Investigating the Relationship Between Movie Popularity and Quality

PRESENTER: NIKITA FERENTS

Project definition

THE MOVIE DB

01.

OBJECTIVE

Determine if movie popularity depends on quality.

02.

SCOPE

- Use data from the TMDB API.
- Apply machine learning models for analysis.

03.

METHODOLOGY

Data acquisition, processing, and machine learning.

Working with API

01 TMDB API OVERVIEW

The Movie Database (TMDB) is a community built movie and TV database. TMDB's strong international focus and breadth of data is largely unmatched.

The API service provides movie, TV show or actor images and/or data in TMDB application. This API is a system provided to programmatically fetch and use TMDB data and/or images.

02 API REGISTRATION AND ACCESS

To register and API for your project, you need to register on a TMDB website and generate your API key in the account setting. TMDB provides all necessary information you need, from overview to the statistics of usage

Once you have been issued a key, an example API key based request looks like this:

```
curl --request GET \
     --url 'https://api.themoviedb.org/3/movie/11?api_key=669a4175a226588523e788ed359dd4ba'
```

API Registration and Access

Example Results Object


```
{
  "poster_path": "/IfB9hy4JH1eH6HEfIgIGORXi5h.jpg",
  "adult": false,
  "overview": "Jack Reacher must uncover the truth behind a major government conspiracy in order to clear his name. On the run as a fugitive from the law, Reacher uncovers a potential secret from his past that could change his life forever.",
  "release_date": "2016-10-19",
  "genre_ids": [
    53,
    28,
    80,
    18,
    9648
  ],
  "id": 343611,
  "original_title": "Jack Reacher: Never Go Back",
  "original_language": "en",
  "title": "Jack Reacher: Never Go Back",
  "backdrop_path": "/4ynQYtSEuU5hyipcGkfD6ncwtwz.jpg",
  "popularity": 26.818468,
  "vote_count": 201,
  "video": false,
  "vote_average": 4.19
}
```

Setting up and connecting to MongoDB

I defined a connection URI to connect to my MongoDB instance. This URI includes the protocol (**mongodb://**), the hostname (**localhost**), and the port number (**27017**). Also, it includes a database name (TMDB).

```
const mongoUri = 'mongodb://localhost:27017/TMDB';
```

I used the ‘MongoClient’ from the ‘mongodb’ package to connect to my MongoDB instance. Here's how I established the connection:

```
const { MongoClient } = require('mongodb');
const client = await MongoClient.connect(mongoUri, { useNewUrlParser: true,
useUnifiedTopology: true });
      const db = client.db();
```

Once connected to the MongoDB database, I created a collection named 'Movies'. Collections in MongoDB are similar to tables in relational databases, but they are schema-less.

```
const collection = db.collection(collectionName);
```

Fetching and Inserting Data

I fetched movie data using the Axios library, filtering and mapping the results to include only relevant fields (title, year, vote_count, vote_average).

```
const axios = require('axios');
const apiKey = '669a4175a226599303e788ed359dd4ba';
const numPages = 500;

const allMovies = [];
for (let page = 1; page <= numPages; page++) {
        const url =
`https://api.themoviedb.org/3/movie/popular?api_key=${apiKey}&page=${page}`;

        const response = await axios.get(url);
```

```
    const response = await axios.get(url);

        const movies = response.data.results
     .filter(movie => movie.vote_count > 100 && movie.vote_average > 0 )
     .map(movie => ({
       title: movie.title,
       year: new Date(movie.release_date).getFullYear(),
       vote_count: movie.vote_count,
       vote_average: movie.vote_average
     }));

    // Accumulate movies from this page
    allMovies.push(...movies);
  }
```

Data Attributes and Data Processing

Attribute	
Attribute	title: Title of the movie.
	year: Year of release.
	vote_count: Number of votes.
	vote_average: Average rating.

Filtering: I filtered out movies with a vote count less than 100 and a vote average of 0. This step ensures that only movies with a **significant** number of votes and **non-zero** ratings are included in the analysis.

```
 const movies = response.data.results
.filter(movie => movie.vote_count > 100 && movie.vote_average > 0 )
```

Transformation involves converting the raw data into a structured format suitable for storage and analysis. Key transformations included:

- **Field Extraction:** Extracting relevant fields such as **title, year, vote_count,** and **vote_average**.
- **Data Mapping:** Converting the release date to a year format and mapping the necessary fields into a new structure.

```
.map(movie => ({
          title: movie.title,
          year: new Date(movie.release_date).getFullYear(),
          vote_count: movie.vote_count,
          vote_average: movie.vote_average
        }));
```

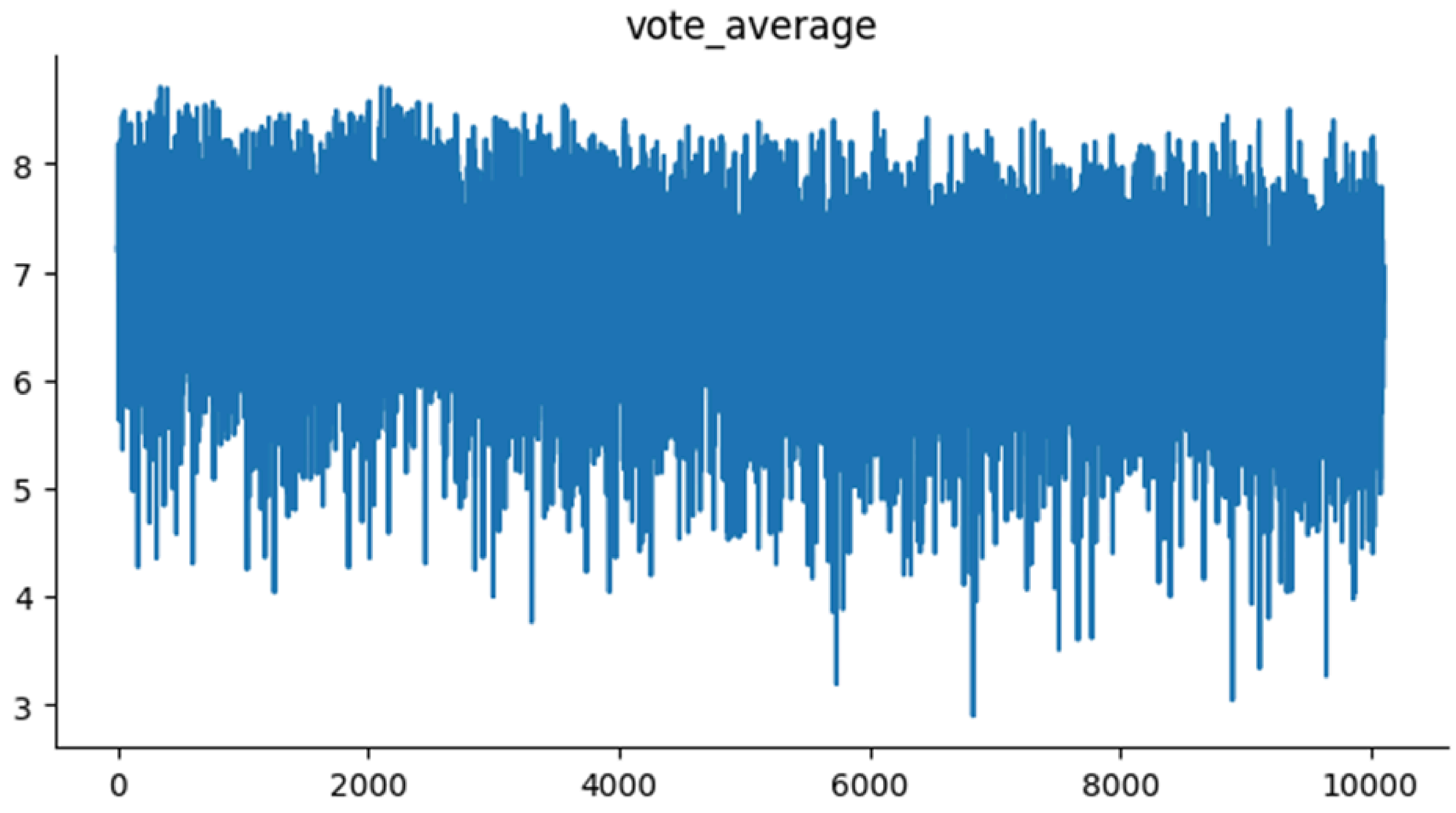
Working with Machine Learning

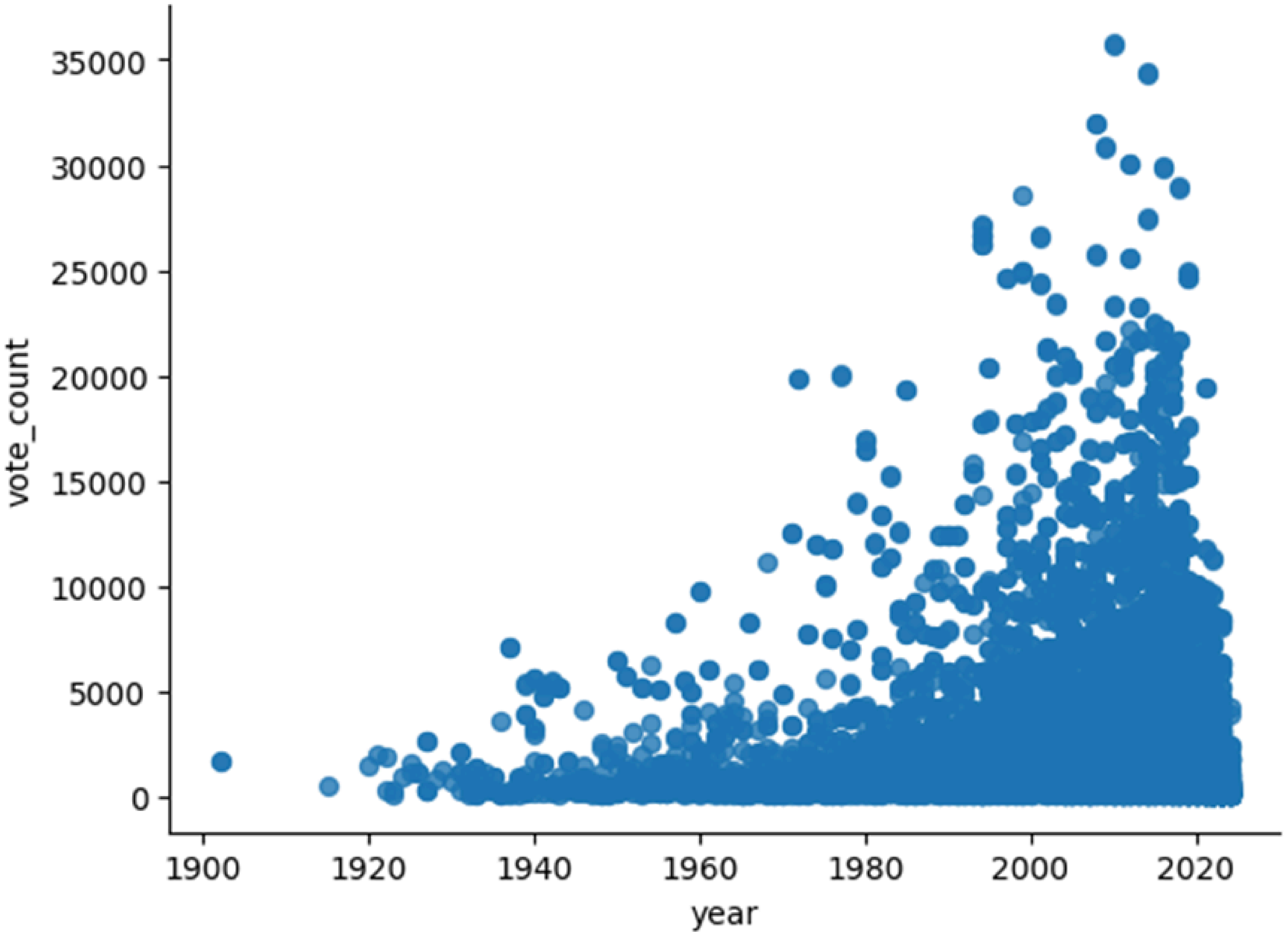
Loading Data: To begin with, I loaded the dataset from Google Drive into a Pandas DataFrame for ease of manipulation and analysis.

```python
import pandas as pd


from google.colab import drive

drive.mount('/content/gdrive')

data_path = '/content/gdrive/My Drive/BigDataAnalysis/TMDB.Movies.BIG.csv'
movie_df = pd.read_csv(data_path)
movie_df.info()
movie_df.head()
```

Initial Data Viualization

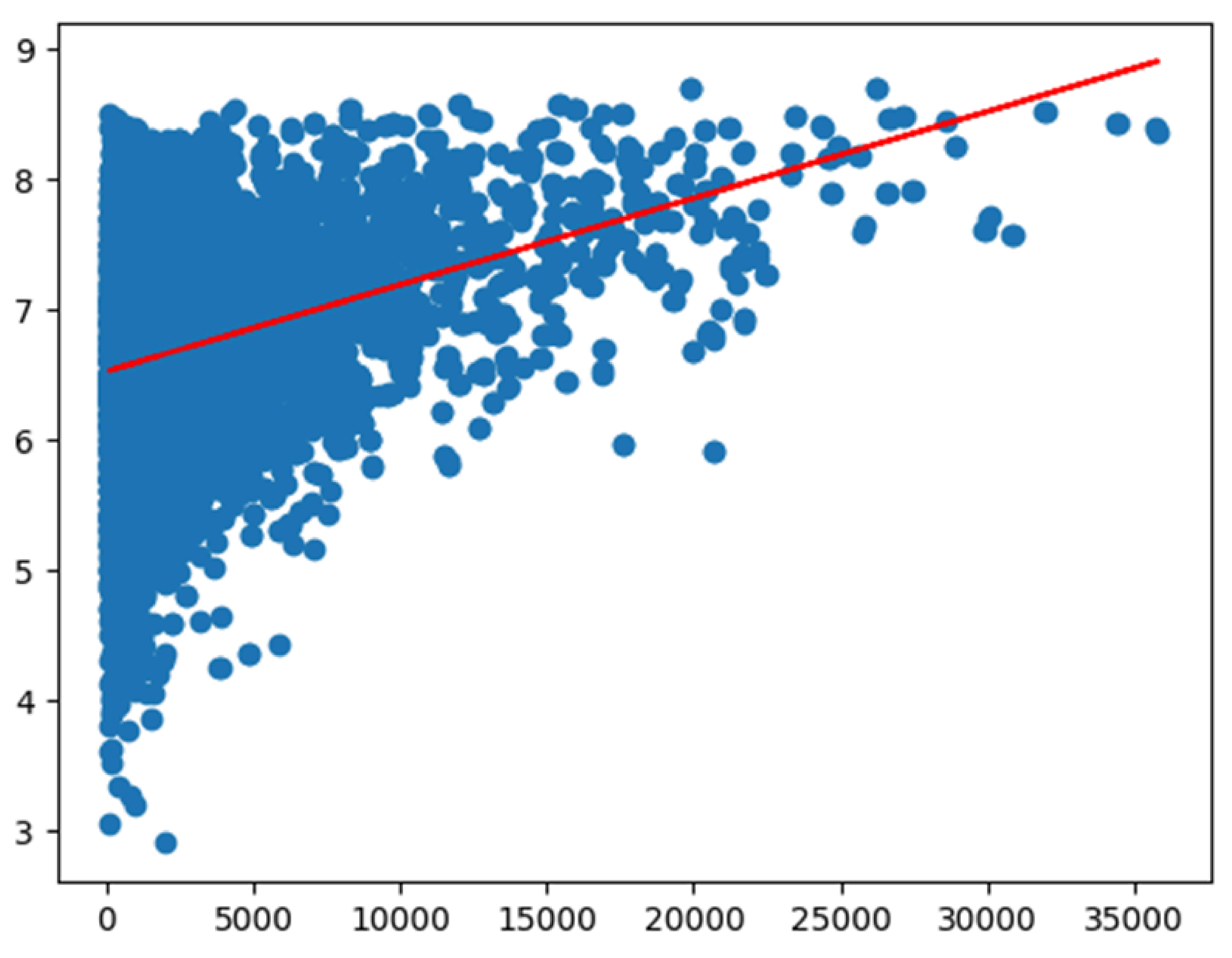
```
from matplotlib import pyplot as plt
movie_df['vote_average'].plot(kind='line', figsize=(8, 4),
title='vote_average')
plt.gca().spines[['top', 'right']].set_visible(False)
```






```
movie_df.plot(kind='scatter', x='year', y='vote_count', s=32, alpha=.8)
plt.gca().spines[['top', 'right',]].set_visible(False)
```

Linear Regression



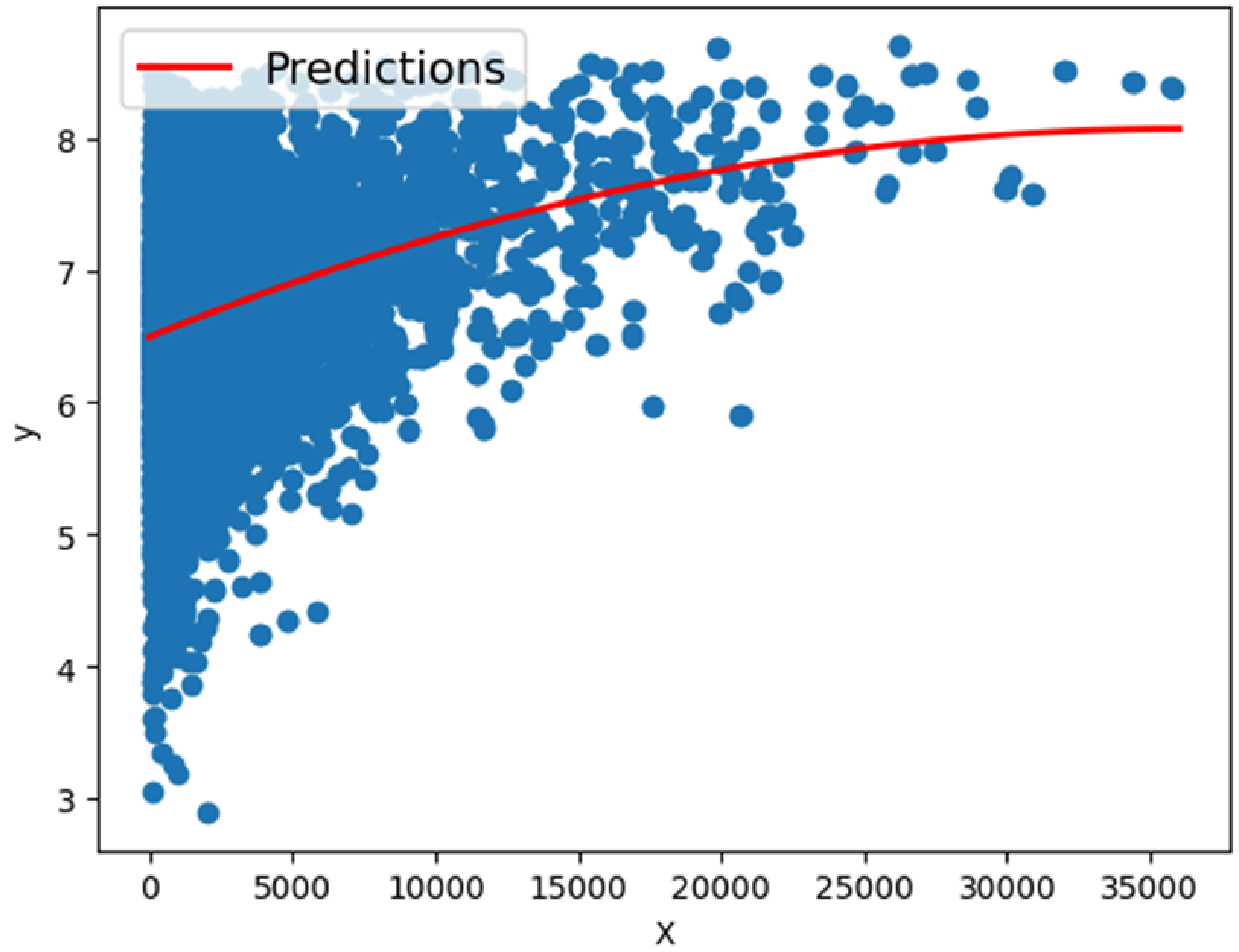

```python
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# Reshape X for Linear Regression
X = X.to_numpy().reshape(-1, 1)
reg = LinearRegression().fit(X, y)

# Print model coefficients
print(f'Slope: {reg.coef_[0][0]}')
print(f'Intercept: {reg.intercept_[0]}')

# Predict and evaluate the model
y_predicted = reg.predict(X)
mse = mean_squared_error(y, y_predicted)
r2 = r2_score(y, y_predicted)
print(f'Linear Regression MSE: {mse}, R-squared: {r2}')

# Visualization
plt.scatter(X, y, label='Actual data')
plt.plot(X, y_predicted, 'r', label='Linear Regression')
plt.xlabel('Vote Count')
plt.ylabel('Vote Average')
plt.legend()
plt.show()
```

Polynomial Regression

```python
from sklearn.preprocessing import PolynomialFeatures

# Transform the features to polynomial features
poly_feature = PolynomialFeatures(degree=2, include_bias=False)
X_poly = poly_feature.fit_transform(X)

# Train the polynomial regression model
lin_reg = LinearRegression()
lin_reg.fit(X_poly, y)

# Predict and evaluate the model
```

```python
y_new = lin_reg.predict(X_poly)
poly_mse = mean_squared_error(y, y_new)
poly_r2 = r2_score(y, y_new)
print(f'Polynomial Regression MSE: {poly_mse}, R-squared: {poly_r2}')

# Visualization
plt.scatter(X, y, label='Actual data')
plt.scatter(X, y_new, label='Polynomial Regression')
plt.xlabel('Vote Count')
plt.ylabel('Vote Average')
plt.legend()
plt.show()

# Generating predictions with distributed points
X_new = np.linspace(0, 36000, 100).reshape(100, 1)
X_new_poly = poly_feature.transform(X_new)
y_new_pred = lin_reg.predict(X_new_poly)

plt.plot(X_new, y_new_pred, "r-", linewidth=2, label="Predictions")
plt.scatter(X, y)
plt.xlabel("Vote Count")
plt.ylabel("Vote Average")
plt.legend(loc="upper left", fontsize=14)
plt.show()
```

Polynomial regression provides a **curved** line that fits the data points better by considering the **polynomial relationship**.

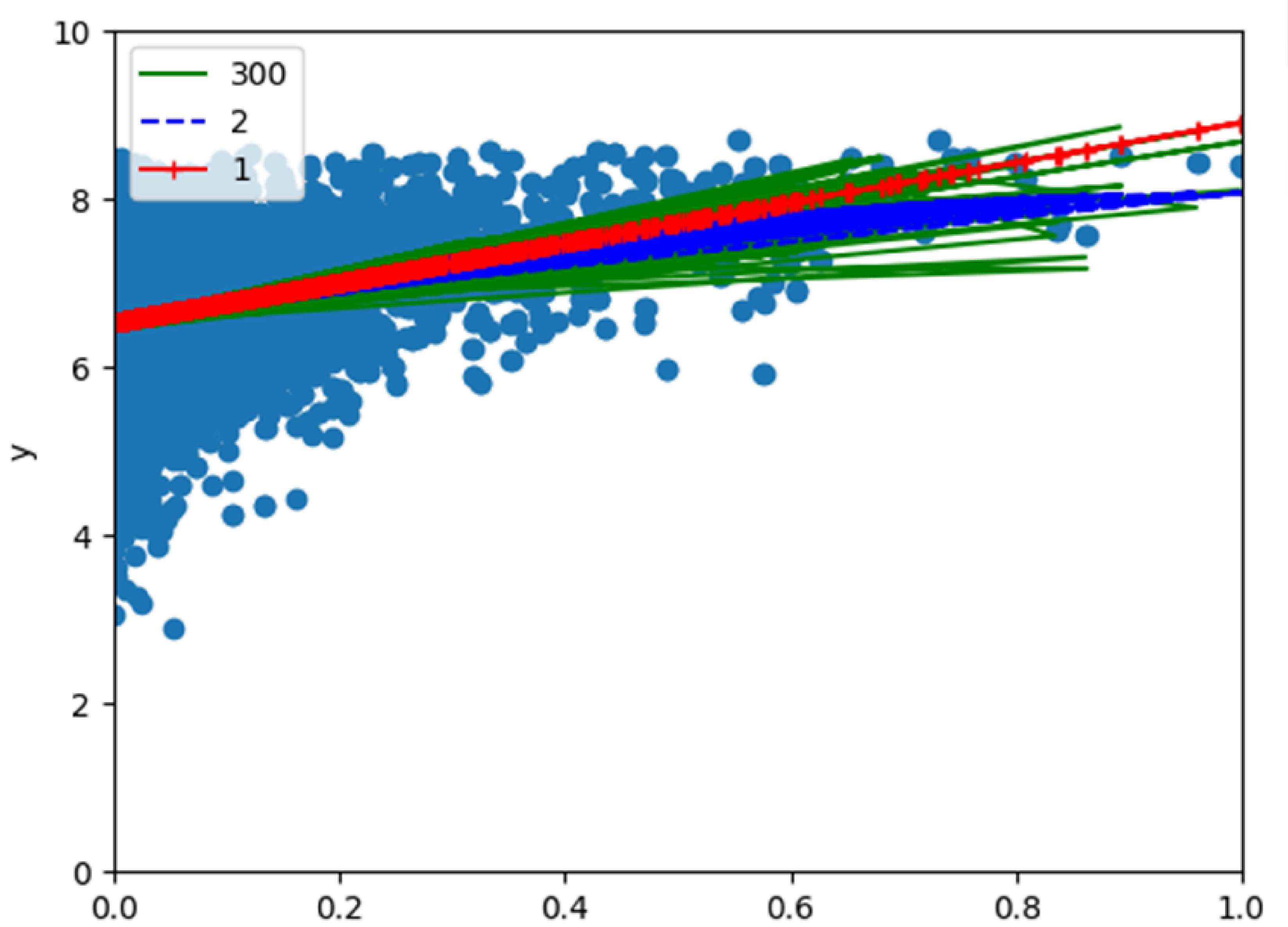


Here, a degree of **2** was used to include both **linear** and **quadratic** terms. The visualization shows how the polynomial model fits the data more closely compared to the linear model.

Pipeline for Polynomial Regression with Different Degrees

To explore the impact of polynomial **degree** on the model performance, I created **pipelines** that standardize the data and apply polynomial transformations of varying degrees before fitting a linear regression model.

The visualization compares the fit of models with different polynomial degrees, highlighting how higher degrees can overfit the data, while lower degrees may underfit.




```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import MinMaxScaler

# Creating pipelines for different polynomial degrees
for style, degree in (("g-", 300), ("b--", 2), ("r-+", 1)):
    polybig_features = PolynomialFeatures(degree=degree, include_bias=False)
    scaler = MinMaxScaler()
    X_scaled = scaler.fit_transform(X)

    lin_reg = LinearRegression()
    polynomial_regression = Pipeline([
        ("poly_features", polybig_features),
        ("scaler", scaler),
        ("lin_reg", lin_reg),
    ])
    polynomial_regression.fit(X_scaled, y)
    y_newbig = polynomial_regression.predict(X_scaled)

    plt.plot(X_scaled, y_newbig, style, label=str(degree))

plt.scatter(X_scaled, y, label='Actual data')
plt.legend(loc="upper left")
plt.xlabel("Vote Count (scaled)")
plt.ylabel("Vote Average (scaled)")
plt.show()
```

Results and Analysis

TMDB

01 LINEAR REGRESSION

The linear model showed a **low** fit with an R-squared value indicating the proportion of variance in the **vote_average** that can be explained by **vote_count**.
reg.score(X,y) = 0.10871758693242362

02 POLYNOMIAL REGRESSION

The polynomial model, especially with **degree 2**, provided a **better** fit with a higher R-squared value, capturing the non-linear relationship more effectively.

03 HIGHER DEGREE POLYNOMIAL

While a higher degree polynomial may fit the training data **better**, it risks overfitting and may not generalize well to unseen data