National University of Computer and Emerging Sciences

MLOP Final Semester Project

Submitted to: Sir Hammad Majeed
Submitted by: Aimen Safdar
Roll #: 21i-0588

Date of submission: 15th December 2024

Table of Contents:

Task 1: Managing Environmental Data with DVC

Pre-tasks:

After cloning the repo, I ran the following commands on the terminal in my directory:

```
python -m venv venv

.\venv\Scripts\activate

pip install requests numpy pandas matplotlib scikit-learn dvc mlflow
```

1.1 Research Live Data Streams

Core Requirement: monitor environmental data (e.g., air quality, weather, and pollution levels) and predict pollution trends

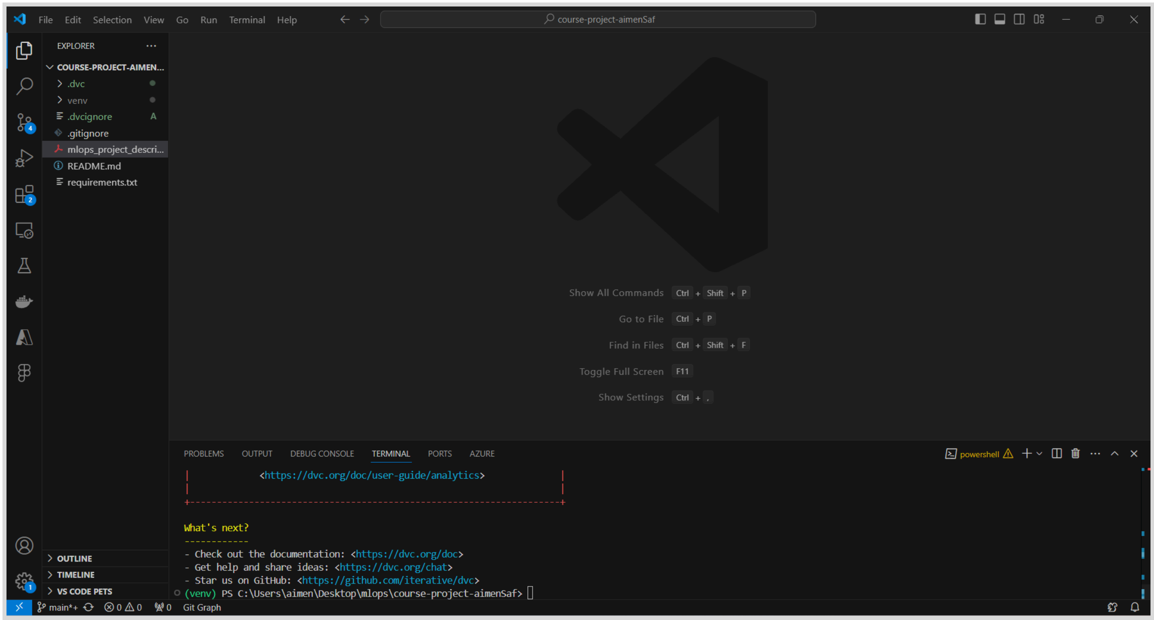
My API: Open Weather API and IQAir
Need to create accounts for them before we can get API.

1.2 Setting up DVC repository:

Used the command:

```
dvc init
git add .dvc .gitignore
git commit -m "Initialize DVC repository"
```

By running the commands, dvc was initialized, and .dvc and .gitignore were committed to the git repo. The prior was responsible for helping git version control files tracked by dvc.



1.3 Remote Storage Configuration:

For my remote storage, I used google drive. In order to use google drive, first i had to get the ID from URL of the designated directory. There is another library that we need to install here:

```
dvc[gdrive]
```

After downloading the above library, I added the ID from my google drive to the command below:

```
dvc remote add -d myremote gdrive://folder-id
```

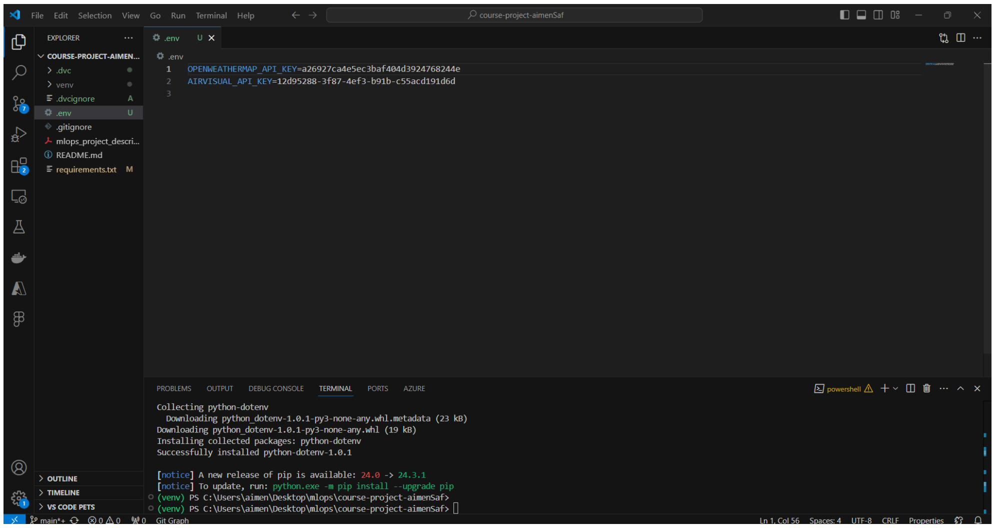
1.4 Data Collection Script:

```
import os
from dotenv import load_dotenv, find_dotenv
import requests
import pandas as pd
from datetime import datetime
```

```python
import time
import logging

load_dotenv(find_dotenv())

logging.basicConfig(filename='data_collection.log', level=logging.INFO,
format='%(asctime)s:%(levelname)s:%(message)s')

if "OPENWEATHERMAP_API_KEY" not in os.environ or "AIRVISUAL_API_KEY" not
in os.environ:
    logging.critical("API keys are not set in environment variables")
    raise ValueError("API keys are not set in environment variables")

# API configuration
API_KEYS = {
    "OpenWeatherMap": os.getenv('OPENWEATHERMAP_API_KEY'),
    "AirVisual": os.getenv('AIRVISUAL_API_KEY')
}
URLS = {
    "OpenWeatherMap": "http://api.openweathermap.org/data/2.5/weather",
    "AirVisual": "https://api.airvisual.com/v2/city"
}

# Parameters for the API calls
PARAMETERS = {
    "OpenWeatherMap": {"q": "London,uk", "appid":
API_KEYS["OpenWeatherMap"]},
    "AirVisual" : {"city": "London", "state": "England", "country":
"United Kingdom","key": API_KEYS["AirVisual"]
}


}

def fetch_data(url, params):
    try:
        response = requests.get(url, params=params)
        response.raise_for_status()
        return response.json()
    except requests.RequestException as e:
        logging.error(f"Error fetching data from {url}: {str(e)}")
```

```
        return None

def save_data(data, filename):
    try:
        df = pd.DataFrame([data])
        df.to_csv(filename, mode='a', header=not
pd.io.common.file_exists(filename), index=False)
        logging.info(f"Data saved successfully to {filename}")
    except Exception as e:
        logging.error(f"Error saving data: {str(e)}")

def main():
    weather_data = fetch_data(URLS["OpenWeatherMap"],
PARAMETERS["OpenWeatherMap"])
    if weather_data:
        save_data(weather_data, "data/weather_data.csv")

    air_quality_data = fetch_data("https://api.airvisual.com/v2/city",
PARAMETERS["AirVisual"])
    if air_quality_data:
        save_data(air_quality_data, "data/air_quality_data.csv")

if __name__ == "__main__":
    while True:
        main()
        sleeping_time = 300
        logging.info(f"Sleeping for {sleeping_time // 60} hour")
        time.sleep(sleeping_time)    # 1 hour for testing
```

The first order of business is to create environment variables for the APIs I am using; I created a folder .env file to store my keys. This approach offers many benefits; the sensitive keys are kept out of the source code and version control, it's easier to update the keys without touching the script, and simplifies deployment and development across different environments.



To use the .env file, first i ran the following command:

```
pip install python-dotenv
```

And then created the .env file with the following contents:

```
OPENWEATHERMAP_API_KEY=openweathermap_api_key
AIRVISUAL_API_KEY=airvisual_api_key
```

Then in the script.py file, it is designed to fetch and manage environmental data, specifically weather and air quality information, using APIs from OpenWeatherMap and AirVisual. It's structured to run continuously, pulling data every hour.

The script uses the python-dotenv library to securely load API keys from the .env file, ensuring these sensitive keys remain outside the source code. For each API, it sends HTTP requests using the requests library, handles potential errors gracefully, and logs activities and errors to a log file for monitoring and debugging purposes.

If the responses are successful, they are converted from JSON and append to CSV files for persistent storage.

After running the script, my .csv files were created and loaded successfully, with the data_collection.log showing no errors.

Below is a picture of the data_collection.log on the first try:

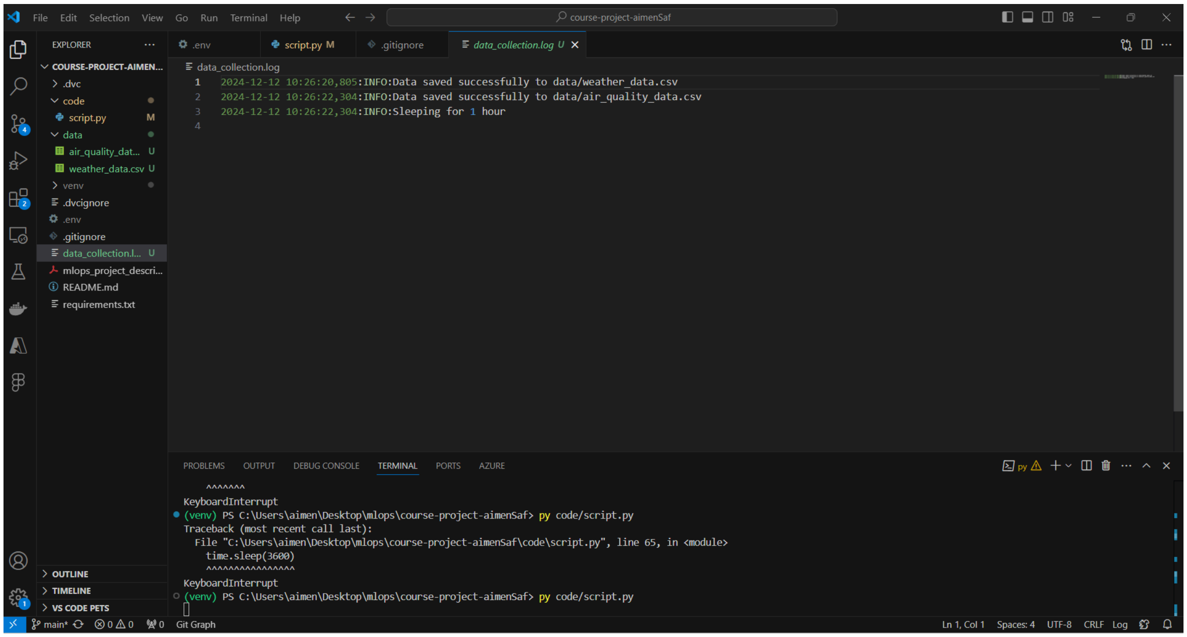


Fig: data_collection.log output

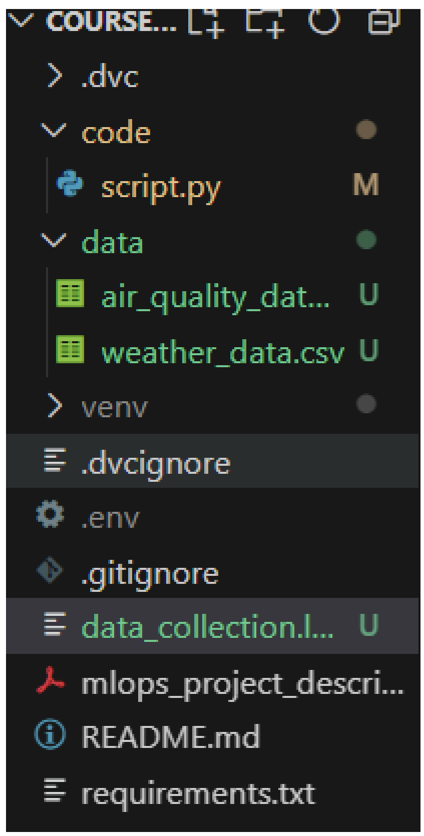


Fig: The .csv files successfully created

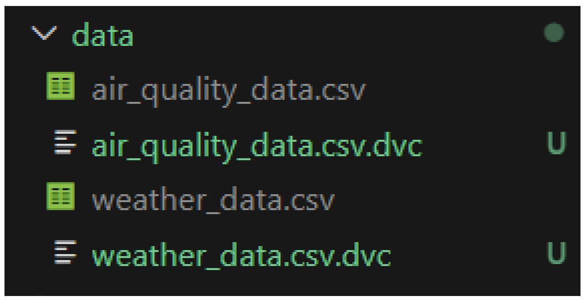
After letting the code run for 5 minutes, two outputs were appended into my .csv files.

1.5 Version Control with DVC

Now it is time to manage the collected data using the following commands:

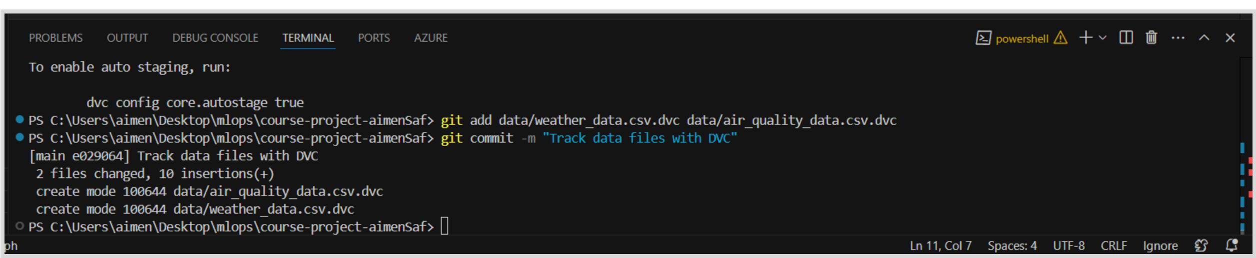
```
dvc add data/weather_data.csv
dvc add data/air_quality_data.csv
```

The commands resulted in two .dvc files being created in the same directory.



But for versioning tracking, we add the .dvc files into git repo. The following commands were used:

```
git add data/weather_data.csv.dvc data/air_quality_data.csv.dvc
git commit -m "Track data files with DVC"
```



After committing the .dvc files, now it was time to push the data onto the remote storage. But these required some additional steps.

Following the documentation provided by the DVC website (https://dvc.org/doc/user-guide/data-management/remote-storage/google-drive#using-a-custom-google-cloud-project-recommended), I created a custom google cloud project to generate OAuth credentials for myr GDrive remote to connect to my Google

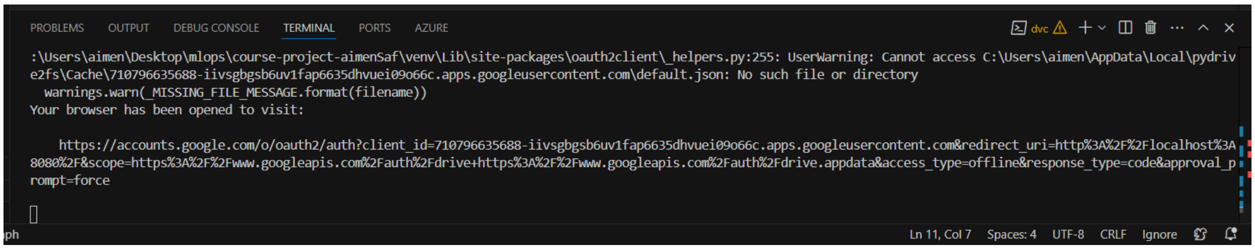
Drive. Through this a JSON file was created that held my credentials; I integrated this file into my project and added the credentials where necessary(.dvc/config).

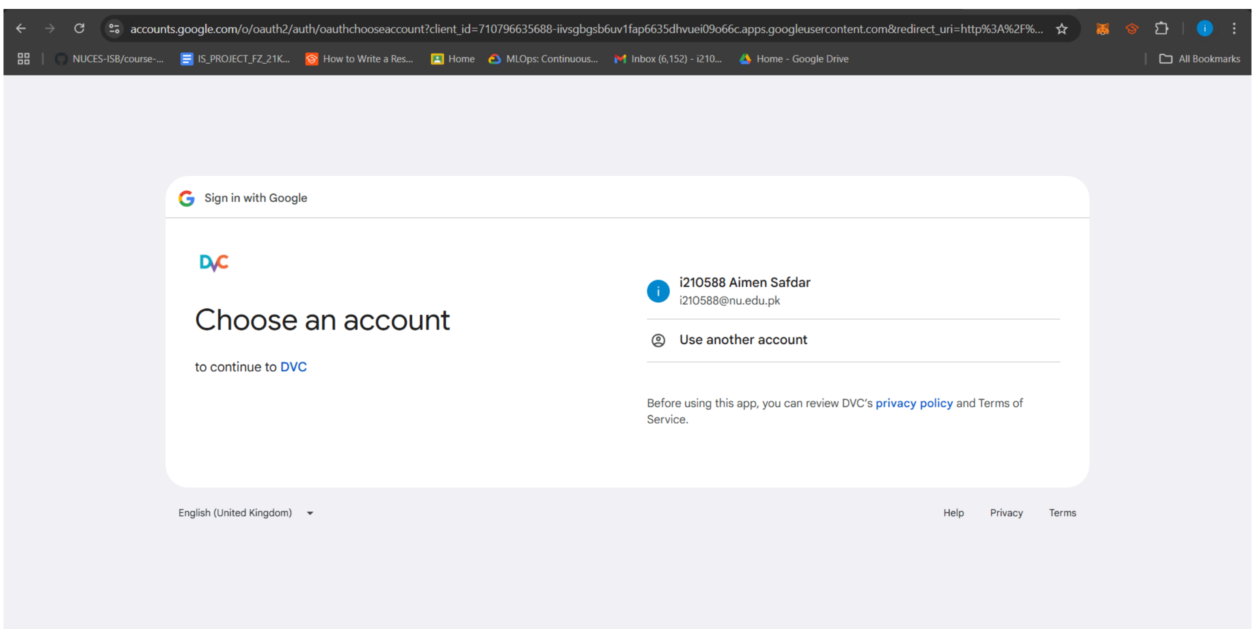
After following the steps, the commands below were run:

```
dvc remote modify myremote gdrive_client_id 'client-id'
dvc remote modify myremote gdrive_client_secret 'client-secret'
dvc push
```

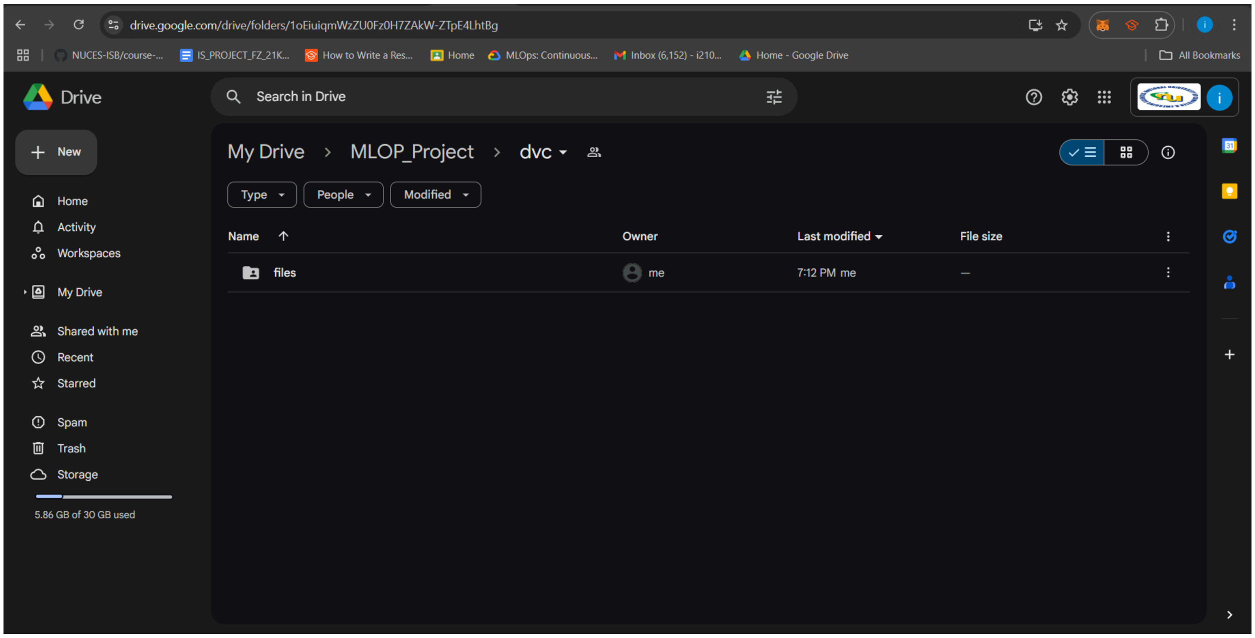
The parameters client-id and client-secret were replaced by the actual values. The result was that I was taken to authenticate and grant access to DVC for using google drive.

The command resulted in a link being generated as seen below:





Files added into my google drive:



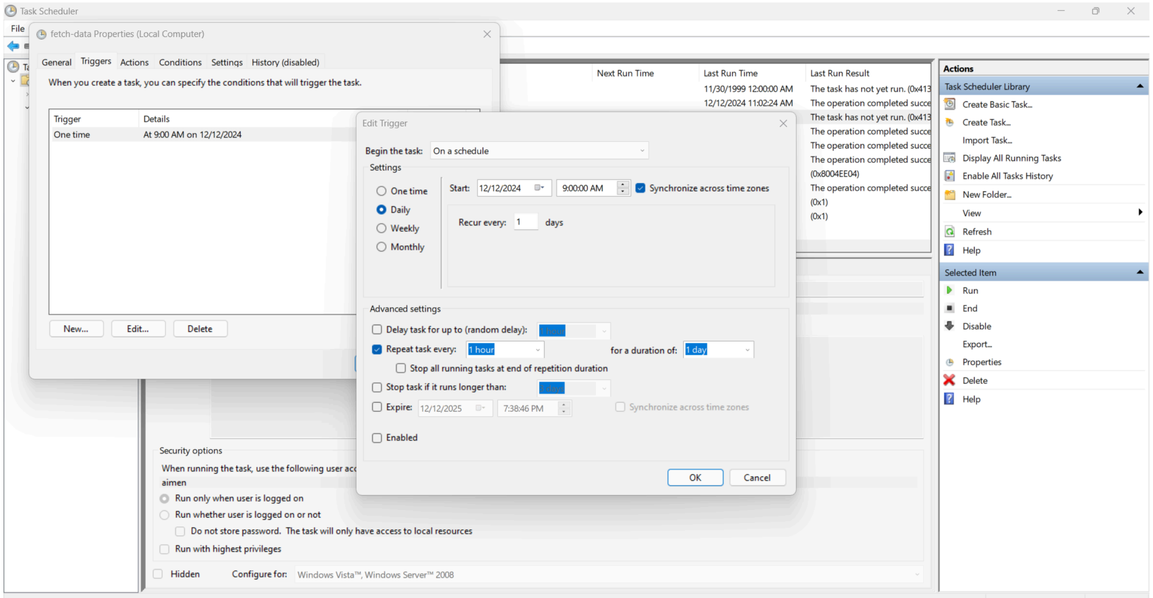
1.6 Automate Data Collection:

For automating the data collection, I needed a scheduler that periodically runs my data fetching script. For this task I used Task Scheduler. I started out by creating a batch file that consisted of shell commands.

fetch_data.bat

```
@echo off
cd C:\Users\aimen\Desktop\mlops\course-project-aimenSaf
call C:\Users\aimen\Desktop\mlops\course-project-aimenSaf\venv\Scripts\activate.bat
py C:\Users\aimen\Desktop\mlops\course-project-aimenSaf\code\script.py
dvc add data\weather_data.csv data\air_quality_data.csv
git add .
git commit -m "Update data files"
dvc push
git push
```

The batch file was then used inside the task scheduler to run the above commands after a certain interval of time.



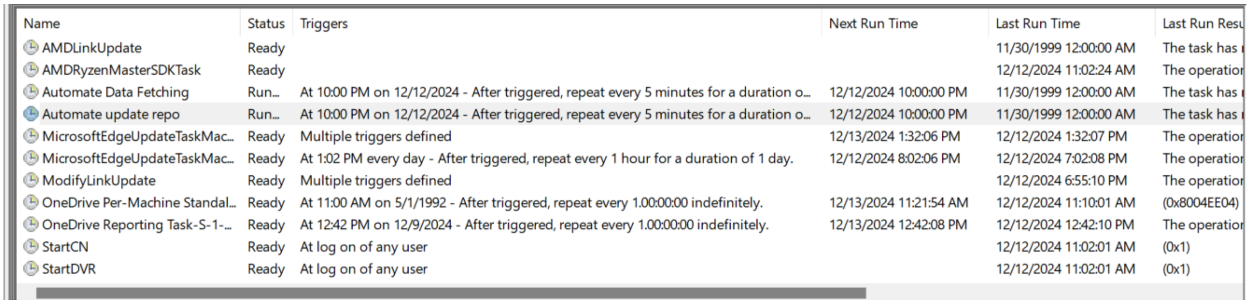
1.7 Update Data with DVC:

Now the data collection has been automated. The next step is to regularly update the DVC repo with new data.

To make things easier, I used shell to my advantage and created a .bat file and added all the necessary commands.

update_dvc_repo.bat

```
@echo off
cd C:\Users\aimen\Desktop\mlops\course-project-aimenSaf
call C:\Users\aimen\Desktop\mlops\course-project-aimenSaf\venv\Scripts\activate.bat

:: Add changes to DVC
dvc add data\weather_data.csv data\air_quality_data.csv

:: Auto-stages DVC files, manually add other important files
git add data\*.dvc dvc_client_id_secret.json fetch_data.bat update_dvc_repo.bat code\script.py
git commit -m "Update data files with DVC and script adjustments"
dvc push
git push

```



Name	Status	Triggers	Next Run Time	Last Run Time	Last Run Resu
AMDLinkUpdate	Ready			11/30/1999 12:00:00 AM	The task has
AMDRyzenMasterSDKTask	Ready			12/12/2024 11:02:24 AM	The operatior
Automate Data Fetching	Run...	At 10:00 PM on 12/12/2024 - After triggered, repeat every 5 minutes for a duration o...	12/12/2024 10:00:00 PM	11/30/1999 12:00:00 AM	The task has
Automate update repo	Run...	At 10:00 PM on 12/12/2024 - After triggered, repeat every 5 minutes for a duration o...	12/12/2024 10:00:00 PM	11/30/1999 12:00:00 AM	The task has
MicrosoftEdgeUpdateTaskMac...	Ready	Multiple triggers defined	12/13/2024 1:32:06 PM	12/12/2024 1:32:07 PM	The operatior
MicrosoftEdgeUpdateTaskMac...	Ready	At 1:02 PM every day - After triggered, repeat every 1 hour for a duration of 1 day.	12/12/2024 8:02:06 PM	12/12/2024 7:02:08 PM	The operatior
ModifyLinkUpdate	Ready	Multiple triggers defined		12/12/2024 6:55:10 PM	The operatior
OneDrive Per-Machine Standal...	Ready	At 11:00 AM on 5/1/1992 - After triggered, repeat every 1.00:00:00 indefinitely.	12/13/2024 11:21:54 AM	12/12/2024 11:10:01 AM	(0x8004EE04)
OneDrive Reporting Task-S-1-...	Ready	At 12:42 PM on 12/9/2024 - After triggered, repeat every 1.00:00:00 indefinitely.	12/13/2024 12:42:08 PM	12/12/2024 12:42:10 PM	The operatior
StartCN	Ready	At log on of any user		12/12/2024 11:02:01 AM	(0x1)
StartDVR	Ready	At log on of any user		12/12/2024 11:02:01 AM	(0x1)

Task 2: Pollution Trend Prediction with MLflow

2.1 Data Preparation:

Started off, by creating a new python file that will pre-process the data in the .csv file. The two already existing .csv files are merged together to form one file and this new file is pushed onto dvc. The following command were used:

```
dvc add output/combined_data.csv
git add output/combined_data.csv.dvc output/.gitignore
git commit -m "Add combined data file to DVC"
dvc push
```

The merged file was stored in a new folder called “output” and after running the commands we have a .dvc and a .csv file. I pushed the csv file onto the remote storage and to track this file, I added the .dcv onto git. But to make sure the right files are tracked I updated the .gitignore file as well.

.gitignore

```
__pycache__/
*.pyo
*.pyd
*.pyc
venv/
.dvc/cache
.dvc/tmp
.env
data_collection.log
*.csv
data/*
!data/*.dvc
output/*
!output/*.dvc
```

2.2 Model Development:

For this task I am using ARIMA and Prophet models; their results are compared in a later step.

ARIMA (AutoRegressive Integrated Moving Average)

Using ARIMA as it is a popular statistical method for time-series forecasting, used extensively in environmental science. The reason why I chose this model is because it models time-series data based on its own past values (autoregression), the differences of the values (integrated), and forecast errors (moving average).

- **Autoregression (AR)**: The model uses the dependency between an observation and a number of lagged observations.
- **Integrated (I)**: This involves differencing the time series to make it stationary, i.e., constant mean and variance over time.
- **Moving Average (MA)**: The model exploits the relationship between the observation and the residual error from a moving average model applied to lagged observations.

ETS:

The ETS model (Error, Trend, Seasonality) model that I used is a time-series forecasting method that decomposes the data into error, trend, and seasonal components. These components can be combined in an additive or multiplicative manner, depending on the nature of the data.

2.3 Train Models with MLflow

I created two separate python files for each model and trained them separately; the results were displayed on mflow's UI.

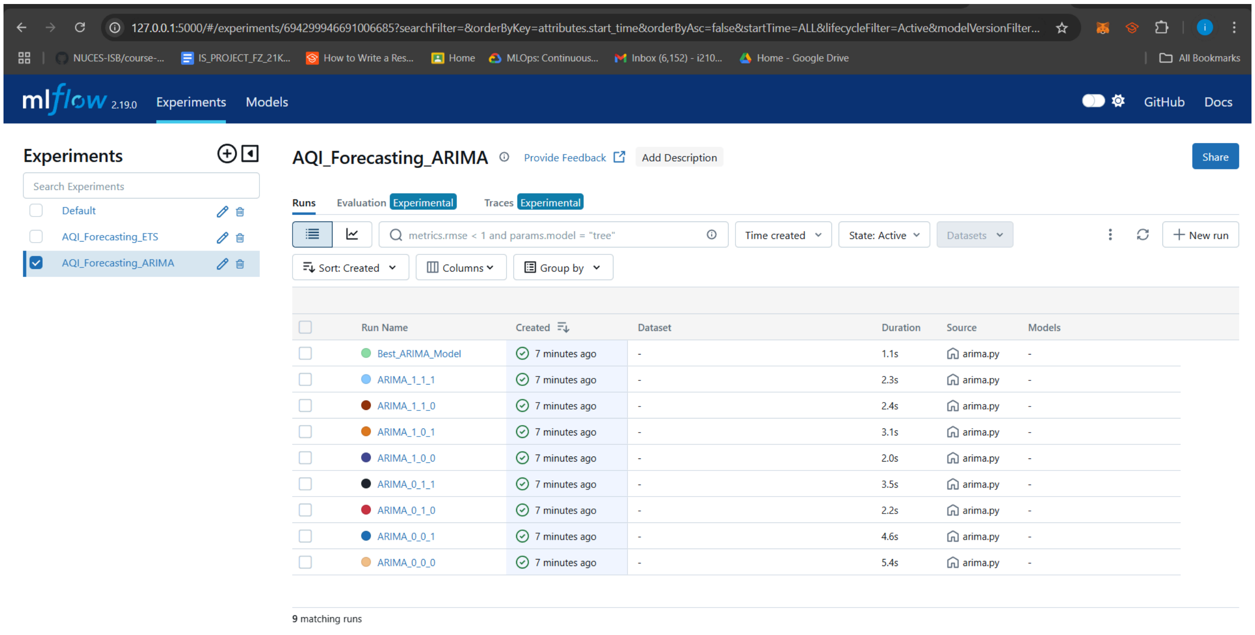
```
python arima_mlflow.py
python ets_mlflow.py

mlflow ui
```

Arima:

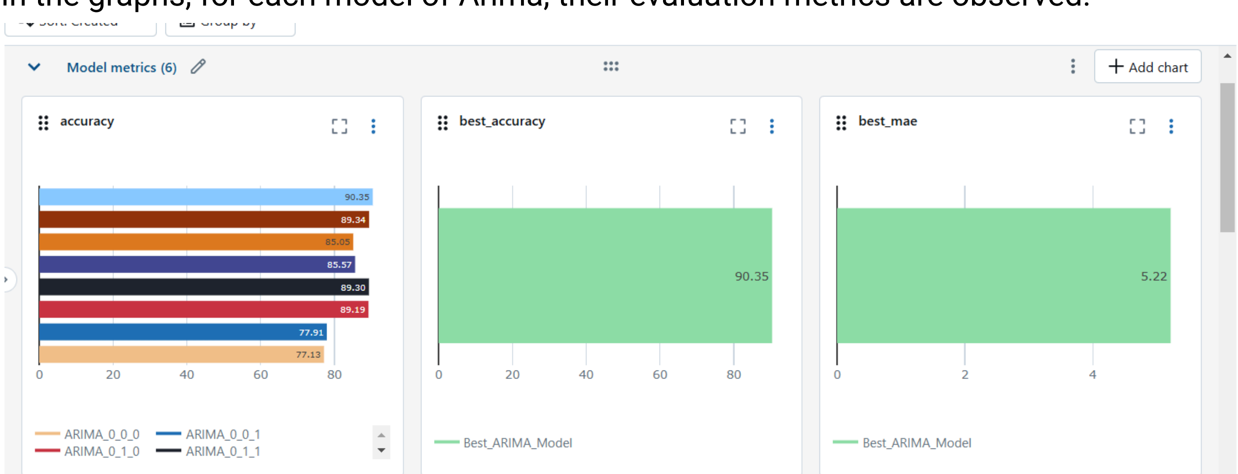
Result of training Arima Model on data collected so far.

```
2024/12/13 23:21:24 INFO mlflow.tracking.fluent: Experiment with name 'AQI_Forecasting_AR
C:\Users\aimen\Desktop\mlops\course-project-aimenSaf\models\arima.py:24: FutureWarning: '
on, please use 'h' instead.
  df = df.resample('H').mean().interpolate(method='linear')
ARIMA(0,0,0) - RMSE: 12.68, MAE: 11.94, Accuracy: 77.13%
ARIMA(0,0,1) - RMSE: 12.46, MAE: 11.57, Accuracy: 77.91%
ARIMA(0,1,0) - RMSE: 7.21, MAE: 5.82, Accuracy: 89.19%
ARIMA(0,1,1) - RMSE: 7.17, MAE: 5.76, Accuracy: 89.30%
ARIMA(1,0,0) - RMSE: 9.19, MAE: 7.71, Accuracy: 85.57%
ARIMA(1,0,1) - RMSE: 9.48, MAE: 7.98, Accuracy: 85.05%
ARIMA(1,1,0) - RMSE: 7.15, MAE: 5.74, Accuracy: 89.34%
ARIMA(1,1,1) - RMSE: 6.65, MAE: 5.22, Accuracy: 90.35%

Best ARIMA Model: (1, 1, 1)
Best RMSE: 6.65
Best MAE: 5.22
Best Accuracy: 90.35%
```

Below is a picture of Mlflow UI showing the results of Arima Model; the results are displayed in both the list format and the graph method.



In the graphs, for each model of Arima, their evaluation metrics are observed.

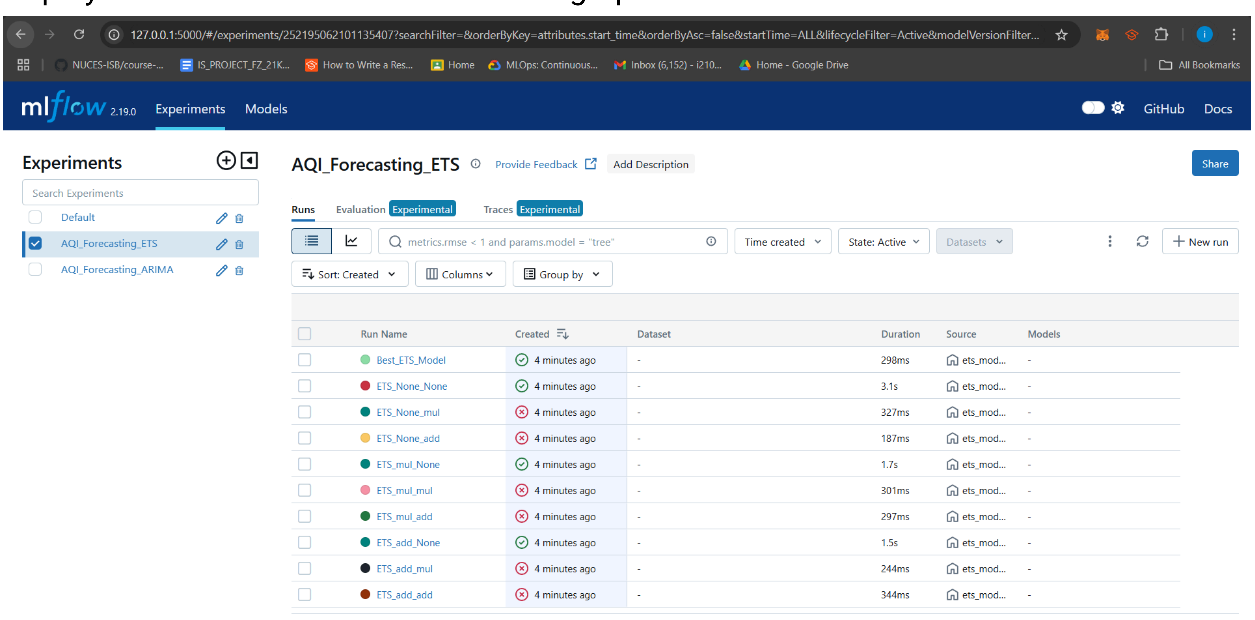


ETS:

Below is the result of training ETS model on the data collected so far.

```
Best ETS Model Parameters:
Trend: mul
Seasonal: None
Best RMSE: 2.75
Best MAE: 1.58
Best Accuracy: 97.21%
```

Below is a picture of Mlflow UI showing the results of Arima Model; the results are displayed in both the list format and the graph method.

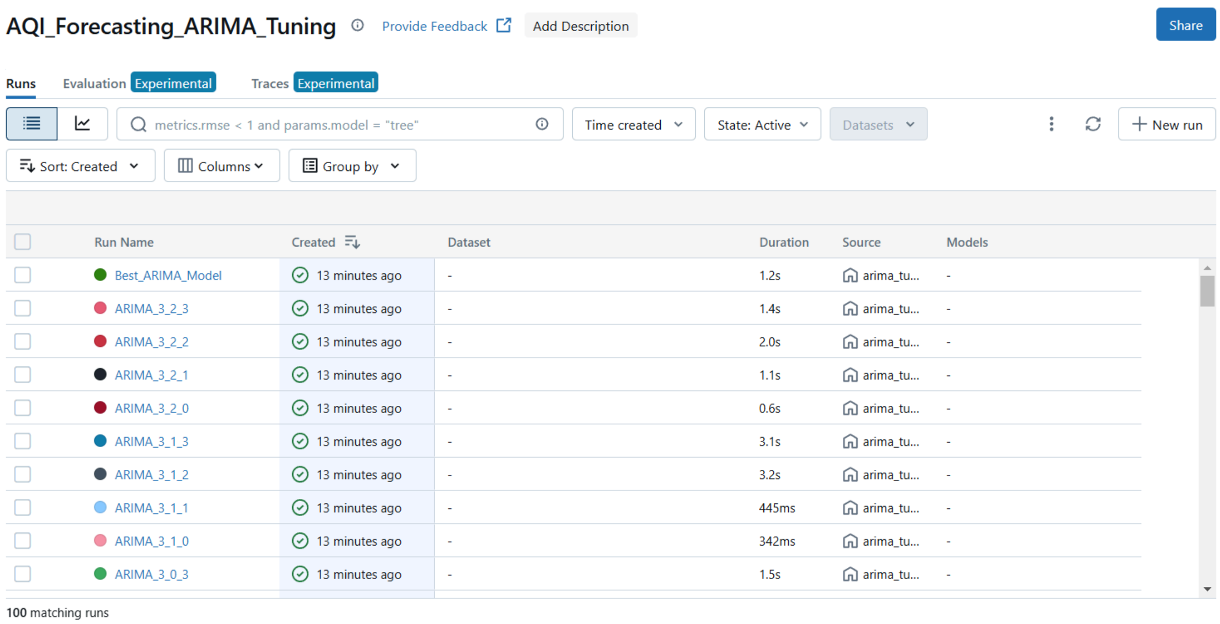


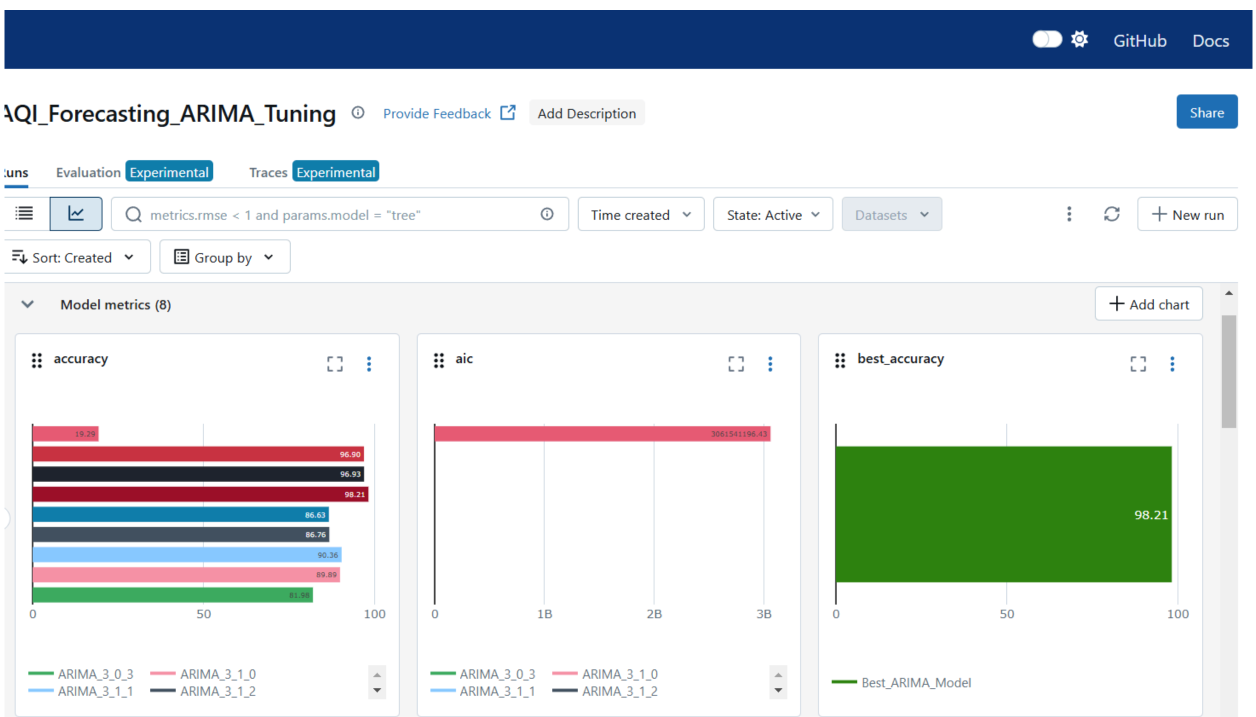
2.4 Hyperparameter Tuning:

After changing some hyperparameters for each model, I trained the model again and visualized the results on mlflow.

Arima:

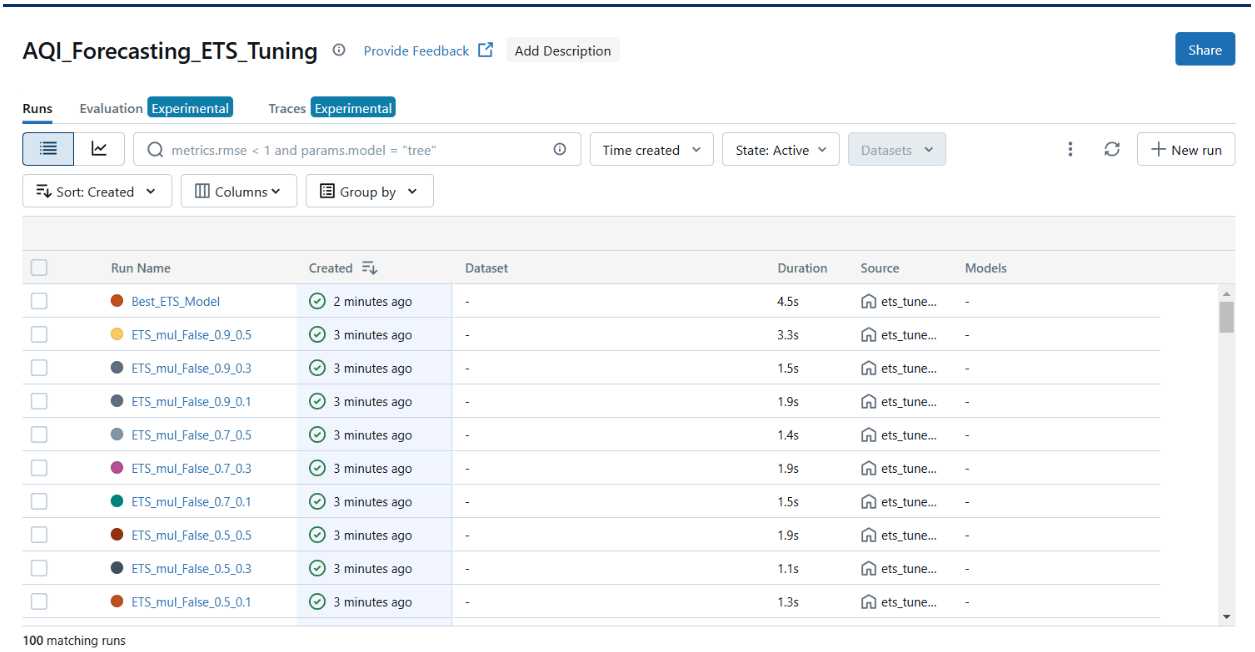
Below is a picture of Mlflow UI showing the results of the fine tuned Arima Model; the results are displayed in both the list format and the graph method.

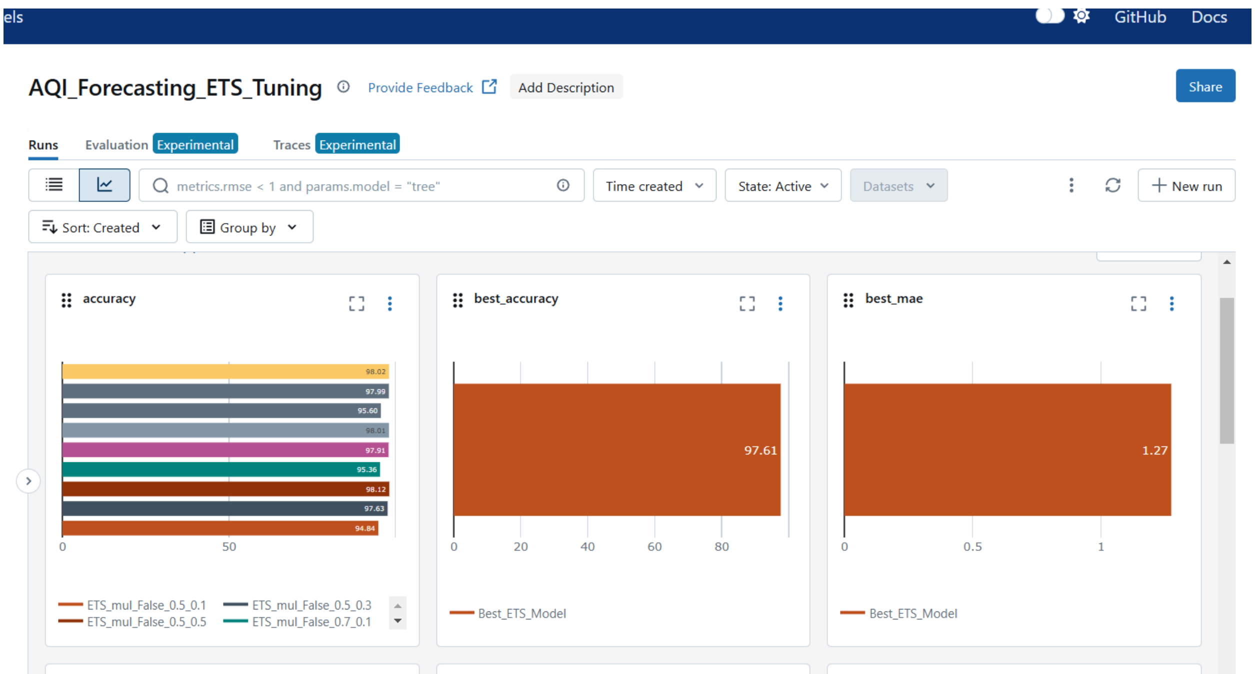




ETS Hyper-tuned model result:

Below is a picture of Mlflow UI showing the results of the fine tuned Arima Model; the results are displayed in both the list format and the graph method.





2.5 Model Evaluation:

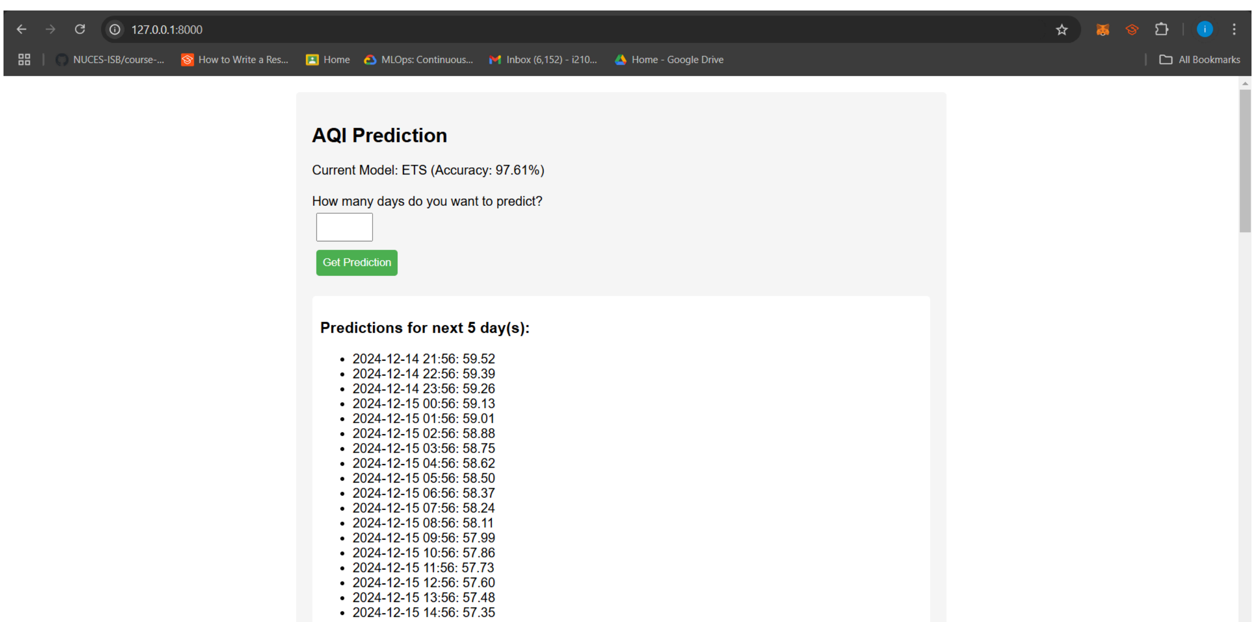
To compare which model presented the best performance, I created a python script to check which model performed the best. The script uses the metrics stored on Mlflow for each model; for comparing, all 4 models are benign compared.

```
KeyError: 'params'
(venv) PS C:\Users\aimen\Desktop\mlops\course-project-aimenSaf> py evaluation.py

Model Comparison:
     Metric      ARIMA       ETS
0      RMSE   2.384746  1.568008
1       MAE   1.053896  1.273160
2  Accuracy  98.211243  97.610980

Best Model: ETS
Best Model Parameters: {'trend': 'add', 'damped_trend': 'False', 'smoothing_slope': '0.5', 'smoothing_level': '0.1', 'seasonal': None, 'seasona
l_periods': None, 'smoothing_seasonal': None}
Best Model Metrics:
RMSE: 1.57
MAE: 1.27
Accuracy: 97.61%
```

The best model I got was ETS with an accuracy of 97.61%.

2.6 Deployment:

For my deployment, I created a simple flask application which predicts the AQI for the x number of days in the future where x is user input.



The output consists of the date, time and the AQI metric in each row.