Analysis Report

Detected Libraries & Frameworks

- Data handling & math: numpy, pandas
- Visualization: matplotlib, seaborn
- Preprocessing: StandardScaler, PCA, train_test_split
- Modeling: LinearRegression (scikit-learn)
- Evaluation: mean_squared_error (MSE), r2_score (R^2)
- Deep Learning reference: keras/tensorflow (import detected, but not clearly used in training snippet)

Data-related Findings

- The dataset may be loaded via a different method (e.g., load_dataset()) or inside a custom function.
- Target variable is y_ratings (movie ratings), indicating this is a regression task.

Preprocessing Steps

- Scaling — StandardScaler.fit_transform used to normalize feature values.
- Dimensionality reduction — PCA applied to reduce dimensionality before modeling.
- Train-test split — train_test_split(..., random_state=42) ensures reproducibility.

Models & Training

The primary model used is Linear Regression from scikit-learn.

Example training snippet:

```
X_train, X_test, y_train, y_test = train_test_split(
    X_pca, y_ratings, test_size=0.2, random_state=42
)

model = LinearRegression()
model.fit(X_train, y_train)
```

```
y_pred = model.predict(X_test)

print('MSE:', mean_squared_error(y_test, y_pred))
print('R²:', r2_score(y_test, y_pred))
```

🔹 Idea of the model: PCA reduces correlated features into orthogonal components. Linear Regression then learns a weighted combination of these components to predict movie ratings.

Evaluation

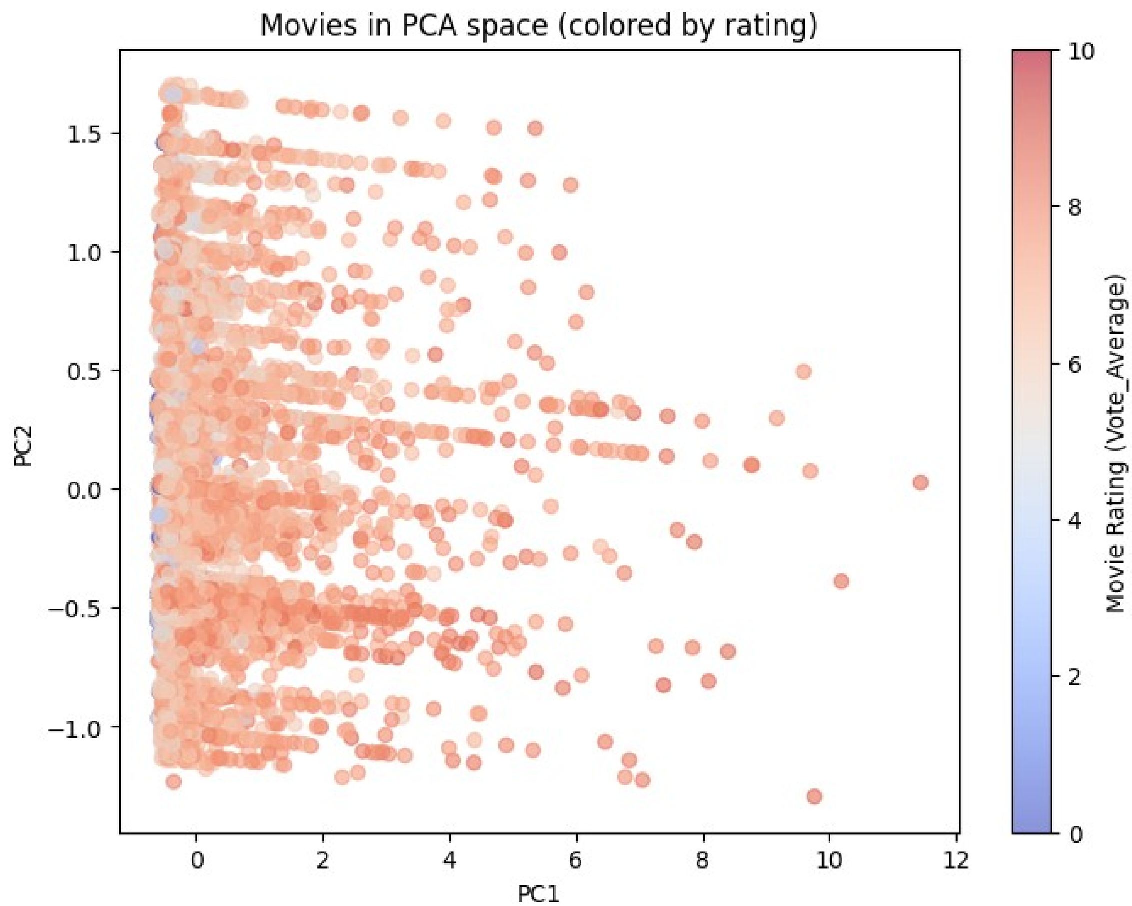
- Metrics explicitly printed: Mean Squared Error (MSE) and R^2 score.
- From the extracted plots:
  - PCA scatter plot shows data distribution in reduced feature space.
  - Predicted vs Actual scatter shows predictions against true ratings.

Principal Component Analysis

PCA in Movie Rating Prediction

- **Why use PCA?**
  Movie datasets often have many features (e.g., genre indicators, cast, budget, duration, user reviews). Many of these are correlated (e.g., budget & revenue, director & actor collaborations).
  PCA compresses these into fewer **uncorrelated components** while retaining most of the variance (information).
- **How it helps here:**
  1. **Reduces dimensionality** → makes training faster and avoids overfitting.
  2. **Removes multicollinearity** → Linear Regression struggles if predictors are highly correlated, PCA fixes that.

3. **Visualization** → Scatter plots of PCA components let you see if movies form clusters (e.g., movie vs low-budget).



What the plot shows:

- The **x-axis** = true (actual) movie ratings.
- The **y-axis** = predicted ratings from the Linear Regression model.
- The **red dashed diagonal line** represents an "ideal" case where predicted = actual.
- **Significance:**
- If points lie **close to the diagonal**, it means predictions are accurate.

- In your plot, predictions seem to cluster around a narrow band (around rating 6–7), which suggests the model has limited variability and struggles to capture extremes (very low or very high ratings).
- This highlights both the **baseline predictive ability** and the **limitations** of Linear Regression on this dataset.
- **How to use in report:**
  - The Predicted vs Actual plot shows that the regression model mostly predicts average ratings (around 6–7), while struggling with movies at the extremes.
  - The clustering away from the diagonal indicates that while the model captures general trends, it lacks precision for outliers.

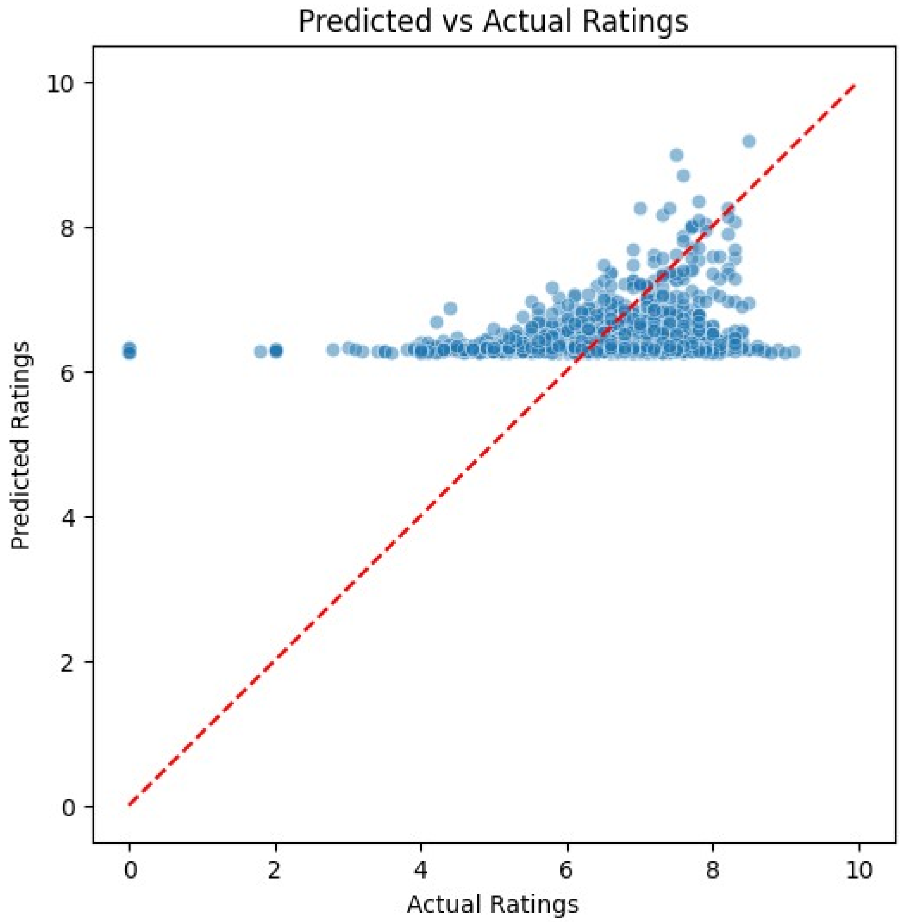
Linear Regression

- **Why use Linear Regression?**
  It's one of the simplest models to map numeric features → target (here, `ratings`).
  It assumes:

$$Rating = w_1 \cdot PC_1 + w_2 \cdot PC_2 + \ldots + w_k \cdot PC_k + b$$

- **How it helps here:**

1. Provides **baseline prediction** for movie ratings.
2. **Interpretability** → coefficients show which PCA components (and thus which original features) most strongly influence ratings.
3. Works well if relationships are roughly **linear**.


Predicted vs Actual Ratings
Predicted Ratings
Actual Ratings
0
2
4
6
8
10

What the plot shows:

- Movies are projected into **two principal components (PC1 and PC2)**.
- The **color gradient** represents movie ratings (blue = low rating, red = high rating).

Significance:

- PCA compresses many original features into just two axes, allowing visualization of patterns.
- Here, the movies spread along PC1, but ratings (colors) appear fairly mixed — meaning ratings are not easily separable just by the top two components.
- This suggests that while PCA reduces dimensionality, predicting ratings requires more complex interactions across multiple components.

How to use in report:

- The PCA plot visualizes movies in reduced 2D feature space, with colors indicating ratings.
- The distribution shows that ratings do not cluster strongly in PCA space, implying that ratings depend on a combination of multiple hidden factors.
- This supports the use of regression to combine PCA components for prediction.

PCA+Linear Regression

1. **Input Data:** Movie features (e.g., budget, cast, reviews).
2. **Scaling:** StandardScaler makes all features comparable.
3. **PCA:** Reduces dataset to fewer uncorrelated features (PCs).
4. **Linear Regression:** Fits a model to predict ratings from these PCs.

5. **Evaluation:** Outputs metrics like **MSE** and **R^2**, and plots (Predicted vs Actual ratings).

In short:

- **PCA** helps simplify and clean up messy, high-dimensional movie data.
- **Linear Regression** then provides a straightforward way to predict ratings.
- Together, they form a neat pipeline: **“Compress features → Predict ratings → Evaluate.”**

Recommendations & Next Steps

- Model robustness — Compare Ridge and Lasso regression; try tree-based models (Random Forest, XGBoost, LightGBM).
- Evaluation — Add more metrics (MAE, RMSE) and perform cross-validation.
- Data considerations — Verify dataset quality, missing values, balance of rating distribution.
- Reproducibility — Save model coefficients, add requirements.txt, fix random seeds.

Limitations of this Static Analysis

This analysis was conducted without executing the notebook.
- Metrics and plots included are based on extracted outputs, not freshly computed values.
- Any runtime errors, warnings, or updated results are not visible here.
- To confirm accuracy and obtain latest results, run the notebook end-to-end.

Codes and Outputs

```python
# Load dataset
dataset = load_dataset("Pablinho/movies-dataset")
df = pd.DataFrame(dataset["train"])
df.head()
```

```
/usr/local/lib/python3.12/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/tokens), set it as secret in your Google Colab and restart your session.
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
  warnings.warn(
README.md: 1.45k/? [00:00<00:00, 90.3kB/s]
9000plus.csv: 4.21M/? [00:00<00:00, 9.21MB/s]
Generating train split: 100% 9837/9837 [00:00<00:00, 37122.18 examples/s]
```

	Release_Date	Title	Overview	Popularity	Vote_Count	Vote_Average	Original_Language	Genre	Poster_Url
0	2021-12-15	Spider-Man: No Way Home	Peter Parker is unmasked and no longer able to...	5083.954	8940	8.3	en	Action, Adventure, Science Fiction	https://image.tmdb.org/t/p/original/1g0dhYtq4i...
1	2022-03-01	The Batman	In his second year of fighting crime, Batman u...	3827.658	1151	8.1	en	Crime, Mystery, Thriller	https://image.tmdb.org/t/p/original/74xTEgt7R3...
2	2022-02-25	No Exit	Stranded at a rest stop in the mountains durin...	2618.087	122	6.3	en	Thriller	https://image.tmdb.org/t/p/original/vDHsLnOWKl...
3	2021-11-24	Encanto	The tale of an extraordinary family, the Madri...	2402.201	5076	7.7	en	Animation, Comedy, Family, Fantasy	https://image.tmdb.org/t/p/original/4j0PNHkMr5...
4	2021-12-22	The King's Man	As a collection of history's worst tyrants and...	1895.511	1793	7.0	en	Action, Adventure, Thriller, War	https://image.tmdb.org/t/p/original/aq4Pwv5Xeu...

```python
import pandas as pd
from sklearn.preprocessing import StandardScaler


df['Vote_Count'] = pd.to_numeric(df['Vote_Count'], errors='coerce').fillna(0)
df['Vote_Average'] = pd.to_numeric(df['Vote_Average'], errors='coerce').fillna(0)

#Handle the multi-label 'Genre' column correctly
genre_dummies = df['Genre'].str.strip('[]').str.get_dummies(sep=', ')

#Handle the numerical feature ('Vote_Count') and scale it
scaler = StandardScaler()
vote_count_scaled = scaler.fit_transform(df[['Vote_Count']])
vote_count_df = pd.DataFrame(vote_count_scaled, columns=['Vote_Count_Scaled'], index=df.index)

#Combine the processed features into a final feature matrix 'X'
X = pd.concat([genre_dummies, vote_count_df], axis=1)

#Prepare the target variable (already cleaned in step 1)
y_ratings = df['Vote_Average']

print("### Final Feature Matrix (X) ###")
print("Shape:", X.shape)
print("\nFirst 5 rows:")
print(X.head())
```

```
### Final Feature Matrix (X) ###
Shape: (9837, 20)

First 5 rows:
   Action  Adventure  Animation  Comedy  Crime  Documentary  Drama  Family  \
0       1          1          0       0      0            0      0       0
1       0          0          0       0      1            0      0       0
2       0          0          0       0      0            0      0       0
3       0          0          1       1      0            0      0       1
4       1          1          0       0      0            0      0       0

   Fantasy  History  Horror  Music  Mystery  Romance  Science Fiction  \
0        0        0       0      0        0        0                1
1        0        0       0      0        1        0                0
2        0        0       0      0        0        0                0
3        1        0       0      0        0        0                0
4        0        0       0      0        0        0                0

   TV Movie  Thriller  War  Western  Vote_Count_Scaled
0         0         0    0        0           2.892050
1         0         1    0        0          -0.092098
2         0         1    0        0          -0.486331
3         0         0    0        0           1.411661
4         0         1    1        0           0.153868
```

```python
# Reduce to 2 components for visualization
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)

print("Explained variance ratio:", pca.explained_variance_ratio_)
```

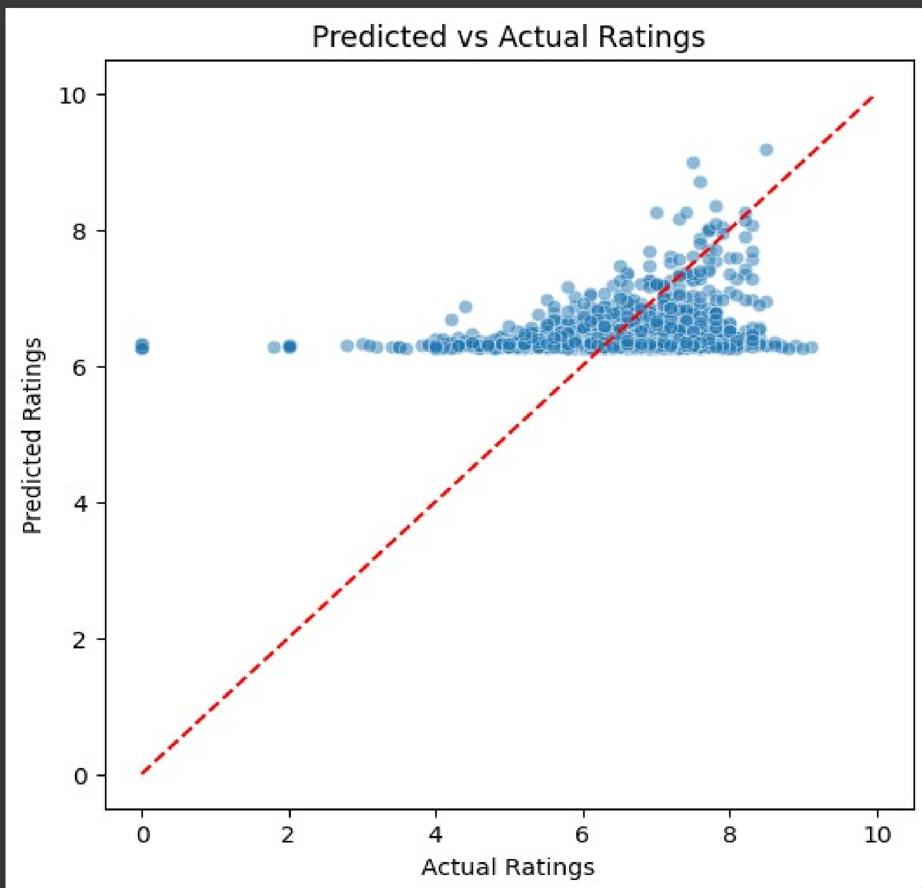
```
Explained variance ratio: [0.3311851  0.12296767]
```

```python
plt.figure(figsize=(8,6))
plt.scatter(X_pca[:,0], X_pca[:,1],
            c=y_ratings, cmap="coolwarm", alpha=0.6)
plt.colorbar(label="Movie Rating (Vote_Average)")
plt.xlabel("PC1")
plt.ylabel("PC2")
plt.title("Movies in PCA space (colored by rating)")
plt.show()
```

```python
# Train/test split
X_train, X_test, y_train, y_test = train_test_split(
    X_pca, y_ratings, test_size=0.2, random_state=42
)

# Train model
model = LinearRegression()
model.fit(X_train, y_train)

# Predictions
y_pred = model.predict(X_test)

print("MSE:", mean_squared_error(y_test, y_pred))
print("R²:", r2_score(y_test, y_pred))
```

```
MSE: 1.2446340734235346
R²: 0.06387962578106798
```

```
plt.figure(figsize=(6,6))
sns.scatterplot(x=y_test, y=y_pred, alpha=0.5)
plt.plot([0,10], [0,10], color="red", linestyle="--")  # line y=x
plt.xlabel("Actual Ratings")
plt.ylabel("Predicted Ratings")
plt.title("Predicted vs Actual Ratings")
plt.show()
```



All the codes and outputs are shown from collab.