**Movie Rating Aggregator - Data Engineering Project Report**

### **1. Introduction**

Ever wondered how movie ratings are collected and analyzed? This project is all about building a **Movie Rating Aggregator** that fetches ratings from The Movie Database (TMDb) API and processes them efficiently. The idea is to automate the data collection process, clean up the information, store it in a structured format, and analyze it to uncover trends in movie ratings.

### **2. Objectives**

* Fetch movie rating data from TMDb API automatically.
* Process and clean the data to make it usable.
* Store the cleaned data in a database for easy access.
* Analyze trends and insights from the ratings.

### **3. Technologies Used**

* **Python**: To interact with the TMDb API and process data.
* **TMDb API**: The source of movie rating data.
* **PySpark**: For handling large datasets efficiently.
* **Pandas**: To manipulate and transform data.
* **SQL Database (MySQL/PostgreSQL)**: To store and query the processed data.

### **4. Implementation**

#### **Step 1: Fetching Movie Data**

We start by using the **TMDb API** to pull movie details, such as title, release year, rating, and popularity. The API request is made using Python's requests library.

import requests

api\_key = "your\_api\_key"

url = f"https://api.themoviedb.org/3/search/movie?api\_key={api\_key}&query=Inception"

response = requests.get(url)

data = response.json()

#### **Step 2: Transforming the Data**

Once we have the data, we clean and organize it using **PySpark**. This helps in handling missing values and structuring the data properly.

from pyspark.sql import SparkSession

from pyspark.sql import Row

spark = SparkSession.builder.appName("MovieAggregator").getOrCreate()

movies = [Row(title="Inception", rating=8.8, release\_year=2010)]

df = spark.createDataFrame(movies)

df.show()

#### **Step 3: Storing the Data**

After processing, we store the data in a **SQL database** to make it easily accessible for future queries and analysis.

CREATE TABLE movie\_ratings (

id SERIAL PRIMARY KEY,

title VARCHAR(255),

rating FLOAT,

release\_year INT

);

#### **Step 4: Analyzing Movie Trends**

With the data stored, we can now analyze trends, such as the average movie rating per year.

avg\_ratings = df.groupBy("release\_year").avg("rating")

avg\_ratings.show()

### **5. Results**

* Successfully extracted and stored movie ratings from TMDb.
* Processed and structured the data using PySpark.
* Stored the data in a SQL database for easy access.
* Performed trend analysis to understand movie ratings over time.

### **6. Conclusion**

This project showcases a complete data engineering pipeline—from fetching raw data to processing and analyzing it. By leveraging **APIs, PySpark, and SQL databases**, we’ve built an efficient system to track movie ratings and gain valuable insights into movie trends. It’s a practical approach to handling real-world data at scale!