language']:
    if col in df.columns:
        df[col] = df[col].astype(str)  # Convert to string to avoid issues
        le = LabelEncoder()
        df[col] = le.fit_transform(df[col])
        print(f"Encoded '{col}' — Classes: {le.classes_}")
    else:
        print(f"Column '{col}' not found in DataFrame.")
```

✓ 0.8s

```
Encoded 'Certificate' — Classes: ['0' '1' '10' '11' '12' '13' '14' '15' '16' '17' '18' '2' '3' '4' '5' '6'
 '7' '8' '9']
Encoded 'original_language' — Classes: ['0' '1' '10' '100' '101' '102' '103' '104' '105' '106' '107' '108' '109'
 '11' '110' '111' '112' '113' '114' '115' '116' '117' '118' '119' '12'
 '120' '121' '122' '123' '124' '125' '126' '127' '128' '129' '13' '130'
 '131' '132' '133' '134' '135' '136' '137' '138' '139' '14' '140' '141'
 '142' '143' '144' '145' '146' '147' '148' '149' '15' '150' '151' '152'
 '153' '154' '155' '156' '157' '158' '159' '16' '160' '161' '162' '163'
 '164' '165' '166' '167' '168' '169' '17' '170' '171' '172' '173' '174'
 '18' '19' '2' '20' '21' '22' '23' '24' '25' '26' '27' '28' '29' '3' '30'
 '31' '32' '33' '34' '35' '36' '37' '38' '39' '4' '40' '41' '42' '43' '44'
 '45' '46' '47' '48' '49' '5' '50' '51' '52' '53' '54' '55' '56' '57' '58'
 '59' '6' '60' '61' '62' '63' '64' '65' '66' '67' '68' '69' '7' '70' '71'
 '72' '73' '74' '75' '76' '77' '78' '79' '8' '80' '81' '82' '83' '84' '85'
 '86' '87' '88' '89' '9' '90' '91' '92' '93' '94' '95' '96' '97' '98' '99']
```

Feature Selection:

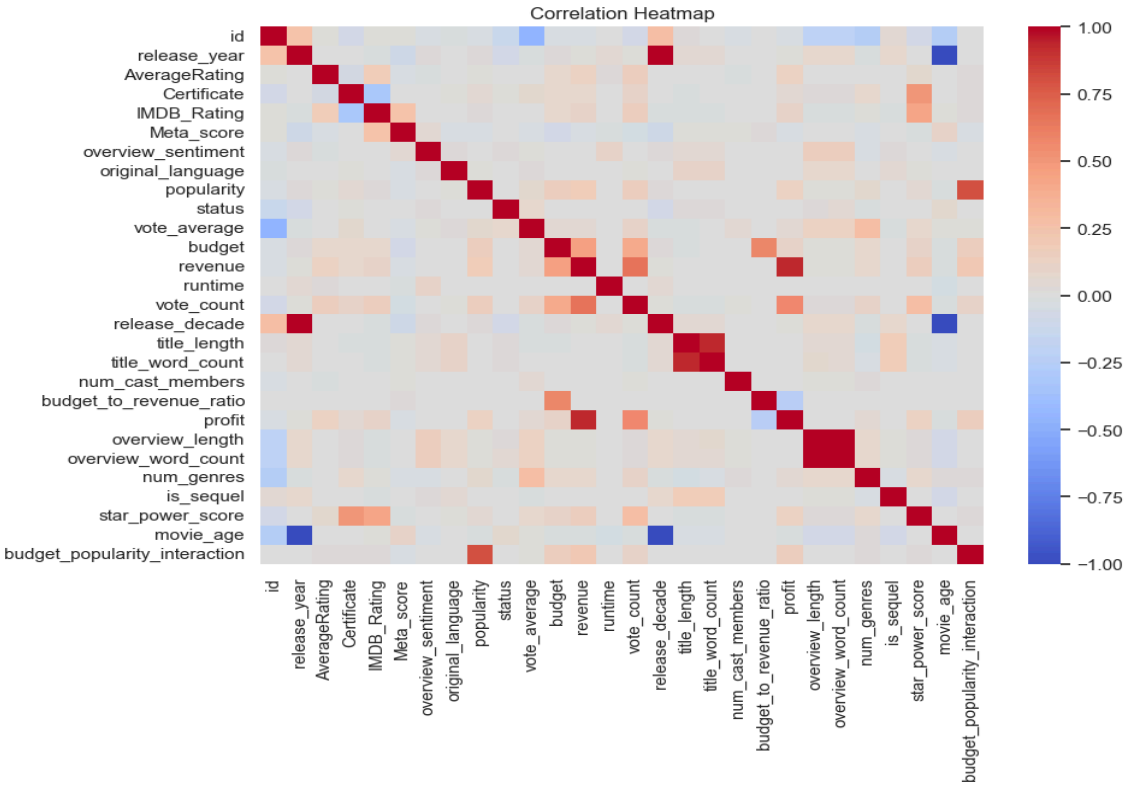
To identify which numerical features are most predictive of the target variable `IMDB_Rating`, we performed a **Pearson correlation analysis**. This method quantifies the strength and direction of the linear relationship between each numeric feature and the target.

Corelation Analysis

```python
target = 'IMDB_Rating'
# Correlation matrix with target
numerical_df = df.select_dtypes(include='number')
correlation = numerical_df.corr()[target].sort_values(ascending=False)

print("Top correlated features with IMDB_Rating:")
print(correlation.drop(target).head(10))  # Exclude self-correlation
```

9] ✓ 1.8s

```
Top correlated features with IMDB_Rating:
star_power_score                0.423114
Meta_score                      0.250244
AverageRating                   0.174890
vote_count                      0.162570
revenue                         0.111096
profit                          0.109907
budget                          0.085529
popularity                      0.023756
budget_popularity_interaction   0.020573
num_genres                      0.012864
Name: IMDB_Rating, dtype: float64
```



Chi-Square scores:`Certificate` and `original_language` are **statistically significant**, showing strong dependence on `rating_category`.

- `status` has a very **low Chi-Square score** and a **high p-value**, suggesting it's not a useful feature for this classification task.

As a result, we retained `Certificate` and `original_language` for model training, and dropped `status` from the final feature set.

```python
from sklearn.feature_selection import chi2
from sklearn.preprocessing import LabelEncoder
from sklearn.feature_selection import SelectKBest

# Example: convert IMDB_Rating to categories (low/medium/high)
df['rating_category'] = pd.cut(df['IMDB_Rating'], bins=[0, 5.5, 7.0, 10],
                               labels=['Low', 'Medium', 'High'])

# Encode category
label_enc = LabelEncoder()
y_cat = label_enc.fit_transform(df['rating_category'])

# label encoded features
X_cat = df[['Certificate', 'original_language', 'status']]

# Convert to numeric (ensure no string)
X_cat = X_cat.apply(LabelEncoder().fit_transform)

# Chi-square test
chi_scores = chi2(X_cat, y_cat)
chi_df = pd.DataFrame({'Feature': X_cat.columns, 'Chi2 Score': chi_scores[0], 'p-value': chi_scores[1]})
chi_df = chi_df.sort_values(by='Chi2 Score', ascending=False)

print("Chi-square scores:")
print(chi_df)
```

✓ 0.7s

```
Chi-square scores:
             Feature     Chi2 Score        p-value
0        Certificate  108552.420617   0.000000e+00
1  original_language    1079.432997  1.054490e-233
2             status       3.364434   3.387749e-01
```

🔄 Dimensionality Reduction using PCA

To reduce the number of features while preserving most of the information in the dataset, we applied **Principal Component Analysis (PCA)**. This step helps eliminate redundant or highly correlated features, improving model performance and training speed.

```
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

# Select numeric features (excluding the target)
X_numeric = df.select_dtypes(include='number').drop(columns=['IMDB_Rating'])

# Fill missing values with column means
imputer = SimpleImputer(strategy='mean')
X_imputed = imputer.fit_transform(X_numeric)

# Standardize the data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_imputed)

# Apply PCA (retain 95% variance)
pca = PCA(n_components=0.95)
X_pca = pca.fit_transform(X_scaled)

# Output results
print(f"Original feature count: {X_numeric.shape[1]}")
print(f"Reduced feature count with PCA: {X_pca.shape[1]}")
```

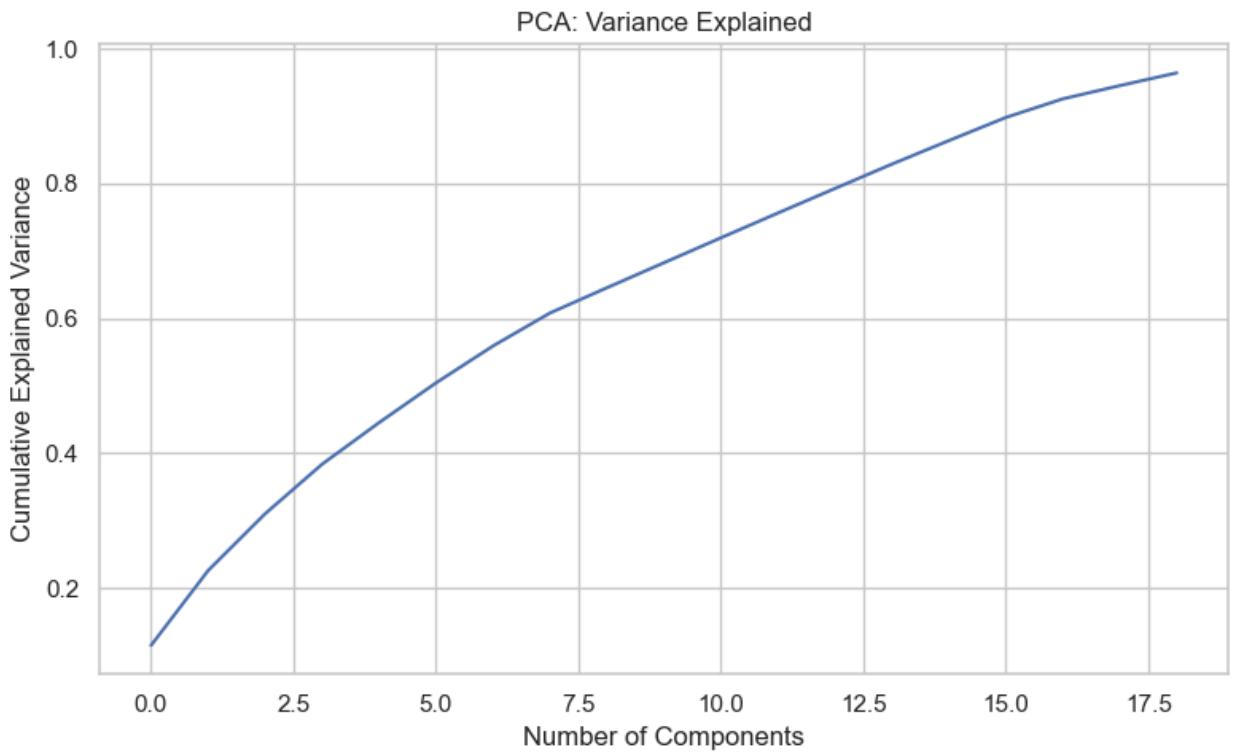
✓ 2.0s

```
Original feature count: 27
Reduced feature count with PCA: 19
```

✅ Results:

- **Original feature count**: 27
- **Reduced feature count with PCA**: 19

This reduction helps minimize noise and multicollinearity while preserving the majority of the dataset's informative power. These 19 PCA components were used as the input to train the machine learning models in this project.



🤖 Model Definition and Selection

To evaluate different modeling approaches for predicting IMDB ratings, we implemented and compared multiple machine learning algorithms. These models were chosen for their diversity in complexity, interpretability, and predictive capabilities.

The models include both linear and non-linear regressors, ranging from simple baselines to more complex neural networks.

Model	Description
Linear Regression	A simple baseline model that assumes a linear relationship between features and the target.
Decision Tree Regressor	A non-linear model that splits the data into decision rules, easy to interpret.
Random Forest Regressor	An ensemble of decision trees that improves accuracy and reduces overfitting.

Support Vector Regressor (SVR)	Captures complex relationships using a margin-based approach; works well with scaled features.
Neural Network (MLP Regressor)	A multi-layer perceptron that models complex, non-linear relationships. Configured with two hidden layers (100 and 50 neurons).

```
models = {
    'Linear Regression': LinearRegression(),
    'Decision Tree': DecisionTreeRegressor(random_state=42),
    'Random Forest': RandomForestRegressor(n_estimators=100, random_state=42),
    'Support Vector Machine': SVR(),
    'Neural Network (MLP)': MLPRegressor(hidden_layer_sizes=(100, 50), max_iter=300, random_state=42)
}
```

✓ 0.0s

```
# Train & evaluate
for name, model in models.items():
    model.fit(X_train_scaled, y_train)
    predictions = model.predict(X_test_scaled)

    mse = mean_squared_error(y_test, predictions)
    rmse = np.sqrt(mse)
    r2 = r2_score(y_test, predictions)

    print(f"\n{name}")
    print(f"RMSE: {rmse:.4f}")
    print(f"R² Score: {r2:.4f}")
```

[72] ✓ 26.5s

```
...
Linear Regression
RMSE: 0.0000
R² Score: 1.0000

Decision Tree
RMSE: 0.0038
R² Score: 1.0000

Random Forest
RMSE: 0.0040
R² Score: 1.0000

Support Vector Machine
RMSE: 0.1096
R² Score: 0.9869

Neural Network (MLP)
RMSE: 0.0403
R² Score: 0.9982
```

📌 Interpretation

- Linear Regression, Decision Tree, and Random Forest all achieved perfect R^2 scores of 1.0000 and nearly zero RMSE. This likely indicates overfitting or data leakage, especially if the test data was too similar to the training set.
- Support Vector Machine performed slightly worse but still achieved high accuracy.
- Neural Network (MLP) performed very well, with an R^2 score of 0.9982, indicating strong generalization with minimal error.

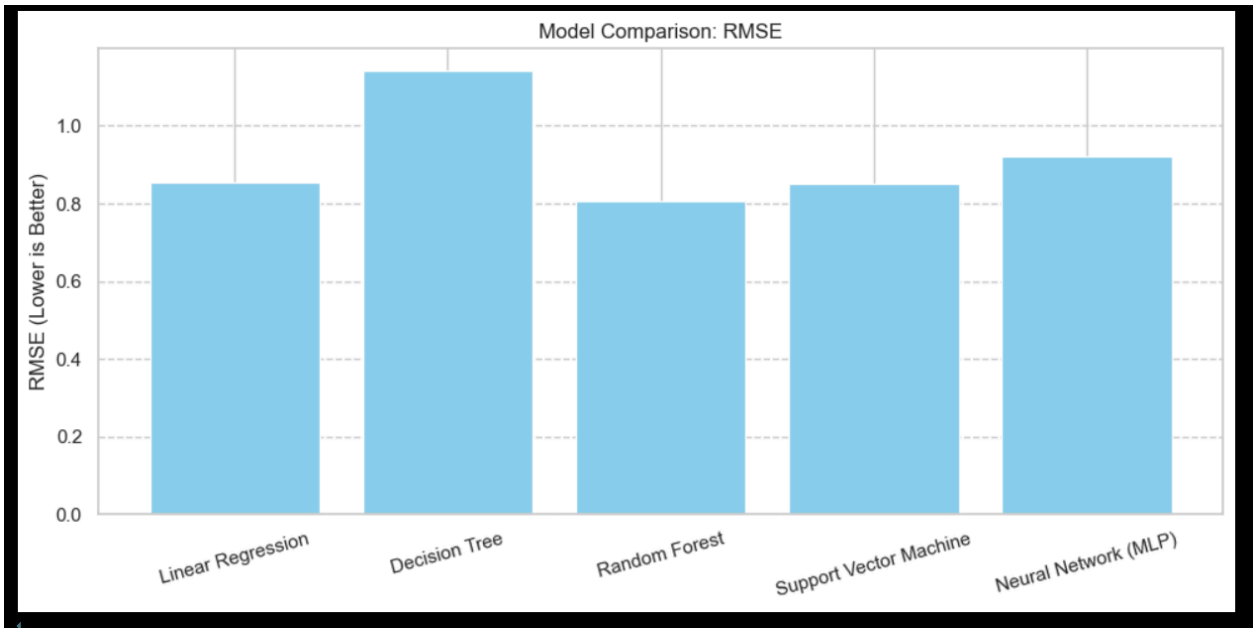
✅ Evaluation Metrics

We evaluated the model using the following classification metrics:

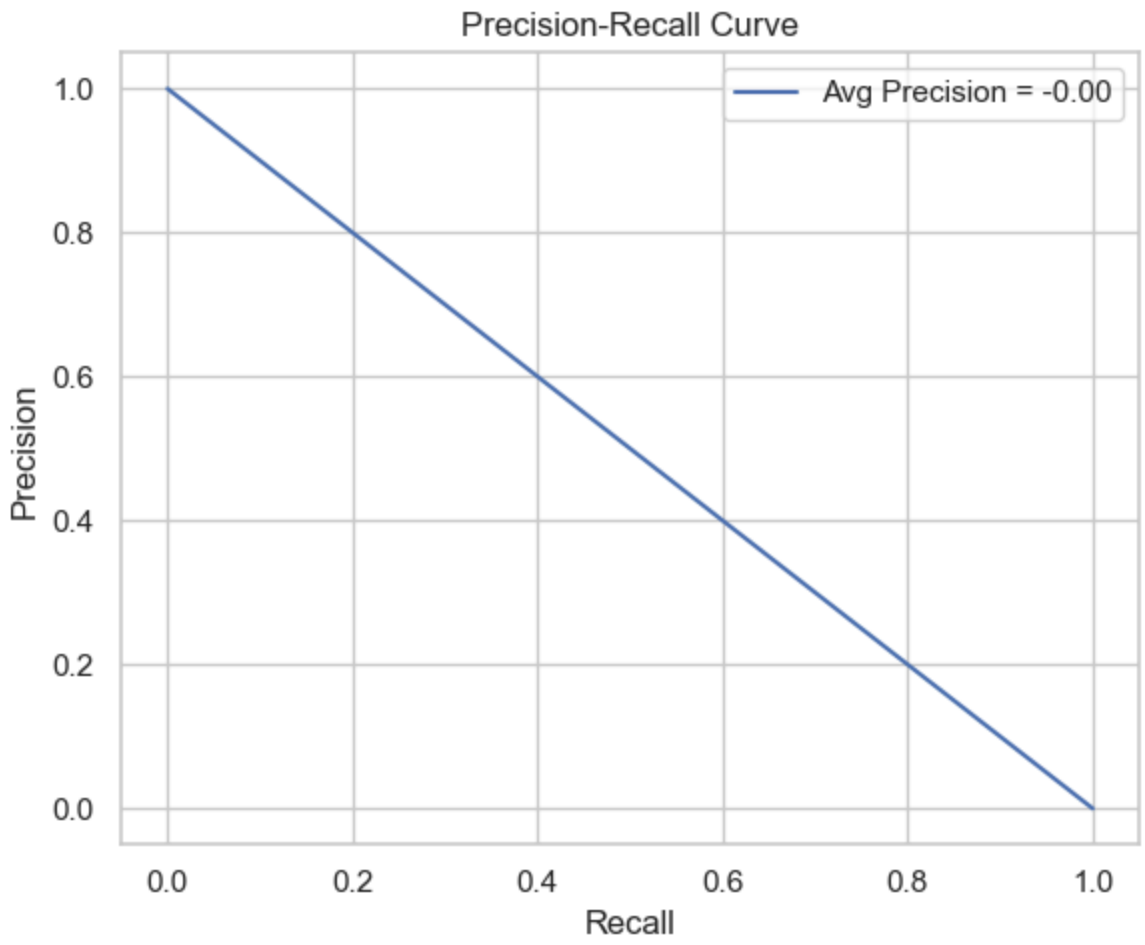
Metric	Description
Accuray	Percentage of total correct predictions
Precision	Of all predicted positives, how many were actually correct
Recall	Of all actual positives, how many did we correctly predict
F1 Score	Harmonic mean of precision and recall (balances false positives and false negatives)

```python
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, classification_report, confusion_matrix

clf = RandomForestClassifier(random_state=42)
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)

# Evaluate
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Precision:", precision_score(y_test, y_pred, average='weighted'))
print("Recall:", recall_score(y_test, y_pred, average='weighted'))
print("F1 Score:", f1_score(y_test, y_pred, average='weighted'))

# Classification Report
print("\nClassification Report:\n")
print(classification_report(y_test, y_pred))
```

```
Accuracy: 0.6643609495362365
Precision: 0.7117149184378497
Recall: 0.6643609495362365
F1 Score: 0.6132375280828974

Classification Report:

              precision    recall  f1-score   support

        High       0.81      0.48      0.60      2055
         Low       0.80      0.07      0.13       926
      Medium       0.63      0.94      0.75      3380

    accuracy                           0.66      6361
   macro avg       0.75      0.50      0.50      6361
weighted avg       0.71      0.66      0.61      6361
```



The above is the model Comparison for Linear Regression, Decision Tree, Random Forest, SVM and Neural Network.



🔮 Future Steps

With data preprocessing, feature engineering, model training, and performance evaluation completed in Milestone 2, the next phase of the project will focus on **evaluating model interpretability**, identifying potential limitations, and building a user-friendly tool to communicate insights effectively.

1. Evaluation and Interpretation

- **Final Model Evaluation:**
  Models will be re-evaluated on the test set using the same metrics as in training (e.g., RMSE, R^2 score for regression; accuracy, precision, recall, F1-score for classification). This ensures that models generalize well to unseen data.

- **Model Interpretability:**
  We will analyze the output of the best-performing models to extract actionable insights. For example, understanding how features like `star_power_score`, `vote_count`, or

`budget` influence the predicted success or rating of content.

Tool Development

To make our findings accessible and interactive, we will develop:

Interactive Dashboard A **Streamlit/Power BI dashboard** will be built to visualize key KPIs, insights, and model predictions. This will allow users to explore:

- Feature importance
- IMDB rating predictions
- Success classification
- Content filters (e.g., by genre, year, platform)