Project Title:

TMDB-RecoFlow: An End-to-End Data Pipeline for Movie Recommendation

---

Prepared by:
Mr. Polakorn Anantapakorn
Data Engineer Intern, Bluebik Vulcan

Supervisor / Advisor
Mr. Napasin Hongngern
Data Engineer, Bluebik Vulcan

Date of Submission:
30-July-2025

Project Duration:
June 2025 – July 2025

Contents

Introduction 4

Objective 4

Output 5

Benefits 5

Data Source 6

Data Characteristic 6

Data Dictionary 6

Column Classification in Dataset 7

Architecture overview 9

Architecture Details 11

Volumes and Mounts (Airflow & Spark) 13

Timeline Project 14

Data Processing Overview 15

ETL_tmdb_dataset DAG 15

DAG Configuration 15

Task Groups and Flow 17

Data Cleansing (clean_data.py) 17

Key Operations 17

Data Transformation 19

Output Tables 19

Load to Data Warehouse (BigQuery) 20

Post-Load Validation 20

Export_Recom_Dataset_from_GCP DAG 21

DAG Task Flow 21

Final Output 22

Data Validation 22

Outcome 23

Result 23

Example Use Case 24

Content-Based Movie Recommendation 24

Dataset Source 24

Workflow Steps 25

How the Model Works 25

Results 26

Summary 26

Introduction

In 2025, **watching movies is a widely accessible and popular activity**. Streaming platforms greatly benefit from this popularity and offer various services, such as providing online movie streaming services, managing subscription payments, and offering comprehensive customer support. One crucial service for retaining customers is **movie recommendation**. This process is driven by **data analysis** and powered by a **recommendation system** that utilizes high-quality data derived from these analytical processes.

The fundamental process that ensures high-quality data for data analytics and building models is **data engineering**. It focuses on efficient data processing and **data pipeline system maintenance**, ultimately leading to actionable, high-quality data.

This project utilizes the **TMDB (The Movie Database) dataset**, a free, community-driven platform providing detailed information about movies, TV shows, and cast members. It includes metadata such as genres, ratings, production details, and images. Developers can access this rich dataset via a public API, making it ideal for building recommendation systems and movie-related projects.

This project presents the architecture and data pipeline system utilizing tools within the **data engineering field**. For example, **Apache Airflow** for orchestrating and managing data pipelines, **Apache Spark** for processes and transforms the large TMDB dataset efficiently, and **BigQuery**, a data warehouse service from Google Cloud Platform (GCP), for storing high-quality data. This project demonstrates **the end-to-end data engineering process**. The outcome of this project is high-quality data prepared for data analytics and recommendation systems.

Objective

1.**To design and implement a robust and scalable data architecture** that integrates all components (data ingestion, processing, storage, and serving) to support the end-to-end data pipeline.

2.**To build a data processing pipeline** that ensures high-quality data is extracted, transformed, and loaded into the data warehouse.

3.**To establish a scalable data warehouse** to house high-quality movie data, optimized to efficiently support a movie recommendation system.

Output

The successful execution of this project, **TMDB-RecoFlow**, will yield the following key outputs:

- **A robust and scalable data pipeline**: This end-to-end pipeline, built with Apache Airflow for orchestration and Apache Spark for processing, ensures continuous data flow from ingestion to serving.
- **An optimized BigQuery data warehouse**: A well-structured central repository for high-quality, transformed movie metadata, designed for efficient data retrieval to support analytical queries and recommendation system training.
- **A high-quality dataset for recommendation**: The project will provide a clean, consistent, and readily accessible dataset specifically prepared to feed into a movie recommendation system for suggesting similar titles.
- **Demonstration of end-to-end data engineering**: This project will serve as a practical showcase of the complete data engineering lifecycle, from raw data acquisition to delivering actionable data for advanced analytics.

Benefits

- **Enhanced Operational Efficiency**: The automated data pipeline, powered by Apache Airflow and Spark, streamlines data processing, reducing manual effort and ensuring timely data availability.
- **Actionable Business Insights**: The project delivers clean, structured data in BigQuery, enabling precise analytics and valuable insights into movie trends and user behavior.
- **Practical Skill Development**: For the developer, it provides invaluable hands-on experience in designing, implementing, and managing an end-to-end data engineering pipeline using industry-standard tools.

Data Source

Name: Full TMDB Movies Dataset 2024 (1M Movies)

Source: https://www.kaggle.com/datasets/asaniczka/tmdb-movies-dataset-2023-930k-movies/data

Data Characteristic

The TMDB Movies Dataset (2023) used in this project is a comprehensive and regularly updated collection of **film information**. It contains a vast number of movies, totaling **1,000,000 entries from the TMDB database**, with daily updates. Each entry includes essential details such as the movie's ID, Title, Average Vote, Vote Count, Status, Release Date, Revenue, and Runtime. Additionally**, the dataset features various other attributes that contribute to effective analysis and the development of robust movie recommendation systems.**

Concern:

- **Secondary Source**: The data is obtained from Kaggle, a secondary source, rather than directly from the TMDB API. This means the project relies on someone else's extraction and aggregation process, which might introduce unforeseen biases or limitations from their collection methodology.
- **Snapshot Nature**: The dataset is a static snapshot, containing data only up to (02-06-2025)

Data Dictionary

The dataset exported from Kaggle contains 1,234,214 rows and 24 columns. This snapshot was created on 02-06-2025 (DD-MM-YYY)

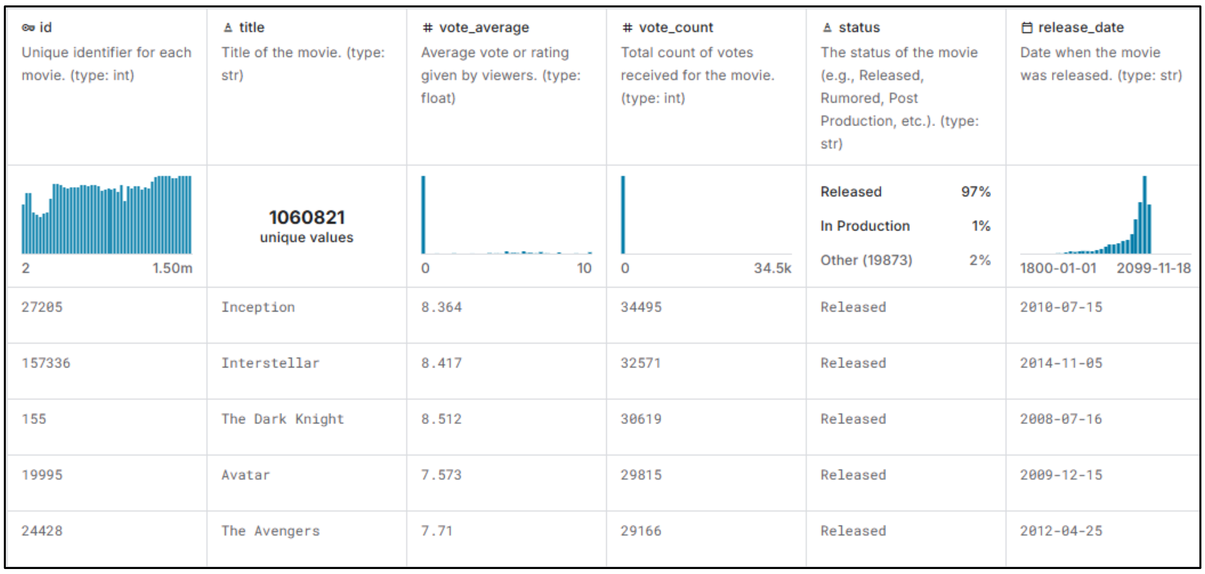
Column Name	Data Type	Description
id	int	Unique identifier for each movie
title	str	Title of the movie
vote_average	float	Average rating given by viewers
vote_count	int	Total number of votes received
status	str	Production status (e.g., Released, Rumored)
release_date	str	Date when the movie was/will be released

revenue	int	Total box office revenue
runtime	int	Duration of the movie in minutes
adult	bool	Whether the movie is categorized as adult-only
backdrop_path	str	URL path to the movie's backdrop image
budget	int	Production budget of the movie
homepage	str	Official homepage URL for the movie
imdb_id	str	IMDb identifier
original_language	str	Original language code (e.g., en, fr)
original_title	str	Original title of the movie
overview	str	Synopsis or description
popularity	float	Popularity score determined by TMDB
poster_path	str	URL path to the movie's poster image
tagline	str	Official tagline or slogan
genres	str/list	List of genres (as strings)
production_companies	str/list	List of production companies
production_countries	str/list	List of production countries
spoken_languages	str/list	List of languages spoken in the movie
keywords	str/list	List of associated keywords

Column Classification in Dataset

Numeric	**Categorical (incl. bool/list)**	**Other/String/ID**
vote_average	status	id
vote_count	adult	title
revenue	original_language	release_date
runtime	genres	backdrop_path

budget	production_companies	homepage
popularity	production_countries	imdb_id
	spoken_languages	original_title
	keywords	overview
		poster_path
		tagline



id	title	# vote_average	# vote_count	status	release_date
Unique identifier for each movie. (type: int)	Title of the movie. (type: str)	Average vote or rating given by viewers. (type: float)	Total count of votes received for the movie. (type: int)	The status of the movie (e.g., Released, Rumored, Post Production, etc.). (type: str)	Date when the movie was released. (type: str)
2 1.50m	1060821 unique values	0 10	0 34.5k	Released 97% In Production 1% Other (19873) 2%	1800-01-01 2099-11-18
27205	Inception	8.364	34495	Released	2010-07-15
157336	Interstellar	8.417	32571	Released	2014-11-05
155	The Dark Knight	8.512	30619	Released	2008-07-16
19995	Avatar	7.573	29815	Released	2009-12-15
24428	The Avengers	7.71	29166	Released	2012-04-25

Figure 1: Sample data from Guest Gale

Architecture overview

This project implements a containerized **ETL (Extract, Transform, Load)** pipeline to prepare data for analytics and recommendation services. The system is built using **Docker Compose** to orchestrate multiple services including Airflow, Spark, and Big - Query integration. Below is a breakdown of the architecture components and their roles in the pipeline.

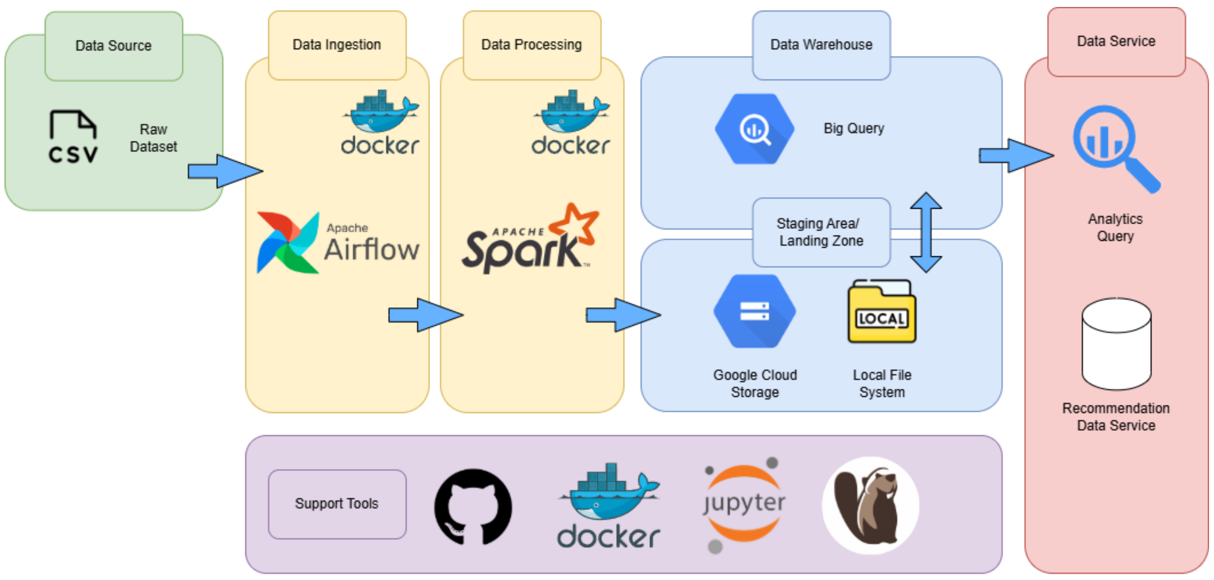


Figure 2: Architecture TMDB RecoFlow Project

Component	**Version/Tag**
Apache Airflow	apache/airflow:2.10.5-python3.12
Apache Spark	bitnami/spark:3.5.2
JupyterLab	jupyter/pyspark-notebook:spark-3.5.0
Docker Engine	Platform base

Pipeline Process

Data Source

- **Format**: CSV files (e.g., TMDB Movie Dataset)
- **Role**: Acts as the raw dataset to be ingested.

Data Ingestion

- **Tool**: Apache Airflow
- **Containerized**: Yes (Docker)

- **Description**: Airflow DAGs orchestrate the loading of raw CSV datasets from external or local sources into the data pipeline. It automates scheduled ingestion and tracks lineage.

Data Processing

- **Tool**: Apache Spark (Bitnami image)
- **Containerized**: Yes (Docker)
- **Description**: Spark processes the ingested raw data by performing transformation, cleaning, and enrichment. The output is a cleaned dataset (CSV format) ready for warehousing.

Data Warehouse

- **Platform**: Google Big Query
- **Zones**:
  - **Staging Area / Landing Zone**: Temporarily holds transformed datasets before final loading.
  - **Cleaned Dataset**: Final structured dataset is loaded into Big Query for analysis and serving.

Data Service

- **Query Layer**: Big Query provides SQL-based analytics queries over the cleaned dataset.
- **Recommendation System Interface**: Cleaned data is used to develop ML-based recommendation systems that query processed features like genres, companies, and languages.

Architecture Details

from Dockerfile, docker compose.yml

This section documents the containerized architecture used for the **ETL_tmdb_dataset** project. It utilizes Docker for environment isolation and service orchestration, consisting primarily of **Apache Airflow**, **Apache Spark**, and **JupyterLab (optional)** components. The configuration is defined in two core files: Dockerfile and docker-compose.yml.

Dockerfile

(Custom Airflow Image)

Base Image

```
FROM apache/airflow:2.10.5-python3.12
```

- The base image is Airflow 2.10.5 with Python 3.12.

System Dependencies

```
USER root
RUN apt-get update && apt-get install -y curl unzip openjdk-17-jdk && apt-get clean
```

- Installs essential tools like curl, unzip, and Java 17 (required by Spark).

Install Spark 3.5.2

```
ENV SPARK_VERSION=3.5.2 ...
RUN curl -L https://.../spark-${SPARK_VERSION}.tgz | tar zx -C /opt
```

- Downloads and installs Spark 3.5.2.
- Spark is installed to /opt/spark

Environment Variables

```
ENV JAVA_HOME=/usr/lib/jvm/java-17-openjdk-amd64
ENV SPARK_HOME=/opt/spark
ENV PATH="${SPARK_HOME}/bin:$PATH"
```

- Sets JAVA_HOME and SPARK_HOME.

- Adds Spark to system PATH.

Install Python Dependencies

```
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt
```

- Installs Python packages needed by the project (likely includes pyspark, google-cloud, etc.).

docker-compose.yml Analysis

The docker-compose.yml file orchestrates the services. Here's a breakdown of the main components:

Service:

airflow-webserver, scheduler, triggerer, worker

- Built using the custom Dockerfile.
- Connected to a shared volume (./dags:/opt/airflow/dags) and /opt/bitnami/spark.
- Uses spark://spark-master:7077 to submit jobs.
- Communicates via the airflow_network.

spark-master

```
image: bitnami/spark:3.5.2
environment:
  - SPARK_MODE=master
```

- Runs Spark in master mode on port 7077 (for job submissions).

spark-worker

```
environment:
  - SPARK_MODE=worker
  - SPARK_MASTER_URL=spark://spark-master:7077
```

- Connects to the Spark master to execute Spark jobs submitted via DAGs.

Volumes and Mounts (Airflow & Spark)

In this architecture, volumes are used to ensure that key files and outputs are shared across containers. This allows Airflow to trigger Spark jobs, Spark to process data, and both to access the same dataset and output.

Host Path	**Container Path**	**Purpose**
./airflow/dags	/opt/airflow/dags	Contains Airflow DAG definitions
./airflow/logs	/opt/airflow/logs	Stores Airflow task logs
./airflow/data	/opt/airflow/data	Used for intermediate data or input/output specific to Airflow tasks
./airflow/plugins	/opt/airflow/plugins	Custom Airflow plugins
./airflow/config	/opt/airflow/config	Custom configuration files for Airflow
./spark/app	/opt/bitnami/spark/app	PySpark application scripts (e.g., clean_data.py, transform_data.py)
./spark/resources	/opt/bitnami/spark/resources	Raw dataset files used by Spark
./spark/output	/opt/shared/output	Output directory for transformed parquet files shared with BigQuery loaders

Timeline Project

Gantt Chart Plan (June–July 2025)

Task	**Week 1**	**Week 2**	**Week 3**	**Week 4**	**Week 5**	**Week 6**	**Week 7**	**Week 8**
Phase 1: Design & Plan (June 2025)	🟩	🟩	🟩	🟩				
Learn Spark & BigQuery	🟩	🟩	🟩	🟩				
Lab Implementation (Component Setup)	🟩	🟩	🟩	🟩				
System Architecture Design			🟩	🟩				
Phase 1 Report Writing			🟩	🟩	🟩			
Phase 2: Implementation (July 2025)					🟩	🟩	🟩	🟩
Component Implementation					🟩	🟩	🟩	🟩
Integration Testing							🟩	🟩
System Demo Preparation								🟩
Final Presentation & Report								🟩

Data Processing Overview

This project follows a standard **ETL (Extract, Transform, Load)** process to ingest and clean movie metadata from a TMDB dataset, structure it using a star schema, and load it into **Google BigQuery** for downstream analytics and machine learning applications. The workflow is fully automated and orchestrated using **Apache Airflow** and **Apache Spark**, and all data exchanges use the **Parquet format** for optimized performance.

ETL_tmdb_dataset DAG

The **ETL_tmdb_dataset** DAG automates the end-to-end ETL pipeline, including validation, cleansing, transformation, and data warehouse loading. It is implemented in Apache Airflow and designed for on-demand execution.

The DAG is defined and executed within **Apache Airflow**, which provides orchestration, scheduling, and monitoring capabilities. It uses the following Airflow features:

DAG Configuration

- **Trigger Mode: Manual (schedule_interval=None)**
- **Catchup: Disabled**
- **Owner: Polakorn Anantapakorn**
- **Tags: project**
- **Connections:**
    - **google_cloud_default: for BigQuery authentication**
    - **spark_default: for submitting Spark jobs**

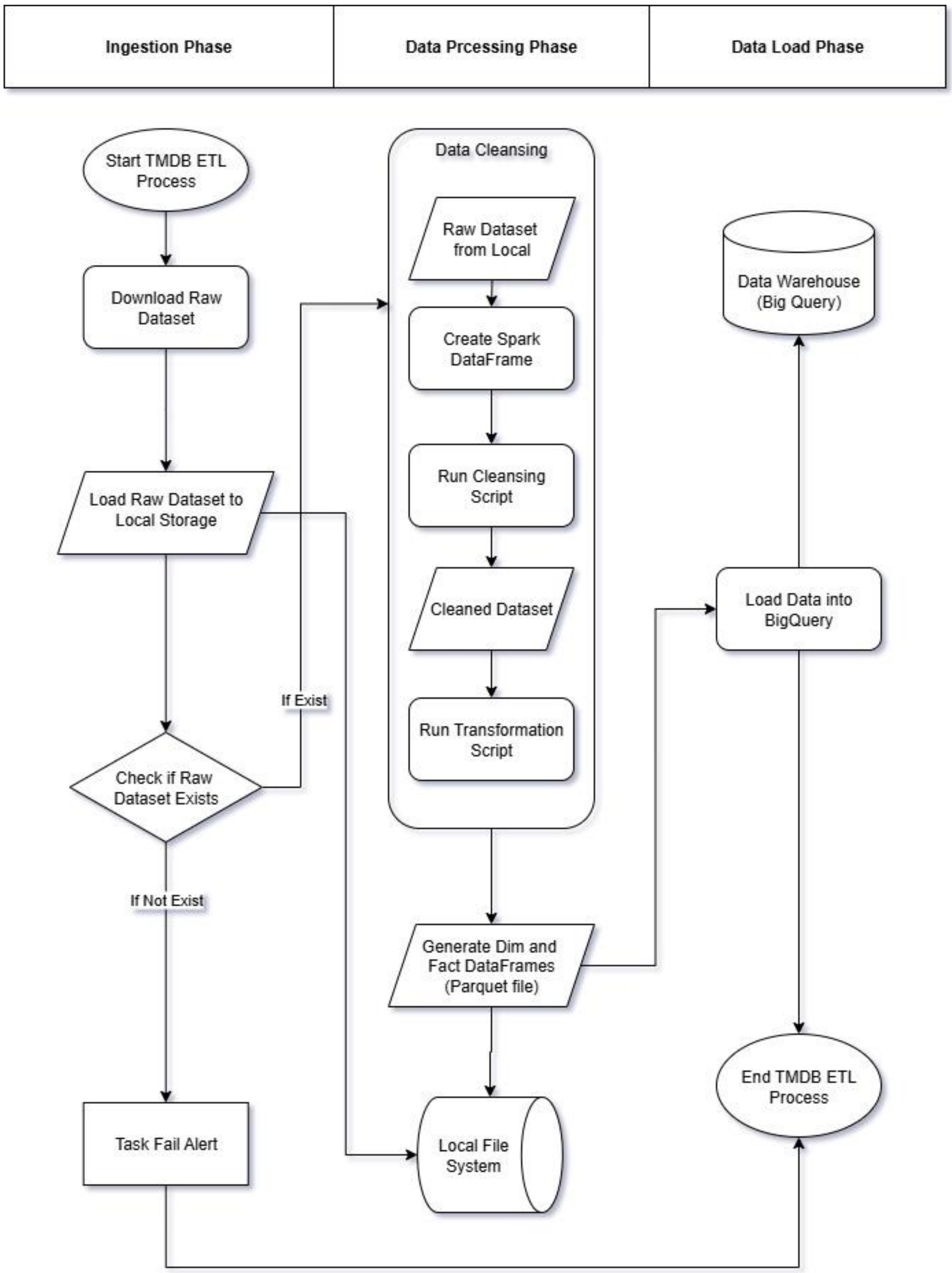


Figure 3: ETL Flow Diagram

Task Groups and Flow

Task	Description
check_dataset_exists	**Validates the existence of the raw CSV file before processing begins**
cleansing_data	**Runs clean_data.py via Spark to clean and standardize raw movie data**
transform_data	**Executes transform_data.py via Spark to reshape cleaned data into dimensional/fact tables**
load_to_bigquery_group	**Uploads each output Parquet directory into its corresponding BigQuery table**
validate_bigquery_group	**Confirms that each BigQuery table exists, has data, and contains the correct schema**

Data Cleansing (clean_data.py)

This step cleans and standardizes the raw dataset (TMDB_movie_dataset_v11.csv) using PySpark. The result is exported as a single Parquet file to reduce complexity when loading into BigQuery.

Key Operations

1. **Column Filtering** Drops unnecessary columns (e.g., poster_path, tagline).
2. **Null Handling** Removes rows missing values in critical fields such as title, genres, overview, and release_date.
3. **Duplicate Removal** Deduplicates entries using movie_id (id column).
4. **Data Type Enforcement** Casts fields to correct types: integers, booleans, and doubles.
5. **Date Validation** Filters movies with release_date between 1700-01-01 and 2025-06-02.
6. **Numeric Field Cleaning** Replaces NA or nulls in budget, revenue, and runtime with 0, then filters invalid (negative) values.
7. **Release Status Filtering** Retains only movies with status = "Released".
8. **String Cleanup** Removes unnecessary quotes from title, overview, and keywords.

9. **Genre Standardization** Cleans genres using a predefined genre list.
10. **Country Normalization** Parses and validates production_countries using an ISO mapping file.
11. **Output Format** Saves the cleaned DataFrame as Parquet using .coalesce(1) to generate a single output file: /opt/shared/output/cleaned_data/

Data Transformation

(transform_data.py)

This step reshapes the cleansed dataset into a star schema using Apache Spark and a modular generator function: process_tables_incrementally().

The script **transform_data.py** performs the following key operations:

Output Tables

1. **Dimension Tables**
   - **dim_movie: Basic movie metadata**
   - **dim_keyword, dim_genre: Exploded from list fields**
   - **dim_production_company, dim_production_country, dim_spoken_language: Derived from normalized string fields**
2. **Bridge Tables**
   - **bridge_movie_genre, bridge_movie_keyword, etc.**
   - **Capture many-to-many relationships between movie_id and attributes**
3. **Fact Table**
   - **fact_movie: Contains metrics like vote_average, budget, revenue, and links to dimension tables**
4. **Output Format**
   Each table is saved to its own Parquet directory:
   - `/opt/shared/output/{table_type}_{table_name}/`

Tables are written incrementally, freeing memory and enabling better Spark performance.

Load to Data Warehouse (BigQuery)

After transformation, the load_to_bigquery_group task group loads each Parquet directory into its corresponding BigQuery table.

Load Behavior

- Uses upload_parquet_folder_to_bq() to detect .parquet files in each output directory
- First file uses WRITE_TRUNCATE; subsequent parts use WRITE_APPEND
- BigQuery table names follow: project.dataset.table

Post-Load Validation

(validate_bigquery_group)

Each loaded table undergoes:

1. **Existence Check** – using BigQueryTableExistenceSensor
2. **Row Count Check** – using BigQueryCheckOperator
3. **Schema Sampling** – using BigQueryGetDataOperator to inspect structure

These checks ensure that data is available, correctly formatted, and query-ready.

Export_Recom_Dataset_from_GCP DAG

This separate DAG handles exporting a **content-based movie recommendation** dataset (view: cbf_movie_recommendations_view) from BigQuery to local storage.

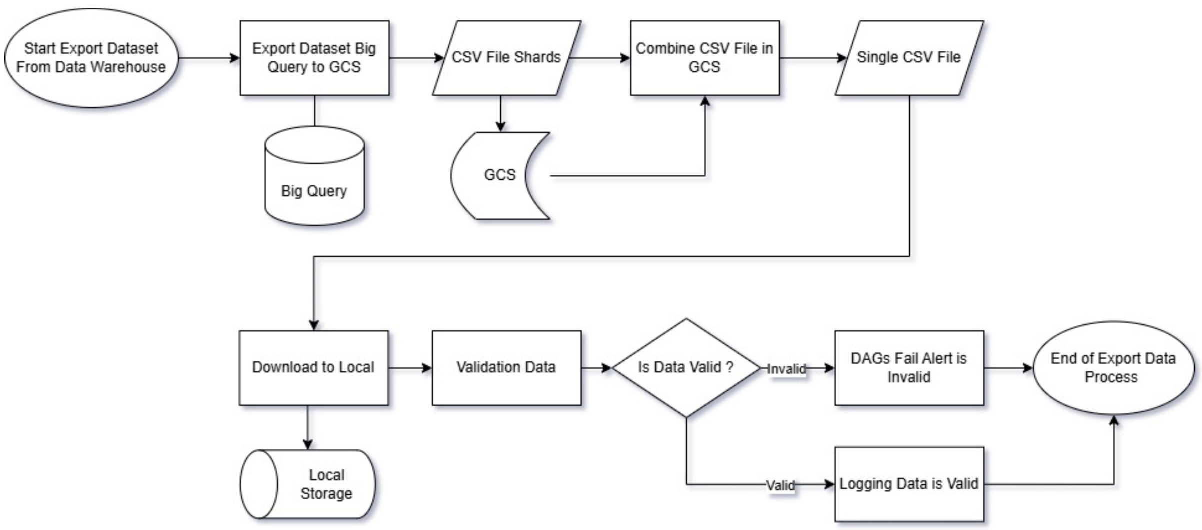


Figure 4: Export Dataset Flow Diagram

DAG Task Flow

Task ID	**Operator**	**Description**
export_view_to_gcs	BigQueryInsertJobOperator	Exports the BigQuery view as multiple **Parquet shards** to a GCS bucket
merge_parquet_shards	PythonOperator (merge_gcs_parquet_shards)	Merges multiple Parquet part files into a single .parquet file in GCS
download_from_gcs	GCSToLocalFilesystemOperator	Downloads the merged Parquet file to the local Airflow instance
validate_data	PythonOperator (validate_parquet)	Validates schema, types, nulls, duplicates, and

		encoding of the final file

All tasks are connected sequentially:

export_view_to_gcs → merge_parquet_shards → download_from_gcs → validate_data

Final Output

The validated dataset is saved to:

```
/opt/airflow/data/cbf_movie.parquet
```

This file is guaranteed to be complete, clean, and formatted for downstream production use.

Data Validation

The final task in the DAG, validate_data, runs a custom Python function validate_parquet to ensure the exported Parquet file is complete, clean, and trustworthy before it is used by downstream systems.

This function performs several **quality checks** on the downloaded file (/opt/airflow/data/cbf_movie.csv) using the pandas library. The validation includes the following steps:

1. **Column Name Validation**
   The function checks whether the columns in the Dataframe exactly match the expected schema:
   ["movie_id", "title", "genres", "keywords", "overview"].
   Any schema mismatch will trigger an exception.

2. **Data Type Validation**

   - movie_id must be of type **integer**.
   - All text-based fields (title, genres, keywords, overview) must be of **string** type.

3. **Null Value Check**
   The function verifies that there are no missing values in any field. If any nulls are found, a detailed report is logged and the DAG fails.

4. **Duplicate ID Check**
   It ensures that all movie_id values are **unique**. Duplicate rows are flagged and shown in the error log.

5. **Text Length Check**
   Empty or whitespace-only values are not allowed in title and overview. The function scans for and flags such records to prevent low-quality descriptions from entering production.

6. **Corrupt Character Detection**
   The function checks for the presence of the Unicode replacement character �, which may indicate encoding issues or corrupt text data. Any field containing this character will fail the validation.

All errors encountered during validation are logged, and an exception is raised to halt the pipeline, ensuring that only high-quality, production-ready data is allowed to pass through.

Outcome

By integrating this validation step, the DAG guarantees that the exported recommendation dataset adheres to strict data quality standards before being shared or served. This helps prevent model degradation, incorrect recommendations, and downstream data integrity issues.

Result

The **Export_Recom_Dataset_from_GCP** DAG successfully delivers a single, validated Parquet file representing the output of a content-based recommendation system. It provides a clean, ready-to-use dataset for use cases such as:

- Deployment to machine learning APIs
- External reporting tools (e.g., Data Studio, Tableau)
- Batch delivery to data consumers or partners

By automating the export and validation steps, this DAG ensures consistency, correctness, and reproducibility of exported recommendation data.

Example Use Case

Content-Based Movie Recommendation

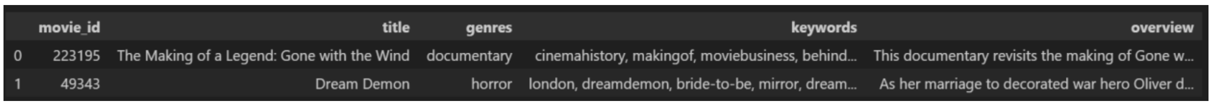
This example demonstrates how to use the final exported dataset (cbf_movie.parquet) from BigQuery to build and evaluate a **content-based recommendation system** using Python and scikit-learn.

Dataset Source

The dataset used: cbf_movie.parquet (cbf/ content-based filtering)

This dataset was previously validated and downloaded from BigQuery via the Export_Recom_Dataset_from_GCP DAG. It includes the following fields:

- movie_id
- title
- genres
- keywords
- overview



	movie_id	title	genres	keywords	overview
0	223195	The Making of a Legend: Gone with the Wind	documentary	cinemahistory, makingof, moviebusiness, behind...	This documentary revisits the making of Gone w...
1	49343	Dream Demon	horror	london, dreamdemon, bride-to-be, mirror, dream...	As her marriage to decorated war hero Oliver d...

Figure 5: Dataset for CBF model

Workflow Steps

Data Loading

- The Parquet file is loaded into a Pandas DataFrame using pd.read_parquet().

Text Preprocessing

- NLTK is used to:
    - Remove stopwords
    - Lemmatize words
- The overview, genres, and keywords are combined into a single combined_text feature for vectorization.

TF-IDF Vectorization

- The combined text is transformed into numerical vectors using TfidfVectorizer.
- This represents the importance of words within each movie's metadata.

Cosine Similarity Matrix

- Cosine similarity is calculated between all movie vectors.
- This matrix allows measuring how similar each movie is to every other movie based on textual metadata.

Recommendation Function

- A function get_recommendations(title) returns the top N most similar movies based on cosine similarity.
- get_recommendations("The Dark Knight")

How the Model Works

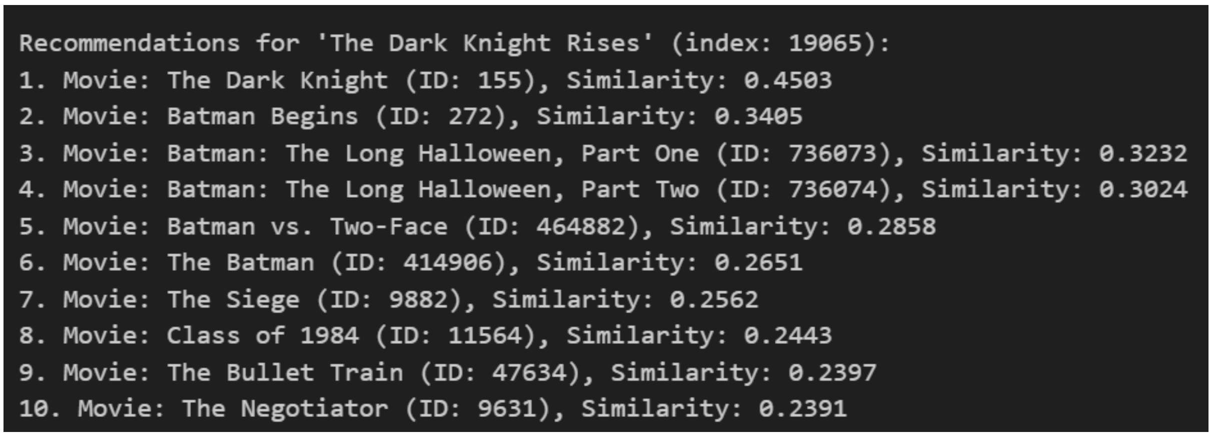
- **Type**: Content-based filtering
- **Input**: Metadata text (overview + genres + keywords)
- **Vectorization**: TF-IDF (Term Frequency–Inverse Document Frequency)
- **Similarity**: Cosine similarity between movies' text features
- **Output**: Ranked list of movies most similar to the input movie

This method doesn't require user interaction history. Instead, it focuses on item (movie) features.

Results

Example output for the input movie "The Dark Knight":

Rank	Recommended Movie	Similarity Score
1	Batman Begins	0.33
2	Batman: The Long Halloween, Part One	0.31
3	The Batman	0.25
...	...	...



```
Recommendations for 'The Dark Knight Rises' (index: 19065):
1. Movie: The Dark Knight (ID: 155), Similarity: 0.4503
2. Movie: Batman Begins (ID: 272), Similarity: 0.3405
3. Movie: Batman: The Long Halloween, Part One (ID: 736073), Similarity: 0.3232
4. Movie: Batman: The Long Halloween, Part Two (ID: 736074), Similarity: 0.3024
5. Movie: Batman vs. Two-Face (ID: 464882), Similarity: 0.2858
6. Movie: The Batman (ID: 414906), Similarity: 0.2651
7. Movie: The Siege (ID: 9882), Similarity: 0.2562
8. Movie: Class of 1984 (ID: 11564), Similarity: 0.2443
9. Movie: The Bullet Train (ID: 47634), Similarity: 0.2397
10. Movie: The Negotiator (ID: 9631), Similarity: 0.2391
```

Figure 6: Example Results of Recommendation Model

The model successfully retrieves thematically and narratively similar movies, reflecting strong alignment with user expectations based on content.

Summary

This notebook demonstrates how the exported dataset from your Airflow pipeline can be directly used to:

- Build practical recommendation models
- Enable intelligent search and personalization features
- Support downstream ML pipelines and analytics

It proves the production readiness and usability of the entire ETL pipeline and exported dataset.