**UNIVERSITY OF RIJEKA  
FACULTY OF INFORMATICS   
AND DIGITAL TECHNOLOGIES**

**Graduate study of computer science**

**Project from the course**

**Big Data Analytics**

**Cinematic Analytics: Investigating the Correlation Between Movie Popularity and Ratings**

**Author: Nikita Ferents**

Mentors: prof. dr. sc. Sanda Martinčić-Ipšić, mag. inf. Kristina Host

Rijeka, May of 2024

Contents

[1. Project Definition 2](#_Toc168490267)

[1.1 Objective: 2](#_Toc168490268)

[1.2 Scope: 2](#_Toc168490269)

[1.3 Methodology: 2](#_Toc168490270)

[2. Working with API 3](#_Toc168490271)

[2.1 TMDB API Overview: 3](#_Toc168490272)

[2.2 API Registration and Access: 3](#_Toc168490273)

[2.3 Search 5](#_Toc168490274)

[2.4 Query For Details 6](#_Toc168490275)

[3. Data Acquisition 6](#_Toc168490276)

[3.1 Tools Used: 6](#_Toc168490277)

[Axios for API requests, MongoDB for data storage. 6](#_Toc168490278)

[3.2 Fetching script: 6](#_Toc168490279)

[3.3 Working with MongoDB 8](#_Toc168490280)

[3.3.1 Setting Up MongoDB 8](#_Toc168490281)

[3.3.2 Connecting to MongoDB 9](#_Toc168490282)

[3.3.3 Creating a Collection 9](#_Toc168490283)

[3.3.4 Fetching Data from TMDB API: 9](#_Toc168490284)

[3.3.5 Inserting Documents into MongoDB: 10](#_Toc168490285)

[3.4 Data Attributes: 10](#_Toc168490286)

[4. Data Proccessing 10](#_Toc168490287)

[4.1 Data Cleaning 11](#_Toc168490288)

[4.2 Data Transformation 11](#_Toc168490289)

[5. Working with Machine Learning 12](#_Toc168490290)

[5.1 Data Preparation for Machine Learning 12](#_Toc168490291)

[5.2 Model Training and Evaluation 14](#_Toc168490292)

[5.3 Results and Analysis 19](#_Toc168490293)

[6. Conclusion 19](#_Toc168490294)

[6.1 Summary: 19](#_Toc168490295)

[6.2 Future Work: 20](#_Toc168490296)

[7. References: 21](#_Toc168490297)

# Project Definition

## Objective:

To investigate the relationship between the popularity of movies and their ratings, and to determine the extent to which movie quality influences audience and critical reception.

## Scope:

Data Acquisition: Utilize the TMDB API to collect real-time data on movies, focusing on metrics such as popularity scores and viewer ratings.

Data Processing: Implement a data engineering pipeline to clean, preprocess, and structure the dataset for analysis.

Data Storage: Employ appropriate storage technologies to manage the streaming or batch data efficiently.

Analytical Model: Develop regression models (linear and polynomial) to analyze the data and extract insights into the correlation between movie popularity and ratings.

## Methodology:

Data Collection: Automated scripts to fetch data from TMDB, ensuring a live and relevant dataset.

Data Engineering: Techniques such as normalization, feature extraction, and transformation to prepare the data for modeling.

Model Training: Application of linear regression to establish a baseline for comparison and polynomial regression to capture non-linear relationships.

Evaluation: Use statistical measures to evaluate the models’ performance and the strength of the correlation.

Expected Outcome: The project aims to reveal patterns and insights that could help understand the factors contributing to a movie’s success and the predictive power of ratings on popularity.

# Working with API

## TMDB API Overview:

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.

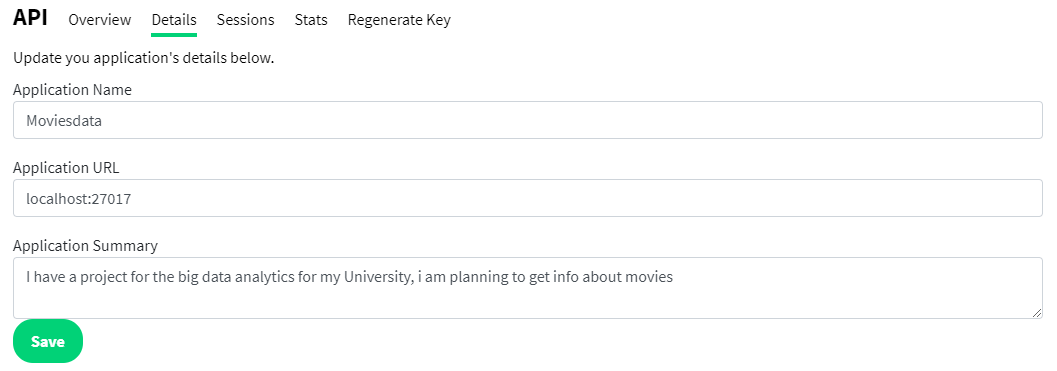
You can apply for an API key by clicking the "API" link from the left hand sidebar within your account settings page. API is free to use for non-commercial purposes as long as you attribute TMDB as the source of the data and/or images.

## API Registration and Access:

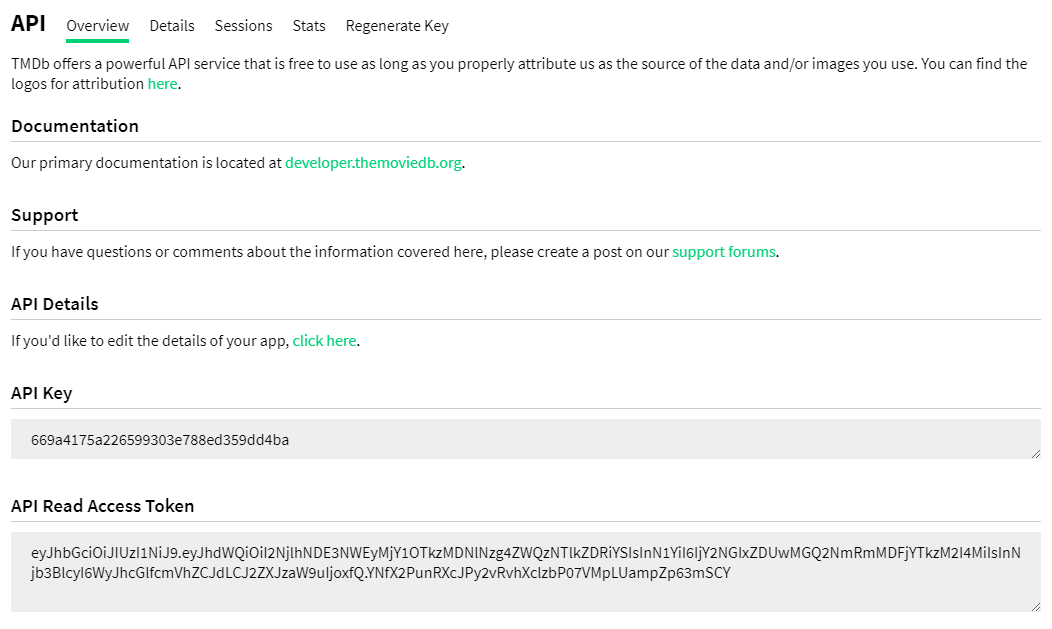
To register API, firstly you need to create an account on a TMDB website (<https://developer.themoviedb.org/>).

Then, you should head to your account and go to tab API (<https://www.themoviedb.org/settings/api>)

Then, you need to generate your API token. To do so, you need to fill all the necessary fields regarding the project, that youwill be using the API for.

Once you do it, you will see the Details tab:

Then, you can head to the Overview tab:



Here, you can find all the necessary information, from the API key itself, to the Statistic for you API usage

Application level authentication would generally be considered the default way of authenticating yourself on the API. Version 3 is controlled by one of either a single query parameter, api\_key, or by using your access token as a Bearer token. You can request an API key by logging in to your account on TMDB and clicking here.

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

cURL Example

curl --request GET \

--url 'https://api.themoviedb.org/3/movie/11?api\_key=669a4175a226588523e788ed359dd4ba'

A common workflow here on TMDB is to search for a movie (or TV show, or person) and then query for the details. Here's a quick overview of what that flow looks like.

## Search

First, you are going to issue a query to one of the movie, TV show or person search methods. We'll use Jack Reacher and the movie method for this example:

**Example Search Request**

curl --request GET \

--url 'https://api.themoviedb.org/3/search/movie?query=Jack+Reacher&api\_key=669a4175a226599303e788ed359dd4ba'

This will return a few fields, the one you want to look at is the results field. This is an array and will contain our standard movie list objects. Here's an example of the first item:

**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

}

## Query For Details

With the item above in hand, you can see the id of the movie is 343611. You can use that id to query the movie details method:

Example Details Query

curl --request GET \

--url 'https://api.themoviedb.org/3/movie/343611?api\_key=669a4175a226599303e788ed359dd4ba'

# Data Acquisition

## ****Tools Used****:

Axios for API requests, MongoDB for data storage.

## Fetching script:

const axios = require('axios');

const { MongoClient } = require('mongodb');

const apiKey = '669a4175a226599303e788ed359dd4ba';

const numPages = 500;

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

const collectionName = 'Movies'; // Adjust collection name if desired

  async function importMovies() {

    try {

      // Connect to MongoDB

      const client = await MongoClient.connect(mongoUri, { useNewUrlParser: true, useUnifiedTopology: true });

      const db = client.db();

      const collection = db.collection(collectionName);

      // Accumulate movies from all pages

      const allMovies = [];

      // Loop through specified number of pages

      for (let page = 1; page <= numPages; page++) {

        // Build the API request URL with page number

        const url = `https://api.themoviedb.org/3/movie/popular?api\_key=${apiKey}&page=${page}`;

        const response = await axios.get(url);

        // Extract only desired data from movies

        const movies = response.data.results

          .filter(movie => movie.vote\_count > 100 && movie.vote\_average > 0 ) // Filter movies with non-zero vote count

          .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);

      }

      // Insert movies into MongoDB collection

      await collection.insertMany(allMovies);

      console.log(`Imported ${allMovies.length} movies with vote count from ${numPages} pages!`);

      // Close MongoDB connection

      await client.close();

    } catch (error) {

      console.error('Error importing movies:', error);

    }

  }

  importMovies();

## Working with MongoDB

As part of my project, I used MongoDB to store and manage the movie data collected from the TMDB API. MongoDB's flexible, JSON-like document structure made it an ideal choice for handling the diverse attributes of movie data. Here's a detailed explanation of how I worked with MongoDB to create documents and collections:

### Setting Up 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). Optionally, it can include a database name (TMDB).

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

### Connecting to MongoDB

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();

**MongoClient.connect()**: This function establishes a connection to the MongoDB server using the provided URI.

**useNewUrlParser: true**: This option ensures the use of the new URL string parser.

**useUnifiedTopology: true**: This option enables the new unified topology layer.

### Creating a Collection

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 Data from TMDB API:

You 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 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);

      }

### **Inserting Documents into MongoDB**:

After accumulating the movie data, I inserted the documents into the 'Movies' collection using the ‘insertMany’ method.

await collection.insertMany(allMovies);

## Data Attributes:

* title: Title of the movie.
* year: Year of release.
* vote\_count: Number of votes.
* vote\_average: Average rating.

# Data Proccessing

Data processing is a crucial step in my project to ensure that the data fetched from the TMDB API is clean, structured, and ready for analysis and machine learning tasks. Here's a detailed breakdown of how I processed the data:

## Data Cleaning

Data cleaning is necessary to ensure the quality and consistency of the dataset. The primary cleaning steps involved filtering out movies with inadequate or unreliable data.

**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 )

## Data Transformation

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

In this section, I detail the steps taken to process the data specifically for machine learning tasks, and the subsequent model training and evaluation based on my Google Colab notebook.

## Data Preparation for Machine Learning

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

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()

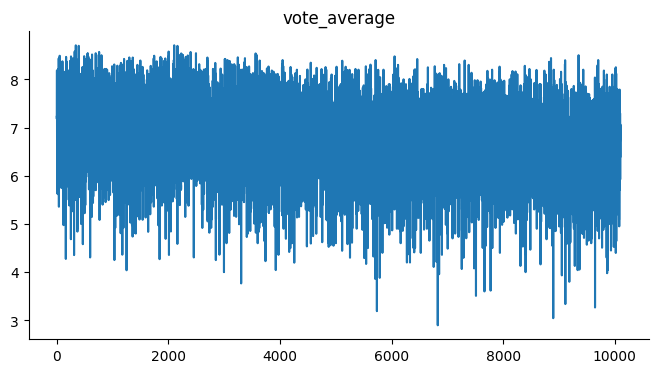
movie\_df.head()

**Visualizing Data:** I performed initial visualizations to understand the distribution and relationship of the data. For instance, plotting the vote\_average and creating a scatter plot of year vs. vote\_count.

from matplotlib import pyplot as plt

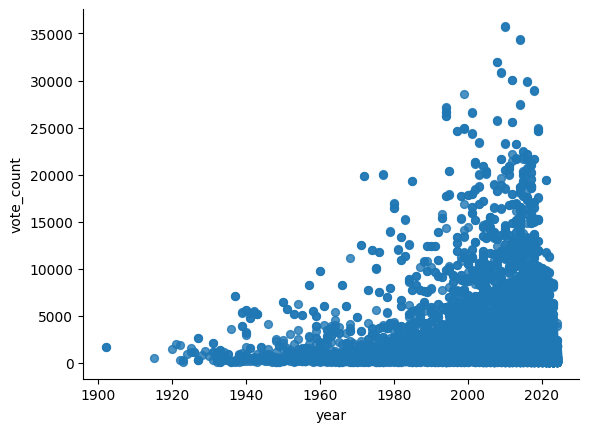
movie\_df['vote\_average'].plot(kind='line', figsize=(8, 4), title='vote\_average')

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)



**Standardizing Features:** To ensure that the features are on a comparable scale, I standardized the vote\_count and vote\_average using the StandardScaler.

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

movie\_df[["vote\_count", "vote\_average"]] = scaler.fit\_transform(movie\_df[["vote\_count", "vote\_average"]])

## Model Training and Evaluation

**Linear Regression:** I trained a linear regression model to understand the linear relationship between movie popularity (vote count) and quality (vote average).

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')

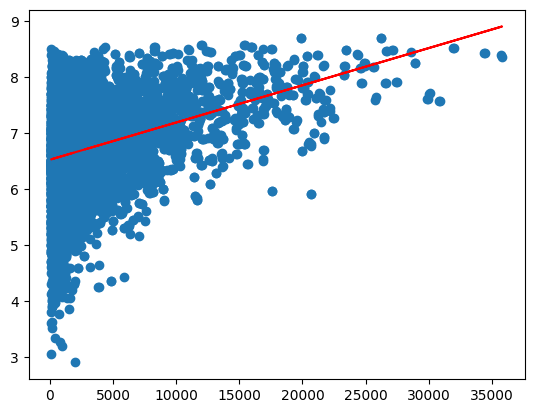
plt.plot(X, y\_predicted, 'r', label='Linear Regression')

plt.xlabel('Vote Count')

plt.ylabel('Vote Average')

plt.legend()

plt.show()



The linear regression model provides a straight line that attempts to best fit the data points by minimizing the mean squared error between the predicted and actual vote\_average. The slope and intercept of the line give insights into how much vote\_average changes with a unit change in vote\_count.

**Polynomial Regression:** Recognizing that the relationship between movie popularity and quality might be non-linear, I also trained a polynomial regression model. This model allows for capturing more complex patterns by including polynomial terms.

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

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")

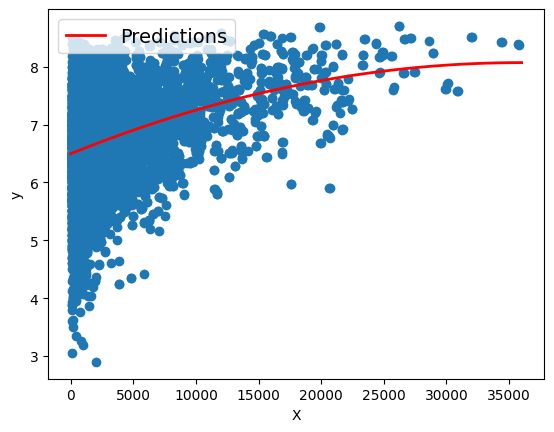
plt.scatter(X, y)

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.

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))

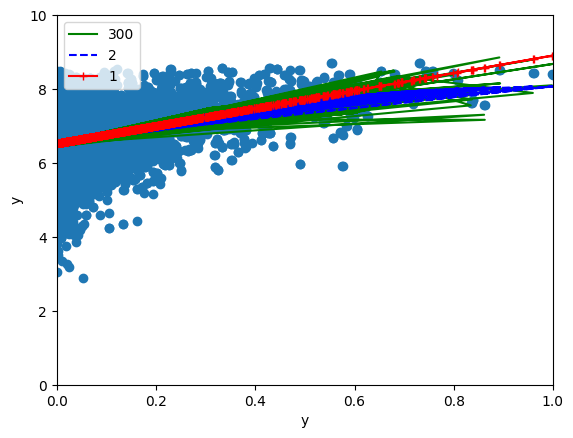
plt.scatter(X\_scaled, y, label='Actual data')

plt.legend(loc="upper left")

plt.xlabel("Vote Count (scaled)")

plt.ylabel("Vote Average (scaled)")

plt.show()



Using the Pipeline and MinMaxScaler, I standardized the features and applied polynomial regression with different degrees (1, 2, and 300). The visualization compares the fit of models with different polynomial degrees, highlighting how higher degrees can overfit the data, while lower degrees may underfit.

## Results and Analysis

* **Linear Regression**: The linear model showed a moderate 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
* **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.
* **Higher Degree Polynomial**: While a higher degree polynomial can fit the training data extremely well, it risks overfitting and may not generalize well to unseen data.

These models provided insights into the relationship between movie popularity and quality, demonstrating that while there is a little positive correlation, the relationship is more accurately captured with polynomial regression, reflecting the complexity of factors influencing movie ratings.

# Conclusion

## **Summary:**

In this project, I aimed to investigate the relationship between the popularity of a movie (measured by the number of votes) and its quality (measured by the average vote score). Using data from the TMDB service, I performed several tasks including data retrieval, processing, and machine learning modeling.

* **Objectives:** The main objective was to determine if and how the popularity of a movie depends on its quality, using machine learning techniques.
* **Methods:**
  + **Data Retrieval:** I collected live data from the TMDB API, storing it in a MongoDB database.
  + **Data Processing:** The data was cleaned and prepared for analysis, including handling missing values, normalizing the features, and splitting the dataset for training and testing.
  + **Machine Learning:** I applied linear regression and polynomial regression models to predict movie quality based on popularity. I evaluated the models using metrics such as Mean Squared Error (MSE) and R-squared.
* **Key Findings:**
  + The linear regression model provided a basic understanding of the linear relationship between movie popularity and quality.
  + The polynomial regression model, particularly with degree 2, captured the non-linear relationship more effectively, suggesting that movie quality does not increase linearly with popularity.

## **Future Work:**

To improve the accuracy and robustness of the analysis, several enhancements could be implemented:

* **Advanced Machine Learning Models:** Utilize more sophisticated models such as decision trees, random forests, or neural networks, which might capture more complex patterns in the data.
* **Additional Features:** Incorporate other relevant features such as genre, runtime, director, budget, and release date. These additional variables can provide a more comprehensive understanding of the factors affecting movie popularity and quality.
* **Temporal Analysis:** Conduct a time-series analysis to understand trends and patterns over time.
* **Sentiment Analysis:** Analyze text data from movie reviews to incorporate qualitative aspects of movie quality.
* **Cross-Validation:** Implement cross-validation techniques to ensure the model's robustness and generalizability to unseen data.

# ****References:****

* Pandas documentation: <https://pandas.pydata.org/>
* Scikit-learn documentation: <https://scikit-learn.org/stable/>
* Matplotlib documentation: <https://matplotlib.org/>
* Google Colab documentation: <https://colab.research.google.com/notebooks/intro.ipynb>
* TMDB API documentation: <https://developers.themoviedb.org/3>