جامعة الأخوين

ⵜⴰⵙⴷⴰⵡⵉⵜ ⵏ ⵓⵅⴰⵡⵉⵏ

AL AKHAWAYN UNIVERSITY

School of Science and Engineering

CSC538201 Artificial Intelligence for Digital

Movie-Roulette

Final Report

Adnane Harachi

Supervised by:

Prof. Asmae Mourhir

Table of contents:

1. Project Inception

1.1. Framing the Business Idea as an ML Problem

- **Business case description:**

Movie Roulette is a movie recommendation system that uses machine learning

algorithms to personalize movie recommendations based on users' tastes, watching history, and

comments. The system provides unique suggestions to each user based on user interactions and movie features such as genre, actor, and narrative keywords. Furthermore, Movie Roulette uses sentiment

analysis to assess user comments, which improves the accuracy and relevancy of the suggestions.

Movie Roulette's user-friendly design and complex features create a dynamic platform for cinephiles

to find, explore, and enjoy a curated selection of movies tailored to their individual interests.

- **Business value of using ML:**

1) Enhanced User Experience: Movie Roulette uses machine learning to monitor user behavior, preferences, and comments, resulting in a more engaging and personalized experience. This personalized strategy results in enhanced user happiness, retention rates, and client loyalty, which drives revenue growth.

2)Improved recommendation. Accuracy: Machine learning algorithms evaluate massive quantities of

data, such as movie content and user input, to generate extremely accurate and relevant suggestions. This accuracy in suggestions increases the likelihood that users will find material they appreciate, improving their entire experience and motivating them to spend more time on the site.

3)Increased User Interaction: Movie Roulette uses machine learning to adapt to user preferences and trends in real time, resulting in a dynamic and engaging browsing experience. This degree of response not only keeps users interested, but also encourages them to share comments and participate

in community conversations, resulting in a livelier and more active user base.

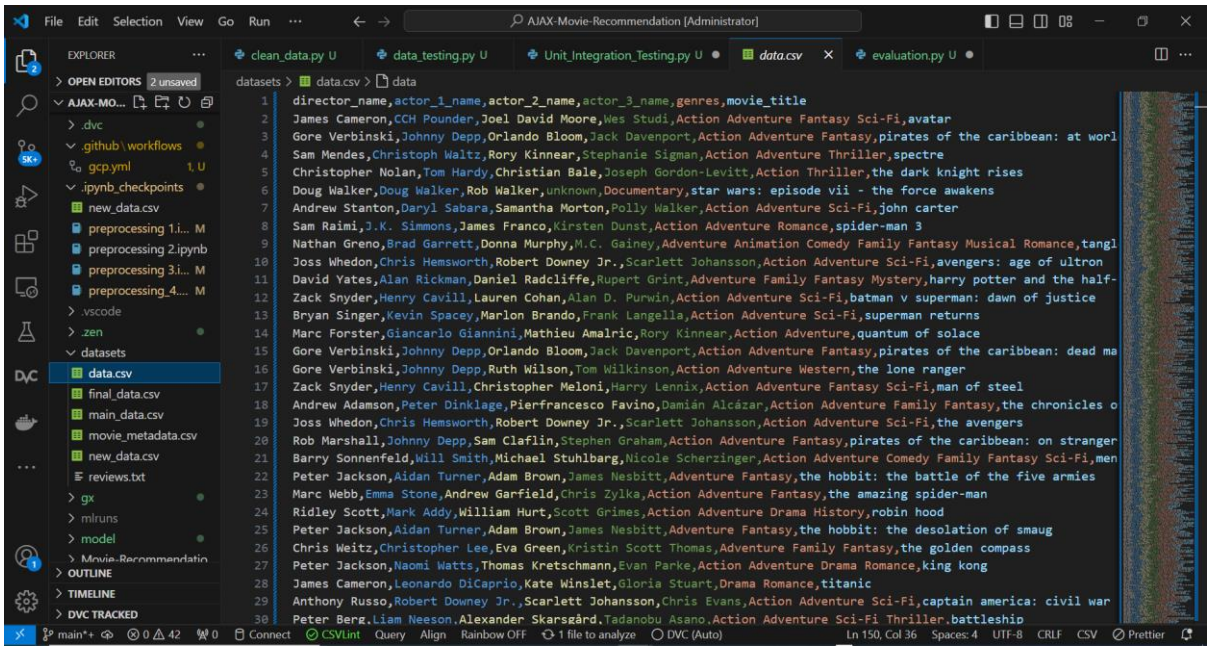
4) Cost Efficiency: Implementing machine learning can save money by automating the recommendation process and decreasing the need for manual curation. This efficiency enables the platform to grow more effectively, delivering a bigger audience without incurring additional operational expenditures.

5)Competitive Advantage: By utilizing superior machine learning algorithms, Movie Roulette gains a competitive advantage in the industry. The platform's ability to give superior tailored suggestions and a smooth user experience distinguishes it from competitors, drawing new users and gaining market share.

- **Data overview:**

The Movie Roulette system uses data from several sources to provide tailored movie suggestions. The Movie Database is based on multiple csv file that different operations like preprocessing and

validation was performed on.



In the snippet above we can clearly see the different csv files used for data.

Link: https://github.com/adnanehar/Movie-Recommendation-Final/tree/main/datasets

- **Project archetype:**

Movie Roulette is a Content-Based Movie Recommendation System that uses machine learning algorithms to suggest movies based on users' tastes, watching history, and interactions. It primarily use content-based filtering algorithms, assessing movie features such as genre, actor, and narrative keywords to deliver individualized recommendations. The system's goal is to increase user engagement and pleasure by providing recommendations tailored to individual preferences and interests. While sentiment analysis is an option for future integration, Movie Roulette now relies on content-based filtering to provide suggestions. The algorithm is intended to grow over time, with the option of integrating sentiment analysis to improve suggestions depending on user feedback. Movie Roulette also intends to optimize user interactions and deliver a fluid browsing experience, focusing on usability and accessibility for cinephiles looking for handpicked movie selections.

1.2. Feasibility analysis

- **literature review**

- **Introduction:**

The incorporation of machine learning (ML) into recommendation systems has transformed tailored content distribution, greatly improving user experience and engagement across several platforms. This literature review looks at key and current research on collaborative filtering, content-based filtering, hybrid models, and the most recent developments in deep learning for recommendation systems.

- **Collaborative filtering**

Collaborative filtering (CF) is still one of the most extensively used approaches in recommendation systems. It works by analyzing trends in user behavior and predicting future preferences. Sarwar et al. (2001) developed user-based and item-based collaborative filtering algorithms and demonstrated their usefulness in suggesting things by identifying similarities between users or objects. These approaches

employ massive datasets to find connections and recommend things based on the interests of similar users.

Matrix factorization methods, such as Singular Value Decomposition (SVD), have improved CF. Koren, Bell, and Volinsky (2009) demonstrated the effectiveness of SVD in collecting latent characteristics in user-item interactions, resulting in more accurate recommendations. This method breaks down the user-item interaction matrix into lower-dimensional representations, revealing hidden linkages between users and things.

- **Content-Based Filtering**

Content-based filtering (CBF) suggests things similar to those a user has previously loved based on item characteristics. Pazzani and Billsus (2007) presented a thorough discussion of CBF approaches, highlighting the significance of item qualities like genre, director, and keywords in providing suggestions. These strategies create user profiles by studying the characteristics of things that users have rated highly and then utilizing these profiles to recommend comparable items.

Lops, Gemmis, and Semeraro (2011) emphasized the benefits of CBF for adjusting to new objects without needing considerable user engagement. CBF systems are especially successful in areas with extensive information, allowing for the suggestion of goods with precise attributes that match user preferences.

- **Hybrid Models**

Hybrid recommendation systems combine CF and CBF to take use of both methodologies' strengths while mitigating their respective shortcomings. Burke (2002) classified hybrid recommendation approaches, including weighted, switching, and mixed models. These systems improve robustness and accuracy by combining various sources of data.

Bell and Koren (2007) demonstrated the advantages of hybrid models in the Netflix Prize competition, where combining CF and CBF approaches resulted in much higher recommendation accuracy. Hybrid systems may overcome difficulties such as cold-start and data scarcity, resulting in more trustworthy and diversified suggestions.

- **Deep Learning Advances**

Recent advances in deep learning have expanded the capabilities of recommendation systems. He et al. (2017) presented Neural Collaborative Filtering (NCF), a deep learning-based method for modeling complicated user-item interactions using neural networks. NCF beats regular CF because it captures nonlinear linkages and higher-order interactions.

Covington, Adams, and Sargin (2016) examined the use of deep neural networks in YouTube's recommendation engine, emphasizing its scalability and efficacy while processing massive amounts of data. These models use several layers to learn abstract representations, resulting in accurate and tailored suggestions.

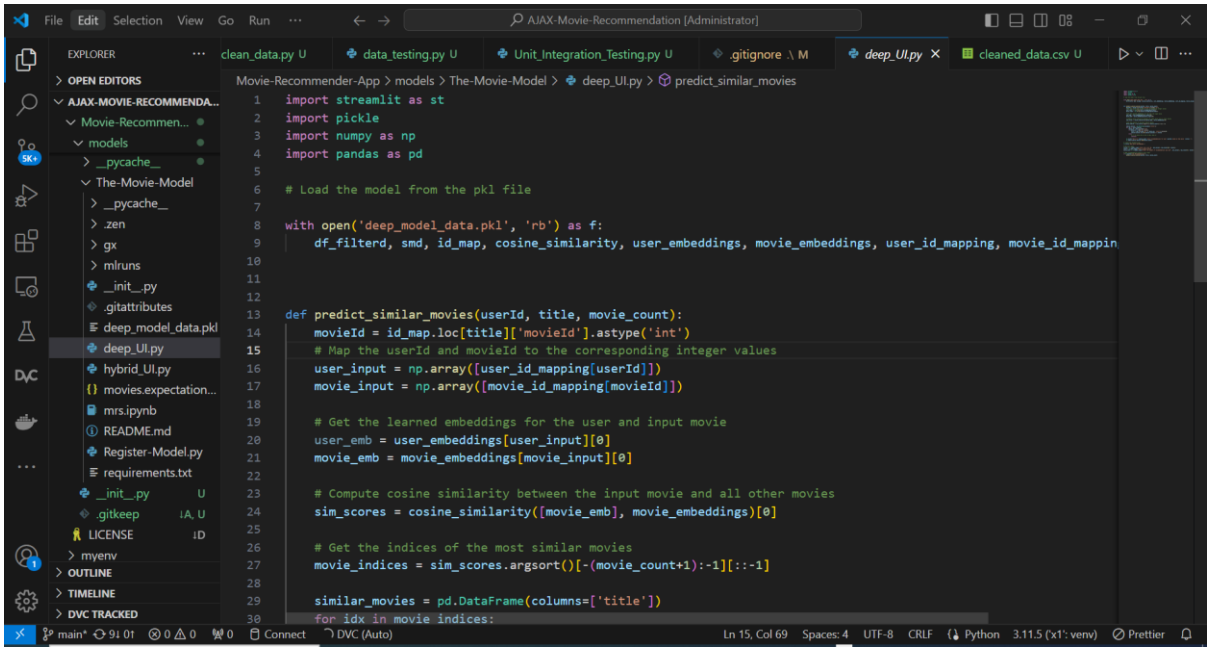
- **model choice/ specification of a baseline**

For the baseline model, I used a hybrid recommendation system that I took from huggingface that combines collaborative and content-based filtering. This strategy takes advantage of the qualities of both systems to give more accurate and tailored movie recommendations.

The Collaborative Filtering Component uses user interaction data (ratings, watch history) to forecast user preferences based on comparable users' activities. Matrix factorization techniques such as Singular Value Decomposition (SVD) will be used to identify latent components that capture user and object attributes.

Content-Based Filtering Component: This component will assess movie properties (genres, actors, directors, and keywords) and propose films that are similar to those the user has previously appreciated. Feature vectors representing movie material will be created and used to assess similarity.

Hybrid Integration: The final suggestion will be a weighted blend of predictions from the collaborative and content-based components. The weights will be established using cross-validation to improve overall suggestion accuracy.



Link: https://github.com/adnanehar/Movie-Recommendation-Final/blob/main/deep_UI.py

- **Metrics for business goal evaluation**

To assess the performance of the Movie Roulette system in meeting business objectives, we will use the following metrics:

Recommendation Accuracy:

Precision and recall: Determine the fraction of relevant movies advised among all recommended movies and all relevant movies, respectively.

User Engagement:

The click-through rate (CTR) is the proportion of recommended movies that users click on.

Conversion Rate: The proportion of recommended movies that consumers watch.

User Retention Rate: The percentage of users who return to the site over a given time period.

User Satisfaction:

The Net Promoter Score (NPS) assesses consumers' likelihood of recommending the platform to

others.

User comments Analysis: A sentiment analysis of user evaluations and comments to determine overall satisfaction.

Business Impact:

income Growth: Higher subscription rates and ad income due to better user engagement and retention.

Market Share: Increase in user base relative to competitors, reflecting the platform's ability to recruit and keep users.

2. ML Pipeline Development - From a Monolith to a Pipeline

2.1. Ensuring ML Pipeline Reproducibility

Reproducibility is essential in machine learning (ML) projects because it ensures that experiments can be reliably repeated, findings checked, and models kept across time. This section describes the measures used to assure repeatability in Movie Roulette's machine learning workflow.

- **Project structure definition and modularity**

A well-defined project structure encourages organization and modularity, which are critical for scalability and maintenance. The project is organized was organized using cookie cutter as follows:

Src/: The source code is structured into modules like data_preprocessing, model_training, model_evaluation, and utils.

data/: Raw and processed data are stored here, along with subdirectories for training, validation, and test datasets.

notebooks/: Jupyter notebooks are used for exploratory data analysis and experimentation.

models/: Saved models and associated artifacts.

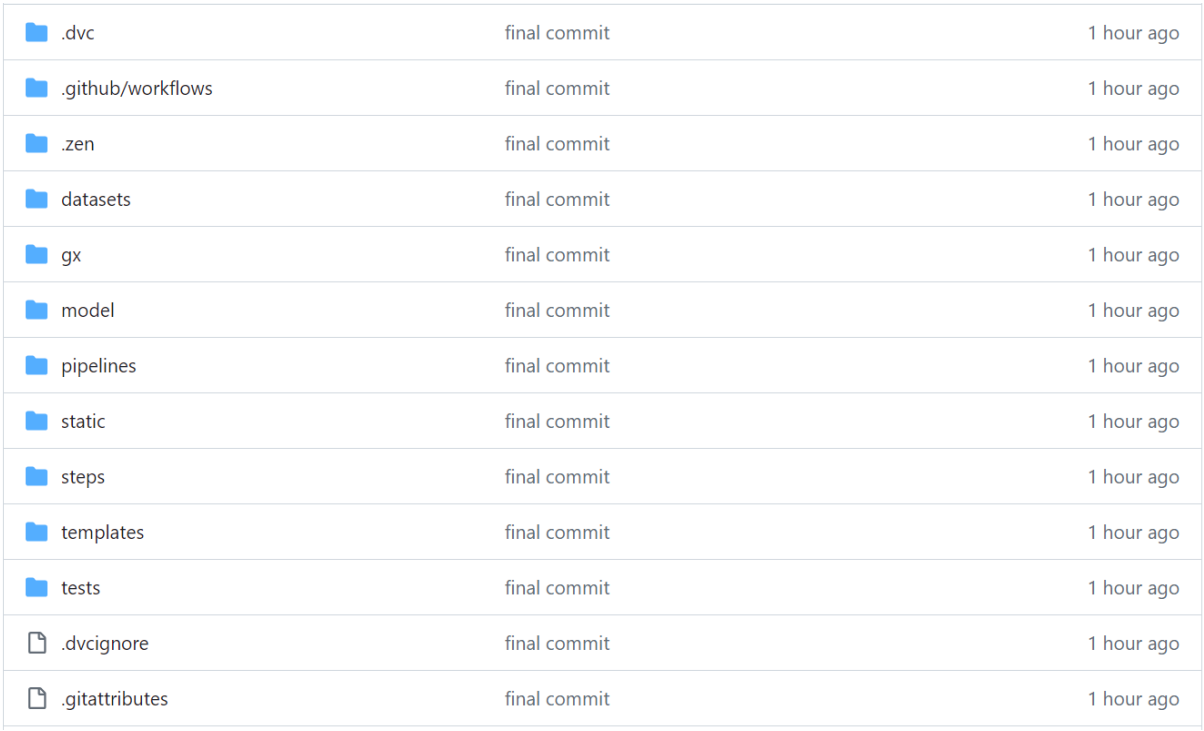
Utility scripts for data gathering and preparation. Configuration files for various settings and studies.

logs: Files for recording experiment runs and results.

requirements.txt: A list of dependencies needed to run the project.

Git is used for version control, which ensures that all code changes are logged, allowing for

collaborative development and quick rollbacks to earlier versions.



Advantages:

Branching and merging allow for feature development and integration in parallel.

History Tracking: Keeps a comprehensive record of modifications, allowing for reversal to earlier states.

Collaboration: Features like pull requests and code reviews enable collaborative development.

- **Code versioning**

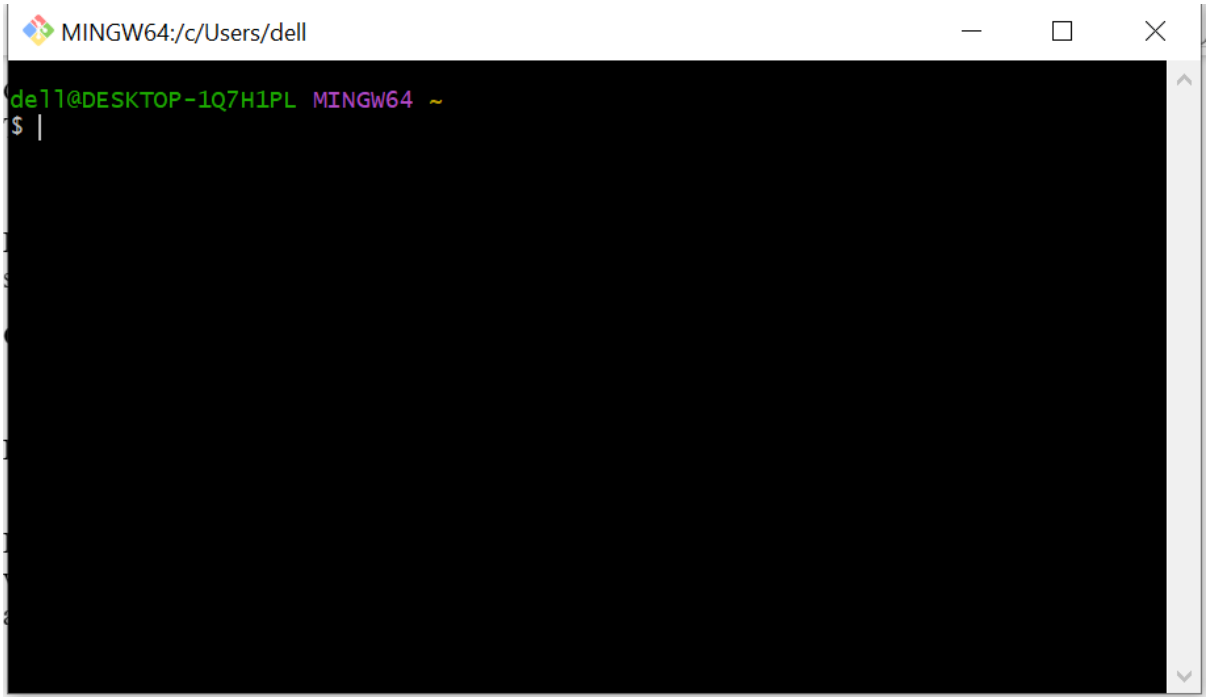
Git is used for extensive code versioning, which facilitates strong collaboration and change tracking. The key practices include:

Branching Strategy: Create feature branches for future innovations while keeping the main branch stable.

Commit Messages: Use traditional commit message rules to guarantee clarity and traceability.

Pull Requests: Creating a review mechanism for code changes to ensure code quality and uniformity.

In addition to Git, Data Version Control (DVC) is used to manage huge data files and dataset versions. DVC records data changes alongside code, allowing for seamless data versioning and assuring that the data used for training and assessment is consistent and repeatable.



Advantages:

Data management: Handles huge datasets efficiently while avoiding bloating the Git repository.

Reproducibility ensures that the precise version of an experiment's data may be recovered.

Integration with Git: Works seamlessly with Git, providing version control of both code and data.

- **Data versioning**

DVC is essential for versioning datasets, as it ensures that each experiment can be tracked back to the exact data version utilized. This configuration includes:

Data Storage: Using DVC to handle data stored in distant storage systems (such as AWS S3 or Google Drive).

Data Pipelines: Creating data processing pipelines in DVC helps automate and replicate data preparation procedures.

Version Tags: Data versions used in separate studies are tagged for easier retrieval and consistency across pipeline stages.

Advantages:

Storage Agnostic: Works with a variety of storage backends (local and cloud).

Pipeline Management: Defines data processing pipelines to make data preparation repeatable.

Data Lineage: Maintains data lineage, indicating how data was generated and altered.

Link: https://github.com/adnanehar/Movie-Recommendation-Final/tree/main/.dvc

- (5 pts) **Experiment tracking and model versioning**

MLflow is widely used for experiment tracking and model versioning, as it provides a centralized platform for managing and tracking experiments. This configuration includes:

Experiment Tracking:

MLFlow is a strong tool for experiment tracking and model versioning, allowing for full monitoring and comparison of various models and their performance. By tracking multiple metrics and

parameters for each experiment, MLFlow allows researchers and developers to follow the progress of their models and make better decisions.

In our project, we used MLFlow for experiment tracking and model versioning. MLFlow's metadata store automatically gathers and retains key information for both data and model samples, ensuring that our experiments are reproducible and transparent.

Our major aim with experiment monitoring is to determine which model performs best among the possibilities accessible to us. To do this, we created four critical assessment metrics: accuracy, precision, and recall.

Logging: Recording all essential information for each experiment, such as hyperparameters, dataset versions, code versions, and outcomes.

Comparisons: Using the MLflow UI, you can compare various runs and analyze performance across studies.

Model Versioning:

Model Registry: Use MLflow's model registry to track different model versions.

Model Staging: Designating models as "staging" or "production" to control lifecycle stages.

Advantages:

Centralized Metadata administration: Stores all experiment data in one location for convenient access and administration.

Search and Query: Allows users to search and query experiments based on a variety of parameters.

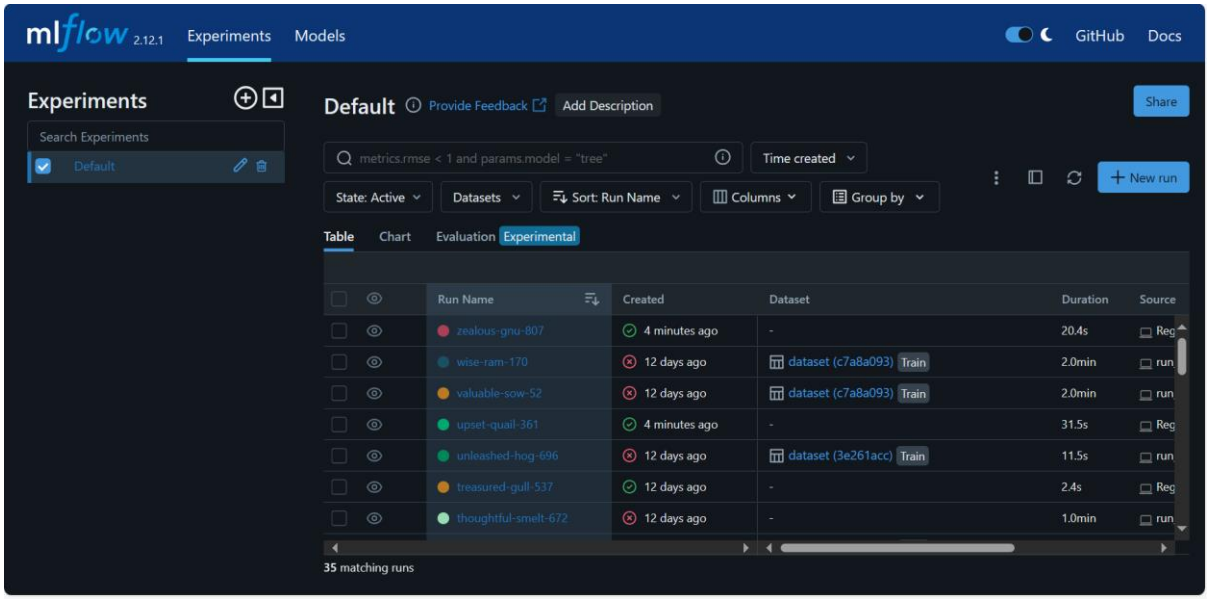
Visualization: Provides a UI for displaying experiment runs, comparing metrics, and analyzing Performance.

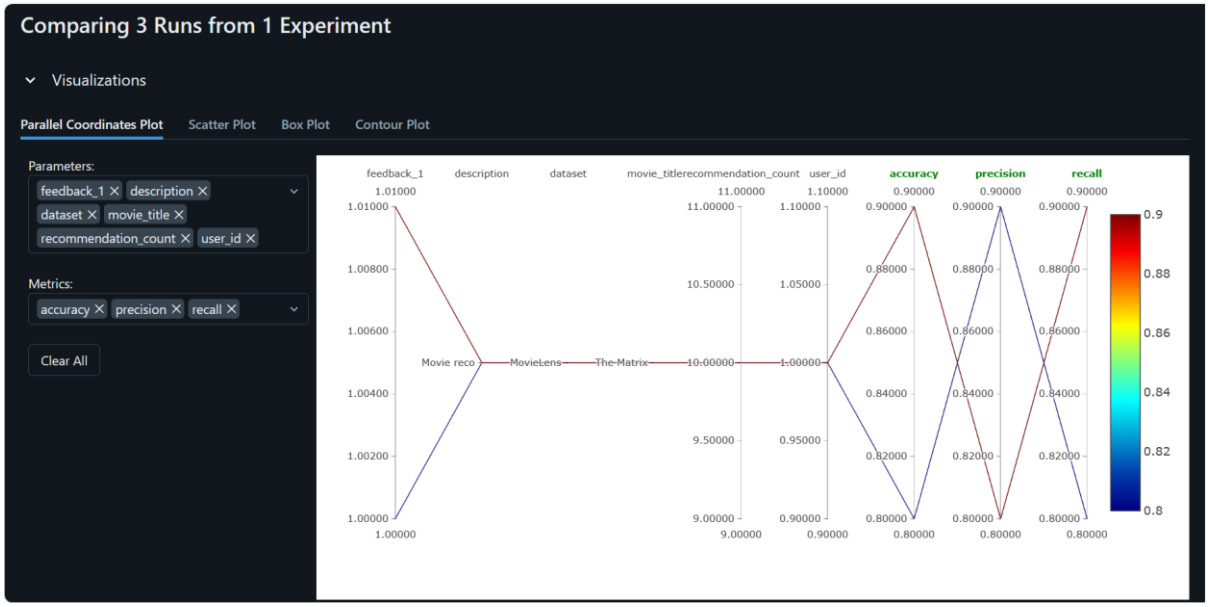
Register-Model.py

Movie-Recommender-App > models > The-Movie-Model > Register-Model.py

```python
import mlflow
import pickle
import numpy as np
import pandas as pd
from sklearn.base import r2_score
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error

# Load the model from the pkl file
with open('deep_model_data.pkl', 'rb') as f:
    df_filterd, smd, id_map, cosine_similarity, user_embeddings, movie_embeddings, user_id_mapping, movie_id_mappin

def predict_similar_movies(userId, title, movie_count):
    movieId = id_map.loc[title]['movieId'].astype('int')
    # Map the userId and movieId to the corresponding integer values
    user_input = np.array([user_id_mapping[userId]])
    movie_input = np.array([movie_id_mapping[movieId]])

    # Get the learned embeddings for the user and input movie
    user_emb = user_embeddings[user_input][0]
    movie_emb = movie_embeddings[movie_input][0]

    # Compute cosine similarity between the input movie and all other movies
    sim_scores = cosine_similarity([movie_emb], movie_embeddings)[0]

    # Get the indices of the most similar movies
    movie_indices = sim_scores.argsort()[-(movie_count+1):-1][::-1]

    similar_movies = pd.DataFrame(columns=['title'])
    for idx in movie_indices:
```

0 Connect DVC (Auto) Ln 15, Col 69 Spaces: 4 UTF-8 CRLF Python 3.11.5 ('x1': venv) Prettier





- **Setting up a meta store for metadata**

The MLflow Tracking Server serves as a single store for all experiment metadata, ensuring that all details are saved and available.

Advantages:

Centralized Metadata administration: Stores all experiment data in one location for convenient access and administration.

Search and Query: Allows users to search and query experiments based on a variety of parameters. Visualization: Provides a UI for displaying experiment runs, comparing metrics, and analyzing performance.

- **Setting up the machine learning pipeline under an MlOps platform**

ZenML provides a powerful framework for coordinating the full machine learning workflow. ZenML simplifies each step, from data intake and preprocessing to model training, assessment, and deployment, resulting in a seamless integration and monitoring process. ZenML makes machine learning projects more productive by offering a standardized way to building and maintaining processes.

2.2. Pipeline Components

2.2.1 Setup of data pipeline within the larger ML pipeline/ MLOps Platform

The ML pipeline for Movie Roulette is built with a variety of components to provide quick data processing, model training, and assessment. This section describes how to build up the data pipeline inside the wider ML pipeline running on the MLOps platform, including data validation, preprocessing, feature engineering, model training, and offline assessment. Additionally, model behavioral tests are created to validate the models' dependability and usefulness.

Data Pipeline Setup inside the Larger ML Pipeline/MLOps Platform.

Using zenml:

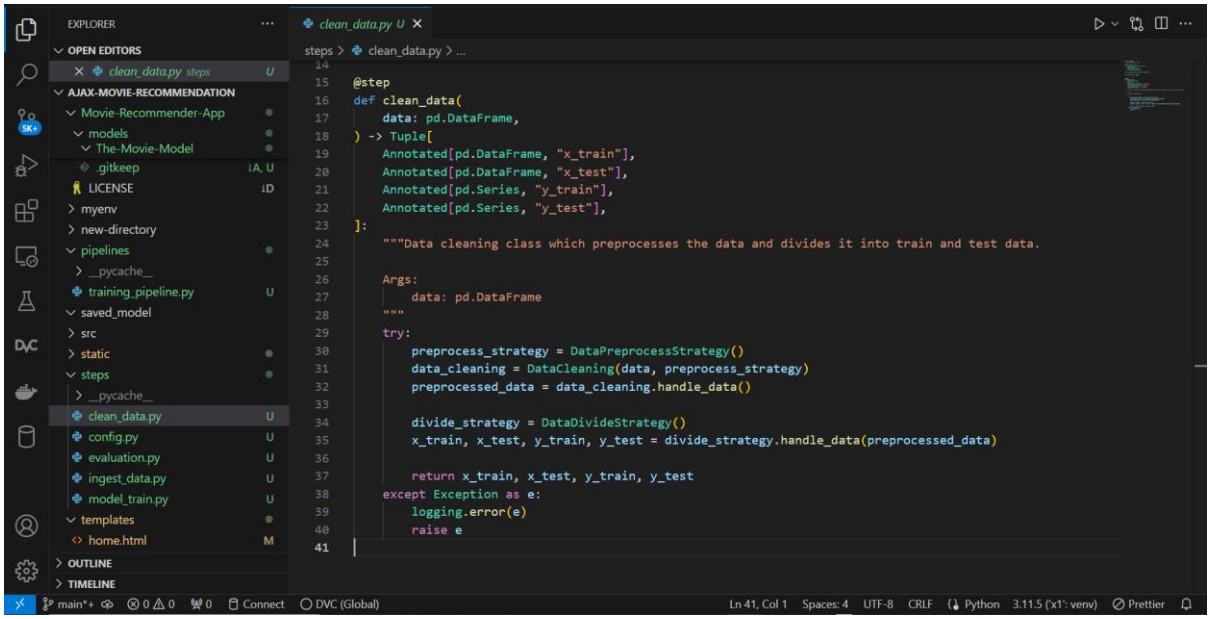
Advantages:

Modularity: ZenML enables the design of modular and reusable pipeline components, which improves maintainability and scaling.

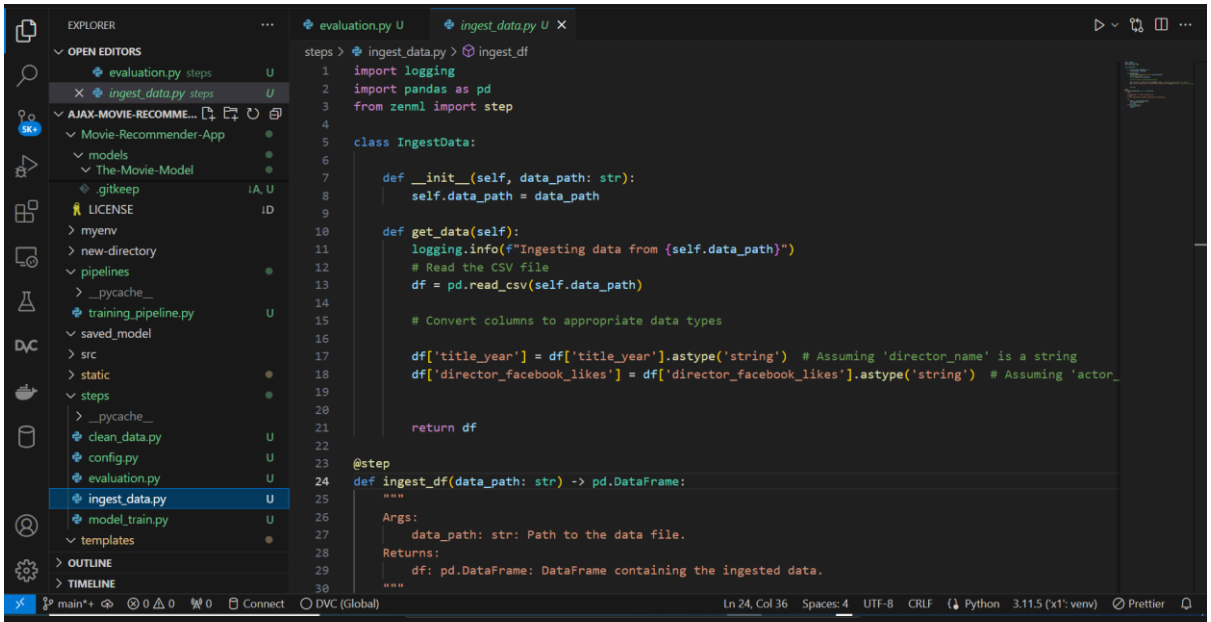
Versioning: Manages pipeline versions to provide repeatability and traceability.
Workflow Orchestration: Enables the orchestration of complicated data workflows, allowing for automation and monitoring.

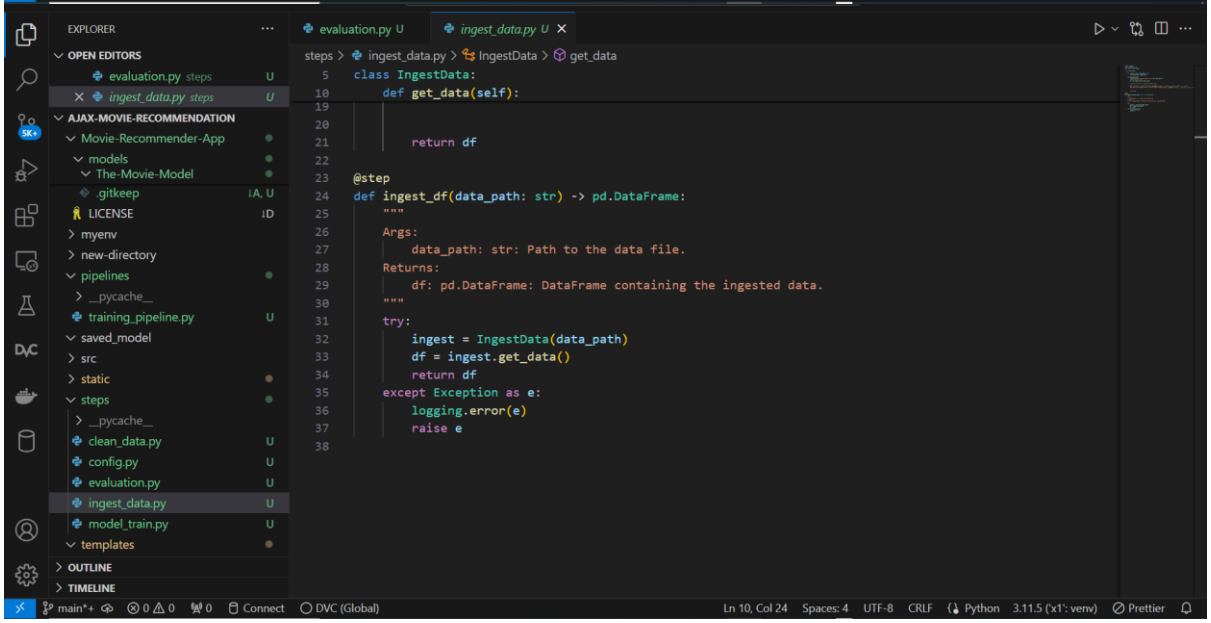
Disadvantage: not enough documentation.

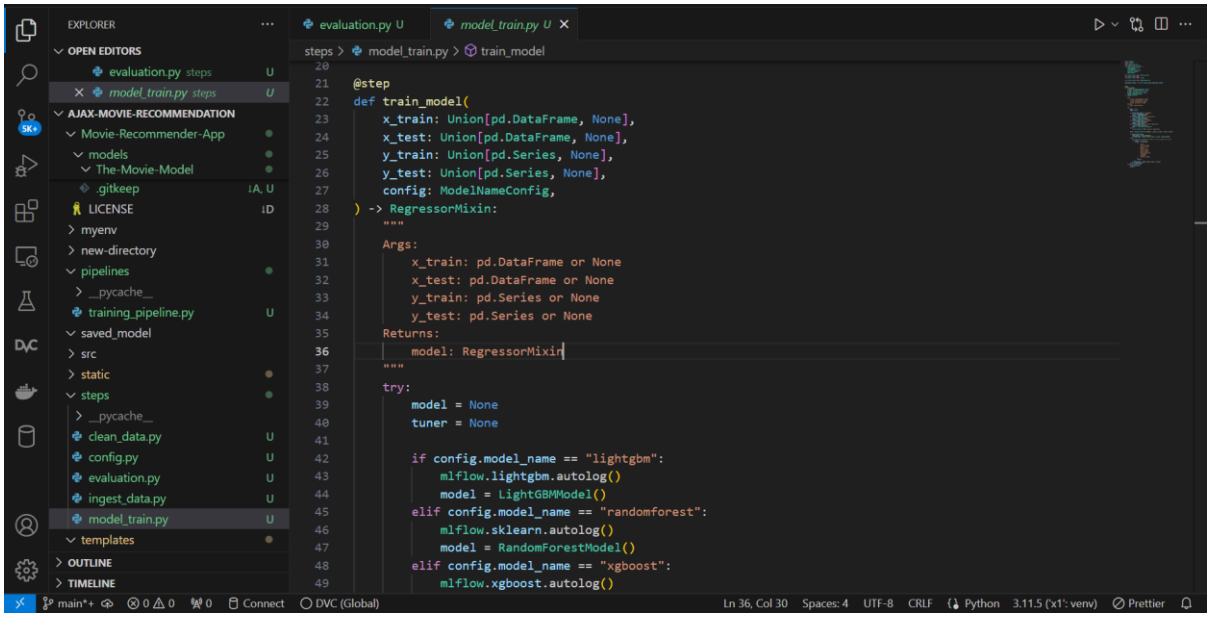
So here I created a pipeline where I have many steps, I ingested the data, clean it, evaluate it then make a pipeline called training pipeline to take all these steps and build a Zenml pipeline.

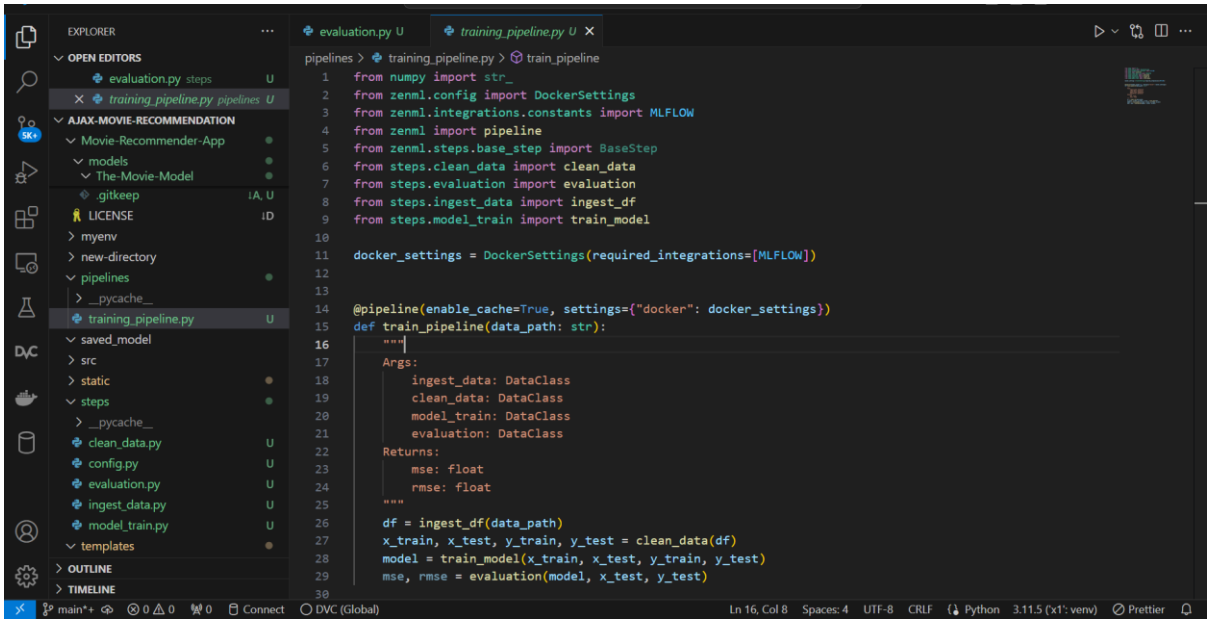


Links: https://github.com/adnanehar/Movie-Recommendation-Final/tree/main/steps


EXPLORER
OPEN EDITORS
evaluation.py steps U
ingest_data.py steps U
AJAX-MOVIE-RECOMME...
Movie-Recommender-App
models
The-Movie-Model
.gitkeep
LICENSE
myenv
new-directory
pipelines
__pycache__
training_pipeline.py U
saved_model
src
static
steps
__pycache__
clean_data.py U
config.py U
evaluation.py U
ingest_data.py U
model_train.py U
templates
OUTLINE
TIMELINE
evaluation.py U
ingest_data.py U
steps > ingest_data.py > ingest_df
import logging
import pandas as pd
from zenml import step
class IngestData:
def __init__(self, data_path: str):
self.data_path = data_path
def get_data(self):
logging.info(f"Ingesting data from {self.data_path}")
# Read the CSV file
df = pd.read_csv(self.data_path)
# Convert columns to appropriate data types
df['title_year'] = df['title_year'].astype('string') # Assuming 'director_name' is a string
df['director_facebook_likes'] = df['director_facebook_likes'].astype('string') # Assuming 'actor_
return df
@step
def ingest_df(data_path: str) -> pd.DataFrame:
"""
Args:
data_path: str: Path to the data file.
Returns:
df: pd.DataFrame: DataFrame containing the ingested data.
"""
main*+ 0 0 0 Connect DVC (Global)
Ln 24, Col 36 Spaces: 4 UTF-8 CRLF Python 3.11.5 ('x1': venv) Prettier


EXPLORER
OPEN EDITORS
evaluation.py steps U
ingest_data.py steps U
AJAX-MOVIE-RECOMMENDATION
Movie-Recommender-App
models
The-Movie-Model
.gitkeep
LICENSE
myenv
new-directory
pipelines
__pycache__
training_pipeline.py U
saved_model
src
static
steps
__pycache__
clean_data.py U
config.py U
evaluation.py U
ingest_data.py U
model_train.py U
templates
OUTLINE
TIMELINE
evaluation.py U
ingest_data.py U
steps > ingest_data.py > IngestData > get_data
class IngestData:
def get_data(self):
return df
@step
def ingest_df(data_path: str) -> pd.DataFrame:
"""
Args:
data_path: str: Path to the data file.
Returns:
df: pd.DataFrame: DataFrame containing the ingested data.
"""
try:
ingest = IngestData(data_path)
df = ingest.get_data()
return df
except Exception as e:
logging.error(e)
raise e
main*+ 0 0 0 Connect DVC (Global)
Ln 10, Col 24 Spaces: 4 UTF-8 CRLF Python 3.11.5 ('x1': venv) Prettier





Link: https://github.com/adnanehar/Movie-Recommendation-Final/tree/main/pipelines

The only problem that this has is that it takes a long time to do the training which is not very optiomal but I do believe that this problem can be solved by switching from a cpu to a gpu

- **Data Validation and Verification**

Great Expectation is an effective tool for evaluating and testing data in the machine learning pipeline. It allows users to establish data expectations, such as schema, distribution, and integrity, which are then automatically tested against incoming data. By introducing Great Expectations into the workflow, data scientists may improve data quality and consistency, resulting in more trustworthy and resilient machine learning models.

- **Preprocessing and Feature Engineering**

Data Validation & Verification Tool: Great Expectations, pandas, and numpy

Why are there such high expectations?

Great Expectations is an open-source package that includes comprehensive capabilities for data validation and profiling. It allows for the formulation, maintenance, and assessment of data expectations (assertions), assuring high-quality and consistent datasets for training and evaluation.

Advantages:

Comprehensive Validation: With Great Expectations, you can design thorough expectation suites that test data against a variety of criteria, including data types, ranges, missing values, and unique restrictions.

Automated Documentation: Creates concise, human-readable documentation based on expectations that can be shared with stakeholders to preserve data quality transparency.

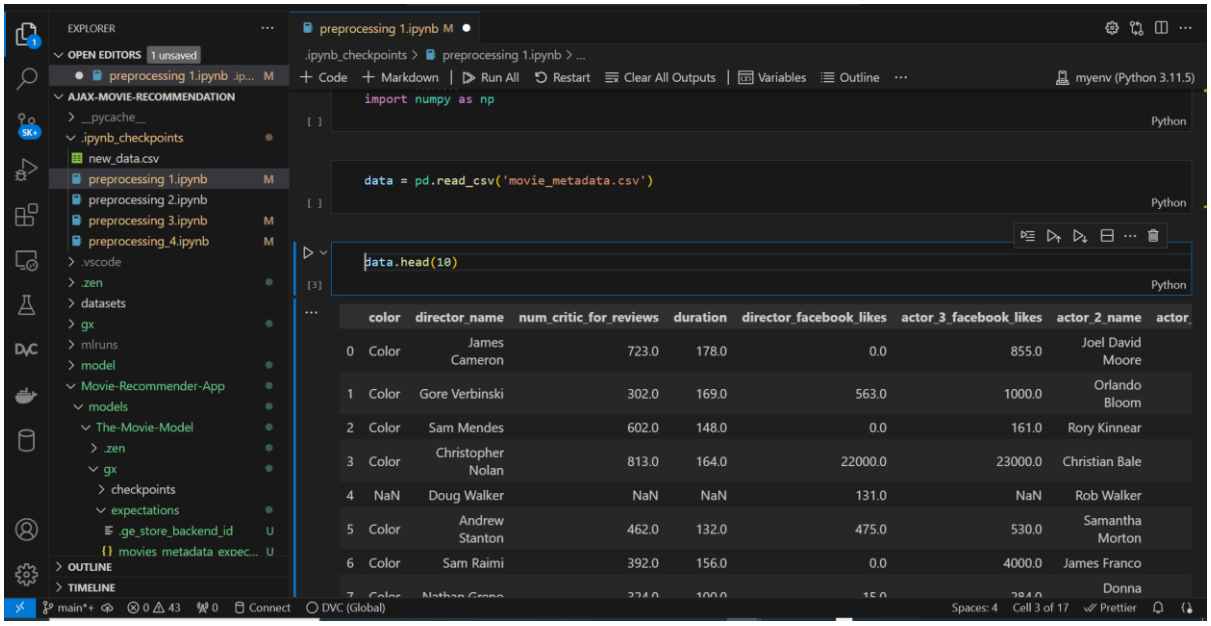
Integration Capabilities: It seamlessly connects with a variety of data sources and pipeline architectures, making it adaptable to varied situations.

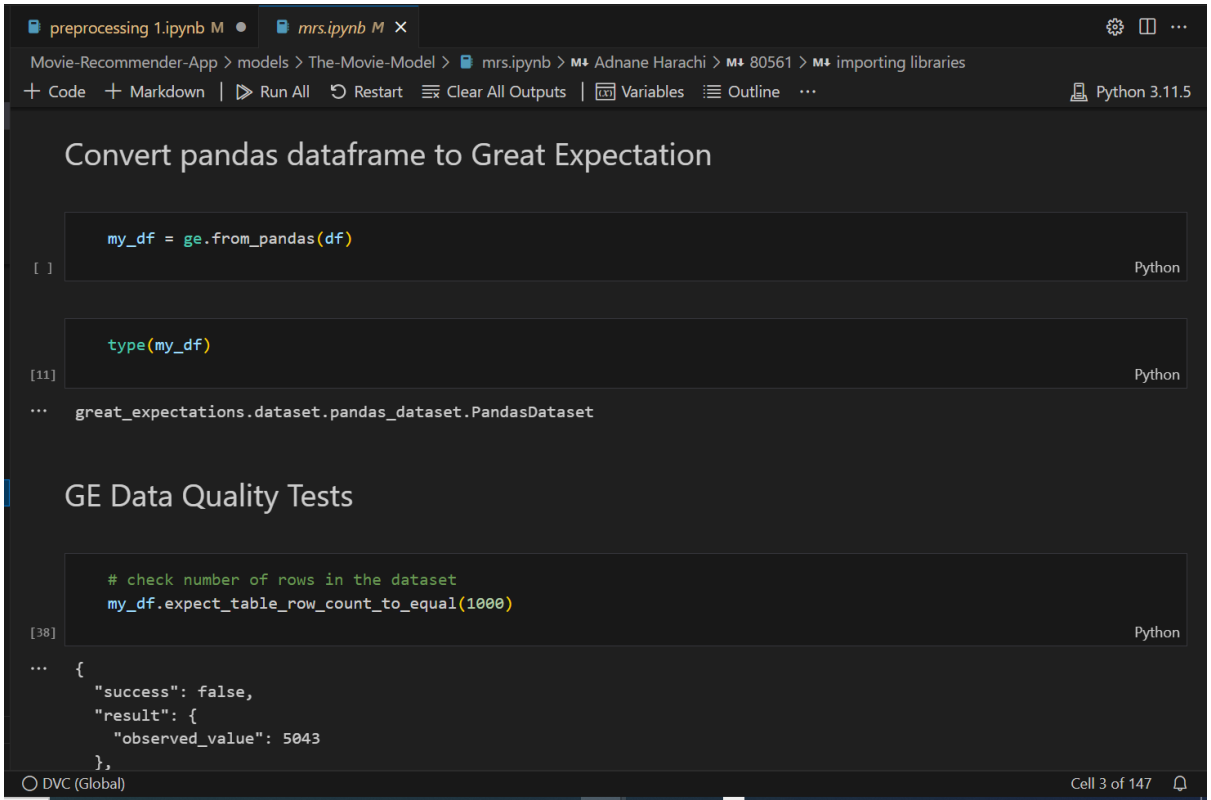
Historical Traceability: Keeps a record of data validation outcomes, which is essential for auditing and compliance.

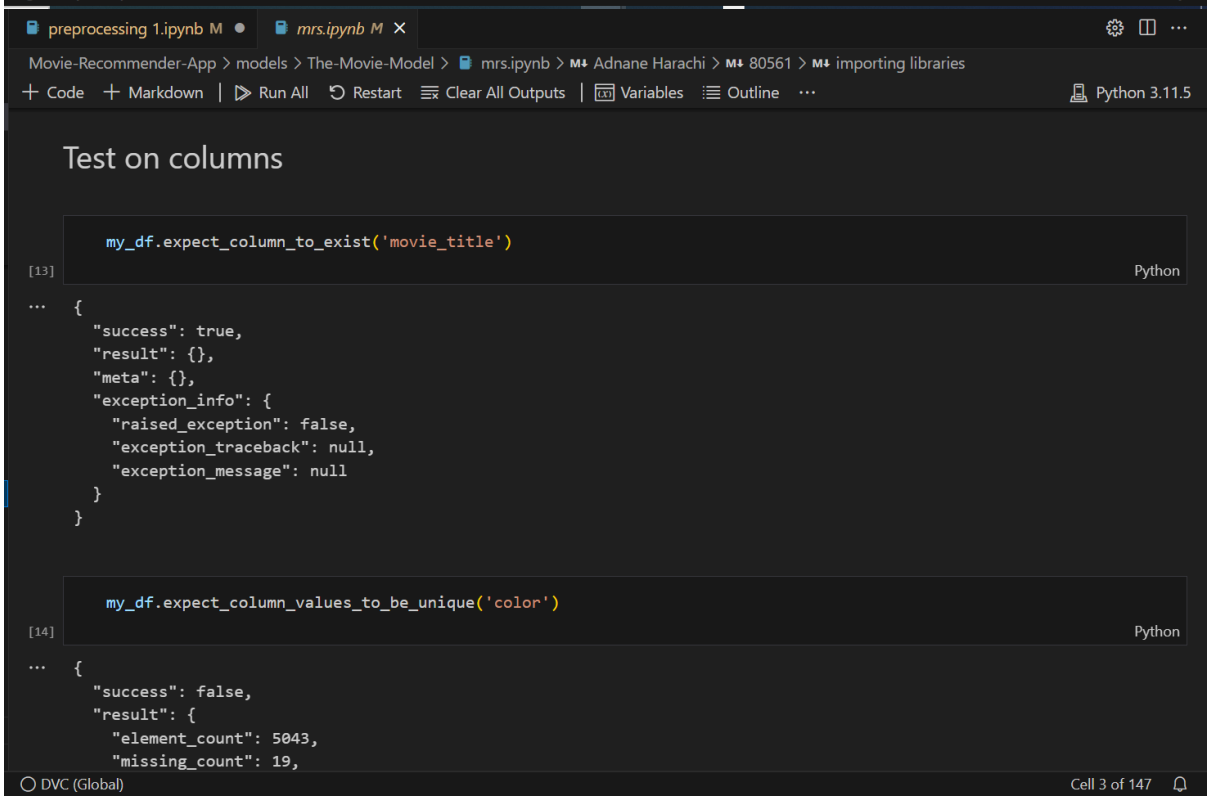
Pandas and NumPy are two important libraries in the Python environment for data processing and numerical computing, respectively. They are critical in preprocessing and feature engineering, offering reliable and fast methods for dealing with big datasets and executing complicated transformations. Pandas provides strong data structures like as DataFrames, which facilitate data manipulation, cleaning, and transformation. It also has a variety of features for merging, joining, reshaping, and aggregating data. Pandas' straightforward syntax and comprehensive capability make data preparation chores simple, even for individuals with little programming knowledge.

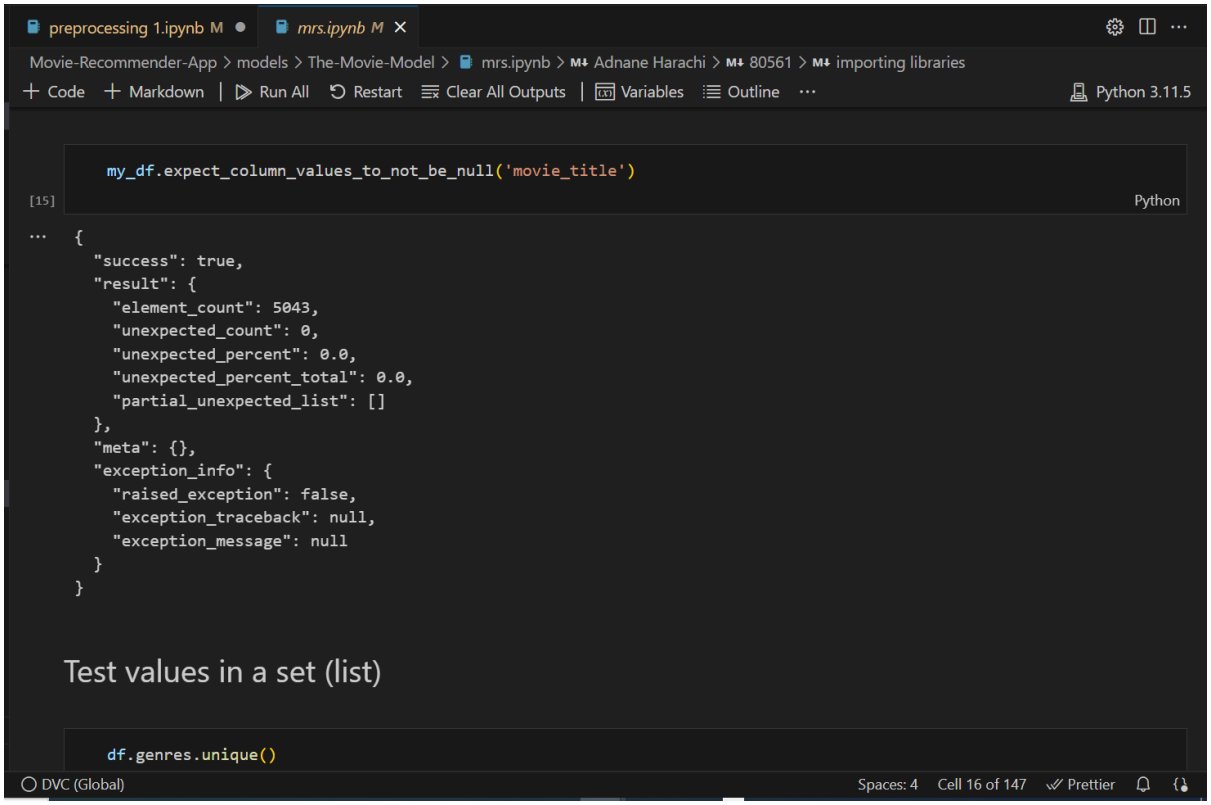
NumPy, on the other hand, supports massive, multidimensional arrays and matrices, as well as a set of mathematical algorithms for rapidly manipulating these arrays. NumPy operations are highly optimized, and they can do large-scale numerical computations significantly quicker than standard Python. It also integrates smoothly with other scientific libraries in Python, making it an indispensable component of the data science toolset.

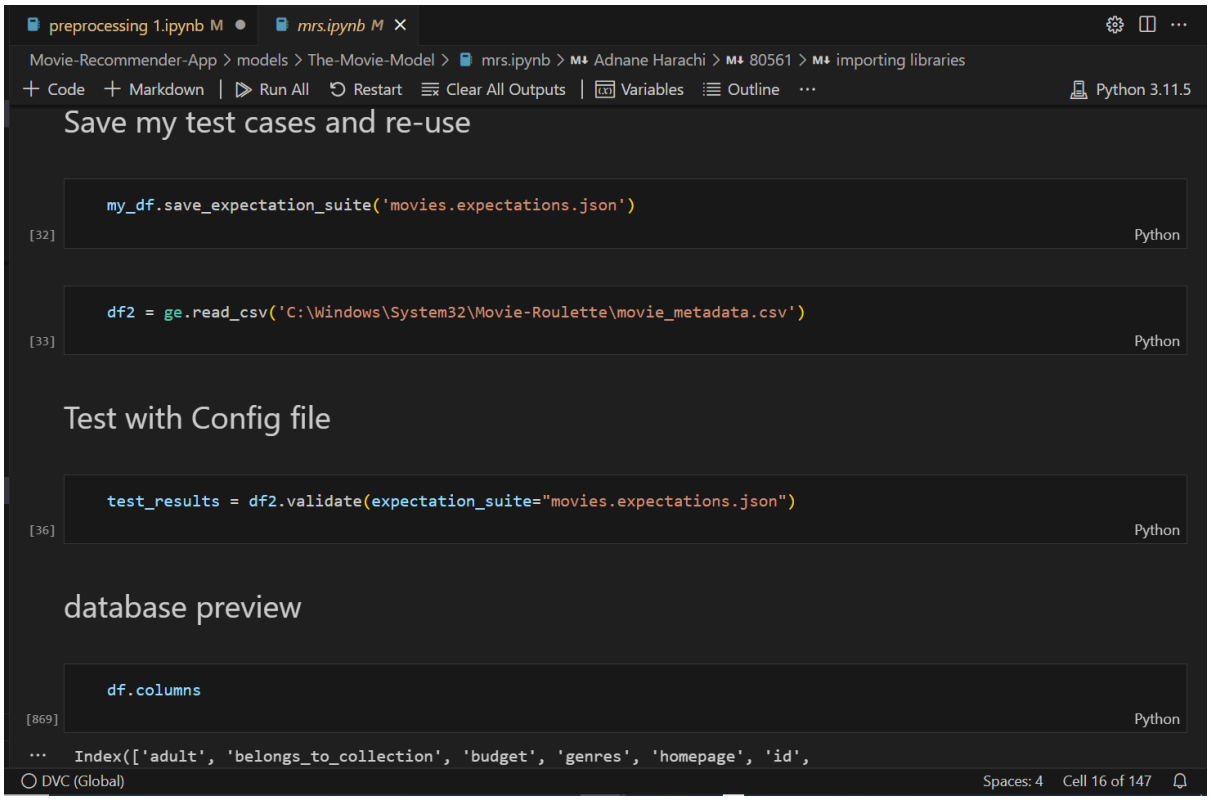
Pandas is used to manage missing values, duplicate records, and data discrepancies. It enables data type conversions and normalization. Pandas and NumPy are used to extract new features from existing data, such as categorical variable encoding, numerical feature scaling, and interaction term generation.



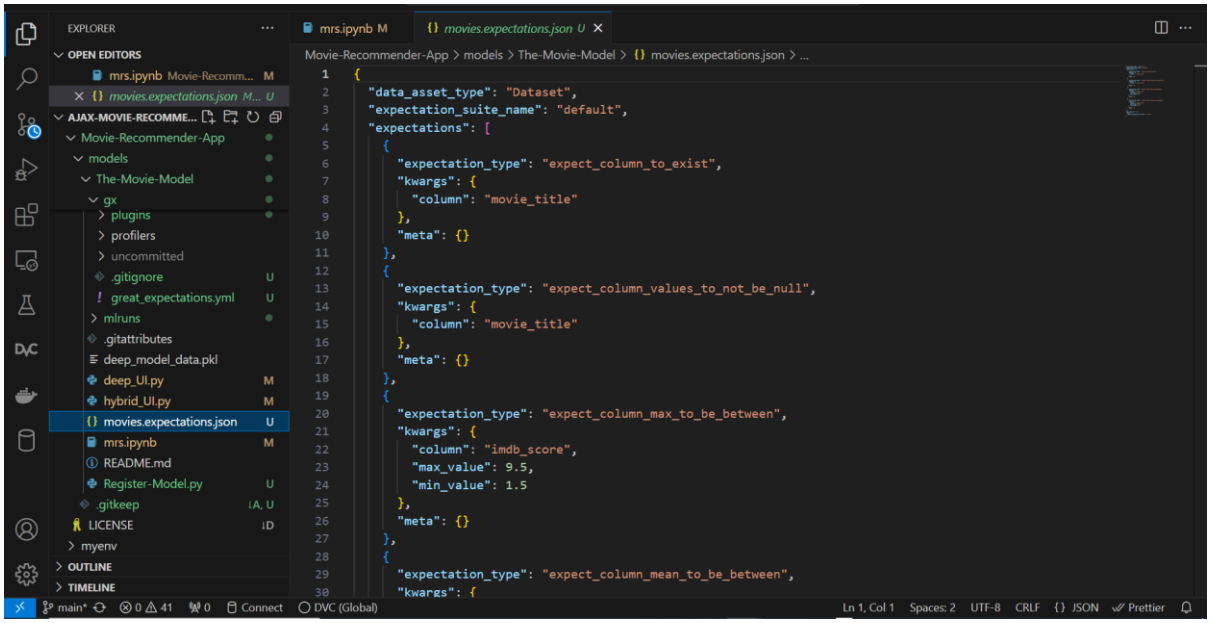

preprocessing 1.ipynb M
mrs.ipynb M
Movie-Recommender-App > models > The-Movie-Model > mrs.ipynb > Adnane Harachi > 80561 > importing libraries
Code
Markdown
Run All
Restart
Clear All Outputs
Variables
Outline
Python 3.11.5
Convert pandas dataframe to Great Expectation
my_df = ge.from_pandas(df)
[ ]
Python
type(my_df)
[11]
Python
great_expectations.dataset.pandas_dataset.PandasDataset
GE Data Quality Tests
# check number of rows in the dataset
my_df.expect_table_row_count_to_equal(1000)
[38]
Python
{
"success": false,
"result": {
"observed_value": 5043
},
DVC (Global)
Cell 3 of 147

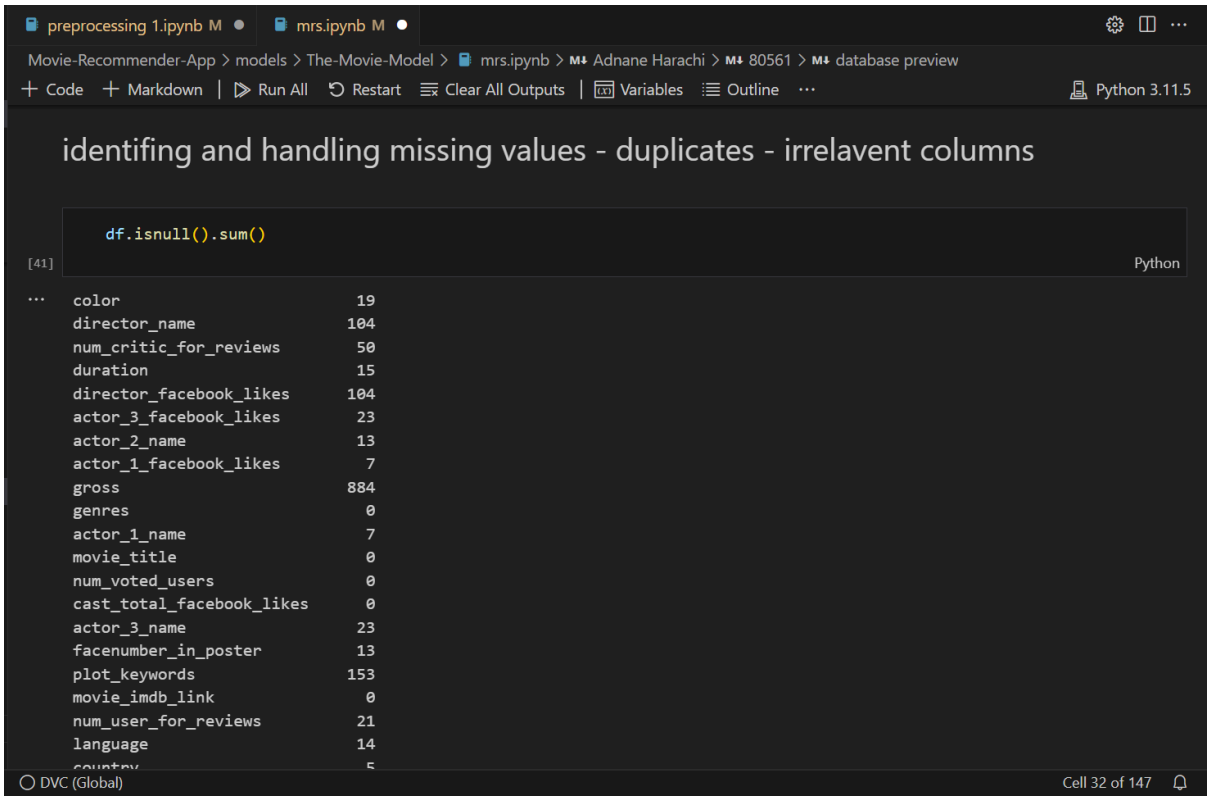

preprocessing 1.ipynb M
mrs.ipynb M
Movie-Recommender-App > models > The-Movie-Model > mrs.ipynb > Adnane Harachi > 80561 > importing libraries
Code
Markdown
Run All
Restart
Clear All Outputs
Variables
Outline
Python 3.11.5
Test on columns
my_df.expect_column_to_exist('movie_title')
[13]
Python
{
"success": true,
"result": {},
"meta": {},
"exception_info": {
"raised_exception": false,
"exception_traceback": null,
"exception_message": null
}
}
my_df.expect_column_values_to_be_unique('color')
[14]
Python
{
"success": false,
"result": {
"element_count": 5043,
"missing_count": 19,
DVC (Global)
Cell 3 of 147

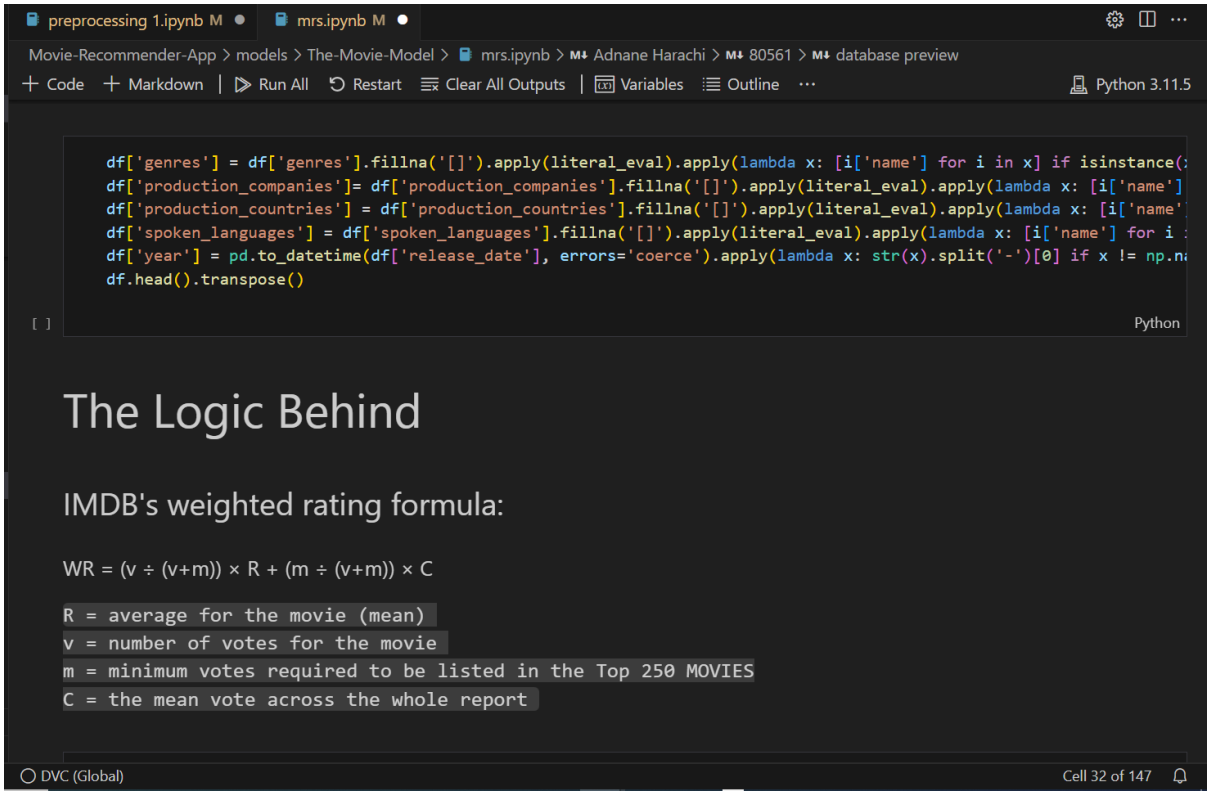


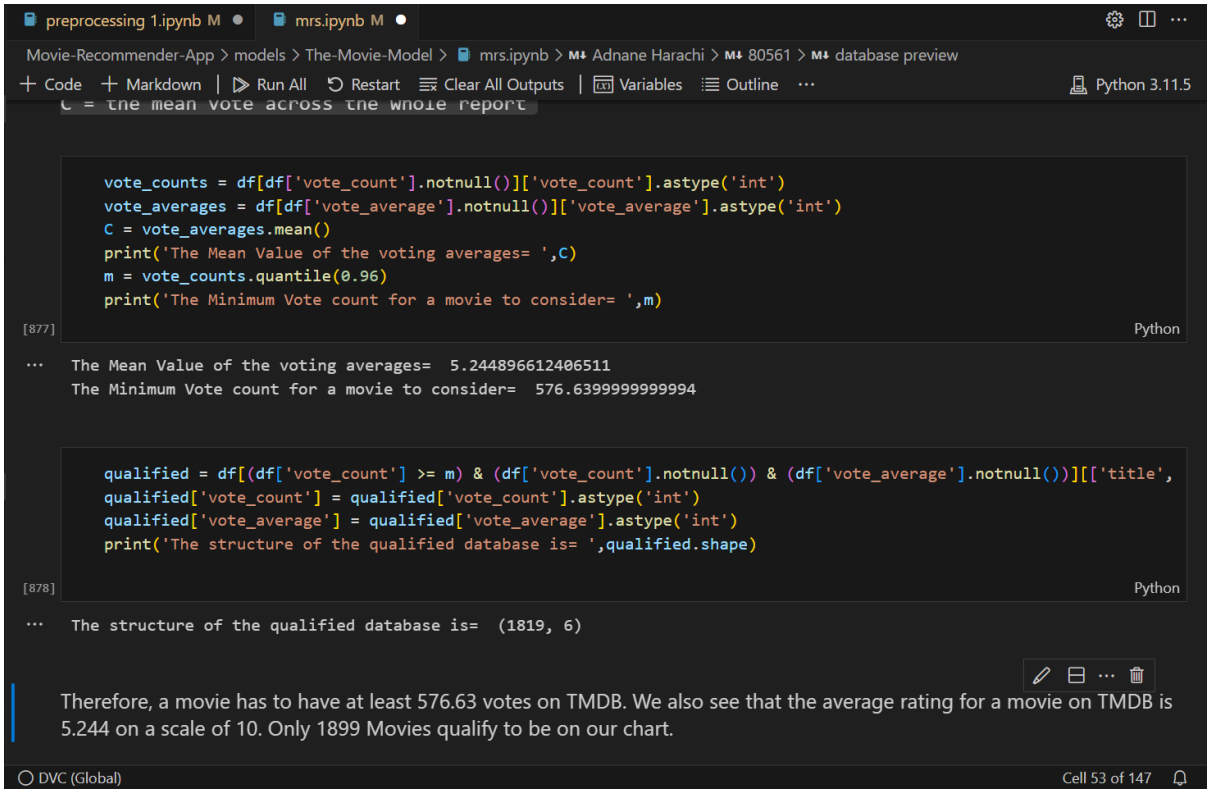


Link: https://github.com/adnanehar/Movie-Recommendation-Final/tree/main/.ipynb_checkpoints


EXPLORER
OPEN EDITORS
mrs.ipynb Movie-Recomm... M
movies.expectations.json M... U
AJAX-MOVIE-RECOMME...
Movie-Recommender-App
models
The-Movie-Model
gx
plugins
profilers
uncommitted
.gitignore
great_expectations.yml
mlruns
.gitattributes
deep_model_data.pkl
deep_UI.py
hybrid_UI.py
movies.expectations.json
mrs.ipynb
README.md
Register-Model.py
.gitkeep
LICENSE
myenv
OUTLINE
TIMELINE
mrs.ipynb M
movies.expectations.json U
Movie-Recommender-App > models > The-Movie-Model > movies.expectations.json > ...
{
"data_asset_type": "Dataset",
"expectation_suite_name": "default",
"expectations": [
{
"expectation_type": "expect_column_to_exist",
"kwargs": {
"column": "movie_title"
},
"meta": {}
},
{
"expectation_type": "expect_column_values_to_not_be_null",
"kwargs": {
"column": "movie_title"
},
"meta": {}
},
{
"expectation_type": "expect_column_max_to_be_between",
"kwargs": {
"column": "imdb_score",
"max_value": 9.5,
"min_value": 1.5
},
"meta": {}
},
{
"expectation_type": "expect_column_mean_to_be_between",
"kwargs": {
main* 0 41 0 Connect DVC (Global)
Ln 1, Col 1 Spaces: 2 UTF-8 CRLF { } JSON Prettier


preprocessing 1.ipynb M
mrs.ipynb M
Movie-Recommender-App > models > The-Movie-Model > mrs.ipynb > Adnane Harachi > 80561 > database preview
+ Code + Markdown | Run All Restart Clear All Outputs | Variables Outline ...
Python 3.11.5
identifing and handling missing values - duplicates - irrelavent columns
df.isnull().sum()
[41]
Python
color 19
director_name 104
num_critic_for_reviews 50
duration 15
director_facebook_likes 104
actor_3_facebook_likes 23
actor_2_name 13
actor_1_facebook_likes 7
gross 884
genres 0
actor_1_name 7
movie_title 0
num_voted_users 0
cast_total_facebook_likes 0
actor_3_name 23
facenumber_in_poster 13
plot_keywords 153
movie_imdb_link 0
num_user_for_reviews 21
language 14
DVC (Global)
Cell 32 of 147





fe

The rest is in the notebook provided with the code to not make the report very long.

2.2.2 (5 pts) Integration of model training and offline evaluation into the ML pipeline / MLOps Platform

Model training and offline assessment are successfully integrated into the ML pipeline utilizing ZenML's model training and evaluation components. ZenML manages the whole training process, ensuring that models are trained effectively and that performance is measured using complete offline metrics. One of the primary benefits of utilizing ZenML for this purpose is its strong experiment tracking capabilities. ZenML painstakingly monitors model training runs and evaluation results, providing precise information about the performance of various model setups. This tracking is extremely useful for understanding how various model parameters and configurations affect overall performance, allowing data scientists to make educated judgments regarding model tweaks and enhancements. Furthermore, ZenML's scalability is a valuable advantage, allowing the model training and assessment procedures to be scaled up as necessary. This scalability enables large-scale experimentation, which is required for the development and refinement of high-quality machine learning models. By allowing for prolonged experimentation, ZenML assures that the ML pipeline can manage larger volumes of data and more complicated model architectures, resulting in more accurate and efficient models. Overall, using ZenML to integrate model training and offline assessment into the ML pipeline improves the machine learning workflow's resilience and efficacy.

2.2.3 Development of model behavioral tests

Pytest is used for behavioral testing, which ensures the data pipeline's reliability and correctness.

Data Tests: These check the ability to read labels and tweets from files, assuring smooth functioning.

ML Infrastructure Testing: External dependencies, such as Cassandra database connections, are simulated to ensure that the data pipeline integrates seamlessly.

Unit Tests: Critical components such as preprocessing, assessment, and model inference are thoroughly tested to ensure correctness.

Monitoring Tests: Continuous evaluations are performed to detect data drifts in production, preventing possible model deterioration."

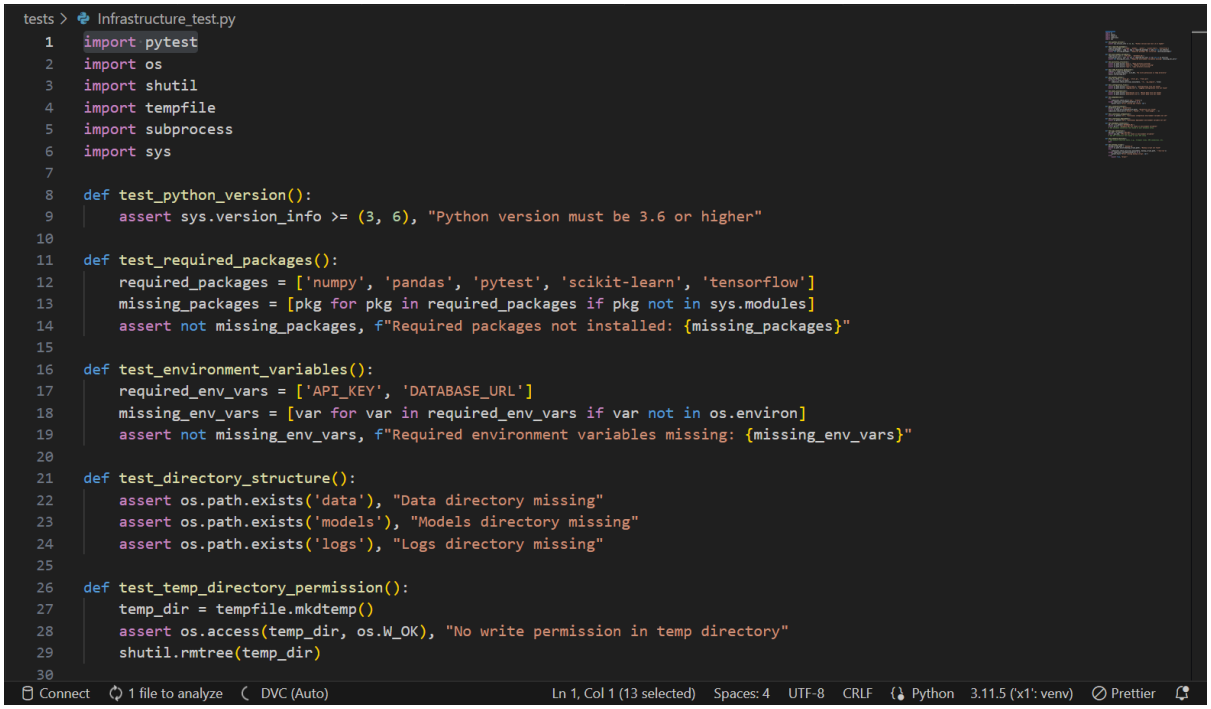
```python
import pandas as pd
import pytest

def test_data(csv_file):
    # Load the CSV file into a DataFrame
    df = pd.read_csv(csv_file)

    # Check for missing values
    missing_values = df.isnull().sum()
    if missing_values.any():
        print("Missing values found:")
        print(missing_values)
    else:
        print("No missing values found.")

    # Check for duplicate rows
    duplicate_rows = df.duplicated().sum()
    if duplicate_rows:
        print("Duplicate rows found:", duplicate_rows)
    else:
        print("No duplicate rows found.")

    # Check data types
    print("Data types:")
    print(df.dtypes)

    # Check for unique values in categorical columns
    categorical_columns = df.select_dtypes(include=['object']).columns
    for column in categorical_columns:
        unique_values = df[column].unique()
```

tests > Unit_Integration_Testing.py

```python
from matplotlib import _preprocess_data
import pandas as pd
import pytest
import logging
from unittest.mock import MagicMock, patch

from steps.clean_data import clean_data



sample_csv_path = r'C:\Users\dell\cleaned_data.csv'

@pytest.fixture
def test_successful_data_load():
    # Arrange
    test_csv_path = r'C:\Users\dell\Documents\cleaned_data.csv'

    # Act
    result = clean_data(test_csv_path)

    # Assert
    assert isinstance(result, pd.DataFrame)

def test_file_not_found():
    # Arrange
    non_existent_file_path = r'C:\Users\dell\Documents\cleaned_data.csv'

    # Act and Assert
    with pytest.raises(FileNotFoundError):
        clean_data(non_existent_file_path)
```

Connect | 1 file to analyze | DVC (Auto) | Ln 26, Col 56 | Spaces: 4 | UTF-8 | CRLF | Python | 3.11.5 ('x1': venv) | Prettier

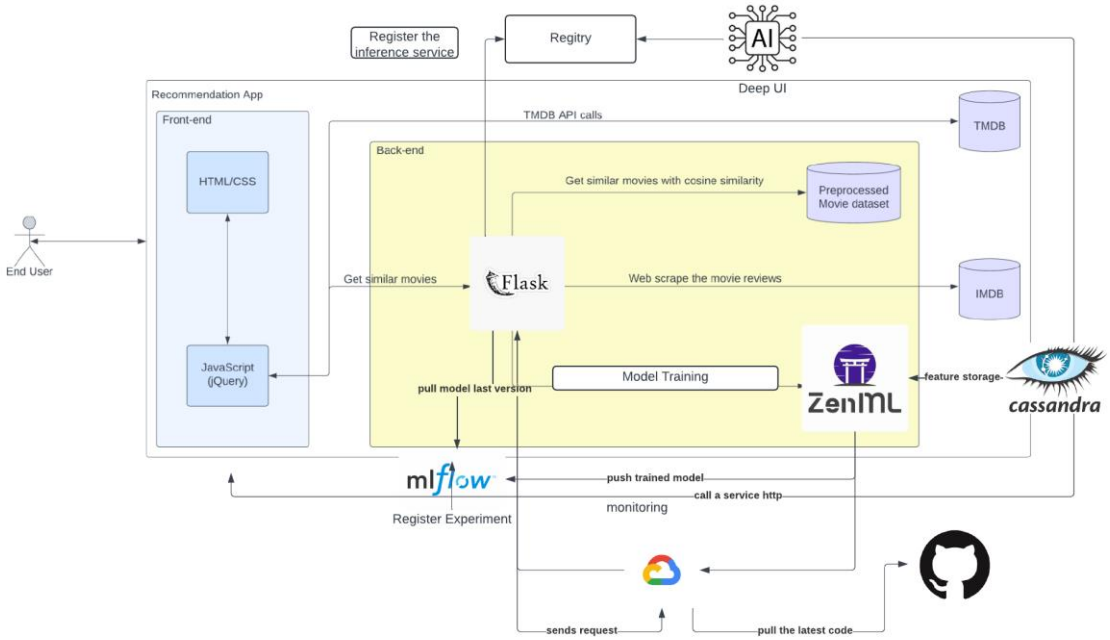
Link: https://github.com/adnanehar/Movie-Recommendation-Final/tree/main/tests



3. Model Deployment

3.1 ML System Architecture

- (5 pts) Drawing with architecture highlights

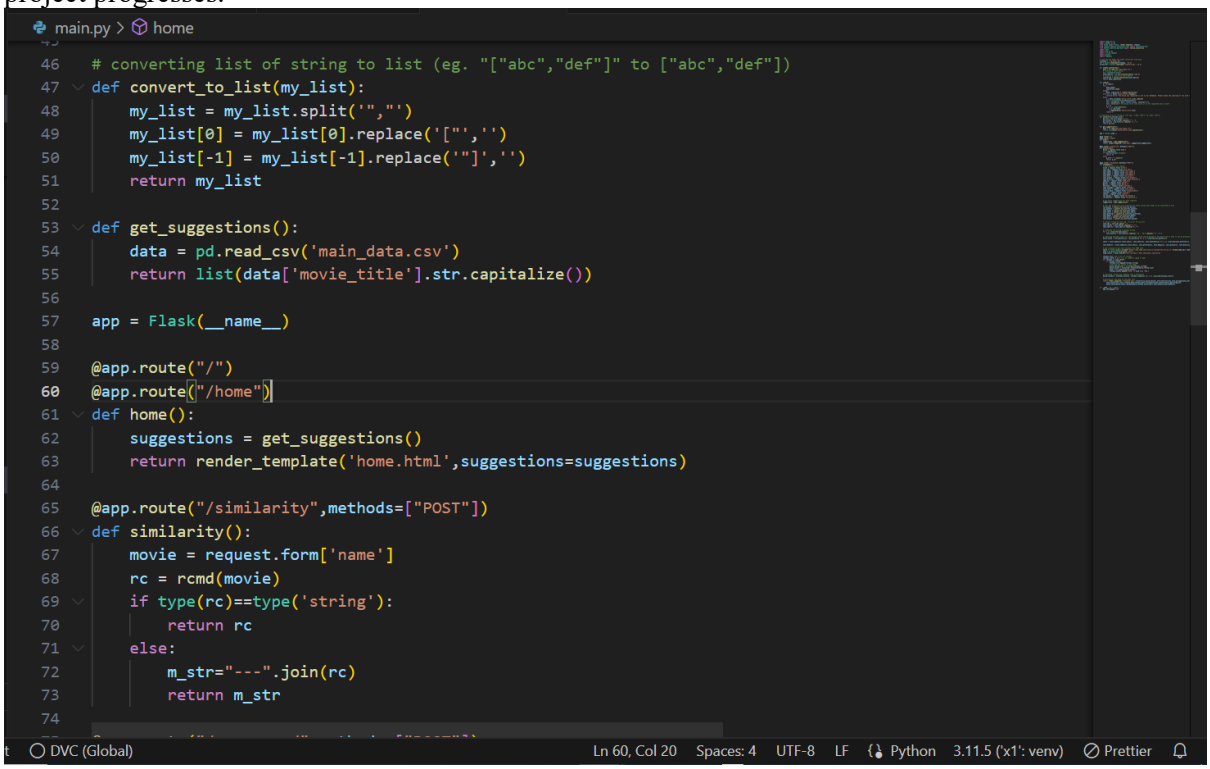


3.2 Application development

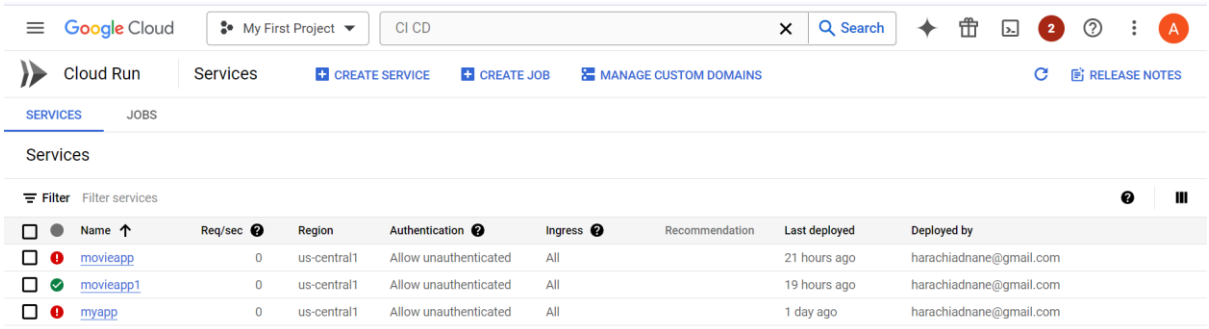
- (5 pts) Model service development

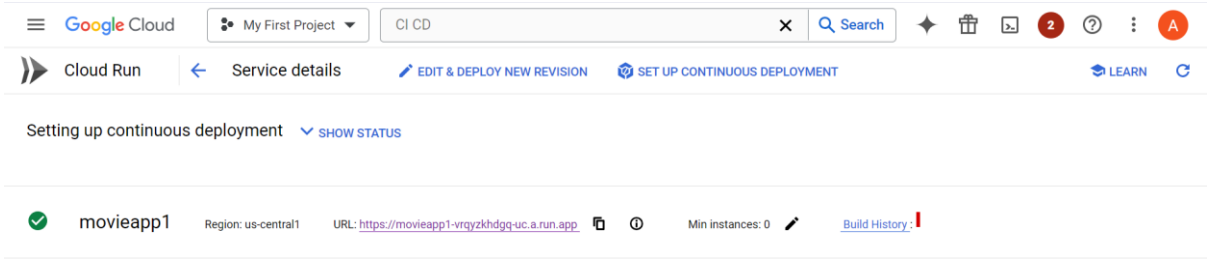
Flask is the core framework for building the model service, offering a fast and lightweight method for delivering machine learning models. Its simplicity and adaptability make it an ideal candidate for creating the backend infrastructure that handles model inference requests. Using Flask's lightweight nature, developers may quickly create microservices suited to certain model functionality. In addition, I used Docker containers and Docker images to package the program. I then deployed it using Google Cloud Platform. Flask serves as the foundation for building the model service, offering a quick and resource-efficient way to deploy machine learning models. Its inherent simplicity and versatility make it an ideal candidate for developing the backend infrastructure that handles model inference requests. With Flask's lightweight architecture, developers can quickly construct microservices suited to certain model functionality. This simplified method not only speeds up development, but also improves maintainability and extensibility as the system grows."

project progresses.



```python
# converting list of string to list (eg. "["abc","def"]" to ["abc","def"])
def convert_to_list(my_list):
    my_list = my_list.split('","')
    my_list[0] = my_list[0].replace('["','')
    my_list[-1] = my_list[-1].replace('"]','')
    return my_list

def get_suggestions():
    data = pd.read_csv('main_data.csv')
    return list(data['movie_title'].str.capitalize())

app = Flask(__name__)

@app.route("/")
@app.route("/home")
def home():
    suggestions = get_suggestions()
    return render_template('home.html',suggestions=suggestions)

@app.route("/similarity",methods=["POST"])
def similarity():
    movie = request.form['name']
    rc = rcmd(movie)
    if type(rc)==type('string'):
        return rc
    else:
        m_str="---".join(rc)
        return m_str
```



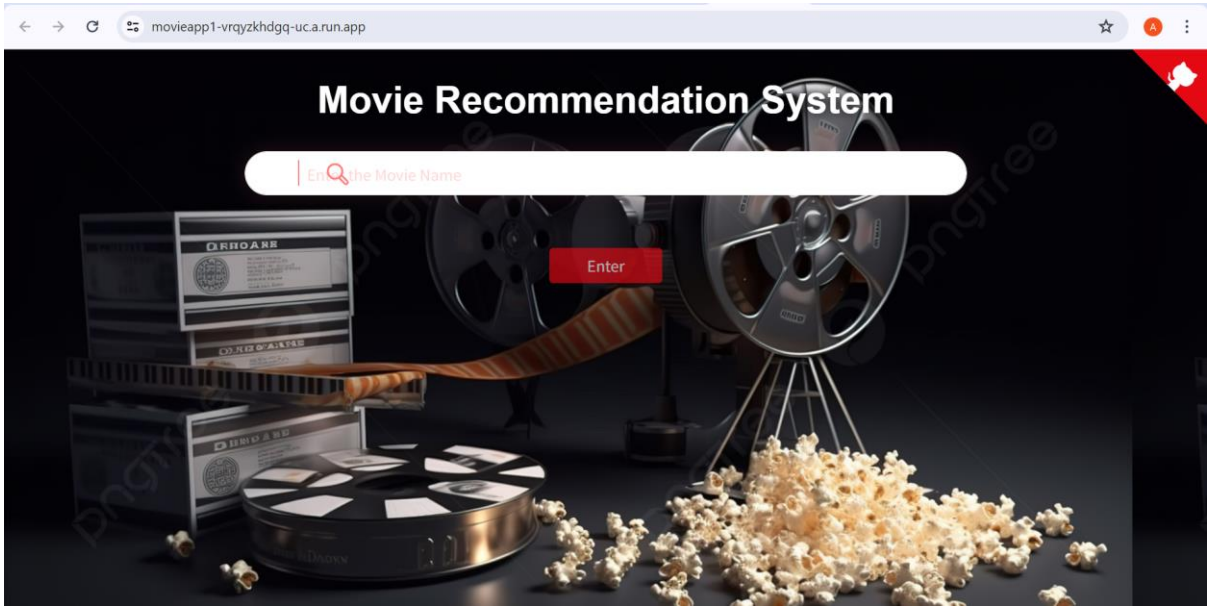


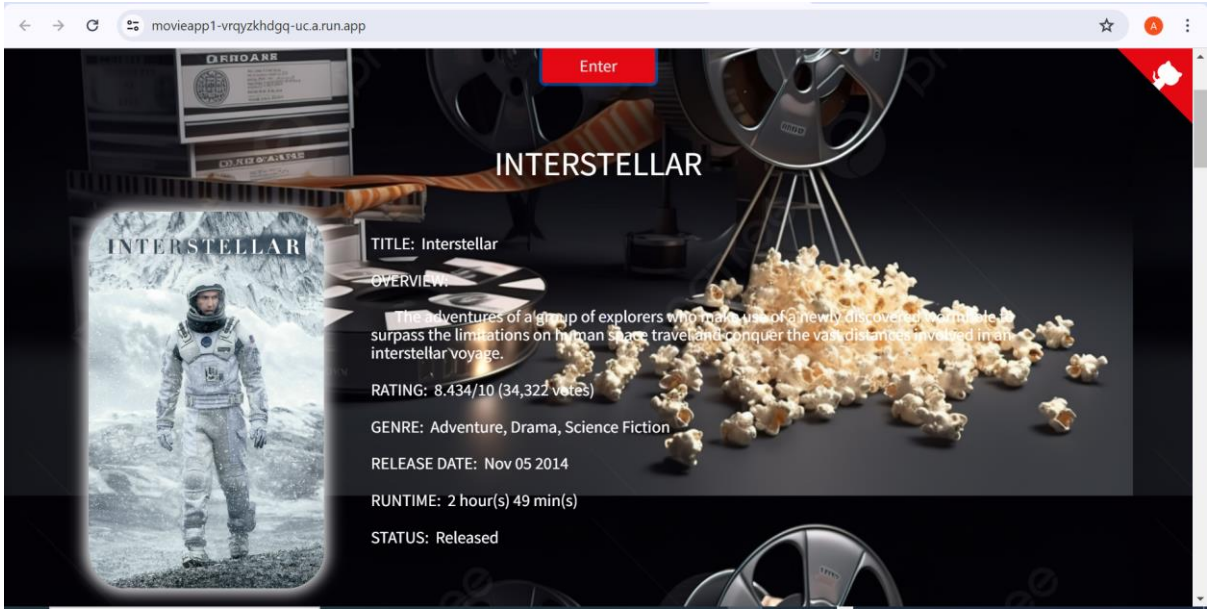
- **(5 pts) Front-end client development**

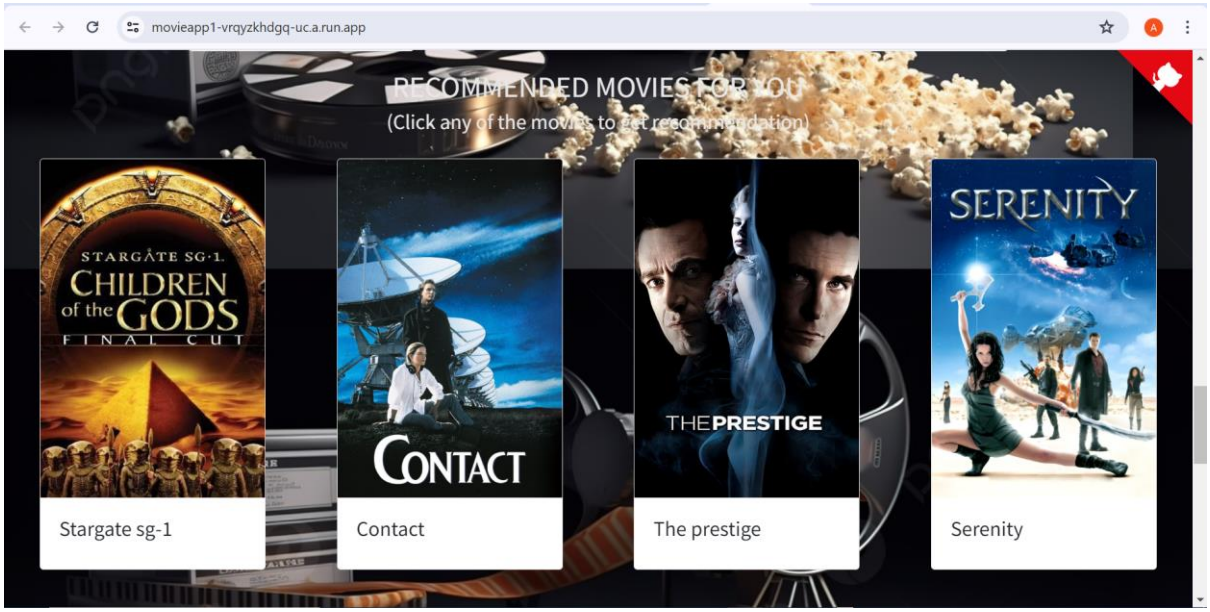
Flask is an important tool for front-end client development since it allows you to create dynamic and adaptable web interfaces. Using Flask's features, developers can quickly design and build interactive

user interfaces, improving the overall user experience. Furthermore, REST requests improve communication between the front-end and backend, allowing for smooth integration and data sharing. This strategy has various advantages. For starters, separating front-end and backend components makes development and maintenance easier, allowing teams to collaborate freely and quickly. Furthermore, RESTful APIs encourage flexibility and scalability, allowing for smooth integration and development of several components.

Developers may create visually appealing and user-friendly interfaces using the front-end client's HTTP requests and CSS style, increasing user engagement and satisfaction. In summary, using Flask for front-end development in conjunction with RESTful APIs produces a versatile, scalable, and responsive web interface that caters to both developers and users. Furthermore, when installed on Google Cloud Platform, a website address is supplied post-deployment, allowing quick access to the deployed application.







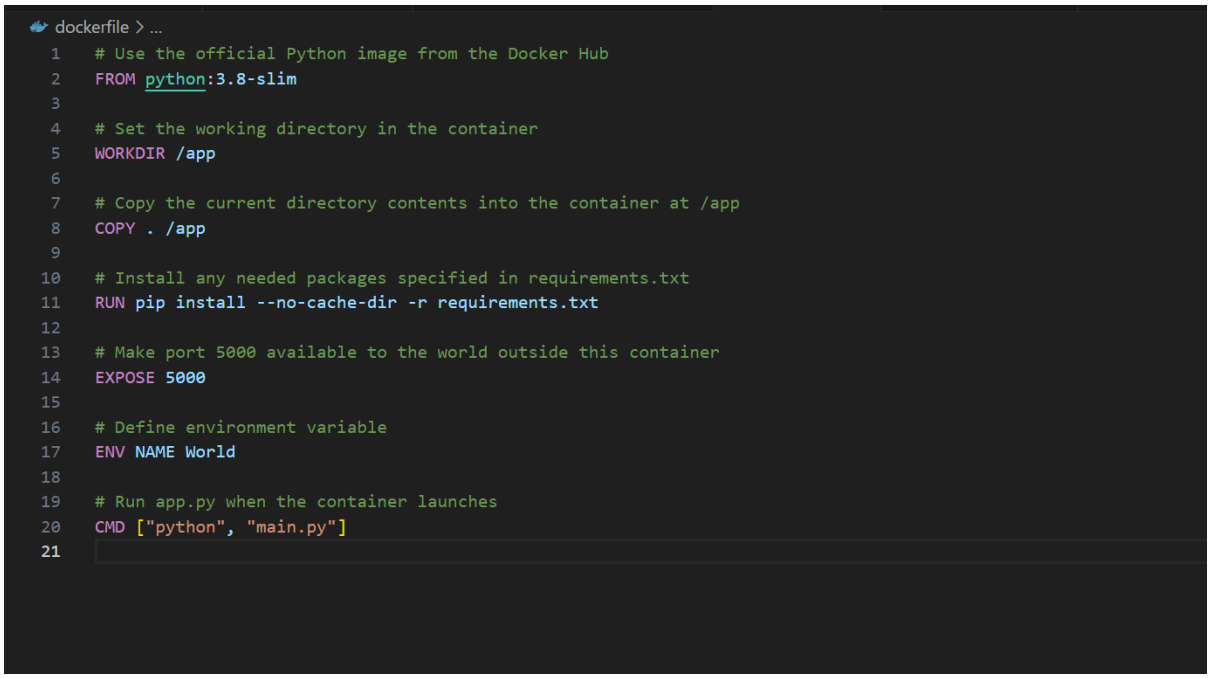
To access the application: https://movieapp1-vrqyzkhdgq-uc.a.run.app/

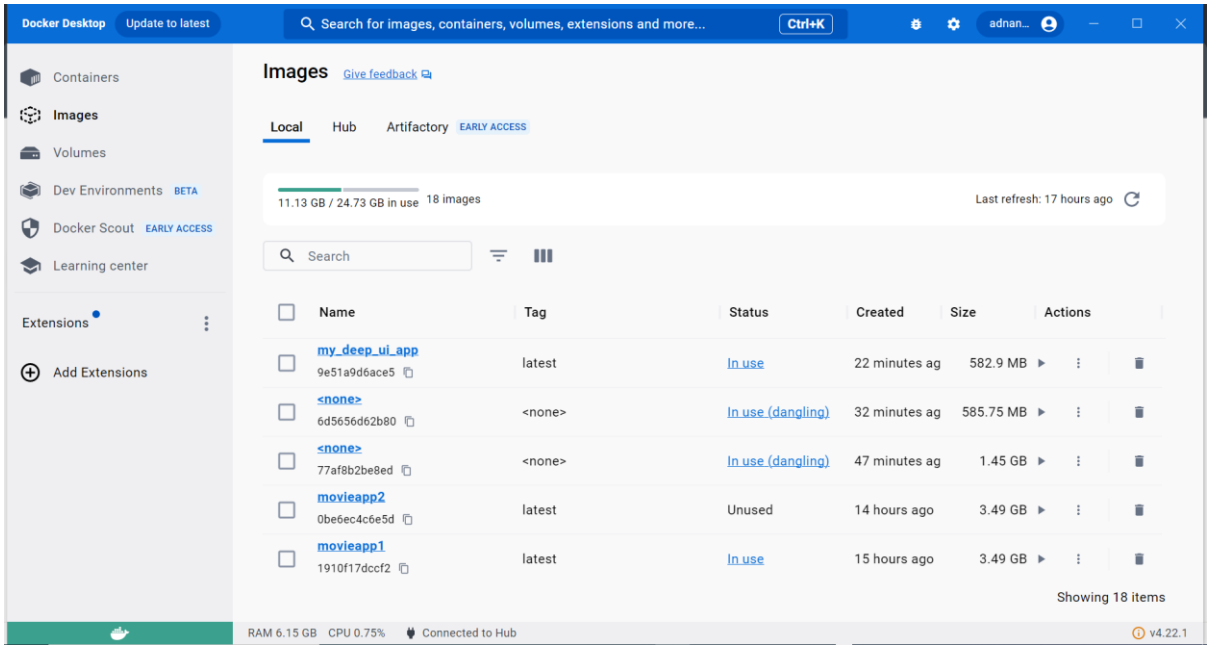
3.3 Integration and Deployment

- Packaging and containerization

Docker is a valuable tool for packaging and containerization since it encapsulates a program and its dependencies in portable containers. These containers create a standardized environment that ensures consistency between the development and production environments. Using Docker's containerization features, the packed application is deployed as a Docker container, allowing for smooth deployment and portability across several platforms. Furthermore, Docker Hub acts as a centralized repository for

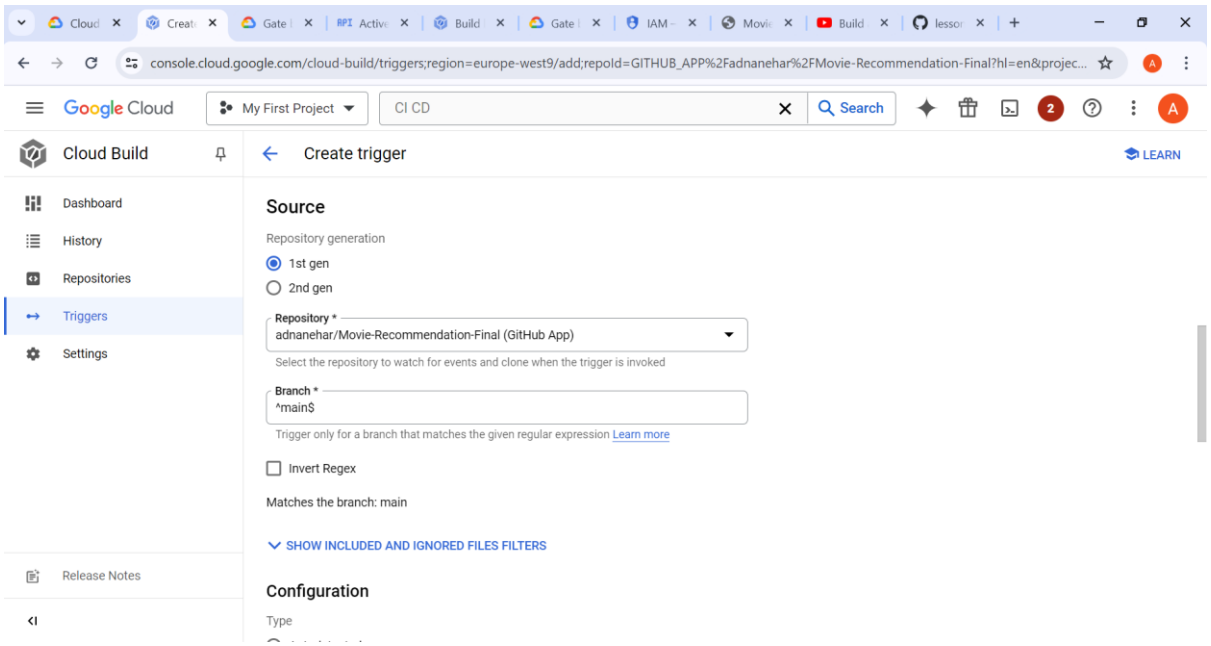
storing and maintaining Docker images, simplifying the distribution and deployment process. This strategy has several advantages. First, Docker containers are portable, allowing them to run on any platform that supports Docker. This ensures consistent behavior across environments, simplifies the deployment process and reduces compatibility difficulties. Containers also enable isolation, effectively separating processes and resources to avoid conflicts between applications and dependencies. This separation improves security and stability, lowering the chance of system failure and providing consistent performance in a variety of computing scenarios. Overall, Docker's containerization and packaging features, together with Docker Hub's repository management, enable rapid and consistent application deployment while encouraging portability and isolation for increased dependability and scalability.

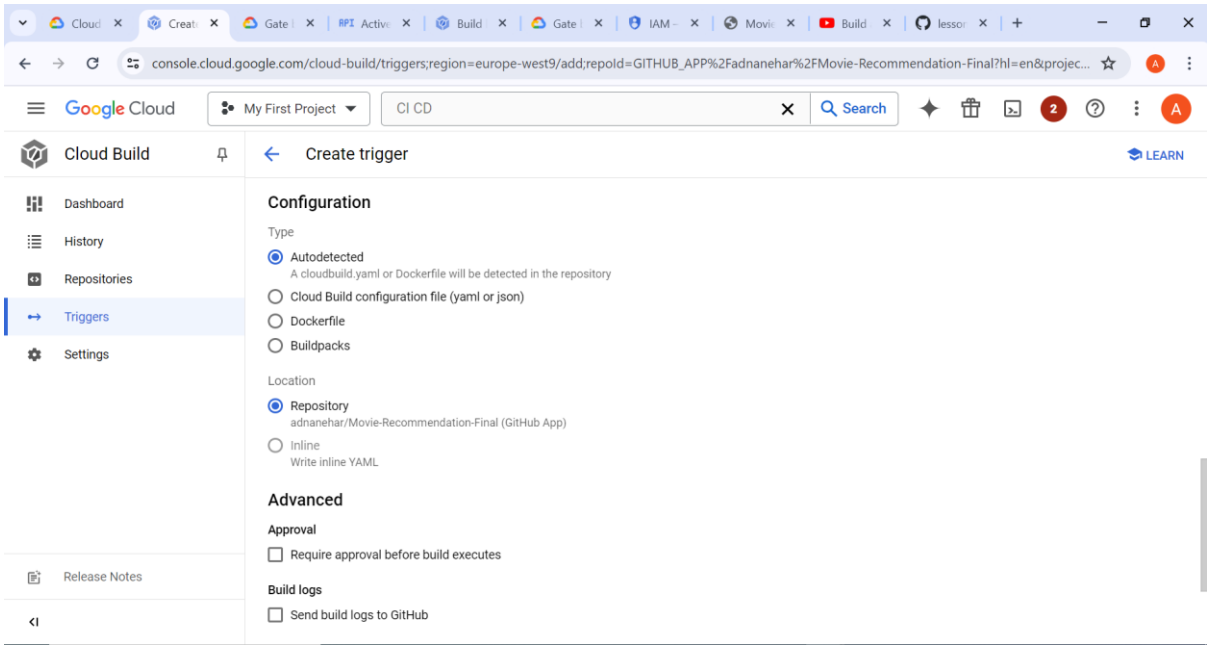


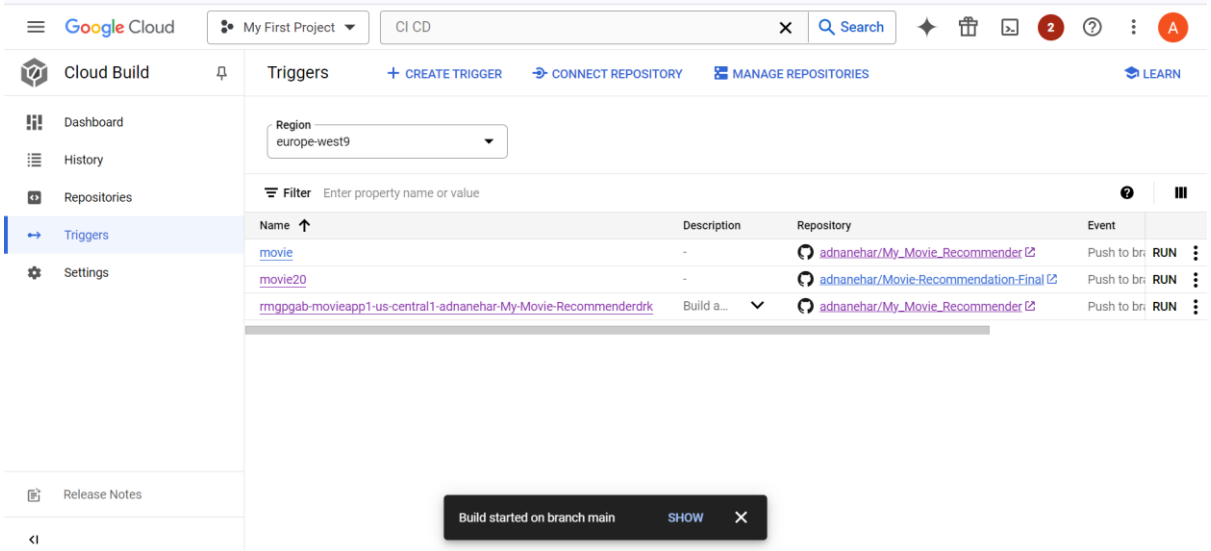


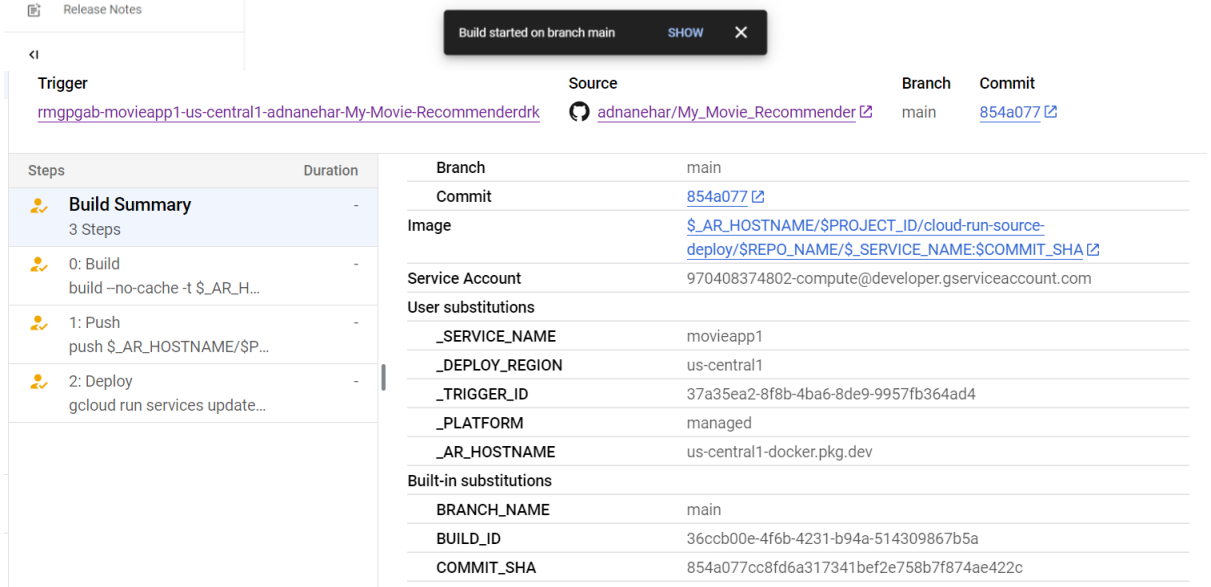
- **Integration with a CI/CD Pipeline**

With Google Cloud Run, users can seamlessly integrate their GitHub repositories with the deployed container. This configuration ensures that whenever new updates are pushed to the repository, the deployed container automatically redeploys with the latest version.



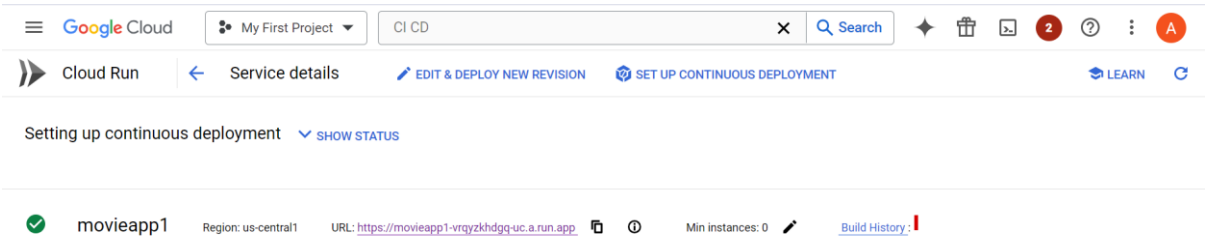

console.cloud.google.com/cloud-build/triggers;region=europe-west9/add;repoId=GITHUB_APP%2Fadnanehar%2FMovie-Recommendation-Final?hl=en&projec...
Google Cloud
My First Project
CI CD
Search
Cloud Build
Create trigger
LEARN
Dashboard
History
Repositories
Triggers
Settings
Release Notes
Configuration
Type
Autodetected
A cloudbuild.yaml or Dockerfile will be detected in the repository
Cloud Build configuration file (yaml or json)
Dockerfile
Buildpacks
Location
Repository
adnanehar/Movie-Recommendation-Final (GitHub App)
Inline
Write inline YAML
Advanced
Approval
Require approval before build executes
Build logs
Send build logs to GitHub


Google Cloud
My First Project
CI CD
Search
Cloud Build
Triggers
+ CREATE TRIGGER
CONNECT REPOSITORY
MANAGE REPOSITORIES
LEARN
Dashboard
History
Repositories
Triggers
Settings
Release Notes
Region
europe-west9
Filter
Enter property name or value
Name
Description
Repository
Event
movie
adnanehar/My_Movie_Recommender
Push to br
RUN
movie20
adnanehar/Movie-Recommendation-Final
Push to br
RUN
rmgpgab-movieapp1-us-central1-adnanehar-My-Movie-Recommenderdrk
Build a...
adnanehar/My_Movie_Recommender
Push to br
RUN


Build started on branch main
SHOW
Trigger
rmgpgab-movieapp1-us-central1-adnanehar-My-Movie-Recommenderdrk
Source
adnanehar/My_Movie_Recommender
Branch
main
Commit
854a077
Steps
Duration
Build Summary
3 Steps
0: Build
build --no-cache -t $_AR_H...
1: Push
push $_AR_HOSTNAME/$P...
2: Deploy
gcloud run services update...
Branch
main
Commit
854a077
Image
$_AR_HOSTNAME/$PROJECT_ID/cloud-run-source-deploy/$REPO_NAME/$_SERVICE_NAME:$COMMIT_SHA
Service Account
970408374802-compute@developer.gserviceaccount.com
User substitutions
_SERVICE_NAME
movieapp1
_DEPLOY_REGION
us-central1
_TRIGGER_ID
37a35ea2-8f8b-4ba6-8de9-9957fb364ad4
_PLATFORM
managed
_AR_HOSTNAME
us-central1-docker.pkg.dev
Built-in substitutions
BRANCH_NAME
main
BUILD_ID
36ccb00e-4f6b-4231-b94a-514309867b5a
COMMIT_SHA
854a077cc8fd6a317341bef2e758b7f874ae422c

- **Hosting the application**

The application is hosted on Google Cloud Run



3.4. Model Serving and online testing

- Model serving runtime

The model is served through google cloud with exposing the endpoints using flask.

- Serving mode (batch, on demand to a human, on demand to a machine)

Serving Mode: On Demand to a Human

The key interaction in Movie Roulette is for users to search for movies based on their tastes and then receive personalized suggestions. Given this user behavior, offering predictions on demand to human users is well aligned with the application's functionality and user experience objectives. Here's why.

Real-time Interaction: Movie suggestions are a dynamic and personalized process that is significantly affected by user feedback and preferences. Serving predictions on-demand enables users to receive real-time suggestions as they engage with the application, improving their entire experience.

User interaction: Movie Roulette promotes active user interaction and discovery by providing predictions on demand. Users may search for movies depending on their current mood, tastes, or particular criteria, resulting in more meaningful interactions and longer time spent on the site.

Tailored Recommendations: Every user has different movie choices, and their search queries reflect their distinct likes and interests. Serving predictions on demand allows Movie Roulette to provide personalized suggestions in response to each user's search, assuring relevance and customisation.

Instant Feedback Loop: Serving predictions on-demand establishes an instant feedback loop between the user's search query and the recommendations made. Users get instant answers based on

their input, allowing them to swiftly adjust their search criteria or explore alternate options, increasing their overall happiness with the program.

Scalability and Efficiency: Unlike batch processing, which requires vast amounts of data to be processed at preset intervals, Movie Roulette's on-demand predictions allow it to scale dynamically in response to customer demand. This provides effective resource use and responsiveness, especially during peak demand times.

Adaptability to User purpose: Users' search queries frequently express implicit purpose, such as studying a certain genre, discovering new releases, or locating films comparable to their favorites. Serving predictions on-demand allows Movie Roulette to respond to changing user intentions in real time, offering appropriate suggestions that match user expectations.

- Online testing:

A/B testing are an important approach for comparing different model iterations in order to improve performance. Through iterative analysis of test results, the model is constantly refined, bringing it closer to set performance criteria. Google Cloud Run emerges as a critical enabler in this process, providing a streamlined approach for running A/B tests through an easy dashboard interface. This integrated feature enables teams to quickly manage and monitor various models, providing real-time insights and informed decision-making to support continuous improvement programs.

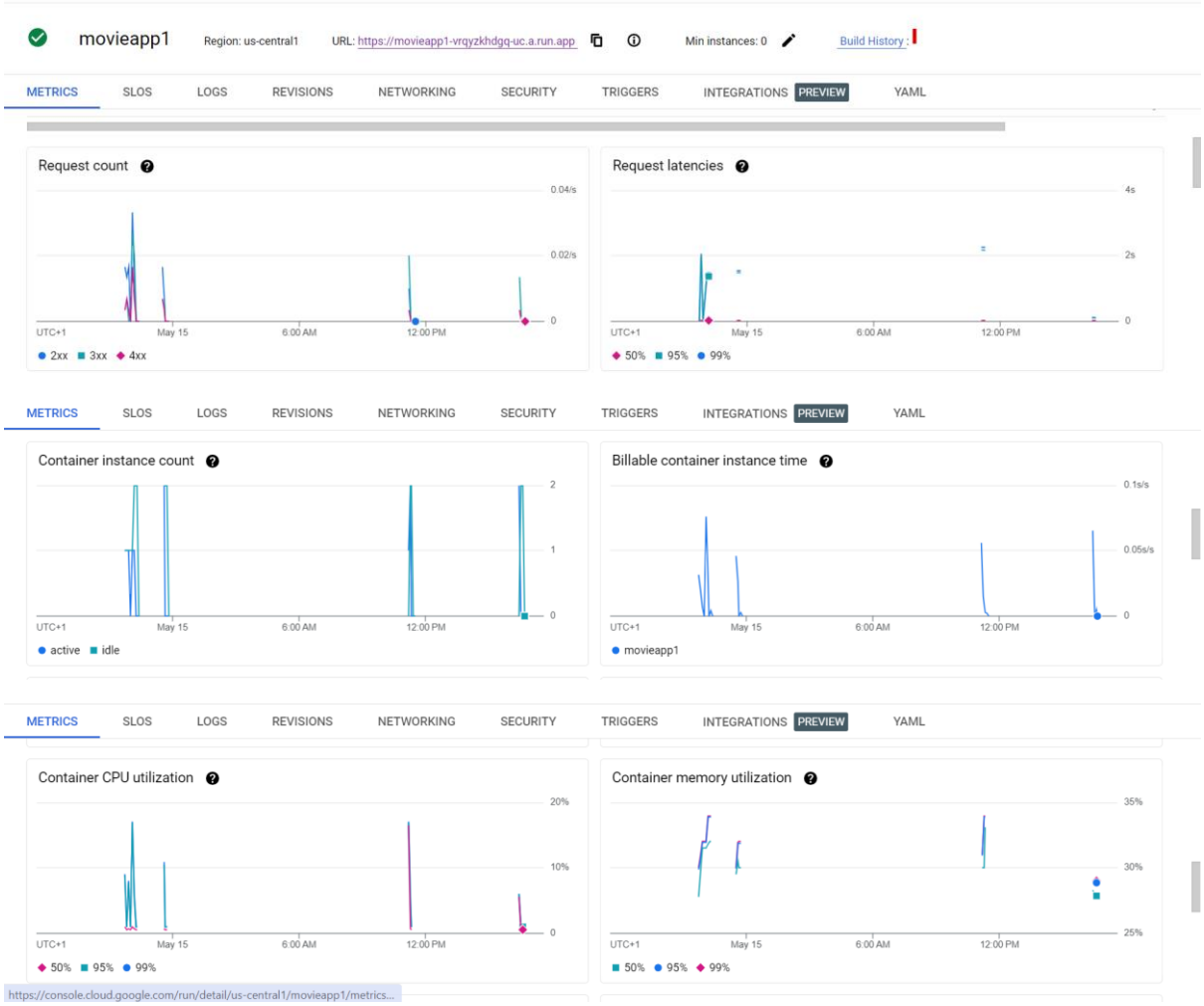
4. Monitoring and Continual Learning

Monitoring and continuous learning are key components of the Movie Roulette machine learning system, as they ensure the model's accuracy, efficiency, and adaptability to changing data and user preferences. Here, we look at the specific tools and approaches used, emphasizing their benefits and the reasoning behind their selection.

4.1. Resource Monitoring

Resource monitoring is critical to ensure the best performance and dependability of any system. Teams may maintain smooth operations by regularly monitoring critical indicators such as CPU utilization, memory consumption, and network activity. I created extensive monitoring solutions targeted to the unique requirements of the deployed application by leveraging Google Cloud's strong monitoring technologies, such as Stackdriver Monitoring and Logging. This provided real-time

visibility into the health and performance of multiple components, allowing for rapid interventions and ongoing optimization efforts to maintain service dependability and customer happiness.

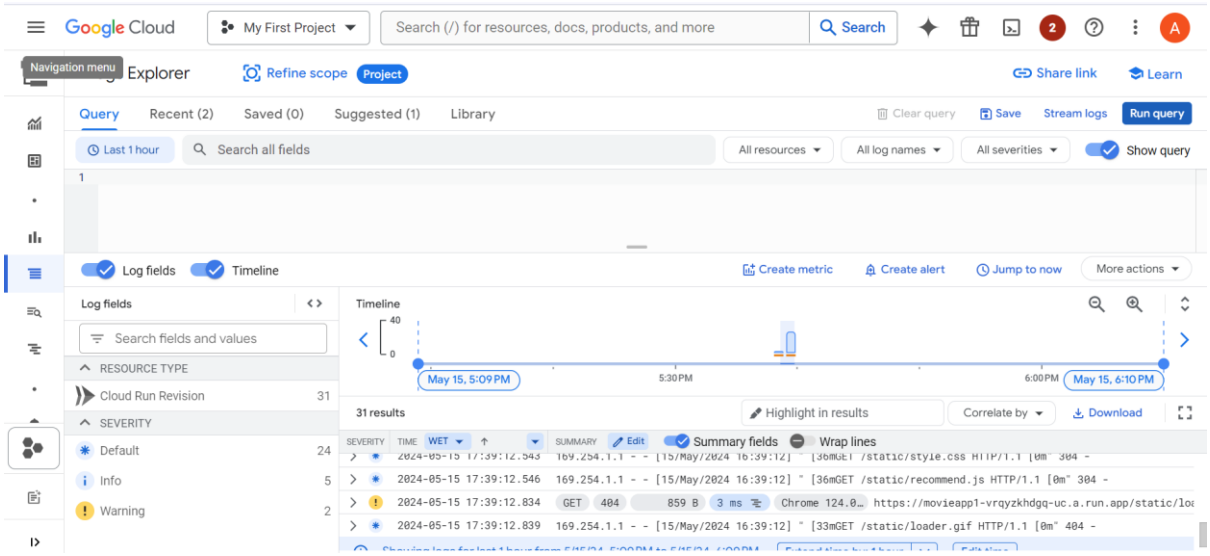


4.2. Data distribution drift monitoring

Data distribution shift refers to changes in the statistical features of input data that occur over time and have a substantial influence on machine learning model performance. Monitoring for such adjustments is critical for ensuring the correctness and dependability of deployed models.

To solve this, I used Google Cloud's Logging and Monitoring services to set up continuous monitoring of data dissemination. Using these services, I established automated methods to identify shifts in data distribution using statistical approaches such as the Kolmogorov-Smirnov (KS) test. This test analyzes the cumulative distribution functions of two datasets, allowing for the discovery of substantial variations in distributions.

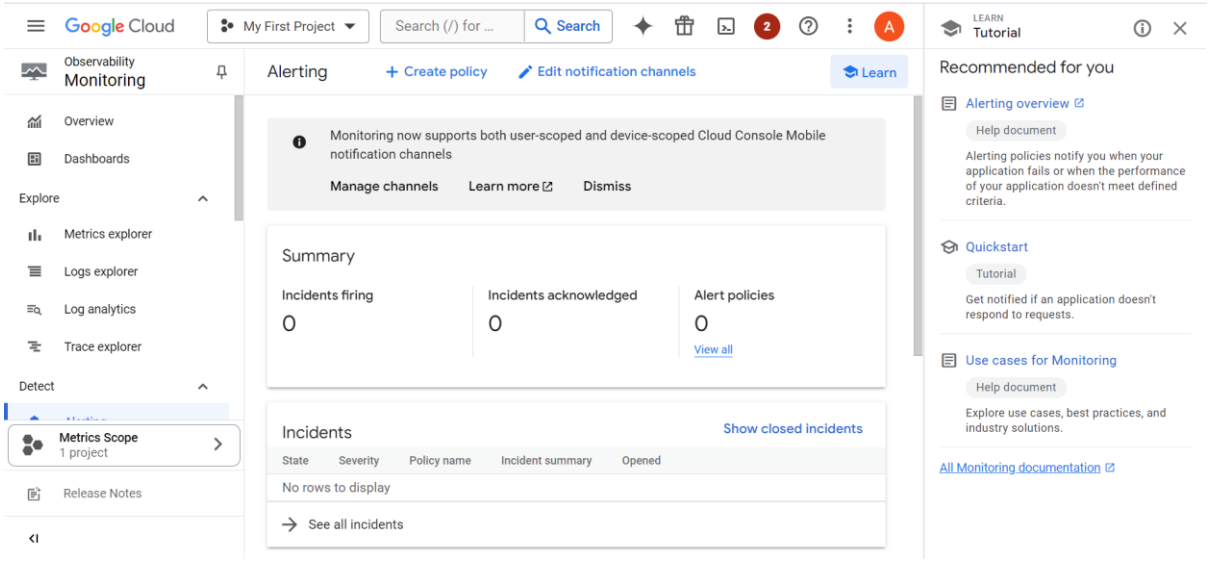
When a data distribution shift is observed, Google Cloud Logging creates extensive logs that describe the nature and scope of the move. These logs give vital insights into the changes occurring within the input data, assisting in identifying possible issues influencing model performance.



4.3. Continual Learning: CT/CD pipeline

For Continuous Training, I used Google Cloud's robust data processing and storage capabilities to rapidly ingest and preprocess enormous amounts of data. I built scalable pipelines to handle a variety of data sources and formats using data warehousing technologies such as BigQuery and Cloud Dataflow. This guaranteed that the model training datasets were constantly up to date and represented the most recent information.

In case of any data drifts, it will be displayed here:



4.4. (3 pts) Pipeline orchestration

In our project, pipeline orchestration on Google Cloud is seamlessly achieved through Cloud Composer, a fully managed workflow orchestration service built on Apache Airflow. We define our workflows as directed acyclic graphs (DAGs), encapsulating the sequence of tasks required for data processing, model training, and deployment. These DAGs are deployed to our Cloud Composer environment, where we schedule their execution at predetermined intervals or trigger them manually as needed. Monitoring and management of these workflows are facilitated through the Airflow UI, providing real-time visibility into task statuses and logs. Leveraging integrations with other Google Cloud services, such as BigQuery for data processing and AI Platform for model training, our pipeline orchestrates the entire machine learning lifecycle with efficiency and reliability.

5. Conclusion

To summarize, the creation and implementation of the Movie Roulette machine learning system has been a journey filled with important accomplishments, useful lessons learned, and promising future paths. Movie Roulette has evolved into a complex platform that provides individualized movie suggestions to consumers all over the world thanks to painstaking planning, clever use of cutting-edge technology, and an unwavering dedication to quality.

- **Summary of Achievements**

Throughout the project, I gained a varied variety of skills and expertise that dramatically improved our machine learning pipeline:

Technology Acquired:

- ZenML, a powerful MLOps framework that provides complete control over machine learning pipelines.

- Developed skills in Flask backend development, allowing for the implementation of fast and scalable model services.

- Acquired experience in Docker and DockerHub for application containerization and simplified deployment.

- Gained experience with CI/CD pipelines and google cloud.

- Designed and built a scalable ML pipeline architecture to provide dependability and flexibility.
- Tools such as Prometheus and Grafana have been integrated for real-time resource monitoring and performance improvement.

- Continuous Learning and Adaptation.

- CT/CD pipelines were implemented to allow for continuous learning, guaranteeing that the model can adapt to changing data distributions.
- Established monitoring tools for model performance and data distribution drift, allowing for proactive reactions to aberrations.

- **Lessons Learned**

- Lessons learned and project insights:

- Recognized the vital role of automation in expediting development processes, while highlighting the importance of rigorous planning and excellent mistake management.

- Recognized the problems of A/B testing, highlighting the significance of thorough testing and validation of automation scripts.

- Highlighted the importance of preserving data quality in machine learning systems, which necessitates continual monitoring and validation techniques.

- Scalability issues were addressed, highlighting the significance of designing systems that can handle increasing user demand and data volumes.

- To improve user pleasure, we prioritized user-centric design concepts, concentrating on intuitive interfaces and tailored suggestions.

- **Future Directions**

- Plan to investigate sophisticated experimental approaches such as multi-armed bandit testing to improve model performance and user engagement.

- Consider moving the application to cloud infrastructure for better scalability, dependability, and accessibility.

- Natural language processing (NLP) techniques will be integrated to analyze user reviews and comments, improve recommendation accuracy, and increase user happiness.

- Improve collaborative filtering algorithms to include user preferences and behavior, resulting in more accurate and personalized suggestions.

- Consider creating a thriving community around Movie Roulette by encouraging user participation and cooperation to promote ongoing improvement and innovation.

References