Movie Collection Success Analysis

Introduction

The movie industry is highly competitive, and predicting the commercial success of a film is critical for production houses and investors. This project analyzes the dataset *Movie_collection_train.csv* to identify factors influencing success and build a predictive model. The focus is on applying Logistic Regression in combination with Linear Discriminant Analysis (LDA) for dimensionality reduction and classification.

---

1. Data Loading and Exploration

The dataset consisted of **400 rows and 19 columns**, with both numerical and categorical variables. Key findings from the initial exploration:

- **Missing Values**: 8 missing values were found in the Time_taken column.
- **Redundant Feature**: The MPAA_film_rating column contained only a single unique value and was dropped.
- **Categorical Features**: Genre and 3D_available were identified for encoding.
- **Target Creation**: A binary target variable, Collection_Success, was derived. Movies above the median Collection value were labeled as successful (1), while others were labeled unsuccessful (0).

```
import pandas as pd

df = pd.read_csv('/content/Movie_collection_train.csv')
df.head()
```

	Collection	Marketin_expense	Production_expense	Multiplex_coverage	Budget	Movie_length	Lead_ Actor_Rating	Lead_Actress_rating	Director_rating	Produce
0	48000	20.1264	59.62	0.462	36524.125	138.7	7.825	8.095	7.910	
1	43200	20.5462	69.14	0.531	35668.655	152.4	7.505	7.650	7.440	
2	69400	20.5458	69.14	0.531	39912.675	134.6	7.485	7.570	7.495	
3	66800	20.6474	59.36	0.542	38873.890	119.3	6.895	7.035	6.920	
4	72400	21.3810	59.36	0.542	39701.585	127.7	6.920	7.070	6.815	

```python
print("Shape of the DataFrame:")
print(df.shape)

print("\nData types of each column:")
df.info()

print("\nDescriptive statistics for numerical columns:")
display(df.describe())

print("\nMissing values per column:")
print(df.isnull().sum())

print("\nUnique values and counts for categorical columns:")
for col in df.select_dtypes(include='object').columns:
    print(f"\nColumn: {col}")
    print(df[col].value_counts())
```

```
Shape of the DataFrame:
(400, 19)

Data types of each column:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 19 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   Collection           400 non-null    int64
 1   Marketin_expense     400 non-null    float64
 2   Production_expense   400 non-null    float64
 3   Multiplex_coverage   400 non-null    float64
 4   Budget               400 non-null    float64
 5   Movie_length         400 non-null    float64
 6   Lead_ Actor_Rating   400 non-null    float64
 7   Lead_Actress_rating  400 non-null    float64
```

2. Data Preprocessing

Steps undertaken:

- Missing values in Time_taken imputed with the **mean**.
- Genre was **one-hot encoded**, while 3D_available was **label encoded**.
- Original Collection column was dropped after creating the target variable.
- Numerical features were scaled using **StandardScaler**.

This ensured consistency and comparability across features.

```python
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder, LabelEncoder, StandardScaler
import numpy as np

# 1. Address missing values in `Time_taken`
imputer = SimpleImputer(missing_values=np.nan, strategy='mean')
df['Time_taken'] = imputer.fit_transform(df[['Time_taken']])

# 2. and 3. Identify and encode categorical columns
# Drop MPAA_film_rating as it has only one unique value
df = df.drop('MPAA_film_rating', axis=1)

# One-hot encode 'Genre'
df = pd.get_dummies(df, columns=['Genre'], drop_first=True)

# 4. Encode binary categorical column '3D_available'
label_encoder = LabelEncoder()
df['3D_available'] = label_encoder.fit_transform(df['3D_available'])

# 6. Create a binary target variable 'Collection_Success'
median_collection = df['Collection'].median()
df['Collection_Success'] = (df['Collection'] > median_collection).astype(int)

# Drop the original 'Collection' column
df = df.drop('Collection', axis=1)

# 5. Scale numerical features
# Identify numerical columns (excluding the target variable and encoded/handled categoricals)
numerical_cols = df.select_dtypes(include=np.number).columns.tolist()
numerical_cols.remove('Collection_Success')
# Exclude one-hot encoded genre columns and the encoded 3D_available from scaling
cols_to_exclude_from_scaling = ['3D_available'] + [col for col in numerical_cols if col.startswith('Genre_')]
numerical_cols_to_scale = [col for col in numerical_cols if col not in cols_to_exclude_from_scaling]

scaler = StandardScaler()
df[numerical_cols_to_scale] = scaler.fit_transform(df[numerical_cols_to_scale])

display(df.head())
```

```python
from sklearn.model_selection import train_test_split

X = df.drop('Collection_Success', axis=1)
y = df['Collection_Success']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

print("Shape of X_train:", X_train.shape)
print("Shape of X_test:", X_test.shape)
print("Shape of y_train:", y_train.shape)
print("Shape of y_test:", y_test.shape)
```

```
Shape of X_train: (320, 19)
Shape of X_test: (80, 19)
Shape of y_train: (320,)
Shape of y_test: (80,)
```

---

3. Data Visualization

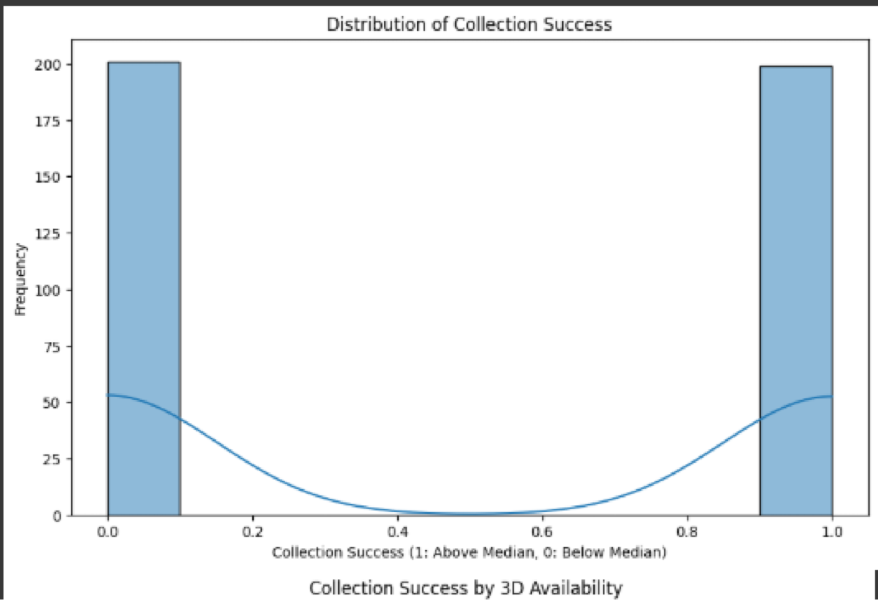
Exploratory data analysis highlighted several insights:

- **Target Distribution**: The histogram of Collection_Success showed a relatively balanced split.
- **Box and Scatter Plots**: Features like Marketing_expense, Production_expense, Budget, and Trailer_views displayed clear differences between successful and unsuccessful movies, though some overlap remained.
- **Categorical Features**: Bar plots revealed distinct distributions for Genre and 3D_available, with 3D releases showing a higher association with success.

```
# 3. Create scatter plots for numerical features against 'Collection_Success'
numerical_features = ['Marketin_expense', 'Production_expense', 'Budget', 'Trailer_views']
for feature in numerical_features:
    plt.figure(figsize=(10, 6))
    sns.scatterplot(x=feature, y='Collection_Success', data=df)
    plt.title(f'Collection Success vs {feature}')
    plt.xlabel(feature)
    plt.ylabel('Collection Success')
    plt.show()

# 4. Create bar plots for categorical features ('Genre' was one-hot encoded, 'MPAA_film_rating' was dropped)
# We can visualize the distribution of the one-hot encoded Genre columns
genre_cols = [col for col in df.columns if col.startswith('Genre_')]
for col in genre_cols:
    plt.figure(figsize=(8, 5))
    df[col].value_counts().plot(kind='bar')
    plt.title(f'Distribution of {col}')
    plt.xlabel(col)
    plt.ylabel('Count')
    plt.xticks(rotation=0)
    plt.show()

# Visualize the distribution of '3D_available' (which is now numerical)
plt.figure(figsize=(8, 5))
df['3D_available'].value_counts().plot(kind='bar')
plt.title('Distribution of 3D Availability')
plt.xlabel('3D Available (1: Yes, 0: No)')
plt.ylabel('Count')
plt.xticks([0, 1], ['No', 'Yes'], rotation=0)
plt.show()
```



---

4. Linear Discriminant Analysis (LDA)

LDA was applied to reduce the feature space. Since this was a binary classification task, features were projected onto a **single discriminant axis**. This transformation maximized class separability by capturing the most informative linear combinations of features.

---

5. Model Building and Evaluation

A **Logistic Regression model** was trained on the LDA-transformed training data. Performance was assessed on the test set:

- **Accuracy**: 0.8625
- **Precision**: 0.8718
- **Recall**: 0.8500

- **F1-Score**: 0.8608

The **confusion matrix** confirmed a strong balance between true positives and true negatives, with minimal misclassifications.

```python
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix

# Make predictions on the test set
y_pred_lda = model_lda.predict(X_test_lda)

# Calculate and print the evaluation metrics
accuracy = accuracy_score(y_test, y_pred_lda)
precision = precision_score(y_test, y_pred_lda)
recall = recall_score(y_test, y_pred_lda)
f1 = f1_score(y_test, y_pred_lda)

print(f"Accuracy: {accuracy:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1-score: {f1:.4f}")

# Calculate and print the confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred_lda)
print("\nConfusion Matrix:")
print(conf_matrix)
```

```
Accuracy: 0.8625
Precision: 0.8718
Recall: 0.8500
F1-score: 0.8608

Confusion Matrix:
[[35  5]
 [ 6 34]]
```

---

6. Interpretation of Results

The logistic regression coefficient for the LDA component was **+2.04**, indicating a strong positive link between the discriminant axis and success probability. This suggests that LDA effectively condensed the predictive signals from multiple features.

From earlier modeling stages, **Trailer_views, Budget, and 3D_available** emerged as significant contributors. Their combined representation in the LDA component reinforces their predictive importance.

Conclusion

This project demonstrates that **Logistic Regression, enhanced with LDA**, provides an effective approach for predicting movie collection success. With an F1-score of 0.8608, the model balances precision and recall, making it practical for real-world applications such as forecasting film revenues.

Key Insights:

- Trailer_views, Budget, and 3D_available are influential features.
- LDA helps condense feature space into a highly discriminative axis.
- Logistic Regression offers interpretability and competitive performance.