riskthinking.AI work sample (Data Engineer) Report:

Problem 1: Raw Data Processing

- I have downloaded the Dataset using the link provided: https://www.kaggle.com/datasets/jacksoncrow/stock-market-dataset and moved it to the required folder.
- Then extract_data.py extracts all folders from a zip file named 'archive.zip' using the zipfile module. The extract_data() function first uses the ZipFile function to open the zip file in read mode and assigns it to the zip_ref variable.
  Then it uses the ZipFile object's extractall method to extract all the files and folders in the zip file to the current directory (".") where the script is running.
  As a result of running this code, all the folders contained within the archive.zip file will be extracted to the current directory this is the first step in the pipeline.
- In the next step in the pipeline, raw_data_processing(), extracts the file paths of CSV files from the etfs and stocks directories and processes them in parallel using multiprocessing and the resulting data frames are merged, it is then merged with the symbols_valid_meta.csv to add the 'Symbol' and 'Security Name' columns and finally saved as CSV and parquet format in the "processed_files" directory, and the path to the CSV file is returned.

Problem 2: Feature Engineering

- On the processed stock market data, feature_engineering.py performs feature engineering. It reads processed data from a CSV file, divides it into chunks, and uses a rolling window function to calculate the rolling average of the trading volume over the last 30 days and the rolling median of the adjusted close values of stocks and ETFs. The results are saved in two new columns: 'vol_moving_avg' and 'adj_close_rolling_med'. The processed data is then saved as a parquet file. For improved performance, the code employs multiprocessing to parallelize the processing of data chunks. The transformed data will be saved in "processed_parquet_file" folder in structured parquet format.
- To ensure the structured parquet file contains the new columns vol_moving_avg and adj_close_rolling_med that are added by the function process_chunk(), I have written unit test, and this can be found in unittest_feature_engineering.py file.

Problem 3: Integrate ML Training

- As a next step in the pipeline ml_training.py file loads processed data from a parquet file, pre-processes it for machine learning, trains a GradientBoostingRegressor model on the data, saves the model to disc using joblib, calculates Mean Absolute Error and Mean Squared Error, logs the training metrics to a file it is saved in logs folder, and it uses multiprocessor to accelerate the training process. The integrate_ml_training() function first loads the processed data from a parquet file, shuffles it, converts the Date column to a datetime type and sets it as the index, removes rows with NaN values, selects features and target, splits the data into train and test sets, trains a

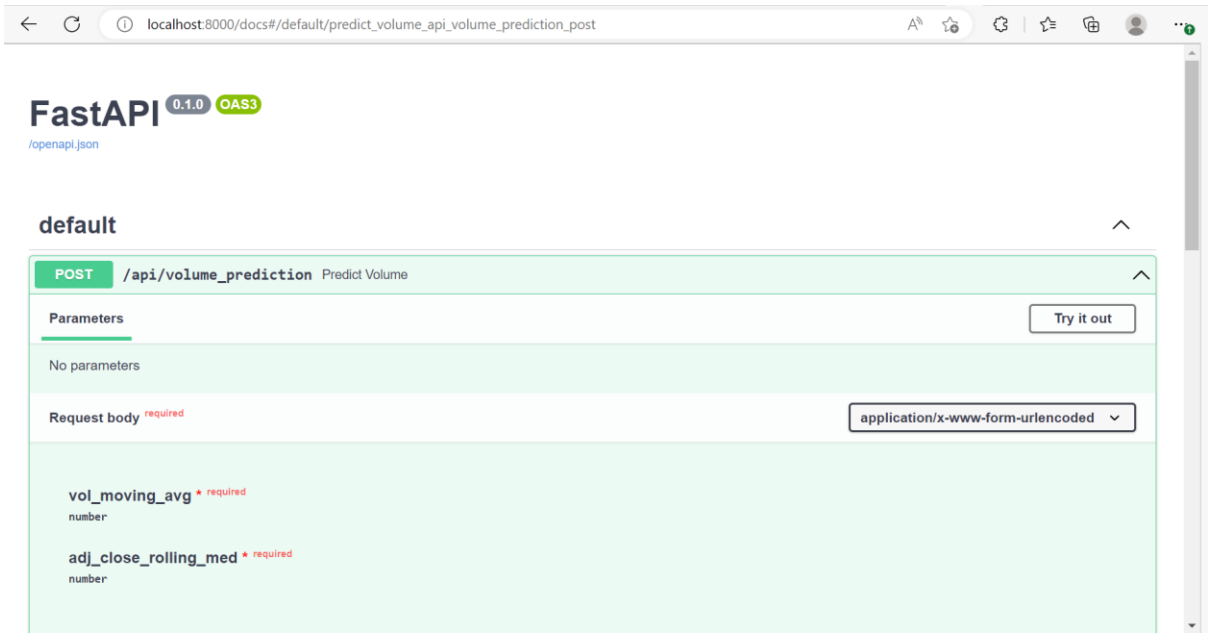
GradientBoostingRegressor model on the training data, predicts on the test data, saves the trained model to disc, calculates the Mean Absolute Error and The if __name__ == '__main__': block creates a multiprocessing pool with four processes, calls the integrate_ml_training function with each process, waits for all processes to complete and collect any errors, and then prints any errors or a success message then finally the model is saved to models folder.

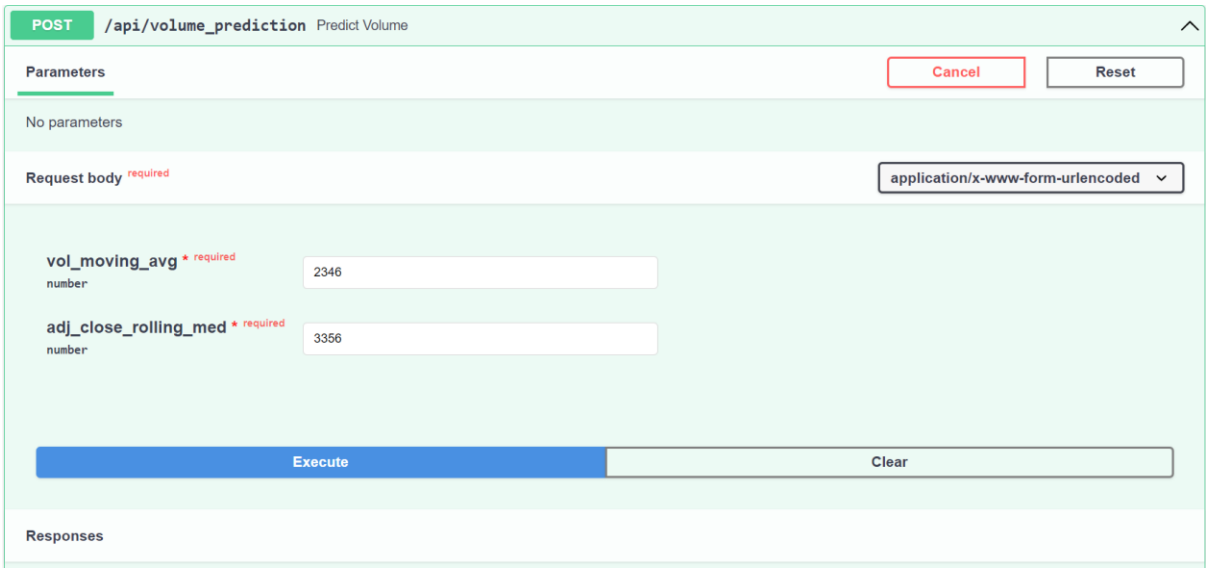
- I chose to use the GradientBoostingRegressor ML model to integrate with the processed data because it is known for its high accuracy and flexibility. This model adds decision trees sequentially to minimize the loss function, which allows it to focus on hard-to-predict data points. Since I used only two features from the parquet file, I preferred GradientBoostingRegressor. However, if there were many features present, I would have chosen the RandomForest ML model.

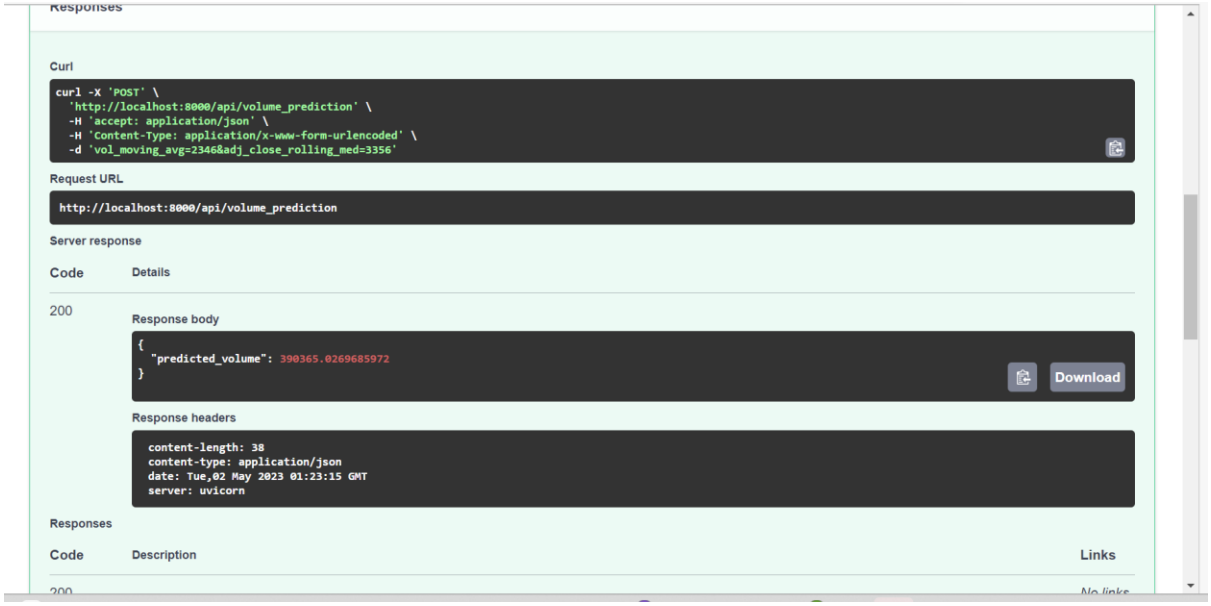
Problem 4: Model Serving

- I have successfully implemented an API service that serves the trained predictive model. To create API endpoints, I have utilized fastAPI, which takes in the values of vol_moving_avg and adj_close_rolling_med as inputs and returns the predicted trading volume using the trained predictive model.
- I have developed a Docker image to containerize the API service that I implemented. Due to the large size of the file, I was unable to host the service on Render, as their free tier does not offer sufficient memory support. Additionally, I attempted to host the service on Heroku, but they do not support any free services, which prevented me from hosting it there as well.
- I have uploaded it to GitHub from there it can be easily accessed and run the container.

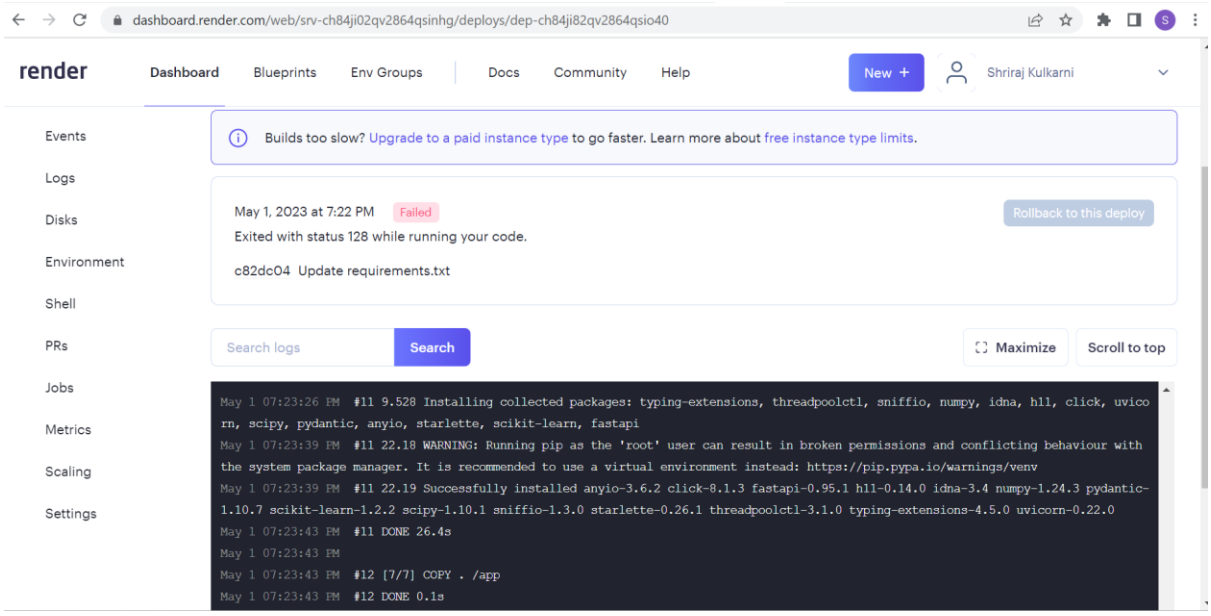
I have attached some screenshots that demonstrate the successful execution of the API service on my local machine.





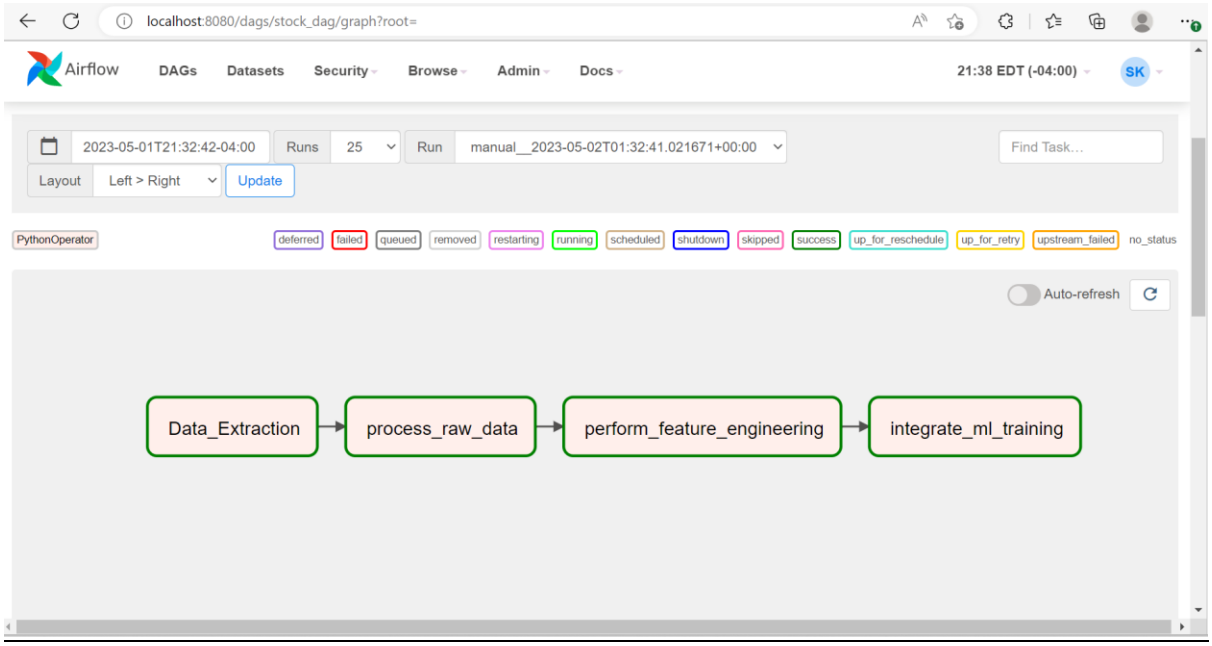


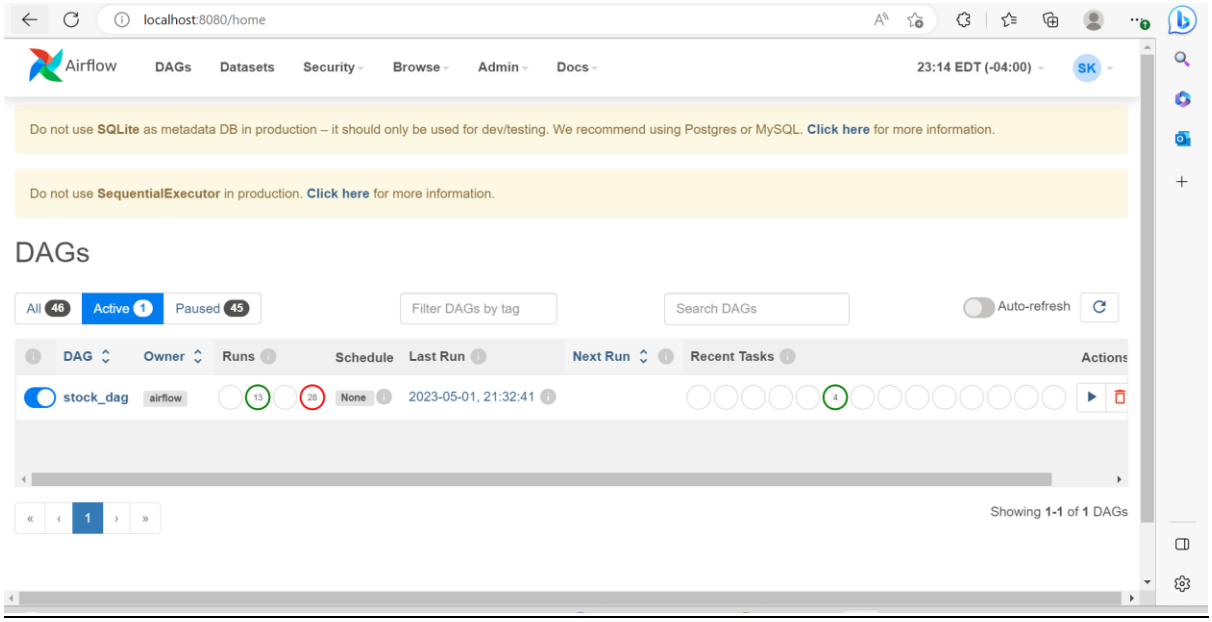
The render error:



RESULT:

Pipeline Diagram





I have uploaded all the necessary files to my GitHub account under the username Shrirajsk7 (Shriraj Kulkarni) with two repositories named **Airflow_Data_Pipeline** and **Trained_predictive_model_API**. The Airflow_Data_Pipeline repository contains the code for solving problems 1 to 4 while the Trained_predictive_model_API repository contains the code for a Dockerized API web-application.

GitHub: Shrirajsk7 (Shriraj Kulkarni) (github.com)

Airflow_Data_Pipeline ➔ Problem 1 to 4

Trained_predictive_model_API ➔ Dockerized API web-application.

Due to size limitations on GitHub, I have stored all large files on Google Drive. You can access the files from the provided link, which includes images of the data pipeline, Airflow logs, processed files, trained models, and model logs.

Google Drive: https://drive.google.com/drive/folders/137BuKy-yZFr3TxfRugajCOMm1VgJwePm?usp=share_link

CONCLUSION:

Based on my work sample as a Data Engineer for riskthinking.AI, I am confident in my ability to develop and implement effective data pipelines that can efficiently process and transform large datasets. Throughout the pipeline, I have utilized various tools and techniques such as multiprocessing, rolling window functions, and machine learning models to extract meaningful insights and predictions from the data. I have also demonstrated my ability to develop and deploy a web application using Docker and fastAPI, which serves the trained predictive model. Although I encountered some challenges while trying to host the application on free platforms such as Render and Heroku, I was able to successfully execute the application on my local machine. Overall, I believe my work sample showcases my expertise in data engineering and my ability to develop end-to-end data solutions. I am eager to continue exploring new technologies and techniques to further enhance my skills and contribute to the field of data engineering.

REFERENCES:

1. **https://www.youtube.com/watch?v=MpnfLcJflaw**
2. **https://www.youtube.com/watch?v=q8q3OFFfY6c**