Report 1 [IoT]

Student ID: 240581505

Student Name: Shobika Rajeskanna

Introduction:

A pipeline to process Internet of Things (IoT) sensor data in an edge-cloud environment has been created. The operation to be performed are pulling and running EMQX, data injection, data preprocessing, pulling and running RABBITMQ, machine learning model and visualization. Firstly, raw data has been fetched from Urban Observatory and from many metrics, need to store and analysis PM2.5 data with timestamp and values only. Then the filtered data has been sent to EMQX service of Azure lab (edge). Further, collect all PM2.5 data published from EMQX service, next filter out outliers which value is greater than 50. Then, the results (averaged data) have been taken into RabbitMQ service on Azure lab(cloud). After performing data preprocessing, collected daily average PM2.5 data. Furthermore, converted timestamp to date time format and using matplotlib displayed line chart of daily average of PM2.5 data. Finally, using feed averaged PM2.5 data to machine learning model to predict the trend of PM2.5 for next 15 days and visualized the predicted result machine learning classifier model.

Task-1

Design a data injector component by leveraging Newcastle Urban Observatory IoT data streams.

Step 1: Pull Docker image emqx/emqx from Dc:

```
docker pull emqx/emqx
```

Step 2: Run Docker image emqx/emqx from Docker Hub:

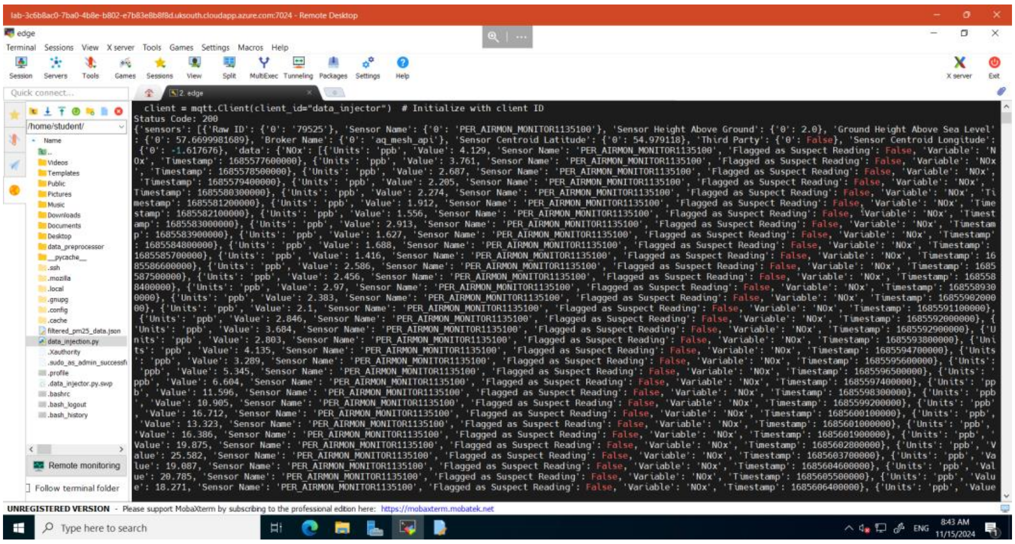
```
docker run -d --name emqx_broker -p 1883:1883 emqx/emqx
```

EMQX helps in filtering relevant data (like pm2.5 data) by acting as a local broker for distributing data to data preprocessing operator.

Step 3: Development of data injector code using python in Azure Lab (Edge):

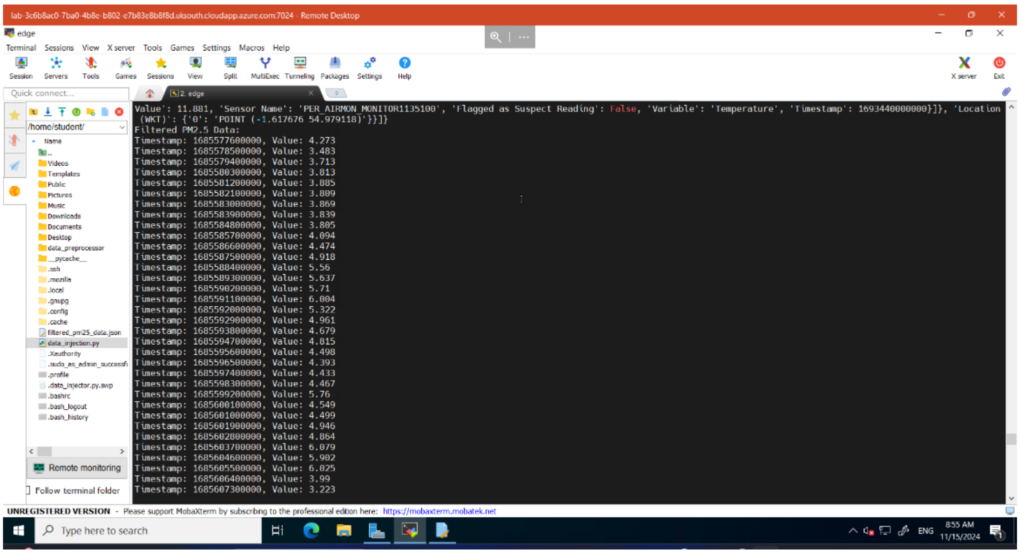
Functions in the code: collect data from Urban Observatory platform by sending HTTP request to the given API. Then fetch the raw data and printed them in the console.

Output:



Step 4: Filtering PM2.5 data:

In Urban Observatory API, there are many air quality sensor data available. We need to print PM2.5 data and in their meta-data, we need to print Timestamp and Values alone.



Step 5: Sending all the data to EMQX service of Azure lab (Edge):

Firstly, need to setup MQTT broker information like Host's IP, Port number and channel topic (in my case the topic name is sensor/pm2.5) where we send our data. Then, connecting to MQTT broker:

client.connect(BROKER_HOST, BROKER_PORT)

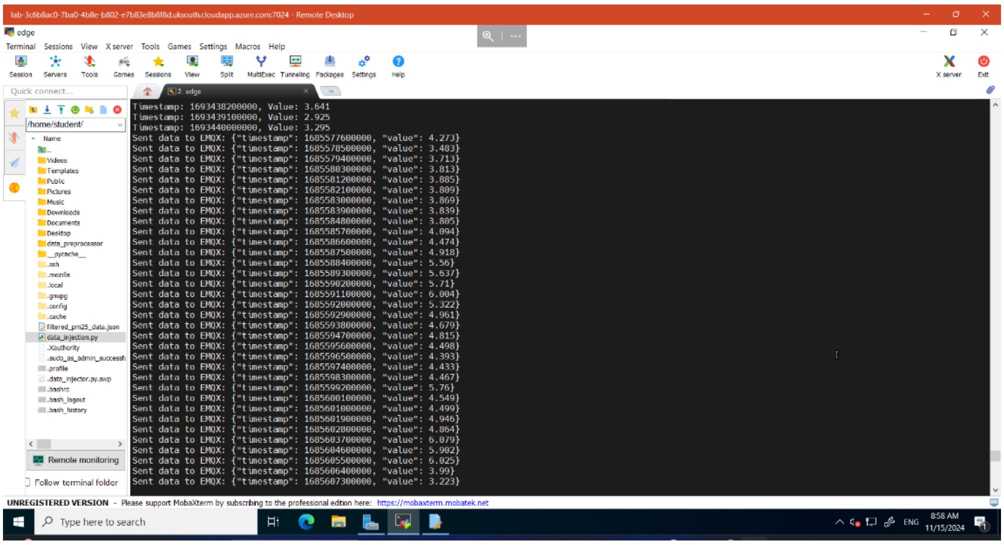
Finally sending data to EMQX:

Code snippet for performing the operation:

for data_point in pm25_data:

```
    message = json.dumps(data_point)
    client.publish(MQTT_TOPIC, message)
    print(f"Sent data to EMQX: {message}")
    time.sleep(0)
```

Output:



In below screenshot I have added the python program that performs all the functions I mentioned earlier.

data_injector.py

```
import json
import requests
import time
import paho.mqtt.client as mqtt

BROKER_HOST = '192.168.0.102'
BROKER_PORT = 1883
MQTT_TOPIC = "sensor/pm2.5"

url = "https://newcastle.urbanobservatory.ac.uk/api/v1.1/sensors/PER_AIRMON_MONITOR1135100/data/json/?starttime=20230601&endtime=20230831"

try:
    client = mqtt.Client(client_id="data_injector")


    response = requests.get(url)
    print(f"Status Code: {response.status_code}")

    if response.status_code == 200:
        raw_data = response.json()
        print(raw_data)
    else:
        print(f"Failed to retrieve data. Status code: {response.status_code}")
        raw_data = {}


    pm25_data = []
    for sensor in raw_data.get("sensors", []):
        data_entries = sensor.get("data", {})

        for key, entries in data_entries.items():
            if isinstance(entries, list):
                for item in entries:
                    if isinstance(item, dict) and item.get("Variable") == "PM2.5":
                        pm25_entry = {"timestamp": item["Timestamp"], "value": item["Value"]}
                        pm25_data.append(pm25_entry)

    print("Filtered PM2.5 Data:")
    for entry in pm25_data:
        print(f"Timestamp: {entry['timestamp']}, Value: {entry['value']}")


    client.connect(BROKER_HOST, BROKER_PORT)
    client.loop_start()


    for data_point in pm25_data:
        message = json.dumps(data_point)
        client.publish(MQTT_TOPIC, message)
        print(f"Sent data to EMQX: {message}")
        time.sleep(0)

except requests.RequestException as e:
    print(f"An error occurred while retrieving data: {e}")

except Exception as e:
    print(f"An error occurred: {e}")

finally:

    try:
        client.loop_stop()
        client.disconnect()
    except Exception as e:
        print(f"An error occurred during disconnect: {e}")
```

Task 2

Data preprocessing operator design

Step1:

In cloud pull and run the RabbitMQ, receive data from the edge (via EMQX) and queue it for further processing and visualizing.

Command to pull RabbitMQ:

docker pull rabbitmq:management

Command to run RabbitMQ:

docker run -d --name rabbitmq -p 5672:5672 -p 15672:15672 rabbitmq:3-management

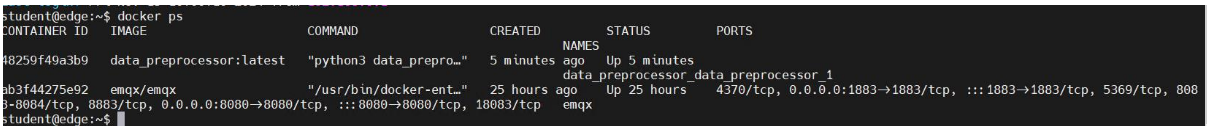
Step2:

Now subscribe all the PM2.5 data that is been sent from task 1 from EMQX service. Containerize this operator using Docker container, so the logs can be seen in the docker logs.

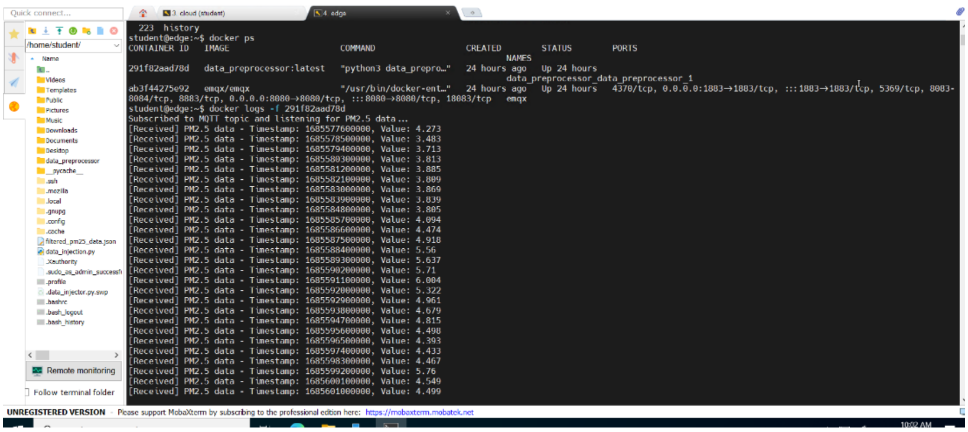
For containerization we need docker-compose, which will help us to start a container in correct order.

Command for docker-compose: docker-compose up

Containers in edge:



Docker log in edge:



The above output is fetching data from EMQX service and as I containerized my data preprocessor it is been logged and logs can be seen automatically.

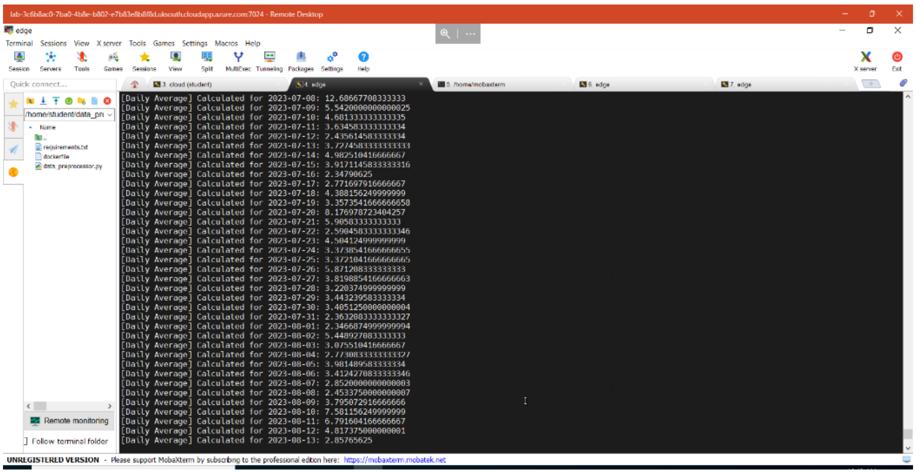
Step 3: Outliers

Perform a function in the code to find the values greater than 50.

Step 4:

Average value of PM2.5 data in daily basis has been created and the result has been printed in docker logs console.

Output:



Step 5:

Finally, the averaged PM2.5 data has been published into RabbitMQ service on Azure lab (Cloud). Connection has been made, for the connection we need RabbitMQ broker information like Host's IP, and rabbitmq _queue (in my code processed_pm25_data is queue).

Connection code snippet:

def publish_to_rabbitmq(data):

connection = pika.BlockingConnection(pika.ConnectionParameters(host=RABBITMQ_HOST))

channel = connection.channel()

channel.queue_declare(queue=RABBITMQ_QUEUE)

for item in data:

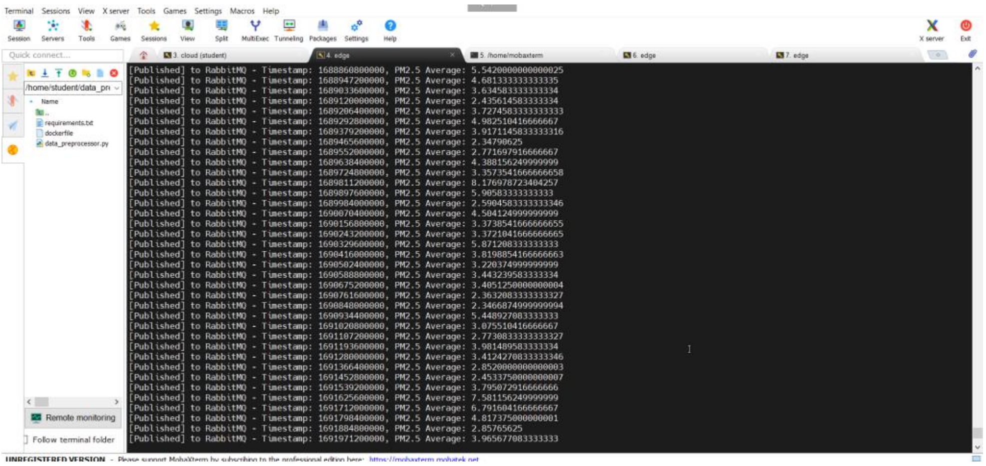
message = json.dumps(item)

channel.basic_publish(exchange='', routing_key=RABBITMQ_QUEUE, body=message)

print(f"[Published] to RabbitMQ - Timestamp: {item['timestamp']}, PM2.5 Average: {item['value']}")

connection.close()

Output:



So, in this task we have created Dockerfile to migrate data preprocessor and docker-compose to run the container.

docker-compose:

```yaml
version: "3"
services:
  data_preprocessor:
    build: ./subscriber
    image: data_preprocessor:latest
    environment:
      rabbitmq_broker: "192.168.0.100"
      exqx_broker : "192.168.0.102"
networks:
  default:
    driver: bridge
```

Dockerfile:

```dockerfile
FROM python:3.8.12

USER root

ADD . /usr/local/source

WORKDIR /usr/local/source

RUN pip3 install -r requirements.txt

CMD ["python3", "data_preprocessor.py"]

```

Code:

Now, the screenshot of entire code that performs task 2 has been added below.

data_preprocessor.py

```python
import json
import paho.mqtt.client as mqtt
import pika
from datetime import datetime, timedelta
import time

BROKER_ADDRESS = "192.168.0.102"
MQTT_PORT = 1883
MQTT_TOPIC = "sensor/pm2.5"

RABBITMQ_QUEUE = 'processed_pm25_data'
RABBITMQ_HOST = '192.168.0.100'

pm25_data = []
outliers = []


def on_message(client, userdata, msg):
    global pm25_data
    data = json.loads(msg.payload.decode())
    print(f"[Received] PM2.5 data - Timestamp: {data['timestamp']}, Value: {data['value']}")


    if data['value'] <= 50:
        pm25_data.append(data)
    else:

        print(f"[Outlier] Filtered out PM2.5 data - Timestamp: {data['timestamp']}, Value: {data['value']}")

client = mqtt.Client()
client.on_message = on_message
client.connect(BROKER_ADDRESS, MQTT_PORT)
client.subscribe(MQTT_TOPIC)
print("Subscribed to MQTT topic and listening for PM2.5 data...")
client.loop_start()

def calculate_daily_averages():
    global pm25_data
    if not pm25_data:
        return []

    pm25_data.sort(key=lambda x: x['timestamp'])

    daily_averages = []
    current_day = datetime.fromtimestamp(pm25_data[0]['timestamp'] / 1000).date()
    daily_sum = 0
    daily_count = 0

    for entry in pm25_data:
        entry_date = datetime.fromtimestamp(entry['timestamp'] / 1000).date()

        if entry_date == current_day:
            daily_sum += entry['value']
            daily_count += 1
        else:
            if daily_count > 0:
                average = daily_sum / daily_count
                daily_averages.append({
                    'timestamp': int(datetime.combine(current_day, datetime.min.time()).timestamp() * 1000),
                    'value': average
                })
                print(f"[Daily Average] Calculated for {current_day}: {average}")

            current_day = entry_date
            daily_sum = entry['value']
            daily_count = 1

    if daily_count > 0:
        average = daily_sum / daily_count
        daily_averages.append({
            'timestamp': int(datetime.combine(current_day, datetime.min.time()).timestamp() * 1000),
            'value': average
        })
        print(f"[Daily Average] Calculated for {current_day}: {average}")


    return daily_averages

def publish_to_rabbitmq(data):
    connection = pika.BlockingConnection(pika.ConnectionParameters(host=RABBITMQ_HOST))
    channel = connection.channel()
    channel.queue_declare(queue=RABBITMQ_QUEUE)

    for item in data:
        message = json.dumps(item)
        channel.basic_publish(exchange='', routing_key=RABBITMQ_QUEUE, body=message)
        print(f"[Published] to RabbitMQ - Timestamp: {item['timestamp']}, PM2.5 Average: {item['value']}")

    connection.close()

try:
    while True:

        time.sleep(300)


        daily_averages = calculate_daily_averages()


        if daily_averages:
            publish_to_rabbitmq(daily_averages)


            with open("filtered_pm25_data.json", "w") as f:
                json.dump(pm25_data, f, indent=4)
            print("[Saved] Filtered PM2.5 data to 'filtered_pm25_data.json'.")

        if outliers:
            print("Outliers:", outliers, flush=True)


        pm25_data = []

except KeyboardInterrupt:
    print("Stopping data processing and disconnecting from MQTT broker.")
    client.loop_stop()
    client.disconnect()
```

Task 3

Time series data prediction and visualization

Step 1:

Firstly, I added pre-defined machine learning engine code in cloud according to my prediction code.

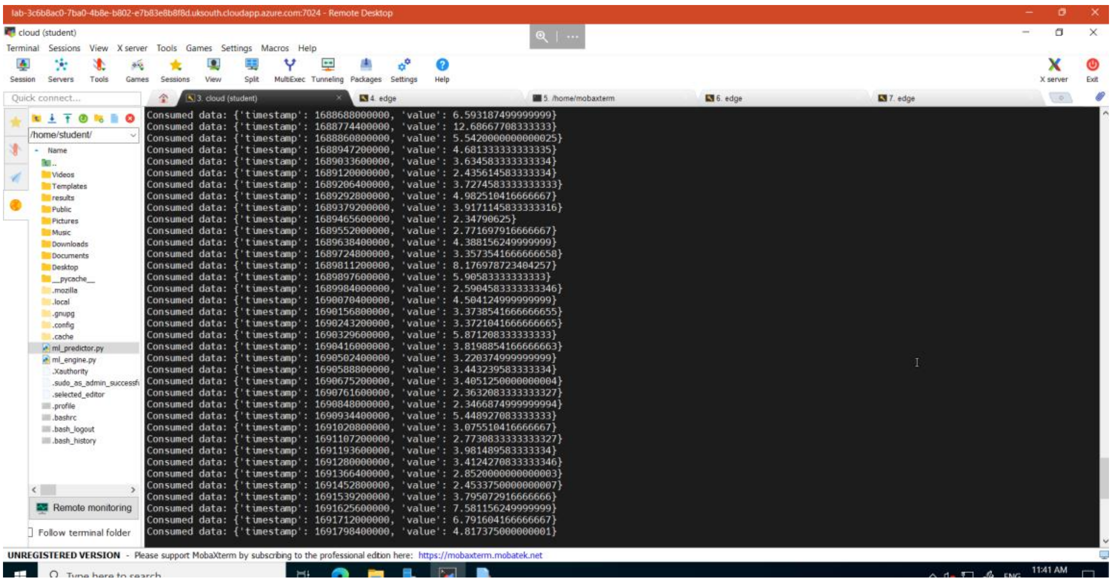
ml_engine.py

```python
import pandas as pd
from prophet import Prophet

class MLPredictor(object):
    def __init__(self, data_df):

        self.__train_data = self.__convert_col_name(data_df)
        self.__trainer = Prophet(changepoint_prior_scale=12)

    def train(self):
        self.__trainer.fit(self.__train_data)

    def __convert_col_name(self, data_df):

        data_df.rename(columns={"timestamp": "ds", "value": "y"}, inplace=True)


        data_df['ds'] = pd.to_datetime(data_df['ds'], unit='ms')

        print(f"After renaming and converting columns:\n{data_df.head()}")
        return data_df

    def __make_future(self, periods=15):
        future = self.__trainer.make_future_dataframe(periods=periods)
        return future

    def predict(self):
        future = self.__make_future()
        forecast = self.__trainer.predict(future)
        return forecast

    def plot_result(self, forecast):
        fig = self.__trainer.plot(forecast, figsize=(15, 6))
        return fig
```

Step 2:

Consume all averaged daily PM2.5 data that has been published by RabbitMQ service in task 2.

Output:



Step 3:

Convert timestamp to date time format.

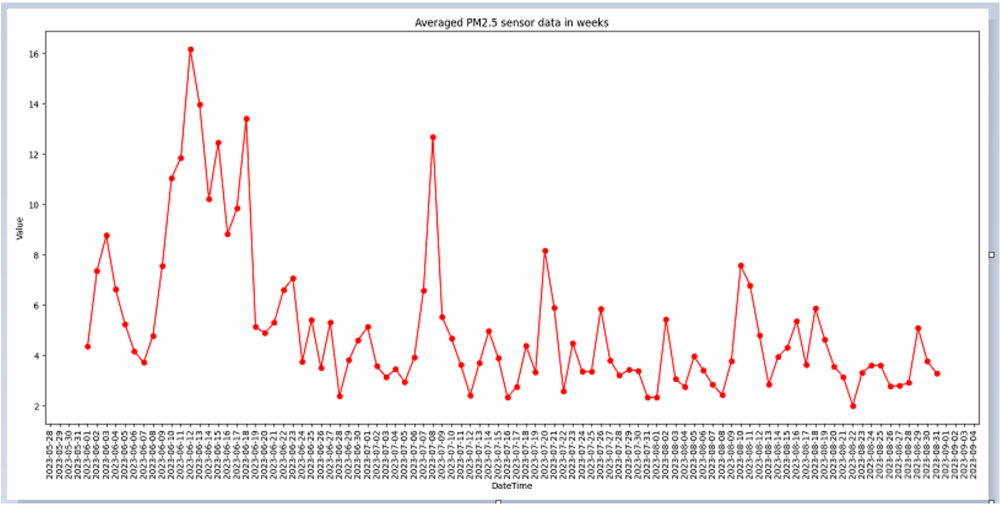
Output:

```
0 2023-06-01  4.385896
1 2023-06-02  7.366385
2 2023-06-03  8.777760
3 2023-06-04  6.630323
4 2023-06-05  5.260854
```

Step 4:

Using matplotlib, visualize the averaged PM2.5 daily data.

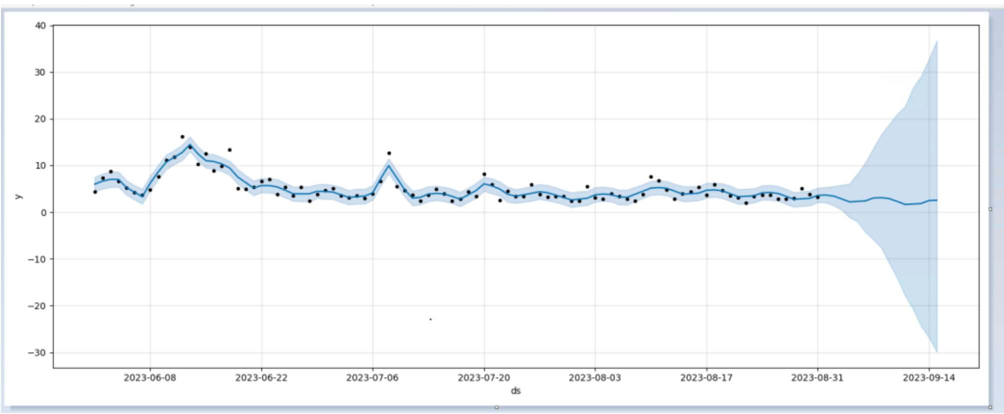
Line chart:



Step 5:

Finally, I feed the averaged data to machine learning model to predict the trend of PM2.5 data for next 15 days. Then, the prediction has been visualized and stored in the directory.

Figure:



Code for task 3:

ml_predictor.py

```python
import matplotlib
matplotlib.use('Agg')
import pika
import json
import pandas as pd
import matplotlib.pyplot as plt
from ml_engine import MLPredictor
import os

RABBITMQ_QUEUE = 'processed_pm25_data'
RABBITMQ_HOST = 'localhost'

def consume_from_rabbitmq():
    connection = pika.BlockingConnection(pika.ConnectionParameters(RABBITMQ_HOST))
    channel = connection.channel()
    channel.queue_declare(queue=RABBITMQ_QUEUE)

    pm25_data = []

    def callback(ch, method, properties, body):
        data = json.loads(body)
        pm25_data.append(data)
        print(f"Consumed data: {data}")

        ch.basic_ack(delivery_tag=method.delivery_tag)

    while True:
        method_frame, _, body = channel.basic_get(queue=RABBITMQ_QUEUE, auto_ack=False)
        if method_frame:
            callback(channel, method_frame, None, body)
        else:
            break

    df = pd.DataFrame(pm25_data)
    connection.close()
    return df

pm25_df = consume_from_rabbitmq()

if not pm25_df.empty:
    pm25_df['timestamp'] = pd.to_datetime(pm25_df['timestamp'], unit='ms')

    if not os.path.exists("results"):
        os.makedirs("results")

    plt.figure(figsize=(16, 8))
    plt.plot(pm25_df['timestamp'], pm25_df['value'], color='red', marker='o', linestyle='-')
    plt.title('Averaged PM2.5 sensor data in weeks')
    plt.xlabel('DateTime')
    plt.ylabel('Value')

    plt.xticks(rotation=90)
    plt.gca().xaxis.set_major_locator(plt.MultipleLocator(1))
    plt.tight_layout()

    daily_plot_path = "results/daily_pm25_averages.png"
    plt.savefig(daily_plot_path)
    print(f"Daily PM2.5 plot saved as '{daily_plot_path}'.")

    predictor = MLPredictor(pm25_df)

    predictor.train()
    print("Training complete.")

    forecast = predictor.predict()
    print("Prediction complete.")

    forecast_plot_path = "results/forecast_result.png"
    fig = predictor.plot_result(forecast)
    fig.savefig(forecast_plot_path)
    print(f"Forecast plot saved as '{forecast_plot_path}'.")

else:
    print("No data available for training and plotting.")
```

Cloud Container:

```
student@cloud:~$ docker ps
CONTAINER ID   IMAGE                 COMMAND                  CREATED        STATUS        PORTS
                                                                                             NAMES
f93b42f21617   rabbitmq:management   "docker-entrypoint.s…"   26 hours ago   Up 26 hours   4369/tcp, 0.0.0.0:1567->1567/tcp, :::1567->1567/tcp, 5671/tcp, 15671-1567
2/tcp, 15691-15692/tcp, 0.0.0.0:5672->5672/tcp, :::5672->5672/tcp, 25672/tcp   rabbitmq
student@cloud:~$
```

Analytics and Conclusion:

In this IOT system, we work with three tiers, they are IoT tier, edge tier and cloud tier. From IoT tier we are accessing the real time data. Then in edge tier we are fetching the real time data and extracting the set of data be processed and, we are publishing the data to preprocessor and then the preprocessor will subscribe the data and removed outlier and finds the average and push the data to cloud tier. Now the cloud tier displays the averaged data into line chart and then using machine learning it predicts for next 15 days, then the predicted image has been stored.