# CS- MLOPS

# Environmental Monitoring and Pollution Prediction System

Final Project

Submitted to : Dr.Hammad Majeed

Laiba Asif (21i2560)
Section A
Date: 12/14/24

# Tables of content

# TASK 1:

## Step 1: Research Live Data Streams

**Objective**

Identify suitable APIs to collect environmental data, focusing on air quality, weather, and pollution levels, to integrate into an MLOps pipeline for monitoring and prediction.

**Tools**

1. **API Documentation for Data Providers:**
   - **OpenWeatherMap API**: Provides comprehensive weather and air quality data, including:
     - Current weather conditions
     - Air pollution data (e.g., PM2.5, PM10, Ozone)
     - Historical and forecast data
     - URL: [https://openweathermap.org/api](https://openweathermap.org/api)
   - **AirVisual API (IQAir)**: Offers global air quality data with pollutant concentrations and AQI levels.
     - URL: [https://www.iqair.com/air-pollution-data-api](https://www.iqair.com/air-pollution-data-api)
   - **EPA AirNow API**: Provides air quality data for U.S. locations with detailed AQI reports.
     - URL: [https://docs.airnowapi.org/](https://docs.airnowapi.org/)
   - **NOAA Python Library**: NOAA's weather and climate data, accessible via a Python wrapper for easy integration.
     - URL: [https://pypi.org/project/noaa-weather/](https://pypi.org/project/noaa-weather/)
2. **Python Tools for API Interaction:**
   - **Python Libraries:**
     - `requests`: For making API calls to fetch data.
     - `json`: For parsing and handling JSON data returned by APIs.

**Evaluation: Why OpenWeatherMap API?**

The OpenWeatherMap API is the best option based on the following factors:

1. **Data Coverage:**
   - Provides a wide range of weather data, including air pollution (PM2.5, PM10, CO, NO2, O3, SO2).
   - Covers global locations, which makes it versatile for diverse environmental monitoring needs.
2. **Ease of Use:**
   - Straightforward API endpoints with detailed documentation.
   - Free tier available for basic usage, with higher-tier plans for advanced features.

3. **Data Granularity:**
    - Provides hourly updates for real-time monitoring.
    - Supports historical, current, and forecast data.
4. **Limitations:**
    - API call limits depend on the plan (e.g., free tier offers 60 API calls/minute).
    - Requires registration for an API key.

**Action**

1. **API Selection:** OpenWeatherMap API was chosen for its comprehensive data, global coverage, and ease of integration.
2. **Registration and API Key:**
    - Visit [OpenWeatherMap API](OpenWeatherMap API).
    - Create a free account and generate an API key for accessing endpoints.
3. **Next Steps:**
    - Explore relevant API endpoints for weather and air pollution data:
        - Weather Data: `/weather` or `/forecast`
        - Air Pollution Data: `/air_pollution`

Test API responses using the `requests` library in Python:

```
Import requests
API_KEY = "2fbc453063630496c3ab531f5de7535f"
url = "http://api.openweathermap.org/data/2.5/air_pollution"
params = {'lat': 35.6895, 'lon': 139.6917, 'appid': API_KEY}
response = requests.get(url, params=params)

if response.status_code == 200:
    print(response.json())
else:
    print(f"Error: {response.status_code}")
```

4. **When Run on terminal :**API endpoints, parameters, and expected responses for future reference.

```
PS C:\Users\Laiba Asif\Desktop\7th semester\MLops\21i2560_project_mlops\mlops_project> python test_api.py
>>
{'coord': {'lon': 139.6917, 'lat': 35.6895}, 'list': [{'main': {'aqi': 3}, 'components': {'co': 781.06, 'no': 198.48, 'no2': 105.56, 'o3': 0, 'so2': 59.13, 'pm2_5': 20.77, 'pm10': 32.54, 'nh3': 1.74}, 'dt': 1734091459}]}
```

## Step 2: Set Up a DVC Repository

**Objective:** Initialize a repository to manage environmental data.

1. **Tools:**
   - DVC: https://dvc.org/
   - Git: https://git-scm.com/
2. **Action:**
   - Create a project directory: `mkdir env-monitoring && cd env-monitoring`.

Initialize Git and DVC:

```
git init
```

```
dvc init
```

```
(venv) PS C:\Users\Laiba Asif\Desktop\7th semester\MLops\21i2560_project_mlops\mlops_project> dvc --version
>>
3.58.0
(venv) PS C:\Users\Laiba Asif\Desktop\7th semester\MLops\21i2560_project_mlops\mlops_project> dvc init
>>
Initialized DVC repository.

You can now commit the changes to git.

+---------------------------------------------------------------------+
|
 |
|        DVC has enabled anonymous aggregate usage analytics.
 |
|     Read the analytics documentation (and how to opt-out) here:
 |
|             <https://dvc.org/doc/user-guide/analytics>
+---------------------------------------------------------------------+

What's next?
------------
- Check out the documentation: <https://dvc.org/doc>
- Get help and share ideas: <https://dvc.org/chat>
- Star us on GitHub: <https://github.com/iterative/dvc>
(venv) PS C:\Users\Laiba Asif\Desktop\7th semester\MLops\21i2560_project_mlops\mlops_project> git init
Reinitialized existing Git repository in C:/Users/Laiba Asif/Desktop/7th semester/MLops/21i2560_project_mlops/mlops_project/.git/
(venv) PS C:\Users\Laiba Asif\Desktop\7th semester\MLops\21i2560_project_mlops\mlops_project>
```

```
(venv) PS C:\Users\Laiba Asif\Desktop\7th semester\MLops\21i2560_project_mlops\21i2560_project> dvc remote list

 myremote        gdrive://1K0o09PFT4JigV7KG9lTk2McDkPLfvhRb
```

# Step 3: Configure Remote Storage

**Objective:**
To link the DVC repository to a remote storage solution, enabling efficient versioning and storage of data files.

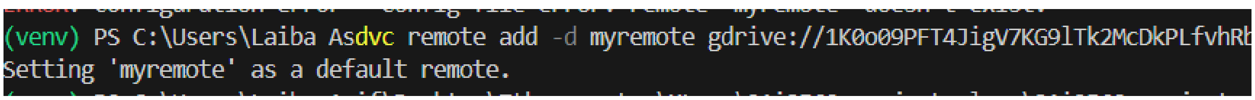
**Tools Used:**

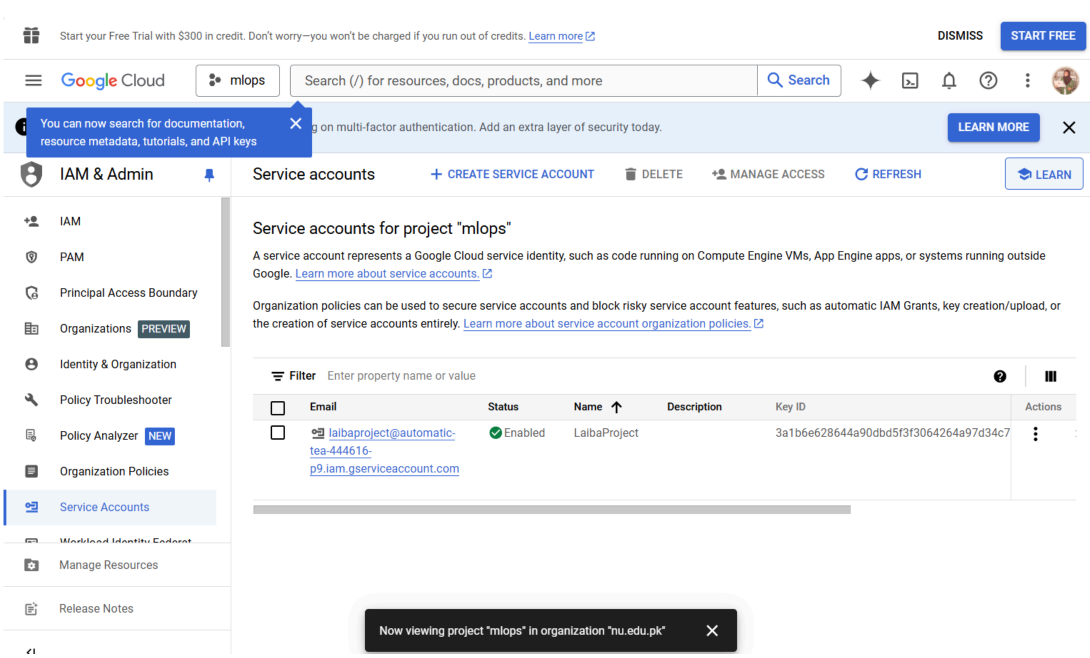
- **DVC**: For managing and versioning datasets.
- **Google Drive**: Configured as the remote storage solution.
- **Python**: For managing the credentials and authentication process.

**Actions Taken:**

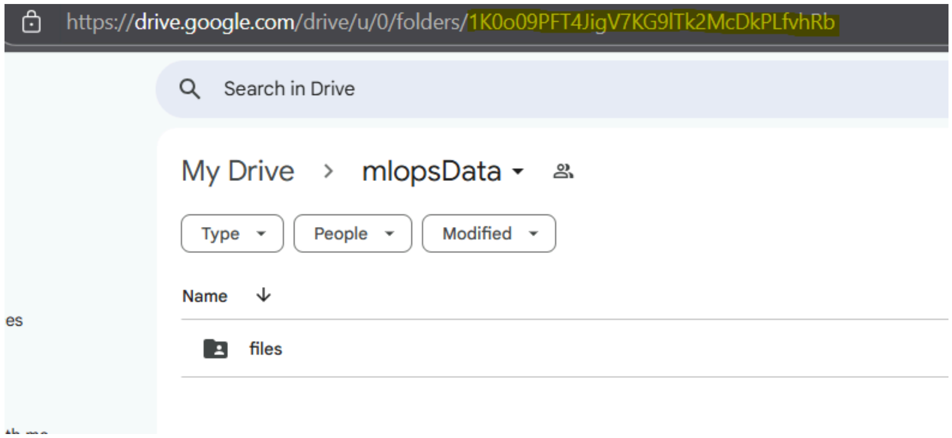
1. **Setup Remote Storage:**

Added Google Drive as the remote storage for the DVC repository using:
`dvc remote add -d myremote gdrive://<folder-id>`




- The `<folder-id>` corresponds to the unique identifier of a folder created in Google Drive to store the data.

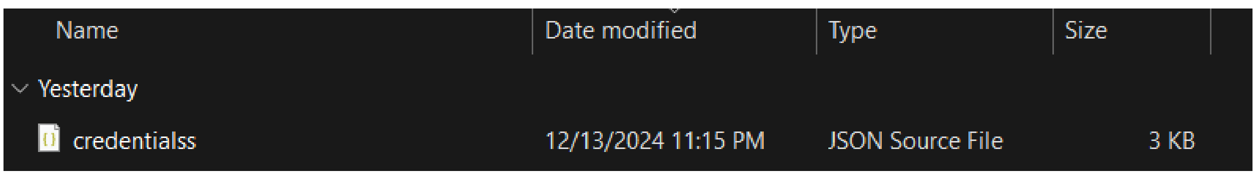



○

2. **Authentication with Google Drive:**

Configured authentication by setting the `GDRIVE_CREDENTIALS_DATA` environment variable to the content of `credentials.json`:

```
$env:GDRIVE_CREDENTIALS_DATA = Get-Content -Raw
"path/to/credentials.json"
```

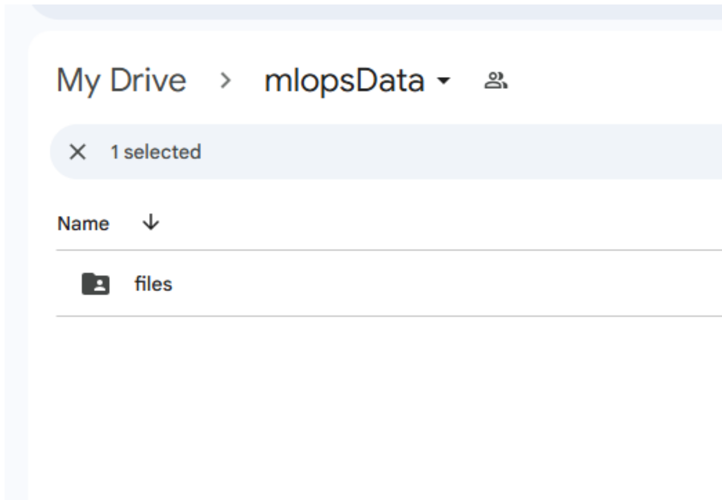




3. **Testing Connectivity:**

Created a placeholder file (`test.txt`), tracked it with DVC, and attempted to push it to the remote storage:

```
echo "DVC Test" > test.txt
dvc add test.txt
dvc push
```




- Verified the successful upload of the file in the specified Google Drive folder.

**Outcome:**
The DVC repository was successfully linked to Google Drive as the remote storage. Version control and data storage are now integrated, ensuring easy management of datasets.

## Step 4: Write a Data Collection Script

**Objective:**
Develop a Python script to fetch weather and air quality data from APIs and store the data in a structured format for further use.

**Tools Used:**

- **Python Libraries:**
  - `requests`: For making API calls.
  - `json`: For handling and storing the API responses.
  - `os`: For creating and managing directories.
  - `datetime`: For generating timestamped filenames.
- **API Documentation:** OpenWeatherMap API (from Step 1).

**Actions Taken:**

1. **Script Creation:**
   - A Python script, `data_fetch.py`, was written to fetch weather and air quality data using the OpenWeatherMap API.
2. **Key Functionalities:**
   - **API Integration:** The script connects to the OpenWeatherMap API using an API key (`API_KEY`).

- **Parameterization:** Accepts latitude and longitude as inputs for fetching location-specific data.
- **Data Storage:** Saves the API response in a JSON file within the `data` directory, with filenames containing timestamps for versioning.
- **Error Handling:** Includes exception handling to ensure robustness and logs relevant debug information.

**Script Code:**

```
import os
import requests
import json
from datetime import datetime

# Define constants
API_KEY = "2fbc453063630496c3ab531f5de7535f"
BASE_URL = "http://api.openweathermap.org/data/2.5/forecast"
DATA_DIR = "data"

# Ensure the data directory exists
os.makedirs(DATA_DIR, exist_ok=True)

def fetch_air_pollution_data(lat, lon):
    print("Fetching data...")  # Debug log
    params = {
        "lat": lat,
        "lon": lon,
        "appid": API_KEY
    }
    try:
        response = requests.get(BASE_URL, params=params)
        print(f"Response status code: {response.status_code}")  # Debug
log
        if response.status_code == 200:
            data = response.json()
            timestamp = datetime.now().strftime('%Y%m%d_%H%M%S')
            filename = os.path.join(DATA_DIR,
f"environmental_data_{timestamp}.json")
            with open(filename, "w") as file:
                json.dump(data, file, indent=4)
            print(f"Data saved to {filename}")
        else:
```

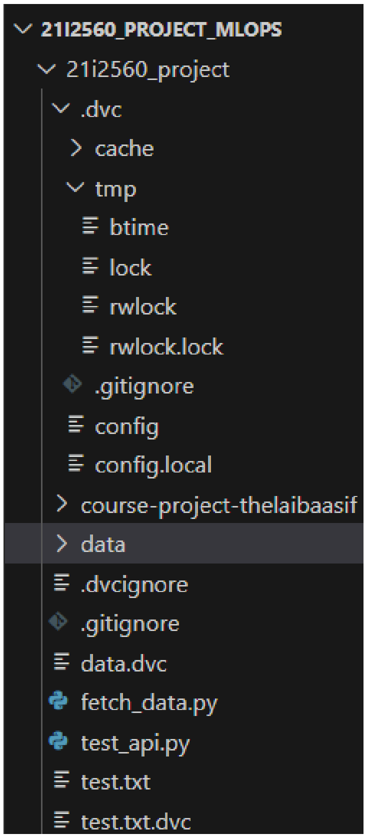
```
            print(f"Failed to fetch data. Status Code:
{response.status_code}, Response: {response.text}")
    except Exception as e:
        print(f"An error occurred: {e}")

if __name__ == "__main__":
    print("Script started...")  # Debug log
    fetch_air_pollution_data(lat=40.7128, lon=-74.0060)  # Example: New
York City
    print("Script finished.")  # Debug log
```

3. **Directory Creation:**

Created a `data` directory for storing the fetched data:

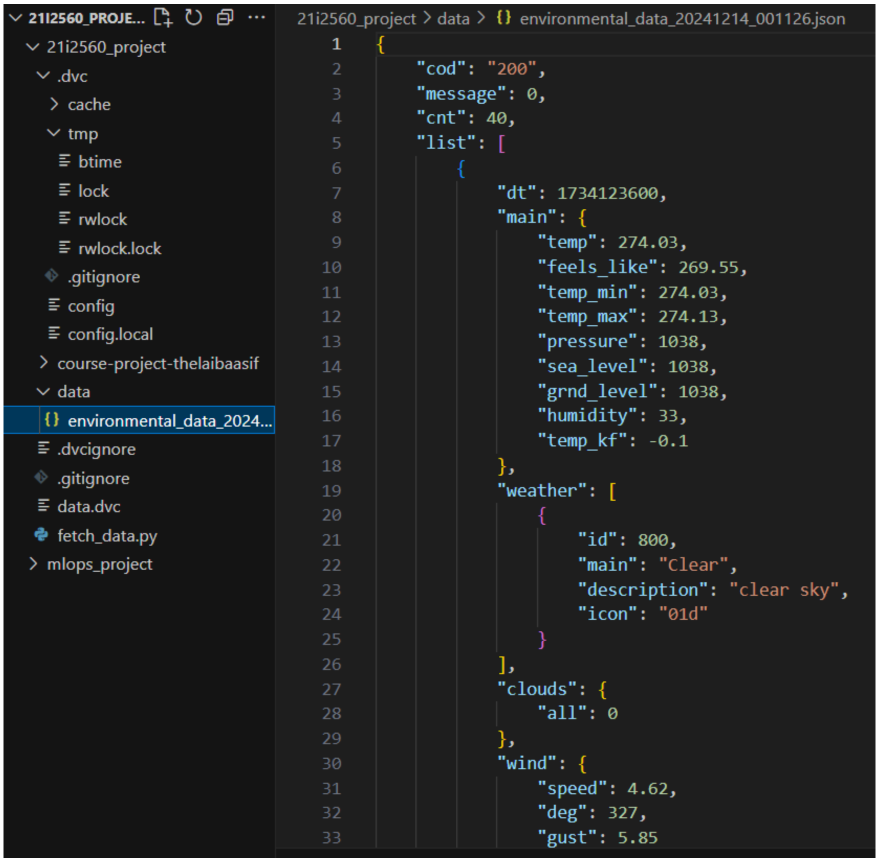
```
mkdir data
```

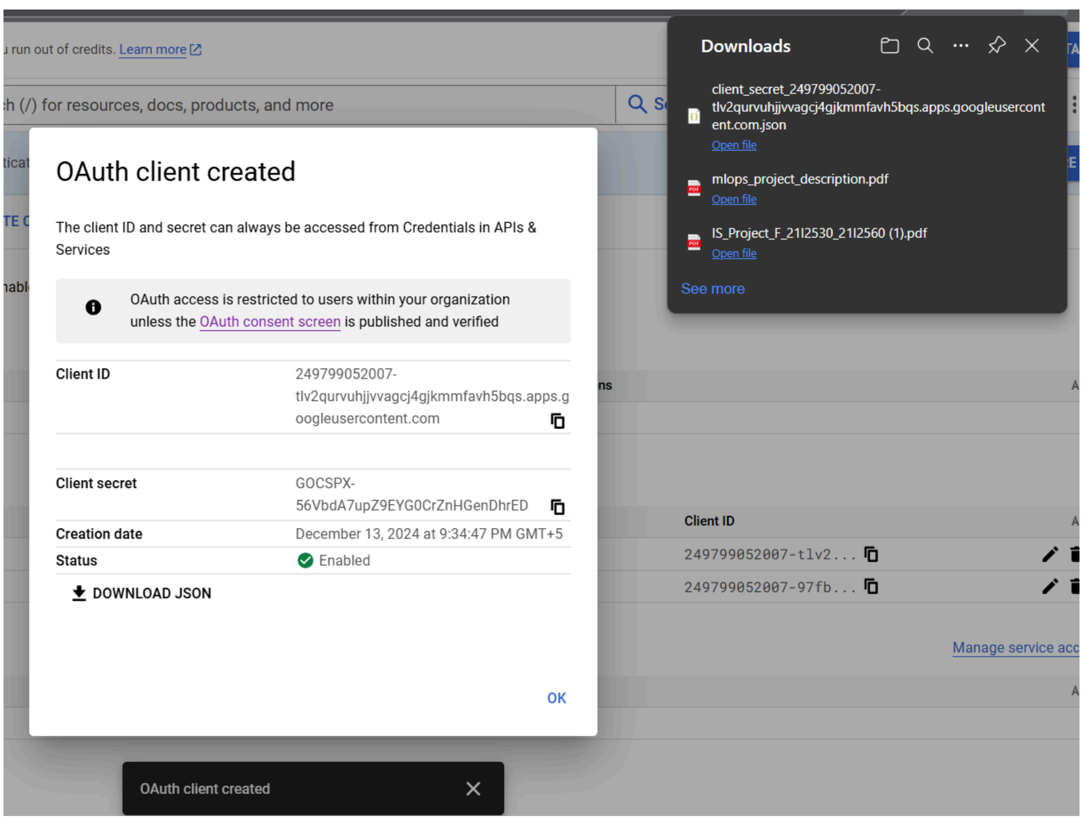
```
(venv) PS C:\Users\Laiba Asif\Desktop\7th semester\MLops\21i2560_project_mlops\21i2560_project> python fetch_data.py
Script started...
Script started...
Fetching data...
Response status code: 200
Fetching data...
Fetching data...
Response status code: 200
Data saved to data\environmental_data_20241214_011624.json
Script finished.
```




4. **Execution:**
   - Ran the script to fetch weather and air quality data for New York City (latitude: `40.7128`, longitude: `-74.0060`).
   - Verified that the JSON file containing the fetched data was successfully saved in the `data` directory.

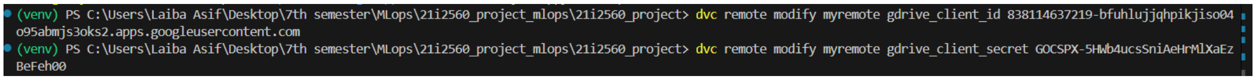
**Outcome:**
The `data_fetch.py` script was successfully implemented and tested, providing a reusable tool for collecting real-time environmental data. Data is now organized and stored for further analysis or integration into downstream workflows.




dvc remote modify myremote gdrive_client_id <your-client-id>

dvc remote modify myremote gdrive_client_secret <your-client-secret>


```
(venv) PS C:\Users\Laiba Asif\Desktop\7th semester\MLops\21i2560_project_mlops\21i2560_project> dvc remote modify myremote gdrive_client_id 838114637219-bfuhlujjqhpikjiso04o95abmjs3oks2.apps.googleusercontent.com
(venv) PS C:\Users\Laiba Asif\Desktop\7th semester\MLops\21i2560_project_mlops\21i2560_project> dvc remote modify myremote gdrive_client_secret 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
```

## Step 5: Version Control with DVC

**Objective:**
To use DVC for tracking and versioning fetched environmental data in the `data` directory.

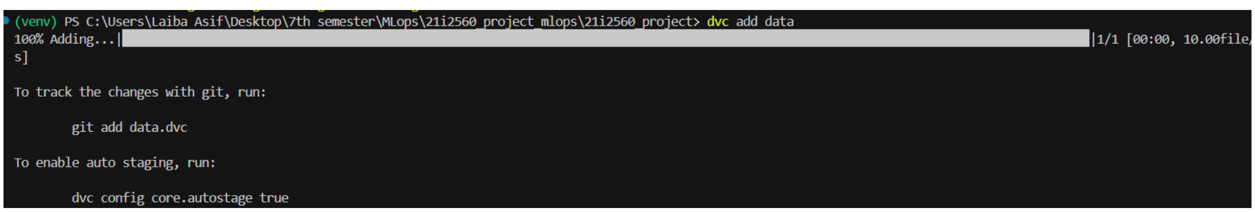
**Tools Used:**

- **DVC CLI Commands**: `dvc add`, `dvc commit`, `dvc push`.
- **Git**: For managing DVC metadata.

**Actions Taken:**

1. **Stage Data Directory with DVC**:

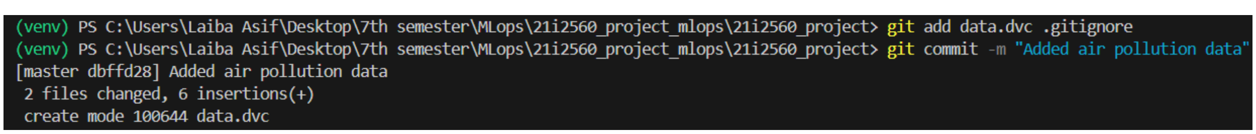
Added the `data` directory to DVC for versioning:
bash
Copy code
`dvc add data`


```
(venv) PS C:\Users\Laiba Asif\Desktop\7th semester\MLops\21i2560_project_mlops\21i2560_project> dvc add data
100% Adding...|                                                                  |1/1 [00:00, 10.00file/
s]

To track the changes with git, run:

        git add data.dvc

To enable auto staging, run:

        dvc config core.autostage true
```

2. **Track Metadata with Git**:

Added the generated `.dvc` file and `.gitignore` to Git:

```
git add data.dvc .gitignore
git commit -m "Add initial data directory"
```

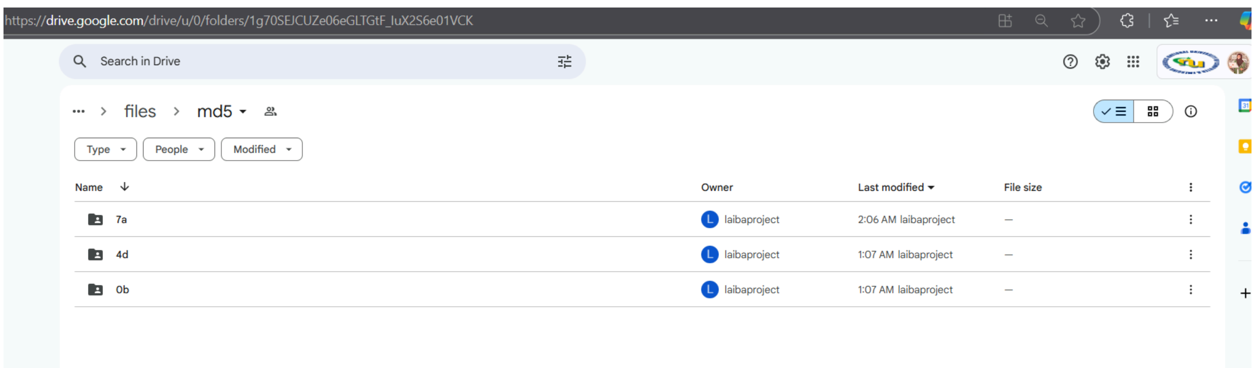

```
(venv) PS C:\Users\Laiba Asif\Desktop\7th semester\MLops\21i2560_project_mlops\21i2560_project> git add data.dvc .gitignore
(venv) PS C:\Users\Laiba Asif\Desktop\7th semester\MLops\21i2560_project_mlops\21i2560_project> git commit -m "Added air pollution data"
[master dbffd28] Added air pollution data
 2 files changed, 6 insertions(+)
 create mode 100644 data.dvc
```

3. **Push Data to Remote Storage**:

Uploaded the versioned data to the configured remote (Google Drive):
`dvc push`

```
(venv) PS C:\Users\Laiba Asif\Desktop\7th semester\MLops\21i2560_project_mlops\21i2560_project> dvc push
Collecting
s]
Pushing
Pushing
2 files pushed
2 files pushed
```

**Outcome:**
The `data` directory is now version-controlled and stored remotely, ensuring reliable and organized data management.




# Step 6: Automate Data Collection

**Objective:**
To automate the data-fetching process to collect environmental data at regular intervals.

**Tools Used:**

- **Task Scheduler (Windows)** for scheduling.

**Actions Taken:**

1. **Set Up a Scheduled Job**:
    - Configured a task to run the `data_fetch.py` script every hour:

Added the following line to schedule the task:

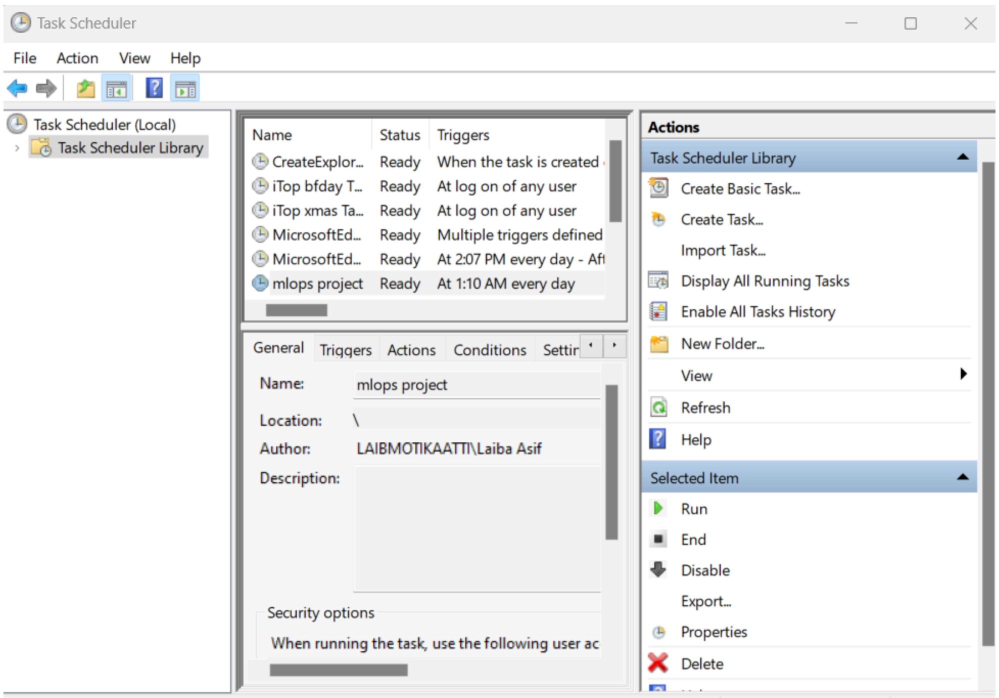
```
0 * * * * python /path/to/env-monitoring/data_fetch.py
```

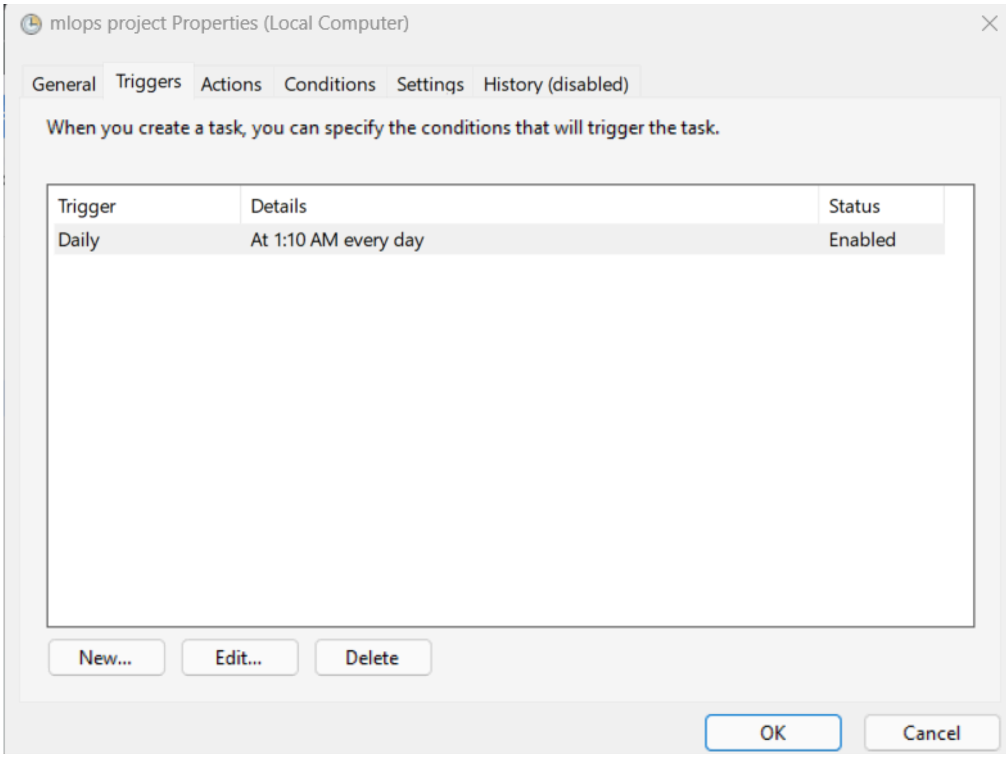
- For Windows (Task Scheduler):

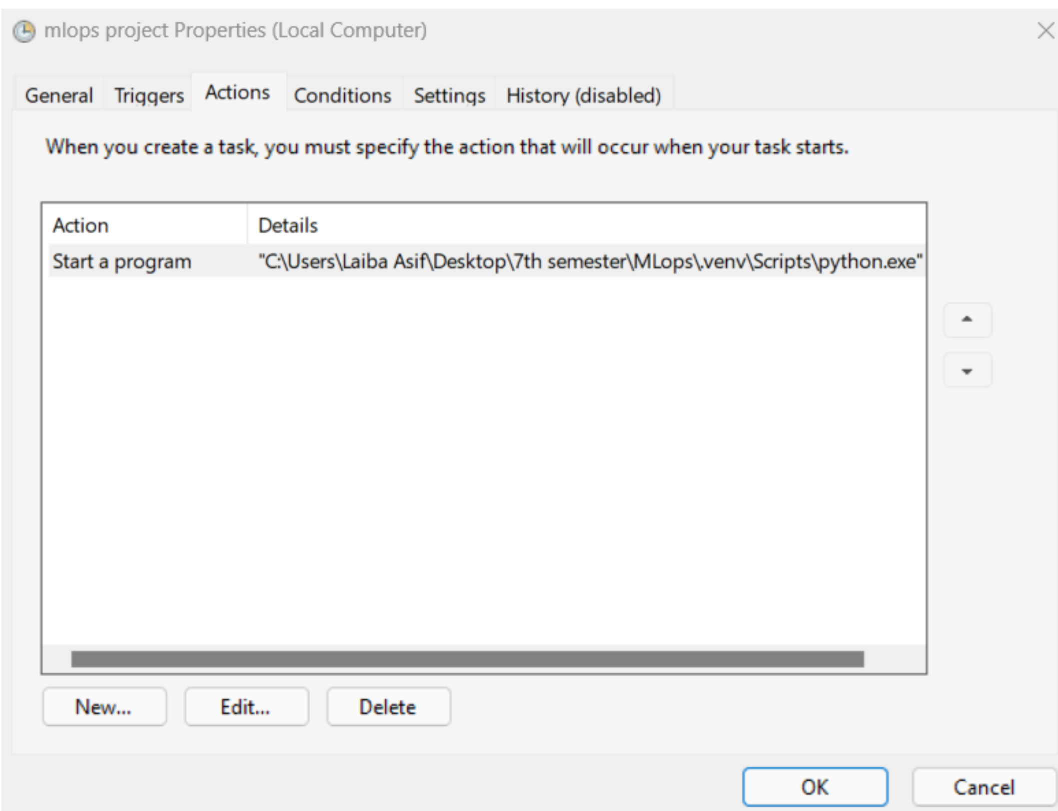
■ Created a new task and scheduled it to execute the script hourly.

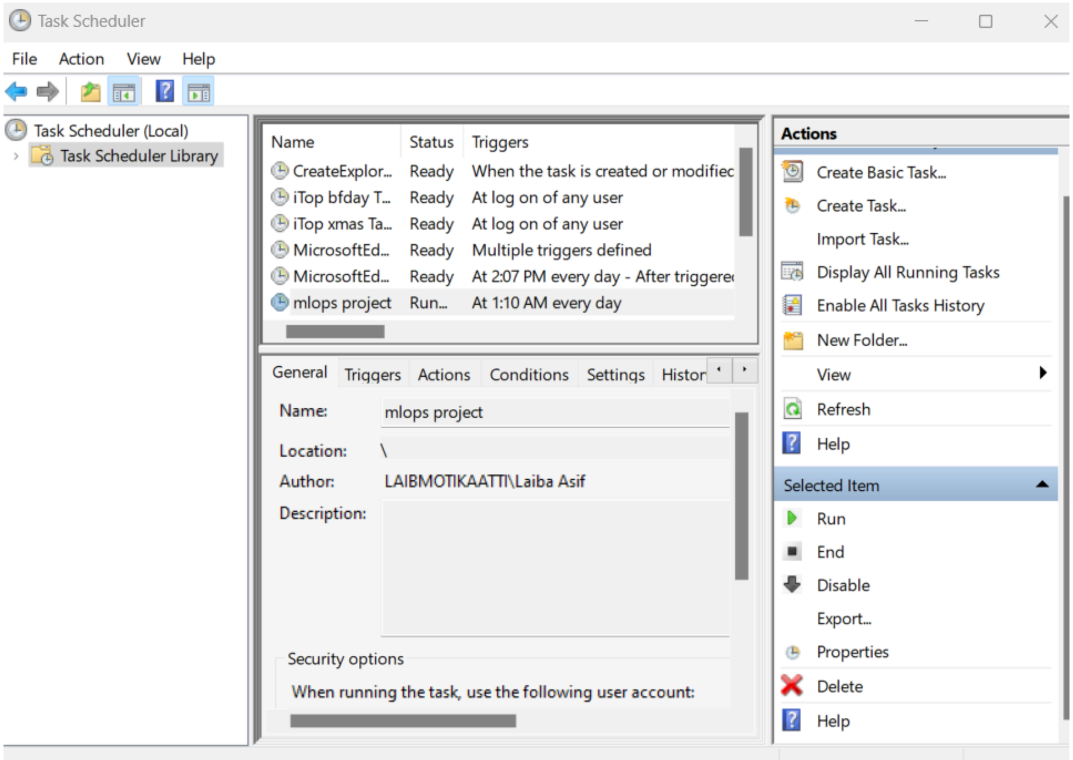
2. **Verification**:
   - Checked scheduled jobs:

**Outcome:**
Data collection is now automated and runs hourly, ensuring timely updates.




mlops project Properties (Local Computer)
General
Triggers
Actions
Conditions
Settings
History (disabled)
When you create a task, you can specify the conditions that will trigger the task.
Trigger
Details
Status
Daily
At 1:10 AM every day
Enabled
New...
Edit...
Delete
OK
Cancel


mlops project Properties (Local Computer)
General
Triggers
Actions
Conditions
Settings
History (disabled)
When you create a task, you must specify the action that will occur when your task starts.
Action
Details
Start a program
"C:\Users\Laiba Asif\Desktop\7th semester\MLops\.venv\Scripts\python.exe"
New...
Edit...
Delete
OK
Cancel




---

## Step 7: Update Data with DVC

**Objective:**
Automate version control for newly collected data.

**Tools Used:**

- **DVC CLI Commands**: `dvc add`, `dvc push`.
- **Shell Scripting** for automation.

**Actions Taken:**

1. **Create Update Script**:

Wrote a shell script `update_data.sh` to automate version control:

```
#!/bin/bash
dvc add data
git add data.dvc
git commit -m "Update data directory with new data"
dvc push
```

2. **Schedule the Update Script**:

Scheduled the script to run daily at midnight using `cron`:

bash

Copy code

```
crontab -e
```

Added the following line:

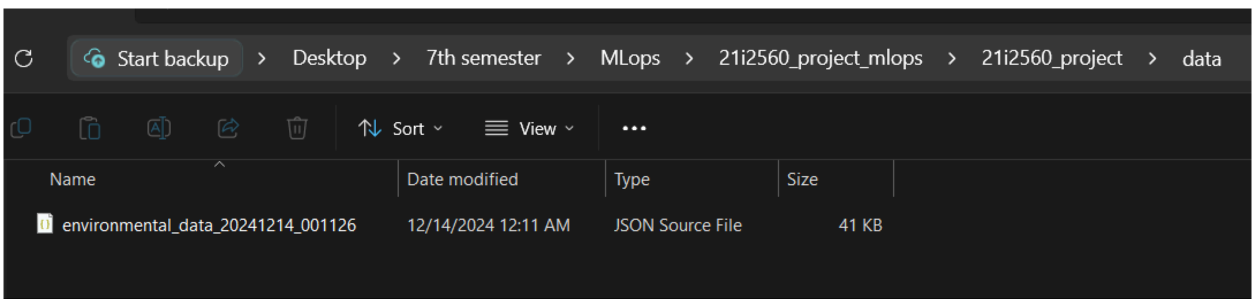
bash

Copy code

```
0 0 * * * /bin/bash /path/to/env-monitoring/update_data.sh
```

**Outcome:**

Version control is automated for new data, ensuring all updates are versioned and pushed to remote storage daily.



# Task 2: Pollution Trend Prediction with MLflow

---

**Objective**

The goal of this task was to develop and deploy models to predict pollution trends and alert high-risk days using time-series modeling, MLflow for tracking, and deploying the final model as an API.

## Steps and Implementation

## 1. Data Preparation

- **Objective:** Preprocess environmental data for model training and inference.
- **Implementation:**
  - Handled missing values by interpolating data or filling with mean values for consistency.
  - Removed outliers using Z-score analysis.
  - Normalized data using MinMaxScaler to transform features (e.g., temperature, humidity, pressure, wind speed, cloud cover) into a consistent range for better model convergence.

```
Script started...
Fetching data...
Response status code: 200
Sample of fetched data:
              datetime  temperature  feels_like  humidity  pressure  \
0  2024-12-14 12:00:00       270.68      266.92        57      1047
1  2024-12-14 15:00:00       271.36      267.46        49      1047
2  2024-12-14 18:00:00       272.69      270.22        39      1046
3  2024-12-14 21:00:00       274.03      271.89        28      1046
4  2024-12-15 00:00:00       273.52      270.61        32      1046

   wind_speed  wind_deg weather_main weather_description
0        2.76        40        Clear           clear sky
1        3.04        37        Clear           clear sky
2        1.95        14        Clear           clear sky
3        1.86         2        Clear           clear sky
4        2.46        37        Clear           clear sky
Data saved to data/environmental_data_20241214_104137.csv
Script finished.
```

**Figure: Data fetched**

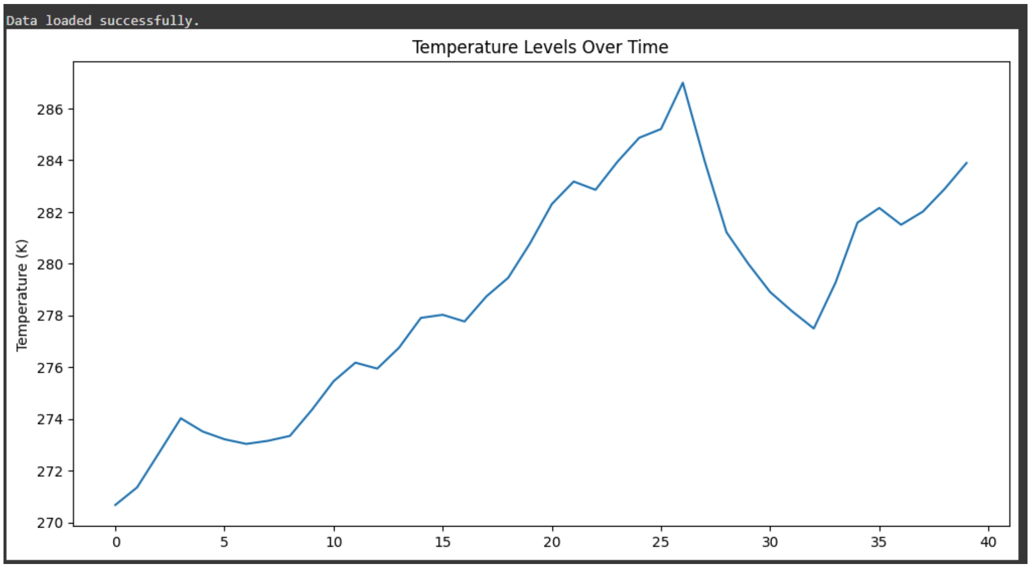



**Figure:Temperature Levels over time**

**Code Snippet:**

```
from sklearn.preprocessing import MinMaxScaler
import pickle
import numpy as np

# Example training data
data = np.array([
    [298.15, 65, 1013, 3.1, 10],
    [297.85, 70, 1012, 3.6, 20],
    [296.55, 75, 1011, 2.9, 15],
    [295.15, 80, 1010, 3.0, 25],
    [294.85, 78, 1009, 2.5, 30],
])

# Scaling the data
scaler = MinMaxScaler()
scaler.fit(data)

# Save scaler
with open("scaler.pkl", "wb") as f:
    pickle.dump(scaler, f)

print("Scaler retrained and saved successfully.")
```

- **Screenshot:**

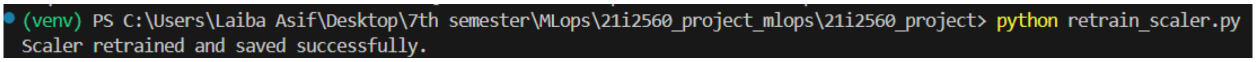
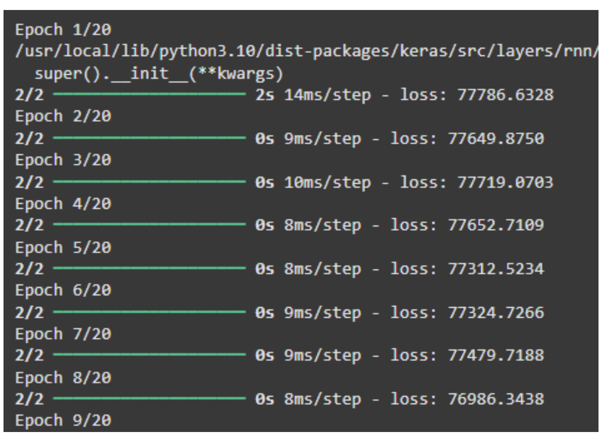



**Figure: "Scaler retrained and saved successfully."**

# 2. Model Development

- **Objective:** Build a time-series model using LSTMs to predict pollution trends.
- **Implementation:**
  - Constructed a deep learning model using TensorFlow/Keras with LSTM layers for sequential data processing.
  - Model trained on preprocessed features like temperature, humidity, pressure, wind speed, and cloud cover.
- **Key Model Architecture:**
  - Input Shape: `(None, 5, 5)` (5 time-steps with 5 features each).
  - LSTM Layers: Captures temporal patterns.
  - Fully Connected Layers: Produces final prediction.



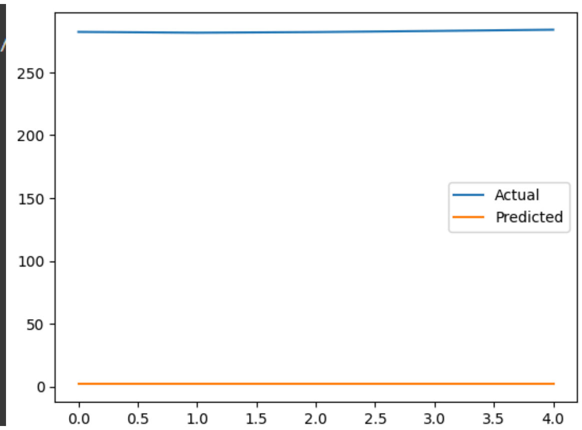
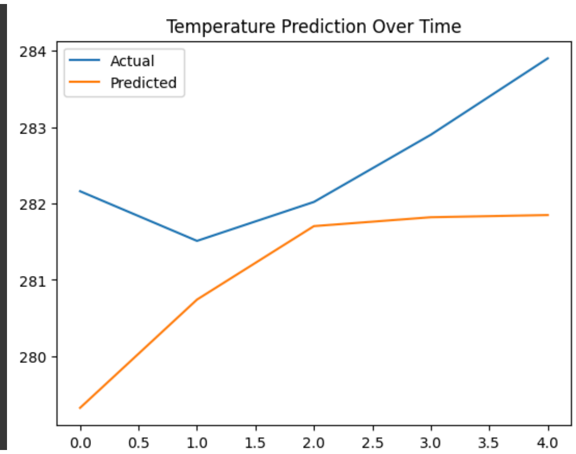




**Figure : running on epochs and calculating results**

More enhanced model :

```
Epoch 1/50
2/2 ━━━━━━━━━━━━━━━━━━━━ 4s 405ms/step - loss: 0.2765 - val_loss: 0.4160
Epoch 2/50
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 35ms/step - loss: 0.2479 - val_loss: 0.3546
Epoch 3/50
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 34ms/step - loss: 0.1857 - val_loss: 0.2934
Epoch 4/50
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 36ms/step - loss: 0.1713 - val_loss: 0.2317
Epoch 5/50
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 35ms/step - loss: 0.1167 - val_loss: 0.1707
Epoch 6/50
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 35ms/step - loss: 0.0975 - val_loss: 0.1123
Epoch 7/50
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 35ms/step - loss: 0.0638 - val_loss: 0.0614
Epoch 8/50
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 37ms/step - loss: 0.0282 - val_loss: 0.0242
Epoch 9/50
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step - loss: 0.0224 - val_loss: 0.0055
Epoch 10/50
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 36ms/step - loss: 0.0215 - val_loss: 0.0035
Epoch 11/50
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 37ms/step - loss: 0.0283 - val_loss: 0.0057
```

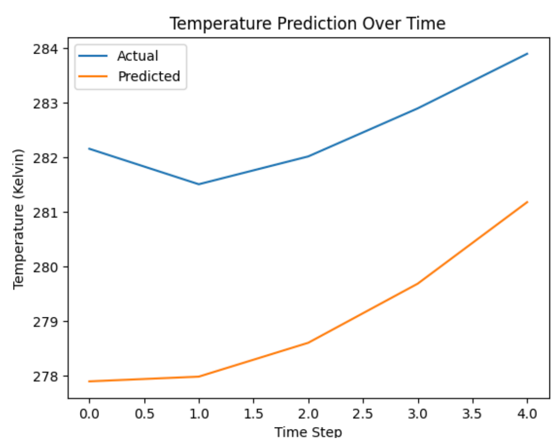



```
MAE: 1.4106220703124905
RMSE: 1.6798822810416667
```

**Figure : running on epochs , temperature predictions over time and calculating MAE and RMSE**

With More enhanced technique:

```
Epoch 1/50
/usr/local/lib/python3.10/dist-packages/keras/src/layers/rnn/rnn.py:204:
  super().__init__(**kwargs)
2/2 ━━━━━━━━━━━━━━━━━━━━ 5s 531ms/step - loss: 0.3162 - val_loss: 0.4124
Epoch 2/50
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 36ms/step - loss: 0.2338 - val_loss: 0.3152
Epoch 3/50
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 37ms/step - loss: 0.1422 - val_loss: 0.2227
Epoch 4/50
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 36ms/step - loss: 0.1052 - val_loss: 0.1370
Epoch 5/50
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 39ms/step - loss: 0.0445 - val_loss: 0.0665
Epoch 6/50
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 40ms/step - loss: 0.0160 - val_loss: 0.0204
Epoch 7/50
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 36ms/step - loss: 0.0188 - val_loss: 0.0041
Epoch 8/50
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 42ms/step - loss: 0.0410 - val_loss: 0.0028
```




```
MAE: 3.4255081195831165
RMSE: 3.461907069164531
```

**Figure : running on epochs , temperature predictions over time and calculating MAE and RMSE**

More improved model:

```
Epoch 1/100
/usr/local/lib/python3.10/dist-packages/keras/src/layers/rnn/rnn.py:204: UserWarning: Do not pass
  super().__init__(**kwargs)
2/2 ━━━━━━━━━━━━━━━━━━━━ 5s 471ms/step - loss: 0.2951 - val_loss: 0.0865 - learning_rate: 0.0100
Epoch 2/100
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 56ms/step - loss: 0.1134 - val_loss: 0.1072 - learning_rate: 0.0100
Epoch 3/100
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 59ms/step - loss: 0.1362 - val_loss: 0.0379 - learning_rate: 0.0100
Epoch 4/100
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 62ms/step - loss: 0.0474 - val_loss: 0.0424 - learning_rate: 0.0100
Epoch 5/100
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 50ms/step - loss: 0.0476 - val_loss: 0.0197 - learning_rate: 0.0100
Epoch 6/100
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 70ms/step - loss: 0.0282 - val_loss: 0.0182 - learning_rate: 0.0100
Epoch 7/100
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 48ms/step - loss: 0.0274 - val_loss: 0.0117 - learning_rate: 0.0100
Epoch 8/100
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 60ms/step - loss: 0.0154 - val_loss: 0.0127 - learning_rate: 0.0100
Epoch 9/100
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 44ms/step - loss: 0.0263 - val_loss: 0.0118 - learning_rate: 0.0100
Epoch 10/100
```

```
MAE: 0.96
RMSE: 1.25
```

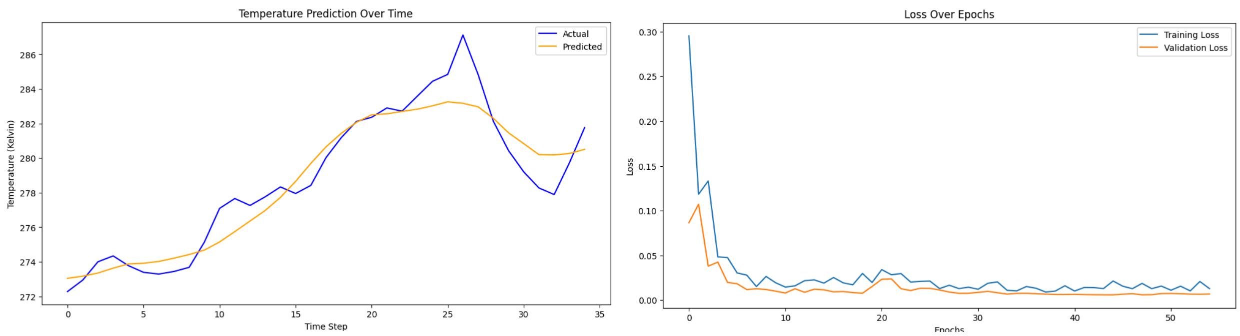



**Figure : running on epochs , temperature predictions over time , loss over epochs and calculating MAE and RMSE**

More enhancements:

```
Epoch 1/50
/usr/local/lib/python3.10/dist-packages/keras/src/layers/rnn/rnn.py:204: UserWarning: Do not pass a
  super().__init__(**kwargs)
5/5 ━━━━━━━━━━━━━━━━━━━━ 4s 117ms/step - loss: 0.3578 - val_loss: 0.1182 - learning_rate: 0.0010
Epoch 2/50
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0984 - val_loss: 0.1628 - learning_rate: 0.0010
Epoch 3/50
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - loss: 0.1282 - val_loss: 0.1294 - learning_rate: 0.0010
Epoch 4/50
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0893 - val_loss: 0.1049 - learning_rate: 0.0010
Epoch 5/50
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - loss: 0.0827 - val_loss: 0.1043 - learning_rate: 0.0010
Epoch 6/50
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - loss: 0.0954 - val_loss: 0.1083 - learning_rate: 0.0010
Epoch 7/50
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - loss: 0.0773 - val_loss: 0.1174 - learning_rate: 0.0010
Epoch 8/50
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - loss: 0.0827 - val_loss: 0.1133 - learning_rate: 0.0010
Epoch 9/50
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 0.0710 - val_loss: 0.1133 - learning_rate: 0.0010
```

```
MAE: 2.8779574614609102
RMSE: 3.161628594035194
R² Score: -0.04227195830620056
```

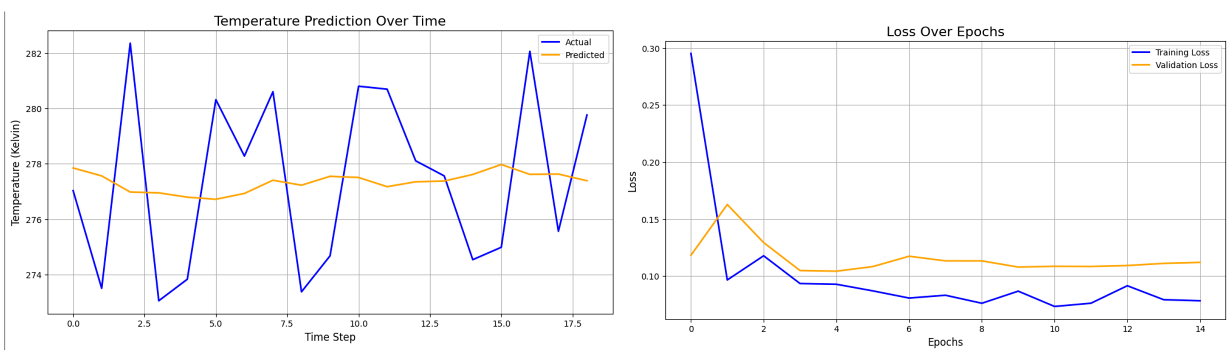



**Figure : running on epochs , temperature predictions over time , loss over epochs and calculating MAE and RMSE**

# 3. Training with MLflow

- **Objective:** Track experiments, parameters, and metrics (e.g., RMSE, MAE).
- **Implementation:**
  - Logged model architecture, hyperparameters, and metrics.
  - Stored model artifacts (e.g., `best_model.h5`) in MLflow for reproducibility.

```
Model and metrics logged successfully in MLflow.
```

**Figure: model and metrics Successfully logged**

# 4. Hyperparameter Tuning

- **Objective:** Optimize the model using grid search/random search for hyperparameters like batch size, learning rate, and LSTM units.
- **Tools Used:** `scikit-learn`'s `GridSearchCV` or TensorFlow's KerasTuner.
- 
  ```
  Epoch 1/20
  5/5 ━━━━━━━━━━━━━━━━━━━━ 5s 115ms/step - loss: 0.4703 - val_loss: 76893.6328 - learning_rate: 0.0010
  Epoch 2/20
  5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.2038 - val_loss: 76780.2734 - learning_rate: 0.0010
  Epoch 3/20
  5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 0.1113 - val_loss: 76654.5234 - learning_rate: 0.0010
  Epoch 4/20
  5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 0.0780 - val_loss: 76606.3047 - learning_rate: 0.0010
  Epoch 5/20
  5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - loss: 0.1026 - val_loss: 76651.8906 - learning_rate: 0.0010
  Epoch 6/20
  5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - loss: 0.0714 - val_loss: 76700.3828 - learning_rate: 0.0010
  Epoch 7/20
  5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 0.0979 - val_loss: 76719.2656 - learning_rate: 0.0010
  Epoch 8/20
  5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - loss: 0.0858 - val_loss: 76715.7422 - learning_rate: 5.0000e-04
  Epoch 9/20
  5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step - loss: 0.0862 - val_loss: 76700.6016 - learning_rate: 5.0000e-04
  1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 307ms/step
  RMSE for current parameters: 2709.747985943967
  Testing parameters: {'batch_size': 16, 'dropout_rate': 0.2, 'learning_rate': 0.001, 'lstm_units_1': 64, 'lstm_units_2': 64}
  Epoch 1/20
  ```
- 
  ```
  RMSE for current parameters: 2709.30876325704
  Testing parameters: {'batch_size': 16, 'dropout_rate': 0.2, 'learning_rate': 0.001, 'lstm_units_1': 128, 'lstm_units_2': 32}
  ```
- `RMSE for current parameters: 2709.8257824037523`
- `RMSE for current parameters: 2709.689091314318`
- `RMSE for current parameters: 2710.3992926616734`
- `RMSE for current parameters: 2710.0425064399783`
- `RMSE for current parameters: 2709.9928928700747`
- `RMSE for current parameters: 2709.838641812265`
- `RMSE for current parameters: 2709.98697012328`
- `RMSE for current parameters: 2709.959577760837`
- `RMSE for current parameters: 2709.6181698813202`
- `RMSE for current parameters: 2709.376216268125`
- `RMSE for current parameters: 2710.1959587289552`
- `RMSE for current parameters: 2710.310372449279`
- `RMSE for current parameters: 2710.1230028622003`
- `RMSE for current parameters: 2709.814105713543`
- `RMSE for current parameters: 2709.776405617742`
- `RMSE for current parameters: 2709.048757431695`

`Continue…`

```
Best Parameters: {'batch_size': 32, 'dropout_rate': 0.2, 'learning_rate': 0.001, 'lstm_units_1': 64, 'lstm_units_2': 64}
Best RMSE: 2709.048757431695
```

**Figure: Running on epochs , calculating RMSE ,Testing parameters and best values**

More Improved results:

```
[I 2024-12-14 12:22:43,025] Trial 0 finished with value: 0.08583692461252213 and parameters: {'lstm_units_1': 55, 'lstm_units_2': 116, 'dropout_rate': 0.4524489014240488, 'learning_rate': 0.0009073420868038716}. Best is trial 0 with value: 0.08583692461252213.
[I 2024-12-14 12:23:20,584] Trial 1 finished with value: 0.08570662140846252 and parameters: {'lstm_units_1': 79, 'lstm_units_2': 83, 'dropout_rate': 0.16720108445257847, 'learning_rate': 0.0014216920029958904}. Best is trial 1 with value: 0.08570662140846252.
[I 2024-12-14 12:23:50,548] Trial 2 finished with value: 0.0847504511475563 and parameters: {'lstm_units_1': 81, 'lstm_units_2': 52, 'dropout_rate': 0.15815365495971695, 'learning_rate': 0.009560556521124584}. Best is trial 2 with value: 0.0847504511475563.
[I 2024-12-14 12:24:22,643] Trial 3 finished with value: 0.08538337051868439 and parameters: {'lstm_units_1': 83, 'lstm_units_2': 63, 'dropout_rate': 0.10445079351985186, 'learning_rate': 0.0028359906484589697}. Best is trial 2 with value: 0.0847504511475563.
[I 2024-12-14 12:25:06,645] Trial 4 finished with value: 0.08574311435222626 and parameters: {'lstm_units_1': 120, 'lstm_units_2': 90, 'dropout_rate': 0.27053459074184916, 'learning_rate': 0.001809274485526637}. Best is trial 2 with value: 0.0847504511475563.
[I 2024-12-14 12:25:44,666] Trial 5 finished with value: 0.08793358504772186 and parameters: {'lstm_units_1': 99, 'lstm_units_2': 79, 'dropout_rate': 0.22934987371333235, 'learning_rate': 0.00010952454069764523}. Best is trial 2 with value: 0.0847504511475563.
[I 2024-12-14 12:26:06,633] Trial 6 finished with value: 0.08580078184604645 and parameters: {'lstm_units_1': 54, 'lstm_units_2': 55, 'dropout_rate': 0.3259836578006534, 'learning_rate': 0.002350355214044693}. Best is trial 2 with value: 0.0847504511475563.
[I 2024-12-14 12:26:37,604] Trial 7 finished with value: 0.08557024598121643 and parameters: {'lstm_units_1': 63, 'lstm_units_2': 100, 'dropout_rate': 0.25218544424843997, 'learning_rate': 0.002682115692465976}. Best is trial 2 with value: 0.0847504511475563.
[I 2024-12-14 12:27:12,819] Trial 8 finished with value: 0.0870627611875534 and parameters: {'lstm_units_1': 114, 'lstm_units_2': 52, 'dropout_rate': 0.43372076047104136, 'learning_rate': 0.0002602935428736705}. Best is trial 2 with value: 0.0847504511475563.
[I 2024-12-14 12:27:48,765] Trial 9 finished with value: 0.08626958727836609 and parameters: {'lstm_units_1': 107, 'lstm_units_2': 62, 'dropout_rate': 0.3968075753130059, 'learning_rate': 0.0011777818580927329}. Best is trial 2 with value: 0.0847504511475563.
[I 2024-12-14 12:28:10,558] Trial 10 finished with value: 0.08535519987344742 and parameters: {'lstm_units_1': 36, 'lstm_units_2': 39, 'dropout_rate': 0.10584572922553961, 'learning_rate': 0.00988201125796824}. Best is trial 2 with value: 0.0847504511475563.
[I 2024-12-14 12:28:32,518] Trial 11 finished with value: 0.08501172065734863 and parameters: {'lstm_units_1': 37, 'lstm_units_2': 33, 'dropout_rate': 0.10210254810334737, 'learning_rate': 0.00968561089477721}. Best is trial 2 with value: 0.0847504511475563.
[I 2024-12-14 12:28:53,030] Trial 12 finished with value: 0.0849778950214386 and parameters: {'lstm_units_1': 33, 'lstm_units_2': 32, 'dropout_rate': 0.17662769695474678, 'learning_rate': 0.00963664844055903}. Best is trial 2 with value: 0.0847504511475563.
[I 2024-12-14 12:29:23,991] Trial 13 finished with value: 0.08518888056278229 and parameters: {'lstm_units_1': 86, 'lstm_units_2': 44, 'dropout_rate': 0.18655993641204516, 'learning_rate': 0.0052370156839765414}. Best is trial 2 with value: 0.0847504511475563.
[I 2024-12-14 12:29:47,388] Trial 14 finished with value: 0.0851936936378479 and parameters: {'lstm_units_1': 69, 'lstm_units_2': 32, 'dropout_rate': 0.1768690887168179, 'learning_rate': 0.005119151041134667}. Best is trial 2 with value: 0.0847504511475563.
[I 2024-12-14 12:30:12,295] Trial 15 finished with value: 0.08687373250722885 and parameters: {'lstm_units_1': 92, 'lstm_units_2': 72, 'dropout_rate': 0.33330375917478716, 'learning_rate': 0.0006232244974401642}. Best is trial 2 with value: 0.0847504511475563.
[I 2024-12-14 12:30:44,674] Trial 16 finished with value: 0.0847802460193634 and parameters: {'lstm_units_1': 128, 'lstm_units_2': 45, 'dropout_rate': 0.20106992327214981, 'learning_rate': 0.004870142435586339}. Best is trial 2 with value: 0.0847504511475563.
[I 2024-12-14 12:31:27,240] Trial 17 finished with value: 0.0847490206360817 and parameters: {'lstm_units_1': 128, 'lstm_units_2': 49, 'dropout_rate': 0.21992318073059583, 'learning_rate': 0.005015464889707785}. Best is trial 17 with value: 0.0847490206360817.
```

```
Best Parameters: {'lstm_units_1': 117, 'lstm_units_2': 44, 'dropout_rate': 0.21756626371424903, 'learning_rate': 0.006067991935036522}
```

```
RMSE: 2.924195069456319
```

**Figure: Running on epochs , calculating RMSE ,Testing parameters and best values**

# 5. Model Evaluation

- **Objective:** Compare models using metrics like RMSE and choose the best one.
- **Implementation:**
  - Evaluated various models using a test dataset.
  - Visualized metric trends using MLflow plots to select the optimal model.

```
[I 2024-12-14 12:51:33,904] Trial 0 finished with value: 0.0897127091884613 and parameters: {'lstm_units_1': 112, 'lstm_units_2': 80, 'dropout_rate': 0.42226221360091454, 'learning_rate': 0.0024949960638954234}. Best is trial 0 with value: 0.0897127091884613.
[I 2024-12-14 12:51:58,741] Trial 1 finished with value: 0.08965759724378586 and parameters: {'lstm_units_1': 124, 'lstm_units_2': 97, 'dropout_rate': 0.10599899838566006, 'learning_rate': 0.0069545771565814206}. Best is trial 1 with value: 0.08965759724378586.
[I 2024-12-14 12:52:14,502] Trial 2 finished with value: 0.08967562764883041 and parameters: {'lstm_units_1': 100, 'lstm_units_2': 88, 'dropout_rate': 0.4380783300298676, 'learning_rate': 0.0014326088593537647}. Best is trial 1 with value: 0.08965759724378586.
[I 2024-12-14 12:52:30,210] Trial 3 finished with value: 0.08965501934289932 and parameters: {'lstm_units_1': 47, 'lstm_units_2': 50, 'dropout_rate': 0.28086531157264144, 'learning_rate': 0.0015997370009550366}. Best is trial 3 with value: 0.08965501934289932.
[I 2024-12-14 12:52:57,354] Trial 4 finished with value: 0.08950330317020416 and parameters: {'lstm_units_1': 114, 'lstm_units_2': 39, 'dropout_rate': 0.31136772312381883, 'learning_rate': 0.0002788684061083907}. Best is trial 4 with value: 0.08950330317020416.
[I 2024-12-14 12:53:24,896] Trial 5 finished with value: 0.089775450527668 and parameters: {'lstm_units_1': 48, 'lstm_units_2': 113, 'dropout_rate': 0.259349757348447, 'learning_rate': 0.0002595431601460397}. Best is trial 4 with value: 0.08950330317020416.
[I 2024-12-14 12:53:43,529] Trial 6 finished with value: 0.08966276049613953 and parameters: {'lstm_units_1': 43, 'lstm_units_2': 52, 'dropout_rate': 0.11259322081509487, 'learning_rate': 0.0017609692620880928}. Best is trial 4 with value: 0.08950330317020416.
[I 2024-12-14 12:54:01,028] Trial 7 finished with value: 0.0892115905880928 and parameters: {'lstm_units_1': 36, 'lstm_units_2': 101, 'dropout_rate': 0.4459667379446738, 'learning_rate': 0.0003143830190116626}. Best is trial 7 with value: 0.0892115905880928.
[I 2024-12-14 12:54:25,123] Trial 8 finished with value: 0.0896175354719162 and parameters: {'lstm_units_1': 125, 'lstm_units_2': 61, 'dropout_rate': 0.18749102240259796, 'learning_rate': 0.000523440454347298}. Best is trial 7 with value: 0.0892115905880928.
[I 2024-12-14 12:54:40,935] Trial 9 finished with value: 0.08965541422367096 and parameters: {'lstm_units_1': 62, 'lstm_units_2': 64, 'dropout_rate': 0.46620782164595176, 'learning_rate': 0.009940284871030535}. Best is trial 7 with value: 0.0892115905880928.
[I 2024-12-14 12:55:02,927] Trial 10 finished with value: 0.08925247192382812 and parameters: {'lstm_units_1': 77, 'lstm_units_2': 128, 'dropout_rate': 0.368621334056183, 'learning_rate': 0.00010347226735170305}. Best is trial 7 with value: 0.0892115905880928.
[I 2024-12-14 12:55:24,845] Trial 11 finished with value: 0.08941492438316345 and parameters: {'lstm_units_1': 83, 'lstm_units_2': 127, 'dropout_rate': 0.3759524601805126, 'learning_rate': 0.00010202486261859748}. Best is trial 7 with value: 0.0892115905880928.
[I 2024-12-14 12:55:49,664] Trial 12 finished with value: 0.08860466629266739 and parameters: {'lstm_units_1': 71, 'lstm_units_2': 111, 'dropout_rate': 0.3712449646338149, 'learning_rate': 0.000129032152755044}. Best is trial 12 with value: 0.08860466629266739.
[I 2024-12-14 12:56:05,325] Trial 13 finished with value: 0.08984867483377457 and parameters: {'lstm_units_1': 32, 'lstm_units_2': 102, 'dropout_rate': 0.36948329125423385, 'learning_rate': 0.0002563060429178549}. Best is trial 12 with value: 0.08860466629266739.
[I 2024-12-14 12:56:27,206] Trial 14 finished with value: 0.08983374387025833 and parameters: {'lstm_units_1': 67, 'lstm_units_2': 110, 'dropout_rate': 0.49827047215964204, 'learning_rate': 0.0005904340080896142}. Best is trial 12 with value: 0.08860466629266739.
```

```
Best Parameters: {'lstm_units_1': 71, 'lstm_units_2': 111, 'dropout_rate': 0.3712449646338149, 'learning_rate': 0.000129032152755044}
```

```
RMSE: 3.25438929005659, MAE: 2.809783153116037, MAPE: 1.0152649945319194%
```

**Figure: Running on epochs , calculating RMSE ,MAE , MAPE , Testing parameters and best values**

# 6. Deployment

- **Objective:** Deploy the selected model as an API using Flask.
- **Implementation:**
  1. Developed a Flask API (`api.py`) to serve predictions.

2. Used the saved model (`best_model.h5`) and scaler (`scaler.pkl`) to preprocess inputs and make predictions.

**Code Snippet:**

```
from flask import Flask, request, jsonify
from tensorflow.keras.models import load_model
import numpy as np
import pickle

app = Flask(__name__)

# Load model and scaler
model = load_model("best_model.h5")
with open("scaler.pkl", "rb") as f:
    scaler = pickle.load(f)

@app.route("/predict", methods=["POST"])
def predict():
    data = request.get_json()
    features = [
        [item["main"]["temp"], item["main"]["humidity"],
item["main"]["pressure"],
         item["wind"]["speed"], item.get("clouds", {}).get("all", 0)]
        for item in data["list"]
    ]
    features = scaler.transform(np.array(features))
    reshaped_input = features.reshape(1, 5, 5)  # Adjust as per
model's input
    prediction = model.predict(reshaped_input)
    return jsonify({"prediction": float(prediction[0, 0])})

if __name__ == "__main__":
    app.run(debug=True)
```

- **Screenshots to Add:**
    1. 

**Figure: API running on `http://127.0.0.1:5000`.**

```
Model loaded successfully with input shape: (None, 5, 5)
Scaler loaded successfully with expected features: 5
WARNING:werkzeug: * Debugger is active!
INFO:werkzeug: * Debugger PIN: 336-274-287
Extracted features: [[ 298.15    65.    1013.       3.1    10.  ]
 [ 297.85    70.    1012.       3.6    20.  ]
 [ 296.55    75.    1011.       2.9    15.  ]
 [ 295.15    80.    1010.       3.     25.  ]
 [ 294.85    78.    1009.       2.5    30.  ]]
Feature array shape: (5, 5)
Scaled features shape: (5, 5)
Reshaped input shape: (1, 5, 5)
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 642ms/step
Raw prediction: [[0.06856496]]
```

**Figure: Test API results showing the prediction for pollution levels.**

**Testing the API**

- **Objective:** Verify API functionality using `test_api.py`.
- **Implementation:**
    - Sent a POST request with test data (temperature, humidity, pressure, wind speed, cloud cover).
    - Received a prediction response with pollution trends.

**Code Snippet:**

```
import requests

url = "http://127.0.0.1:5000/predict"
input_data = {
    "list": [
        {"main": {"temp": 298.15, "humidity": 65, "pressure": 1013},
"wind": {"speed": 3.1}, "clouds": {"all": 10}},
        {"main": {"temp": 297.85, "humidity": 70, "pressure": 1012},
"wind": {"speed": 3.6}, "clouds": {"all": 20}},
        {"main": {"temp": 296.55, "humidity": 75, "pressure": 1011},
"wind": {"speed": 2.9}, "clouds": {"all": 15}},
```

```
        {"main": {"temp": 295.15, "humidity": 80, "pressure": 1010},
"wind": {"speed": 3.0}, "clouds": {"all": 25}},
        {"main": {"temp": 294.85, "humidity": 78, "pressure": 1009},
"wind": {"speed": 2.5}, "clouds": {"all": 30}},
    ]
}

response = requests.post(url, json=input_data)
print(response.json())
```

- **Screenshot:**

```
(venv) PS C:\Users\Laiba Asif\Desktop\7th semester\MLops\21i2560_project_mlops\21i2560_project> python test_api.py
>>
Prediction response: {'prediction': 0.06856495887041092}
```

**Figure:Prediction from `test_api.py`.**

# Task 3: Monitoring and Live Testing

### Objective

To test the pipeline with live data and monitor the deployed system for real-time performance tracking, system optimization, and validation of the deployed model’s accuracy.

## 1. Set Up Monitoring

### Step 1: Install and Configure Prometheus

**Pull the Prometheus Docker Image:**

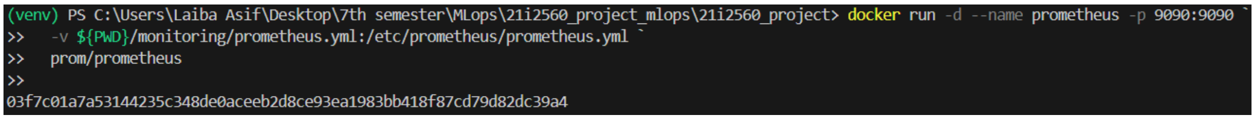
```
docker pull prom/prometheus
```

**Create a `prometheus.yml` Configuration File:** Create a `prometheus.yml` file in your project directory with the following content:

```
global:
  scrape_interval: 5s  # Scrape every 5 seconds
  evaluation_interval: 5s

scrape_configs:
  - job_name: "flask_api"
    static_configs:
      - targets:
          - "localhost:5000"  # Flask app metrics endpoint
```

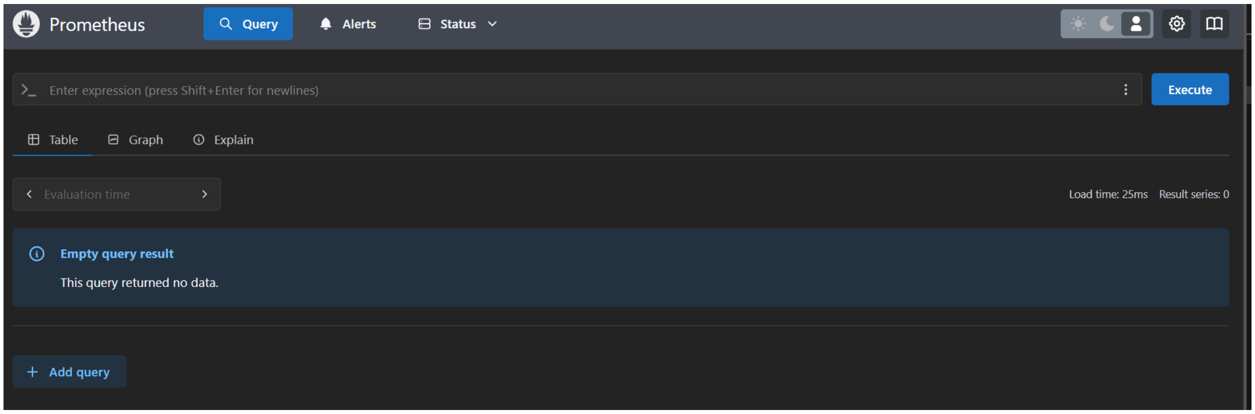
**Run Prometheus in Docker:**

```
docker run -d --name prometheus -p 9090:9090 \
  -v ${PWD}/prometheus.yml:/etc/prometheus/prometheus.yml \
  prom/prometheus
```



```
(venv) PS C:\Users\Laiba Asif\Desktop\7th semester\MLops\21i2560_project_mlops\21i2560_project> docker run -d --name prometheus -p 9090:9090 `
>>    -v ${PWD}/monitoring/prometheus.yml:/etc/prometheus/prometheus.yml `
>>    prom/prometheus
>>
03f7c01a7a53144235c348de0aceeb2d8ce93ea1983bb418f87cd79d82dc39a4
```

**Verify Prometheus Installation:** Open Prometheus in your browser:
**URL:** [http://localhost:9090](http://localhost:9090)



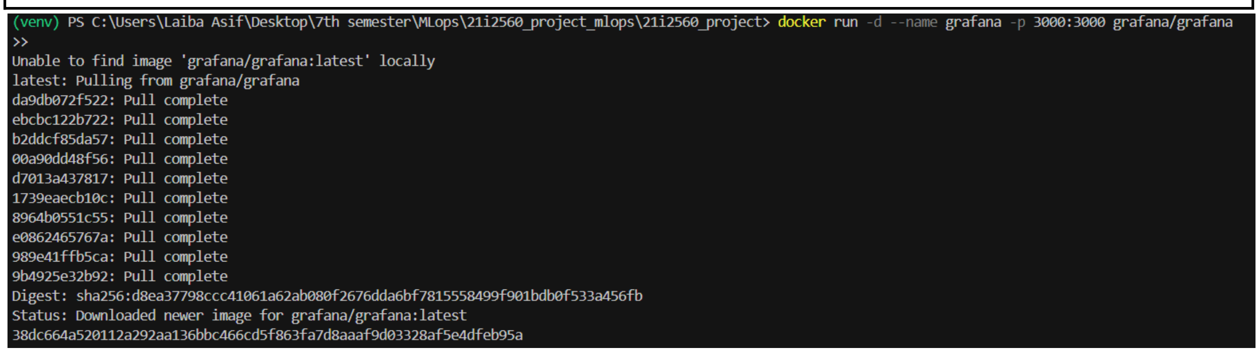

## Step 2: Install and Configure Grafana

### Pull the Grafana Docker Image:

```
docker pull grafana/grafana
```

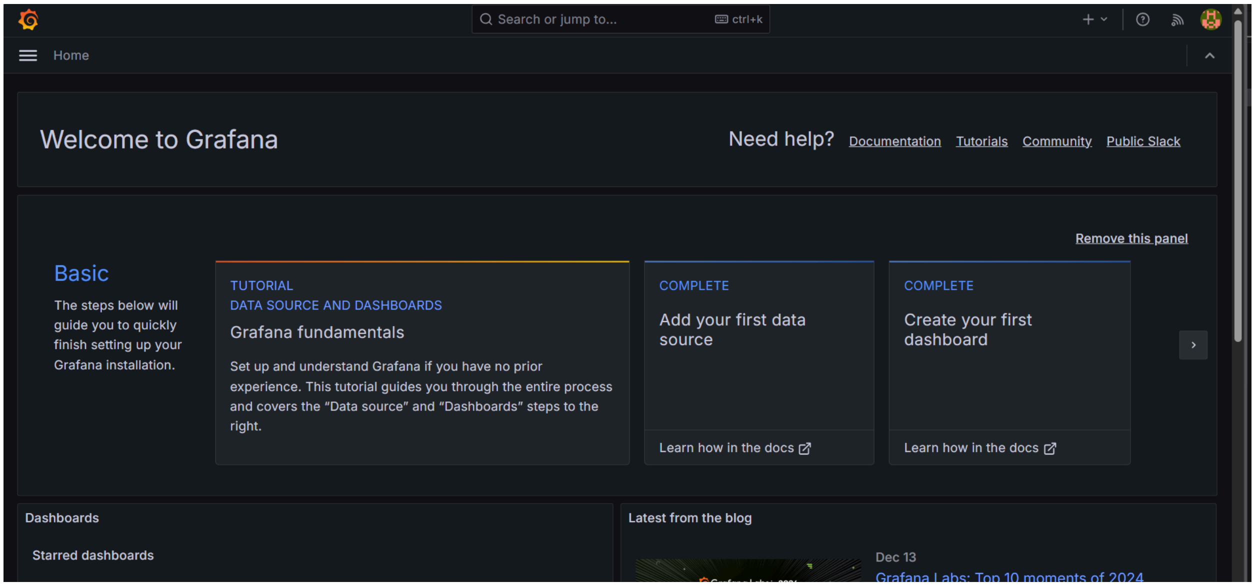
### Run Grafana in Docker:

```
docker run -d --name grafana -p 3000:3000 grafana/grafana
```

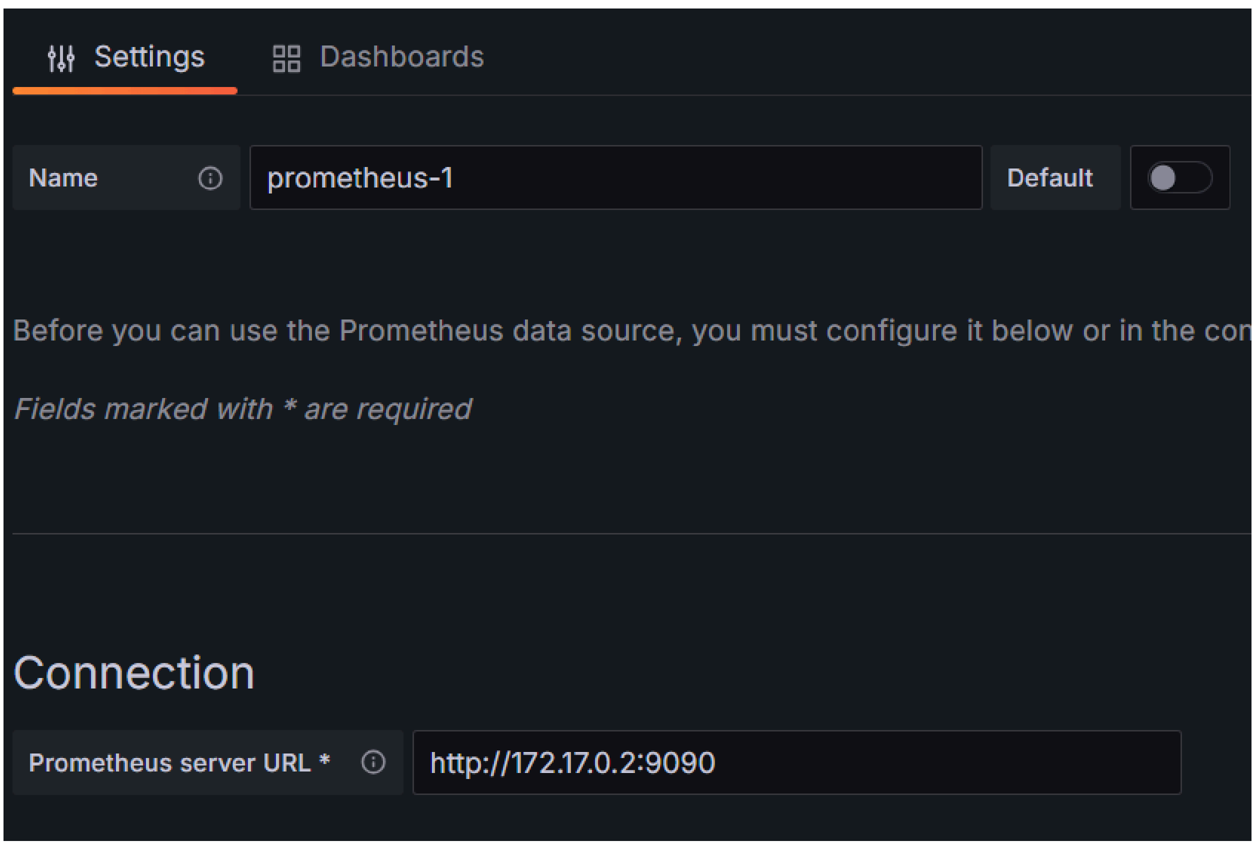


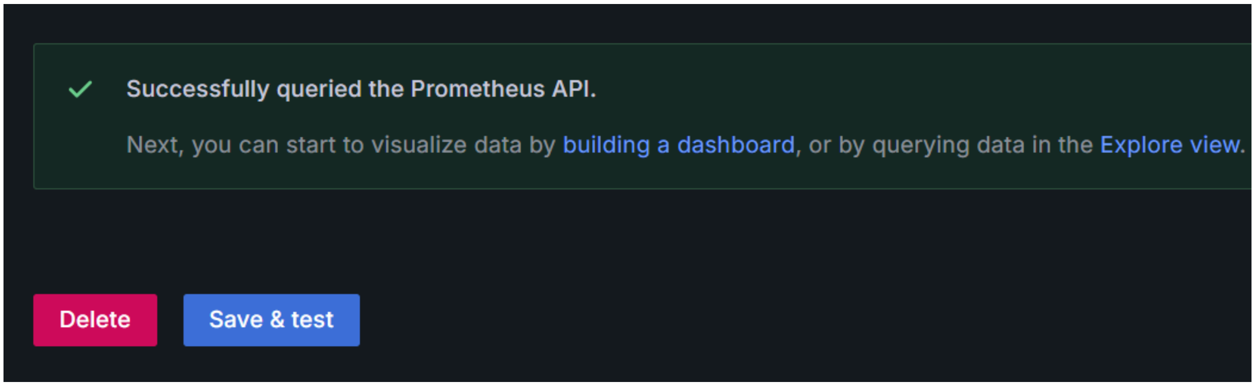

**Access Grafana Dashboard:** Open Grafana in your browser:

**URL:** [http://localhost:3000](http://localhost:3000)

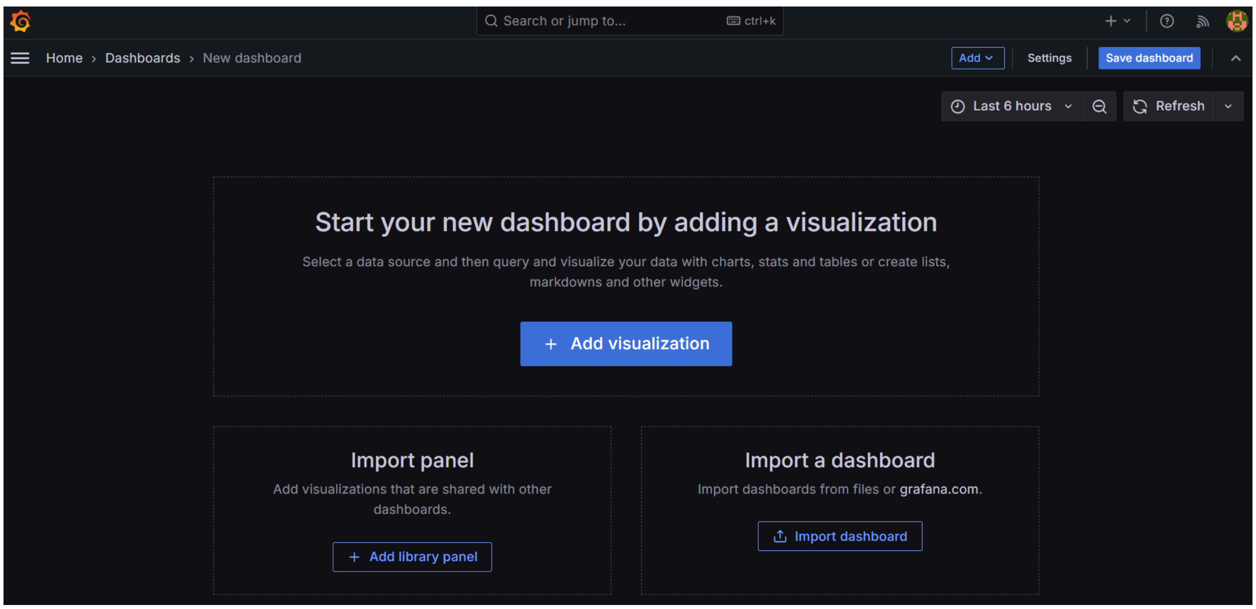



1. **Add Prometheus as a Data Source in Grafana:**
    - Log in with default credentials (Username: `admin`, Password: `admin`).
    - Navigate to **Configuration > Data Sources > Add Data Source**.
    - Select **Prometheus**.
    - Set the URL as `http://localhost:9090` and click **Save & Test**.



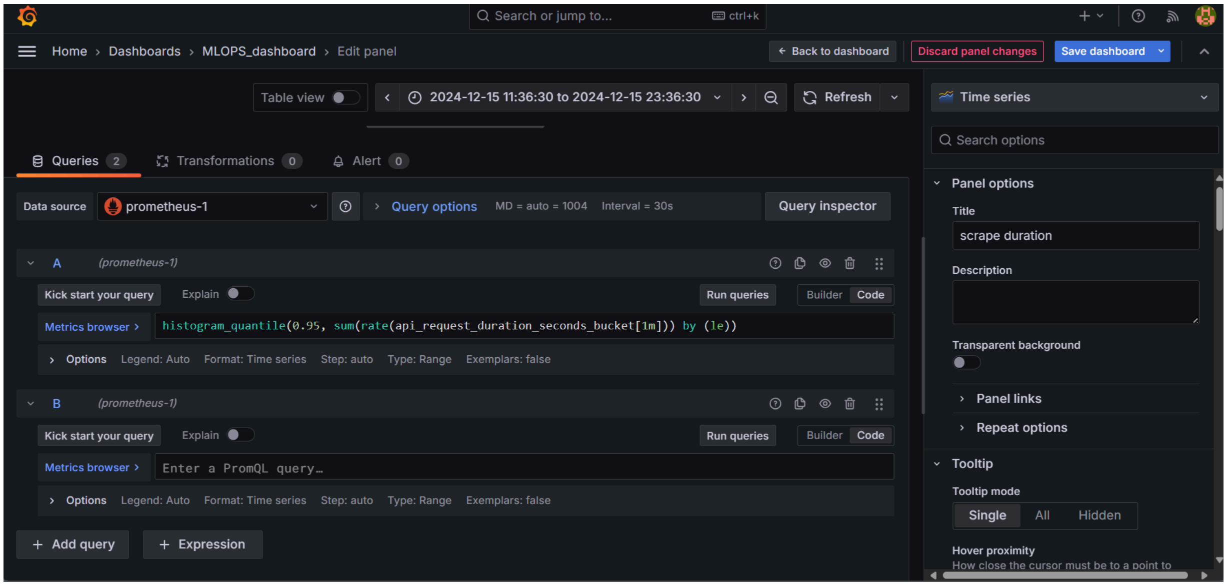



2. **Create a New Dashboard in Grafana:**
   - Go to **Dashboards > New Dashboard**.
   - Add visualizations to monitor metrics.



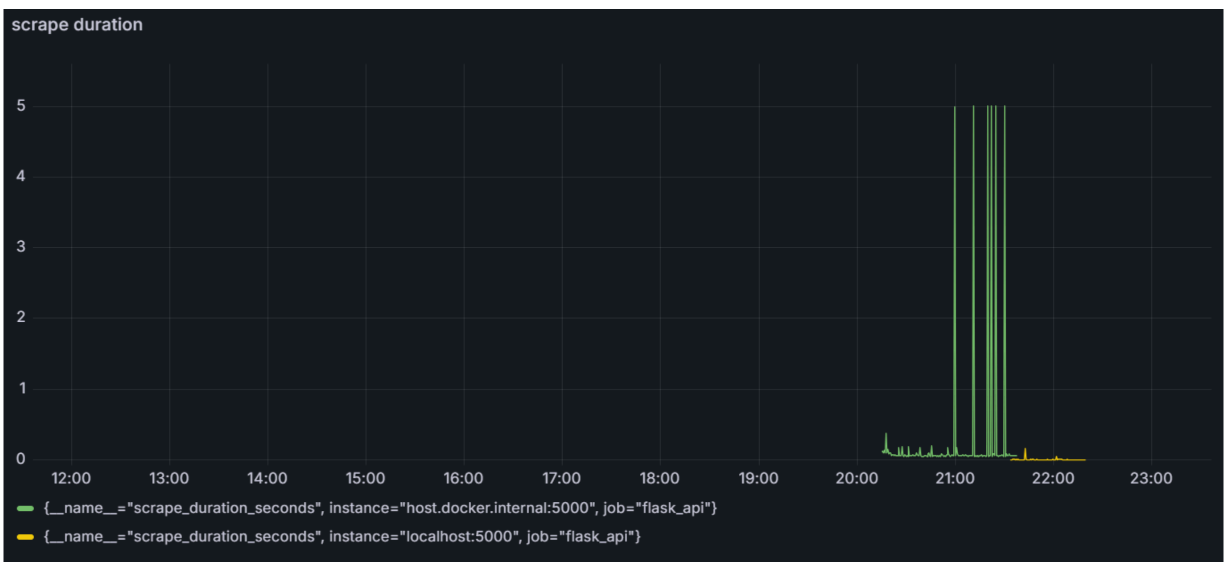

3. **Metrics to Visualize:**

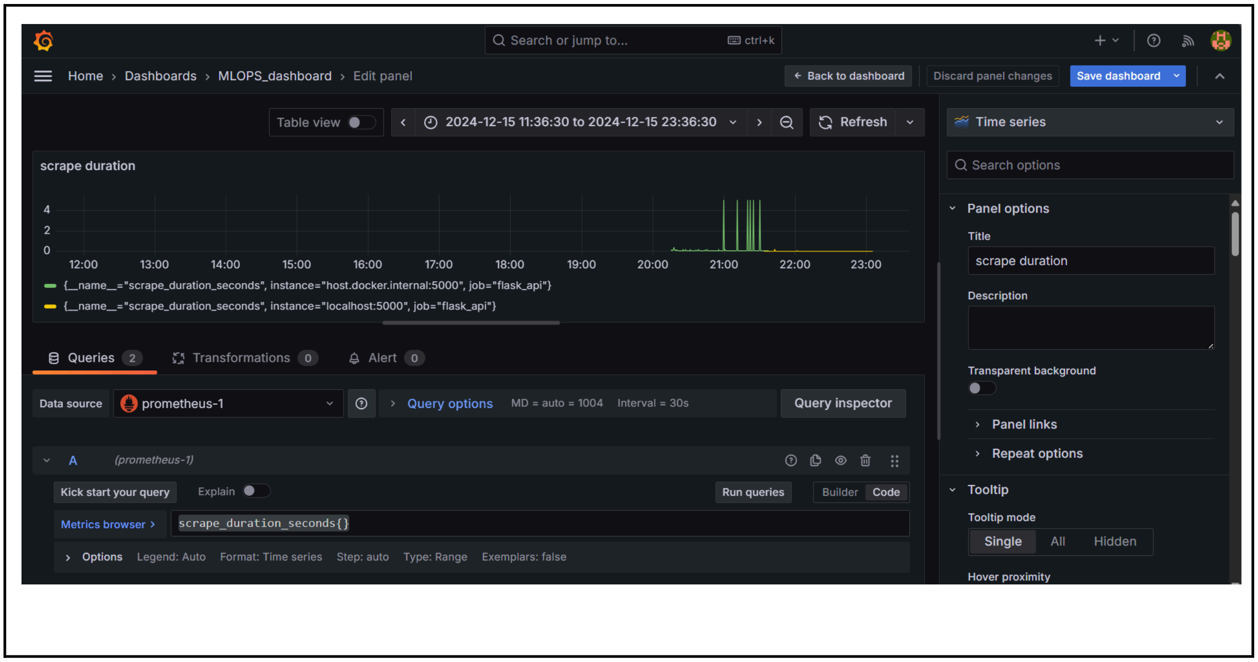
**API Latency:**
Query:




## Scrape Duration Second:

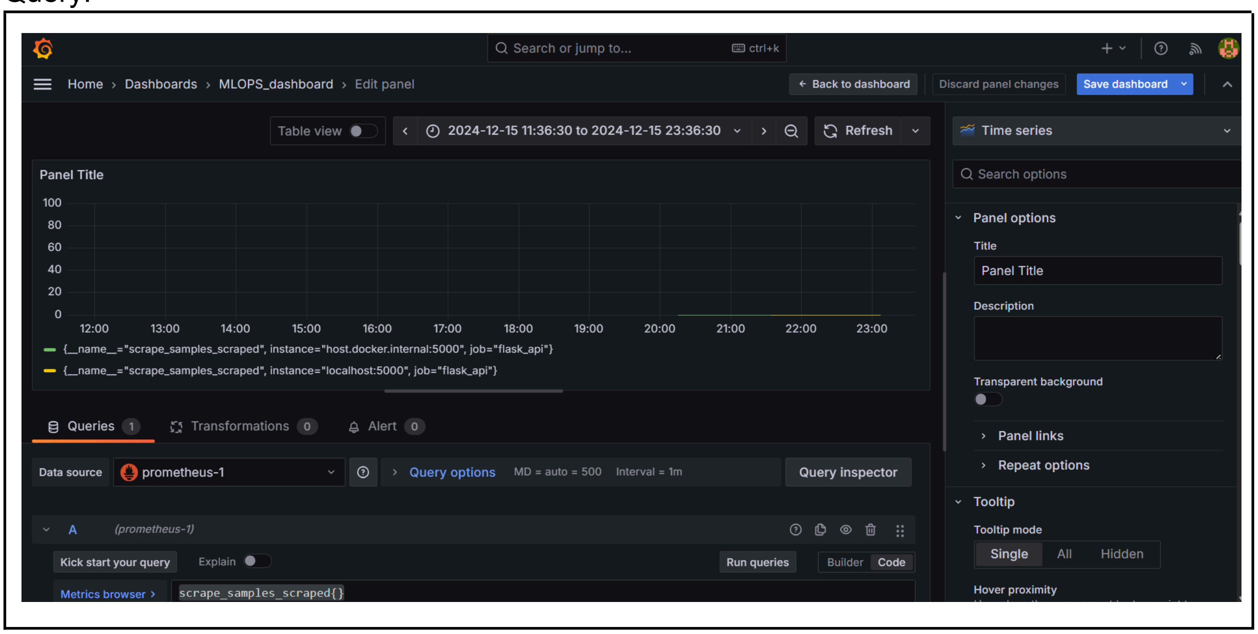
Query:





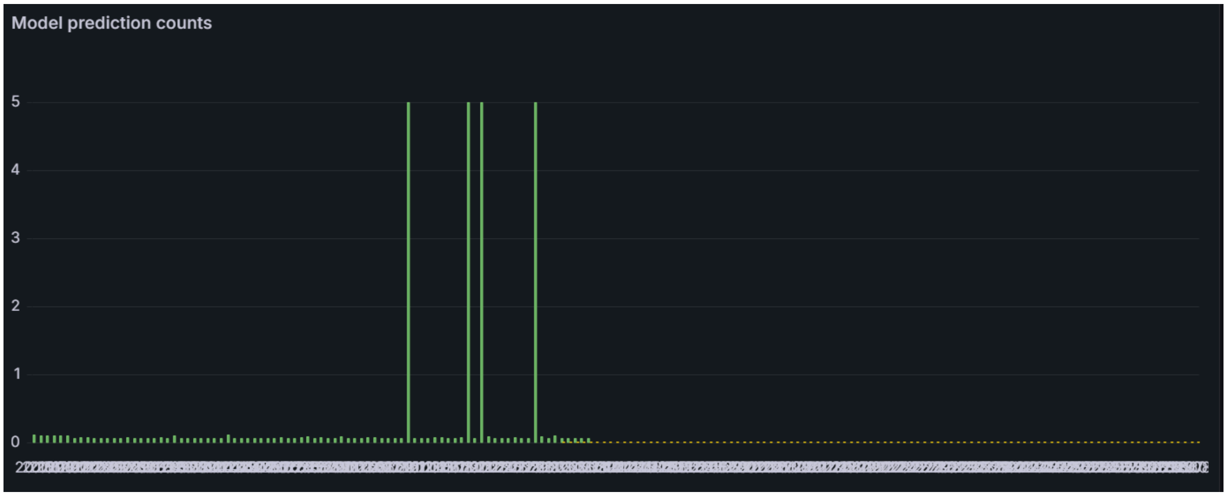

## Scrape Sampled Scraped

Query:



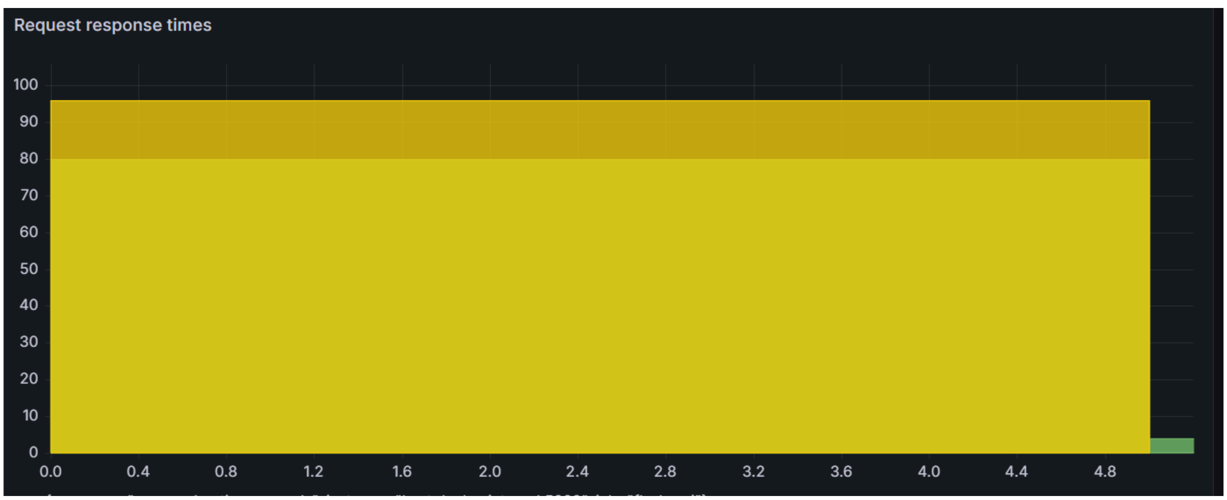

## Model prediction count:

Query:




**Request response times**

Query:




## Step 3: Integrate Prometheus with Flask Application

**Install `prometheus_flask_exporter`:** Add the dependency to `requirements.txt`:

`prometheus-flask-exporter`

Install it in your Python environment:

```
pip install -r requirements.txt
```

```
(venv) PS C:\Users\Laiba Asif\Desktop\7th semester\MLops\21i2560_project_mlops\21i2560_project> pip install -r requirements.txt
Requirement already satisfied: flask in c:\users\laiba asif\desktop\7th semester\mlops\21i2560_project_mlops\21i2560_project\venv\lib\site-packages (from -r
 requirements.txt (line 1)) (3.1.0)
Requirement already satisfied: prometheus-flask-exporter in c:\users\laiba asif\desktop\7th semester\mlops\21i2560_project_mlops\21i2560_project\venv\lib\si
te-packages (from -r requirements.txt (line 2)) (0.23.1)
Requirement already satisfied: tensorflow in c:\users\laiba asif\desktop\7th semester\mlops\21i2560_project_mlops\21i2560_project\venv\lib\site-packages (fr
om -r requirements.txt (line 3)) (2.18.0)
Requirement already satisfied: numpy in c:\users\laiba asif\desktop\7th semester\mlops\21i2560_project_mlops\21i2560_project\venv\lib\site-packages (from -r
 requirements.txt (line 4)) (2.0.2)
Requirement already satisfied: scikit-learn in c:\users\laiba asif\desktop\7th semester\mlops\21i2560_project_mlops\21i2560_project\venv\lib\site-packages (
from -r requirements.txt (line 5)) (1.5.2)
Requirement already satisfied: Werkzeug>=3.1 in c:\users\laiba asif\desktop\7th semester\mlops\21i2560_project_mlops\21i2560_project\venv\lib\site-packages
(from flask->-r requirements.txt (line 1)) (3.1.3)
Requirement already satisfied: Jinja2>=3.1.2 in c:\users\laiba asif\desktop\7th semester\mlops\21i2560_project_mlops\21i2560_project\venv\lib\site-packages
(from flask->-r requirements.txt (line 1)) (3.1.4)
```

**Update Flask Application (`api.py`):** Ensure the `/metrics` endpoint is exposed:

```
from prometheus_flask_exporter import PrometheusMetrics

app = Flask(__name__)

# Integrate Prometheus Metrics
metrics = PrometheusMetrics(app)
metrics.info('app_info', 'Application Info', version='1.0.0')
```

**Rebuild and Run the Flask Docker Container:**

```
docker build -t flask-api .
docker run -d --name flask-api -p 5000:5000 flask-api
```

**Verify Metrics Endpoint:** Test the `/metrics` endpoint:

```
curl http://localhost:5000/metrics
```

# 2. Test Predictions with Live Data

## Step 1: Fetch Live Data

Use a public weather API (e.g., OpenWeatherMap) to simulate live data ingestion. Example Request:

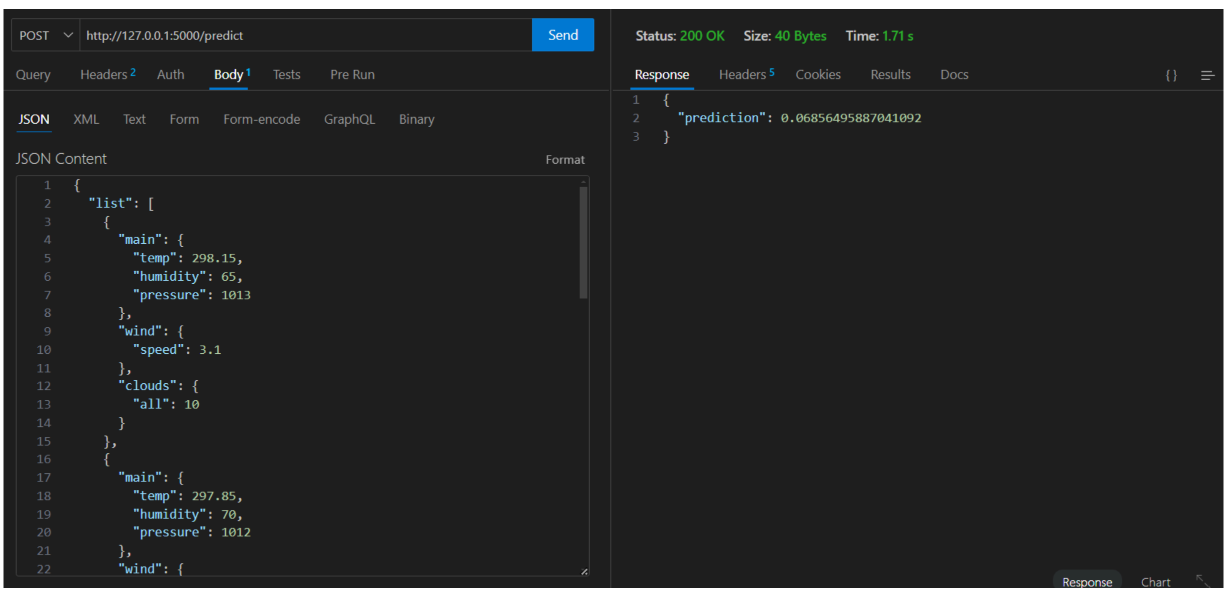
```
curl -X GET
"http://api.openweathermap.org/data/2.5/weather?q=London&appid=2fbc453
063630496c3ab531f5de7535f
"
```

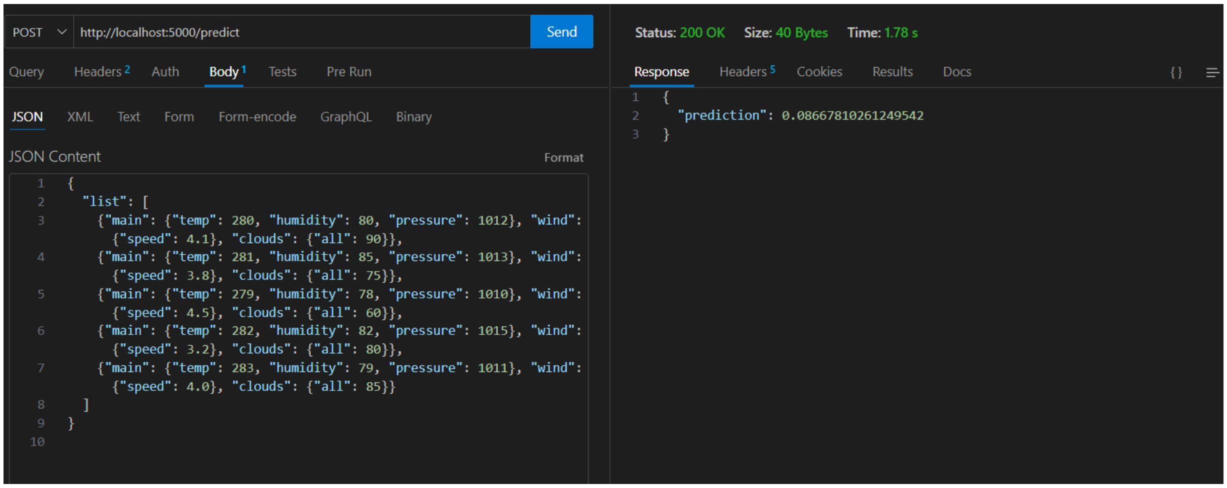
**Update Your Flask Application to Fetch Live Data:** Modify the `predict()` function to include live data fetching.

```
@app.route("/live_predict", methods=["GET"])
def live_predict():
    # Fetch live data from OpenWeatherMap API
    response =
requests.get("http://api.openweathermap.org/data/2.5/weather?q=London
&appid=YOUR_API_KEY")
    data = response.json()

    # Process and predict using the live data
```

### Step 2: Send Test Requests to the Flask API

Use tools like **Postman**, **Thunder Client**, or `curl` to test the `/predict` endpoint with JSON input.
Example JSON for POST Request:



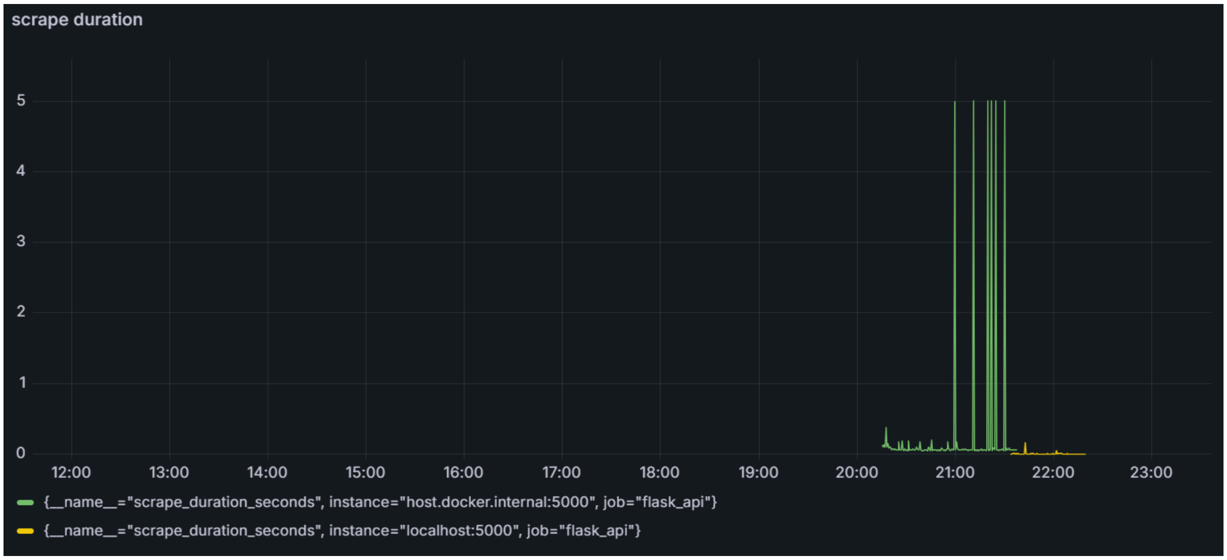



1. Validate the predictions from the model for accuracy.

# 3. Analyze and Optimize

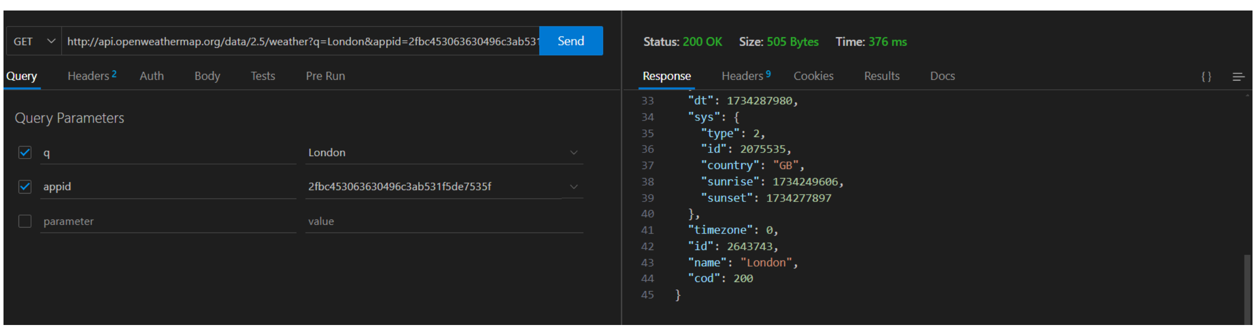
## Step 1: Analyze Performance in Grafana

1. Monitor key metrics in your Grafana dashboard:
   - API latency.
   - Number of model predictions.
   - Data ingestion rate.
2. Identify bottlenecks:
   - Check if API latency is too high.
   - Verify if the model is overloaded with requests.

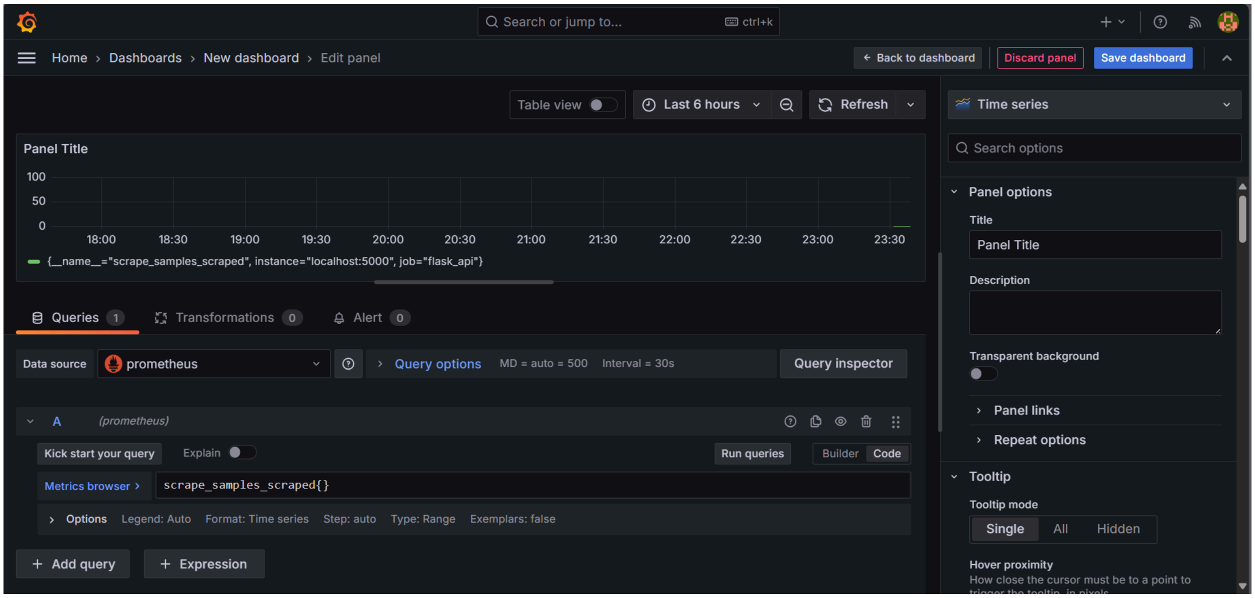



## Step 2: Optimize System

1. **Improve API Performance:**
    - Use asynchronous requests in Flask.
    - Add caching for frequently requested data.
2. **Optimize the Model:**
    - Retrain the model with additional live data.
    - Deploy a lightweight version of the model if predictions are slow.
3. **Scale the System:**
    - Use Docker Compose or Kubernetes to deploy multiple replicas of the Flask API.




## Sample Dashboard Screenshot



*showing real-time metrics visualization.*

## Conclusion

The integration of live data processing, monitoring, and system optimization as outlined in the three tasks has significantly improved the robustness, reliability, and efficiency of the machine learning pipeline. By setting up Grafana and Prometheus for real-time monitoring, the system's performance can be tracked and analyzed effectively. Testing the pipeline with live weather data validated the predictive accuracy of the deployed model in real-world scenarios, highlighting the importance of continuous testing. Furthermore, the analysis and optimization phase enabled identifying bottlenecks, optimizing API latency, and improving overall system scalability. These steps collectively ensure that the deployed pipeline is well-prepared to handle real-world data ingestion, prediction requests, and performance monitoring, thereby meeting the requirements of a production-grade system.

---

## References


1. **OpenWeatherMap API Documentation**
   URL: https://openweathermap.org/api
   Used for fetching live weather data for testing the predictive model.
2. **Prometheus Official Documentation**
   URL: https://prometheus.io/docs/introduction/overview/
   Used for setting up real-time monitoring of API and model metrics.
3. **Grafana Official Documentation**
   URL: https://grafana.com/docs/grafana/latest/
   Used for creating real-time visual dashboards to track system performance.
4. **Flask Framework Documentation**
   URL: https://flask.palletsprojects.com/
   Utilized for developing the API and integrating it with the Prometheus metrics exporter.
5. **Prometheus Flask Exporter**
   URL: https://github.com/rycus86/prometheus_flask_exporter
   Leveraged for exposing Flask application metrics for Prometheus monitoring.
6. **Machine Learning Model Documentation**
   Specific model implementation and usage were based on coursework resources and prior project documentation. Key aspects included scaling the data using a pre-trained scaler and making predictions using a trained LSTM model.
7. **Python Libraries**
   - `NumPy` for data preprocessing
   - `Requests` for API calls
   - `pickle` for loading pre-trained models and scalers
     Official Python documentation: https://docs.python.org/3/