CSC8112 Report1 Boyan Li (IoT)

Student ID：23010958（c30109586）

Student Name：Boyan Li

During the demonstration, some problems were found in the implementation of Task2's "data preprocessing operator" part, which has been corrected now:

1. data preprocessing, which should be implemented to fliter different types of data into different data sets.
2. Send data to RabbitMQ in a package instead of sending each piece of data individually.

To fix the above problems, I improved the python code in the following ways:

1. Clearer naming
2. Try to implement all logic in one main function per file to avoid unnecessary bugs caused by parameter passing
3. Correct the code logic of "data preprocessing operator" in Task2
4. Send the data to the RabbitMQ message queue in the correct way
5. Confirm the right format of 'Timestamp'.

Part A. Experimental Results Show

Content of Part A including:

1. Detailed response to each task and related sub-tasks.
2. Screenshots of running services in the Docker Environment.
3. Screenshots of Funtional Code (Files)

    “task1_iot_edge.py”: the data injector component;

    “task2_edge_cloud.py”: the data preprocessing operator;

    “task3_cloud_processor.py”: time-series data prediction and visualization;

(with task2_edge_cloud.py, there are Dockerfile, docher-compose.yml, and requirements.txt)

Task 1: Design a data injector component by leveraging Newcastle Urban Observatory IoT data streams (by using the back up url)

1. Pull and run the Docker image "emqx/emqx" from Docker Hub in the virtual machine running on Azure lab (Edge). Perform this task first using the command line interface (CLI).

- pull the Docker image "emqx/emqx" by using：“docker pull emqx/emqx”
- run the Docker image "emqx/emqx" by using： "docker run -d --name emqx -p 1883:1883 -p 18083:18083 emqx/emqx"

  (eg. "1883:1883", the left one is for the server, and the right one is for the docker)

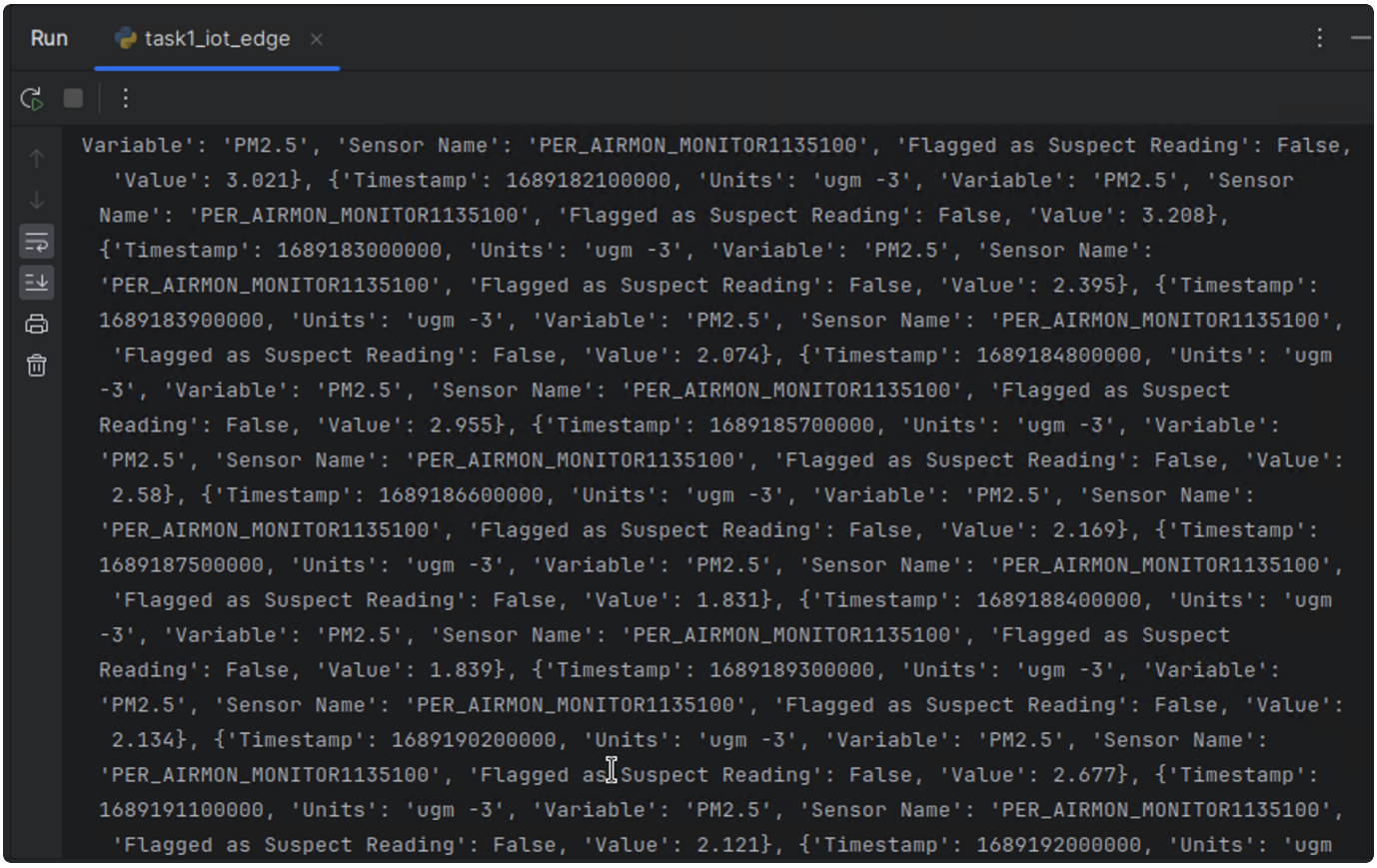
```
student@edge:~$ docker ps
CONTAINER ID   IMAGE       COMMAND                  CREATED      STATUS        PORTS
                                                                                NAMES
086f3553c9be   emqx/emqx   "/usr/bin/docker-ent…"   2 days ago   Up 33 hours   4370/tcp, 0.0.0.0
:1883->1883/tcp, :::1883->1883/tcp, 5369/tcp, 8083-8084/tcp, 8883/tcp, 0.0.0.0:8081->8081/tcp, :
::8081->8081/tcp, 0.0.0.0:18083->18083/tcp, :::18083->18083/tcp, 11883/tcp   emqx
student@edge:~$
```

2. Develop a data injector component with the following functions (Code) in Azure Lab (Edge):

“task1_iot_edge.py”: the data injector component;

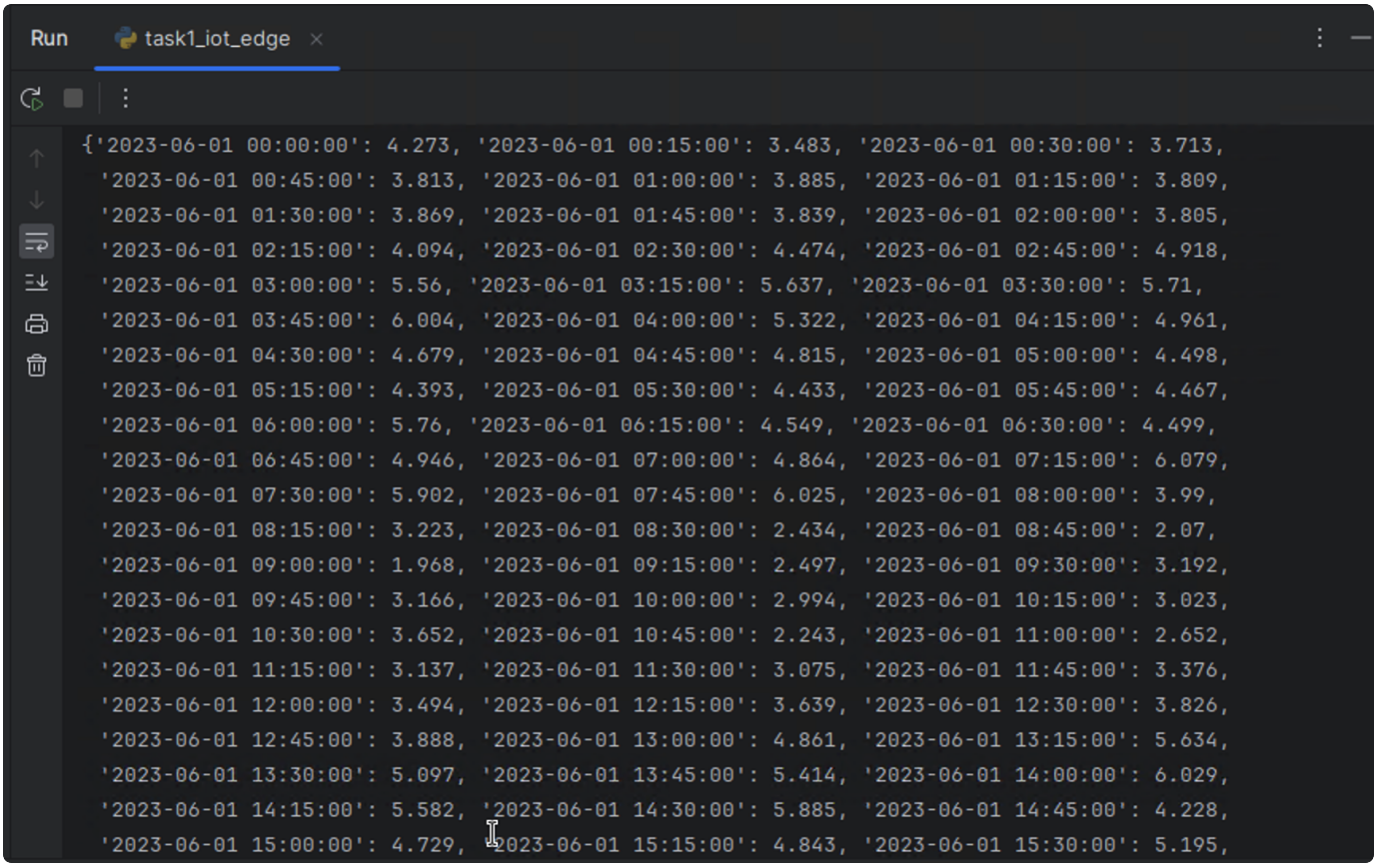
(a) Collect data from Urban Observatory platform by sending HTTP request. Following that, please print out the raw data streams that you collected on the console.

```
# Collect raw data from Urban Observatory (IoT Layer Back up url)
url = "https://gist.githubusercontent.com/ringosham/fbd66654dc53c40bd4581d2828acc94e/raw/d56a0fcfd27ff7ea31e2aec3765eb2c5d64a
response = requests.get(url)
raw_data_dict = json.loads(response.text)
print(raw_data_dict)
```



(b) Although the raw air quality data you collected from the Urban Observatory API contains many metrics including$N\ O_2$, NO,CO_2, PM2.5, and PM10, among others, for the purpose of this coursework you only need to store and analyze PM2.5 data. While many meta–data are available for PM2.5 data, such as sensor name, timestamp, value, and location, you only need to store the metrics related to the Timestamp and Value meta–data fields.

```
# Extract PM2.5 data
pm25_data = {}
for data in raw_data_dict["sensors"]:
    for items in data["data"]["PM2.5"]:
        timestamp = items['Timestamp']
        value = items['Value']
        timestamp = datetime.utcfromtimestamp(timestamp / 1000).strftime('%Y-%m-%d %H:%M:%S')
        pm25_data[timestamp] = value
print(pm25_data)
```



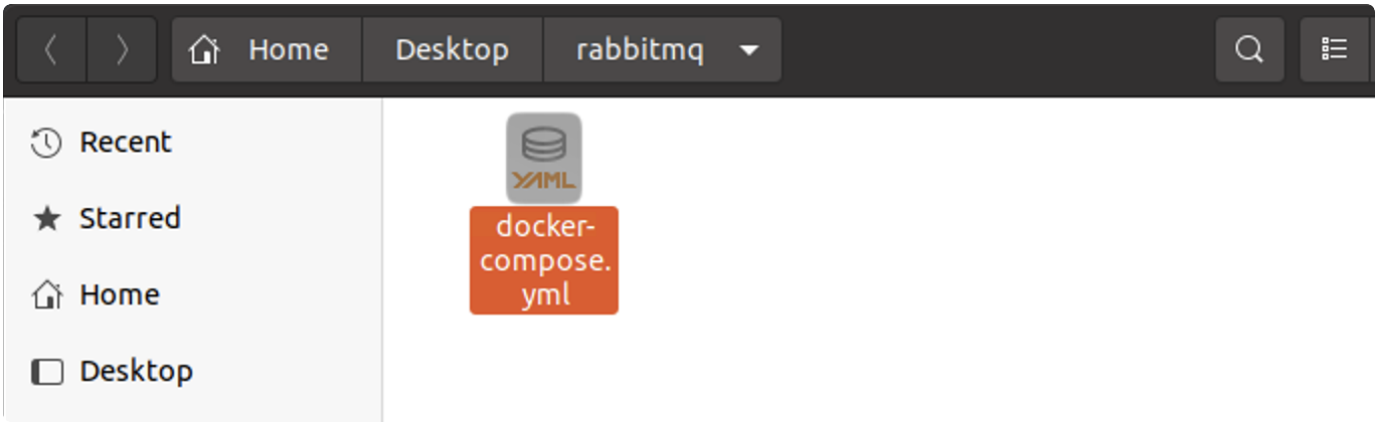
(c) Send all PM2.5 data to be used by Task 2.2 (a) to EMQX service of Azure lab (Edge).

```
# MQTT (Publisher)
mqtt_ip = "192.168.0.102"  # edge vm ip address
mqtt_port = 1883
topic = "CSC8112"

# Create a mqtt client object
client = mqtt_client.Client()

# Connect to MQTT service
client.on_connect = on_connect
client.connect(mqtt_ip, mqtt_port)

# Publish message to MQTT
# Note: MQTT payload must be a string, bytearray, int, float or None
msg = json.dumps(pm25_data)
client.publish(topic, msg)
```

Task 2: Data preprocessing operator design

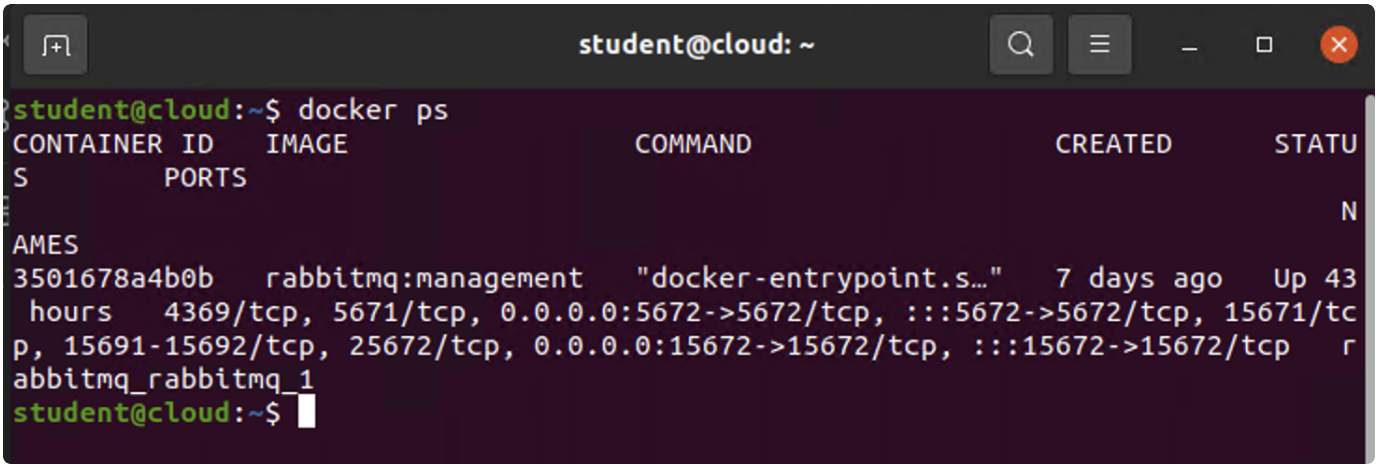
1. Define a Docker compose file which contains the following necessary configurations and

instructions for deploying and instantiating the following set of Docker images (as shown
in Figure 1) on Azure lab (Cloud):




```yaml
version: '3'
services:
  rabbitmq:
    image: rabbitmq:management
    ports:
      - "5672:5672"
      - "15672:15672"
```

(a) Download and run RabbitMQ image (rabbitmq:management);



2. Design a data preprocessing operator with the following functions (code) in Azure Lab (Edge):

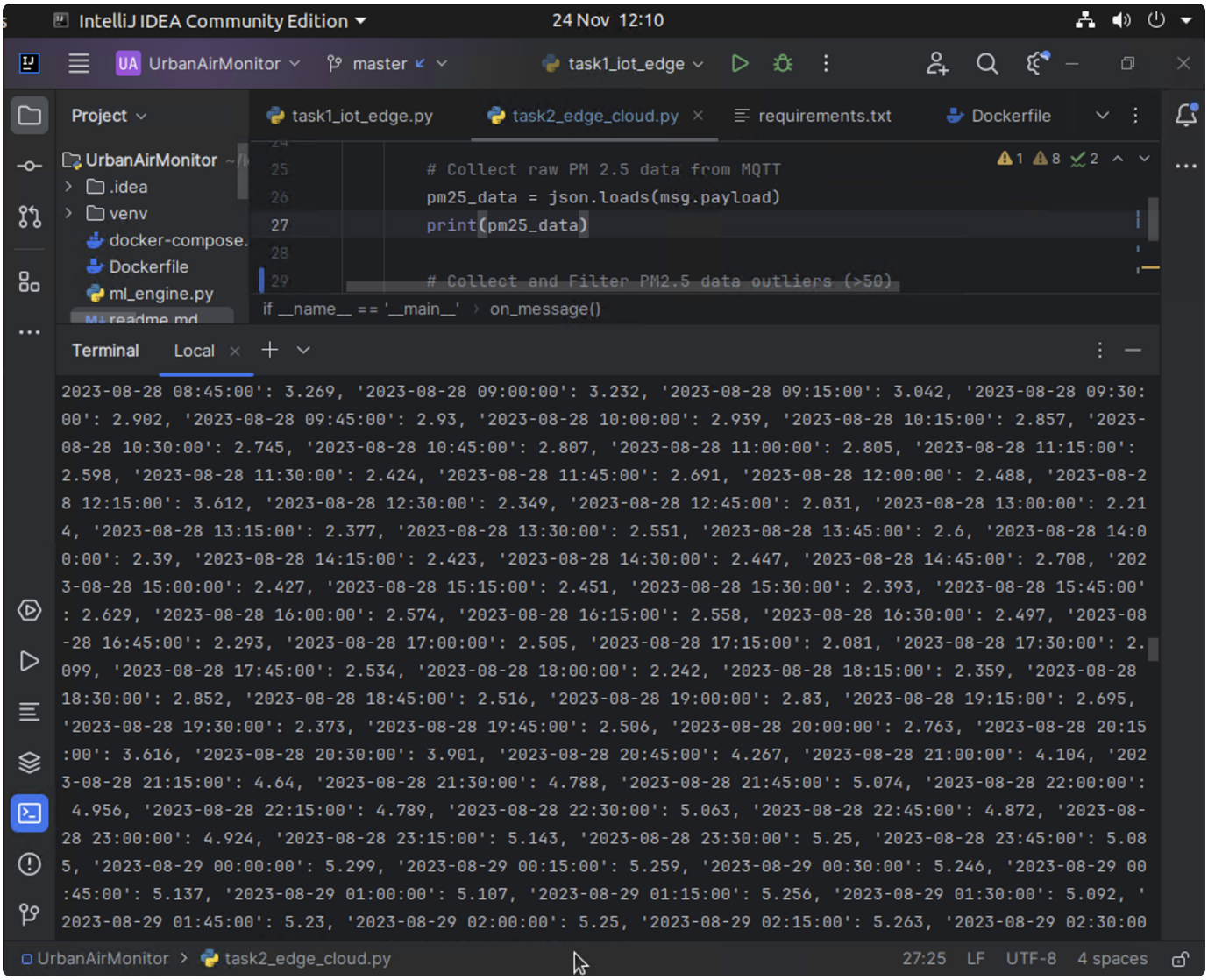
"task2_edge_cloud.py": the data preprocessing operator;

a. Collect all PM2.5 data published by Task 1.2 (c) from EMQX service, and please print out the PM2.5 data to the console (this operator will run as a Docker

container,

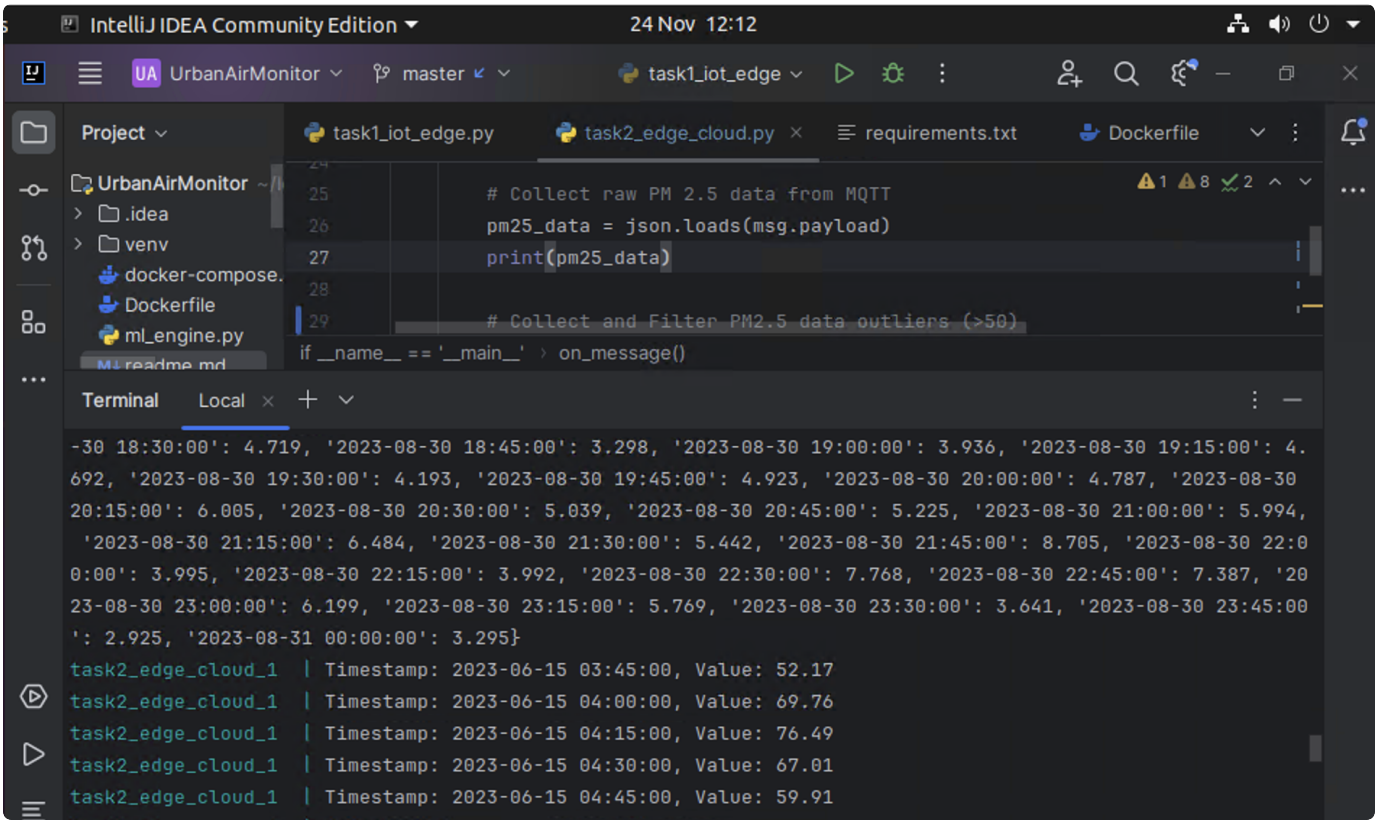
so the logs can be seen in the docker logs console automatically).

```
    # MQTT (Subscriber)
    # Create a mqtt client object
    client = mqtt_client.Client()
    # Callback function for MQTT connection
    client.on_connect = on_connect
    client.connect(mqtt_ip, mqtt_port)
    # Subscribe MQTT topic
    client.subscribe(topic)
    client.on_message = on_message
    # Start a thread to monitor message from publisher
    client.loop_forever()

# Callback function will be triggered
def on_message(client, userdata, msg):
    print(f"Get message from publisher {json.loads(msg.payload)}")

    # Collect raw PM 2.5 data from MQTT
    pm25_data = json.loads(msg.payload)
    print(pm25_data)
```



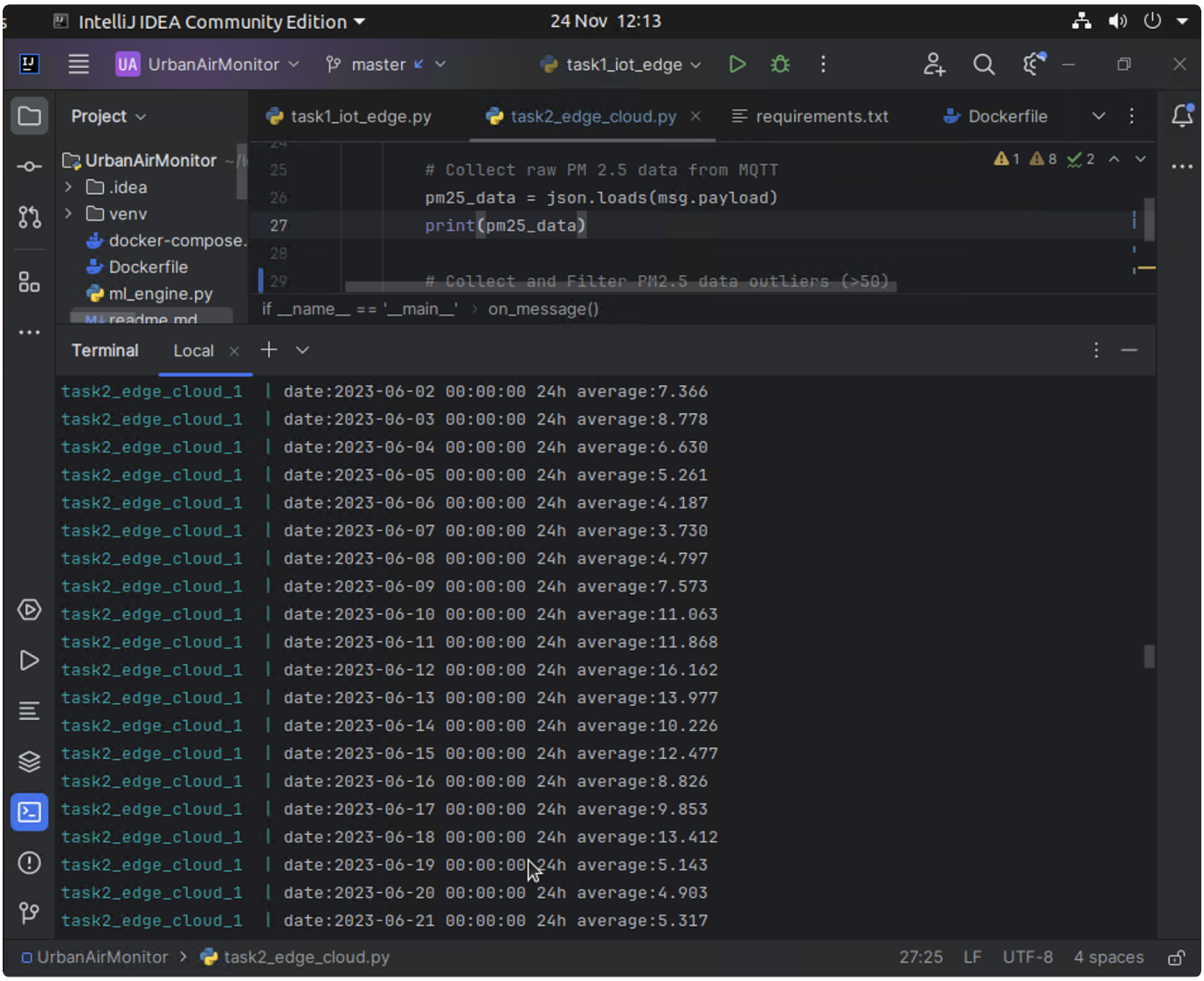
b. Filter out outliers (the value greater than 50), and please print out outliers to the console (docker logs console).

```
# Collect and Filter PM2.5 data outliers (>50)
pm25_outliers = []
for key, value in pm25_data.items():
    if value > 50:
        pm25_outliers.append(key)
        print(f"Timestamp: {key}, Value: {value}")
for key in pm25_outliers:
    pm25_data.pop(key)  # Filter the outliers!
```



c. Since the original PM2.5 data readings are collected every 15 mins, so please imple–
ment a python code to calculate the averaging value of PM2.5 data on daily basis (every 24 hours), please pick the first start date of a day or a 24 hours interval as the new timestamp of averaged PM2.5 data, and please print out the result to the console (docker logs console)

```python
# Collect valid daily data
daily_data = {}
for key, value in pm25_data.items():  # Outliers already filtered.
    key = datetime.strptime(key, '%Y-%m-%d %H:%M:%S')
    # Converts the timestamp to a date object, extracting only the date part(year, month, and day).
    date = key.date()
    if date not in daily_data:
        daily_data[date] = {'Timestamp': key, 'Value': [], 'Average': None}
    daily_data[date]['Value'].append(value)

# Calculate the daily average and use the first datetime of each day as the new timestamp
average_data = {}
for date, day_data in daily_data.items():
    day_data['Average_Value'] = sum(day_data['Value']) / len(day_data['Value'])
    day_data['Timestamp'] = datetime(date.year, date.month, date.day)
    average_data[f"{day_data['Timestamp']}"] = day_data['Average_Value']
    print(f"date:{day_data['Timestamp']} 24h average:{day_data['Average_Value']:.3f}")
```



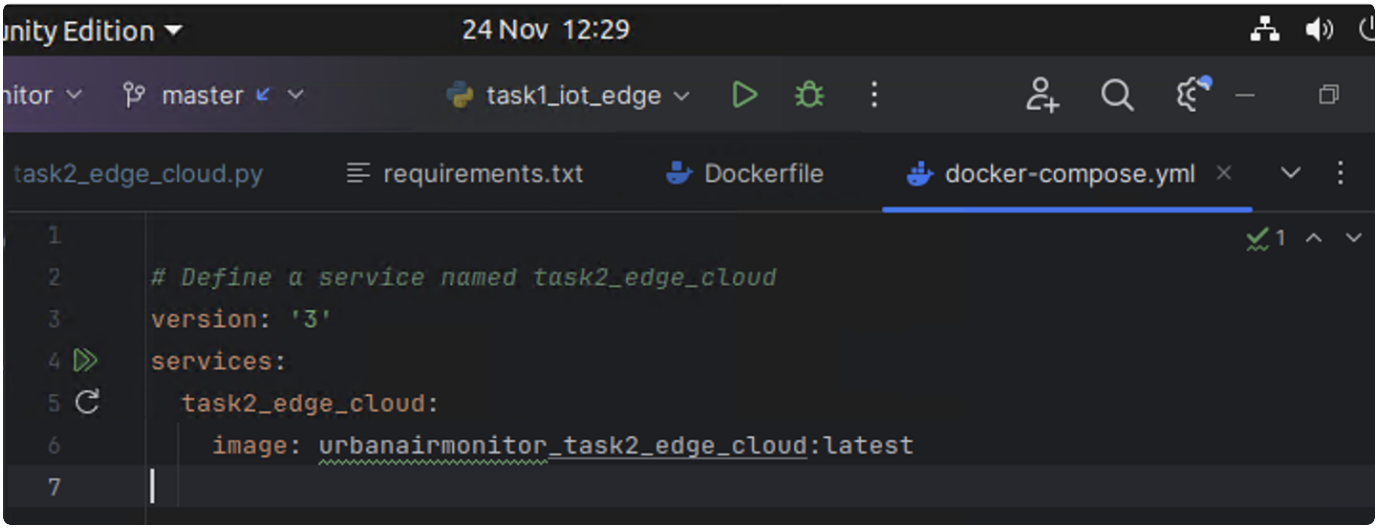
d. Transfer all results (averaged PM2.5 data) to be used by Task 3.2 (a) into RabbitMQ

service onAzure lab (Cloud).

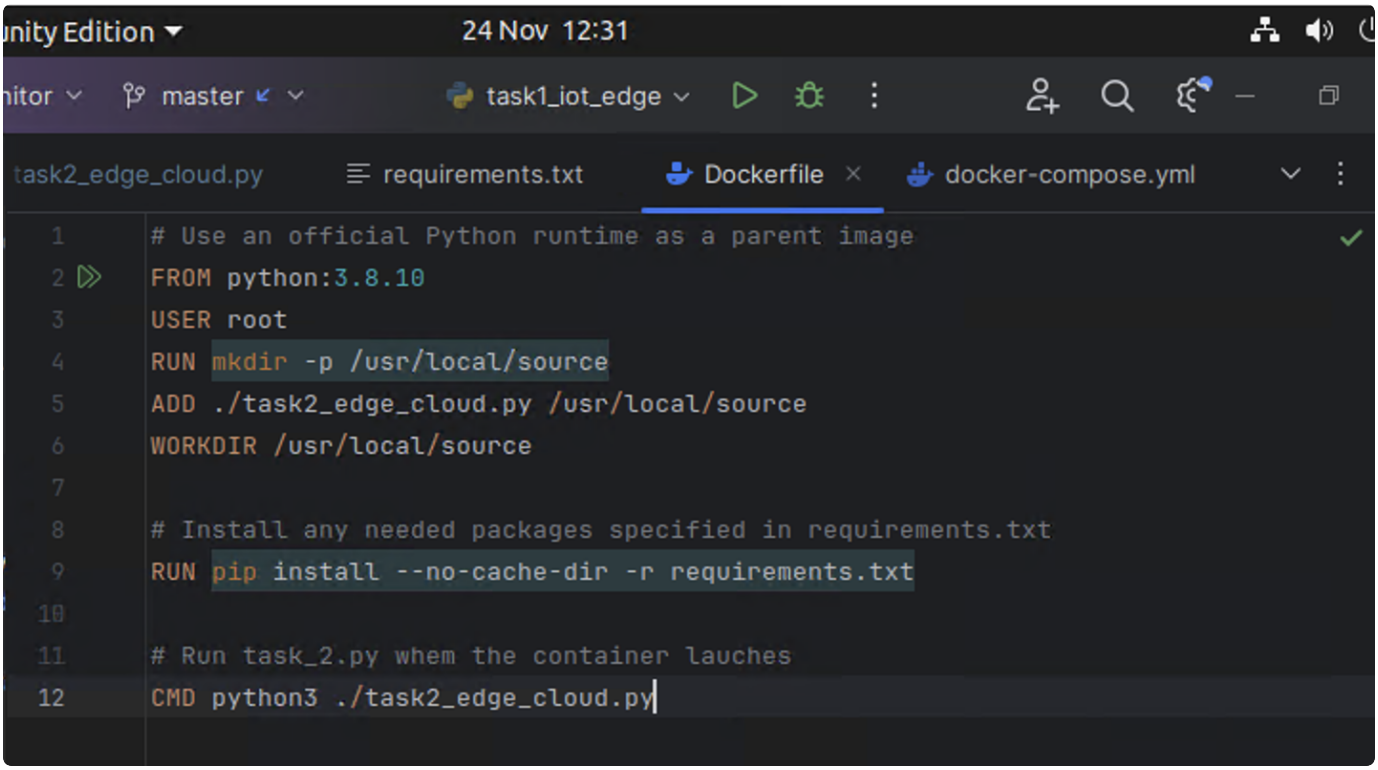
```
# RabbitMQ (Producer)
rabbitmq_ip = "192.168.0.100"
rabbitmq_port = 5672
# Queue name
rabbitmq_queque = "CSC8112"
# Connect to RabbitMQ service
connection = pika.BlockingConnection(pika.ConnectionParameters(host=rabbitmq_ip, port=rabbitmq_port))
channel = connection.channel()
# Declare a queue
channel.queue_declare(queue=rabbitmq_queque)
# Produce message
channel.basic_publish(exchange='',
                      routing_key=rabbitmq_queque,
                      body=json.dumps(average_data))
connection.close()
```

3. Define a Dockerfile to migrate your "data preprocessing operator" source code into a Docker
   image and then define a docker-compose file to run it as a container locally on the Azure
   lab (Edge).

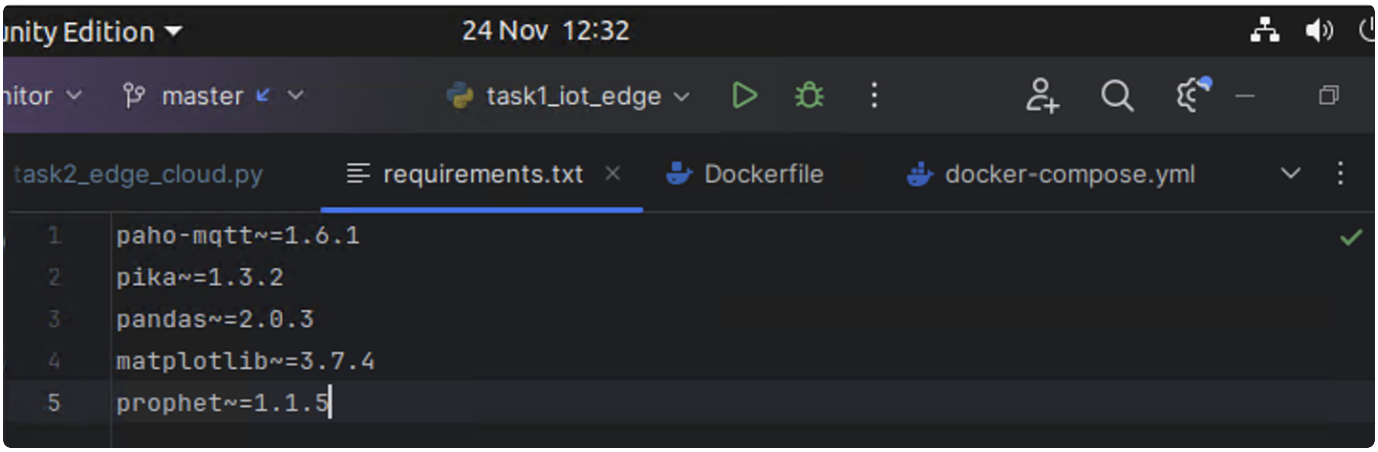
· *docker-compose.yml*



· *Dockerfile*

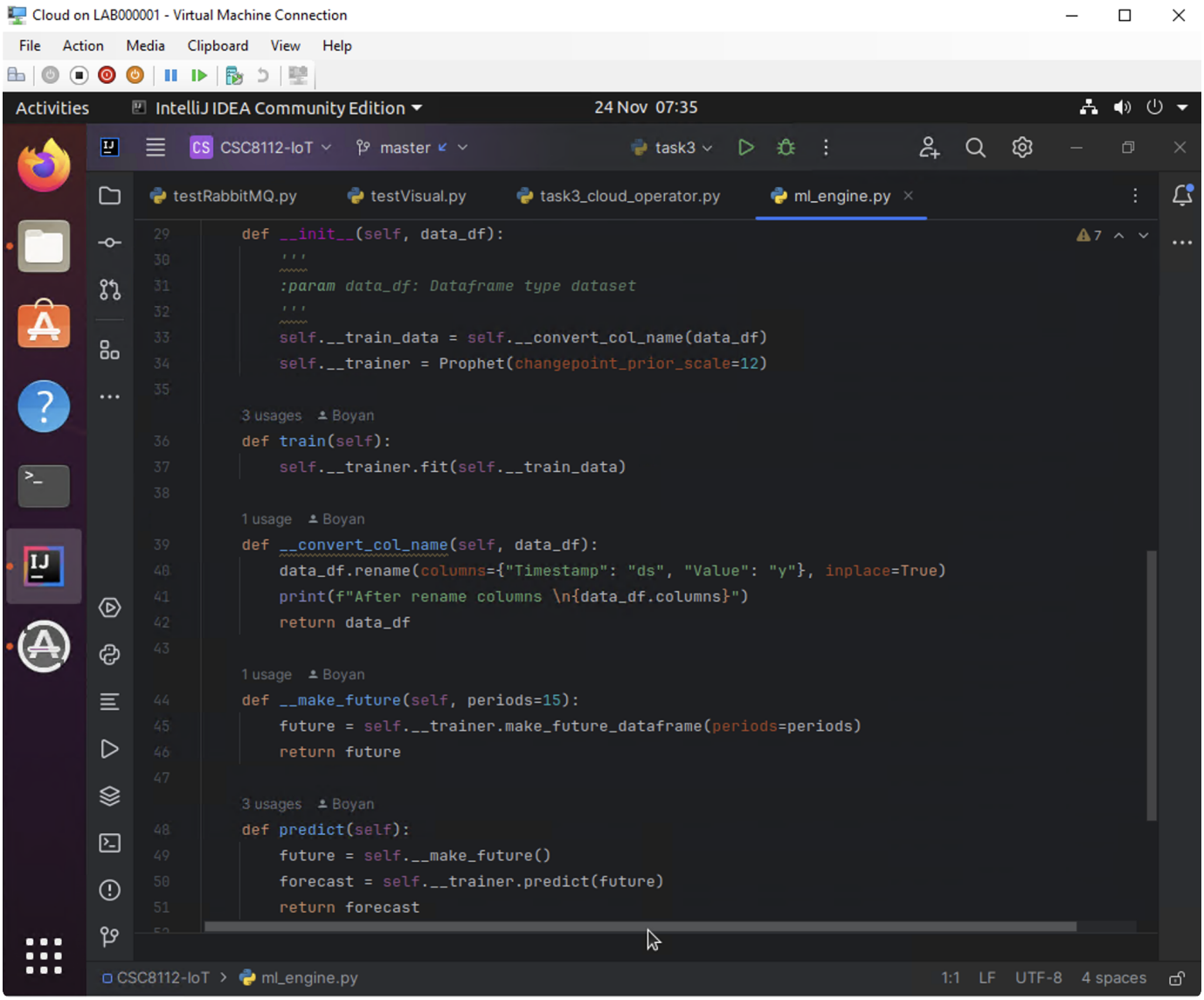


· *requirements.txt*



Task 3: Time-series data prediction and visualization

1. Download a pre-defined Machine Learning (ML) engine code from [https://github.com/ncl-iot-team/CSC8112_MLEngine]. And put it in the same directory with task3_cloud_operator.py.



2. Design a PM2.5 prediction operator with the following functions (code) in Azure Lab (Cloud) or the Azure Lab localhost:

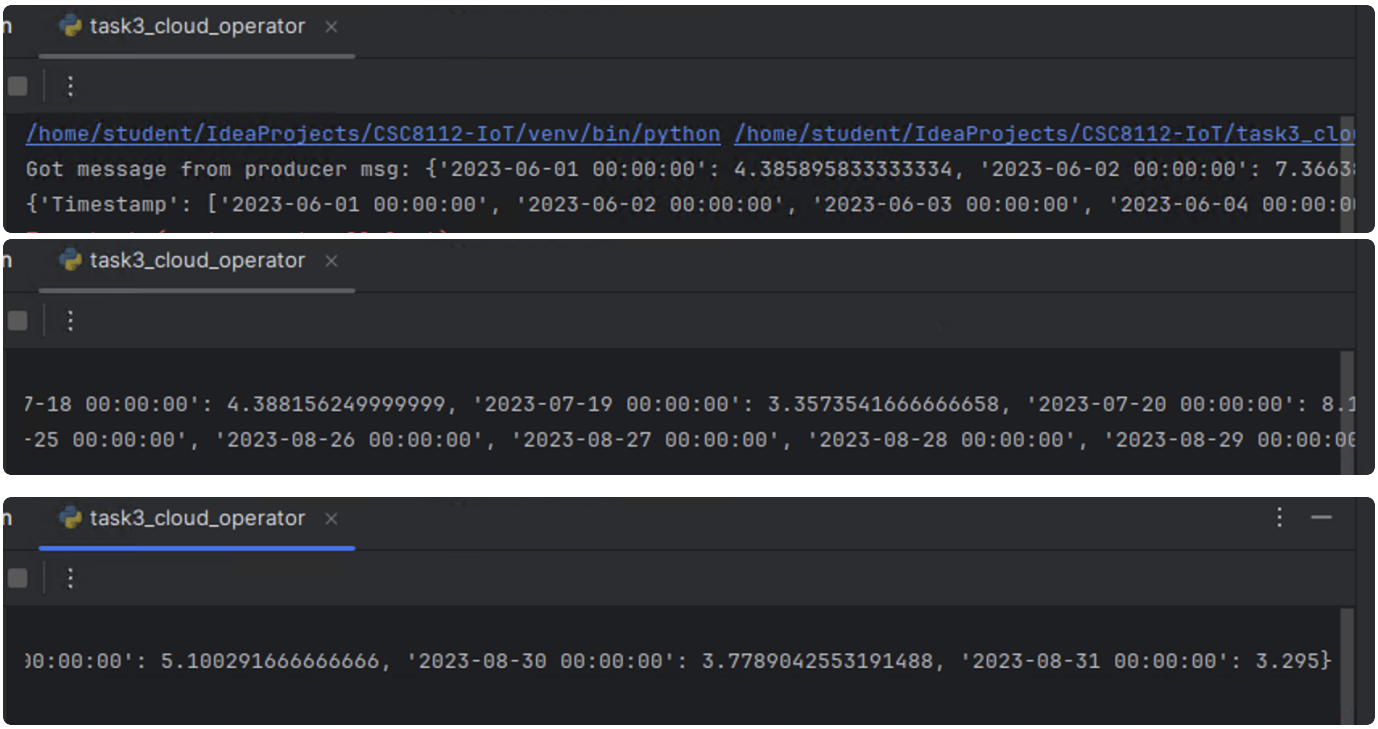
"task3_cloud_processor.py": time-series data prediction and visualization;

a. Collect all averaged daily PM2.5 data computed by Task 2.2 (d) from RabbitMQ service, and please print out them to the console.

b. Convert timestamp to date time format (year-month-day hour:minute:second), and please print out the PM2.5 data with the reformatted timestamp to the console.

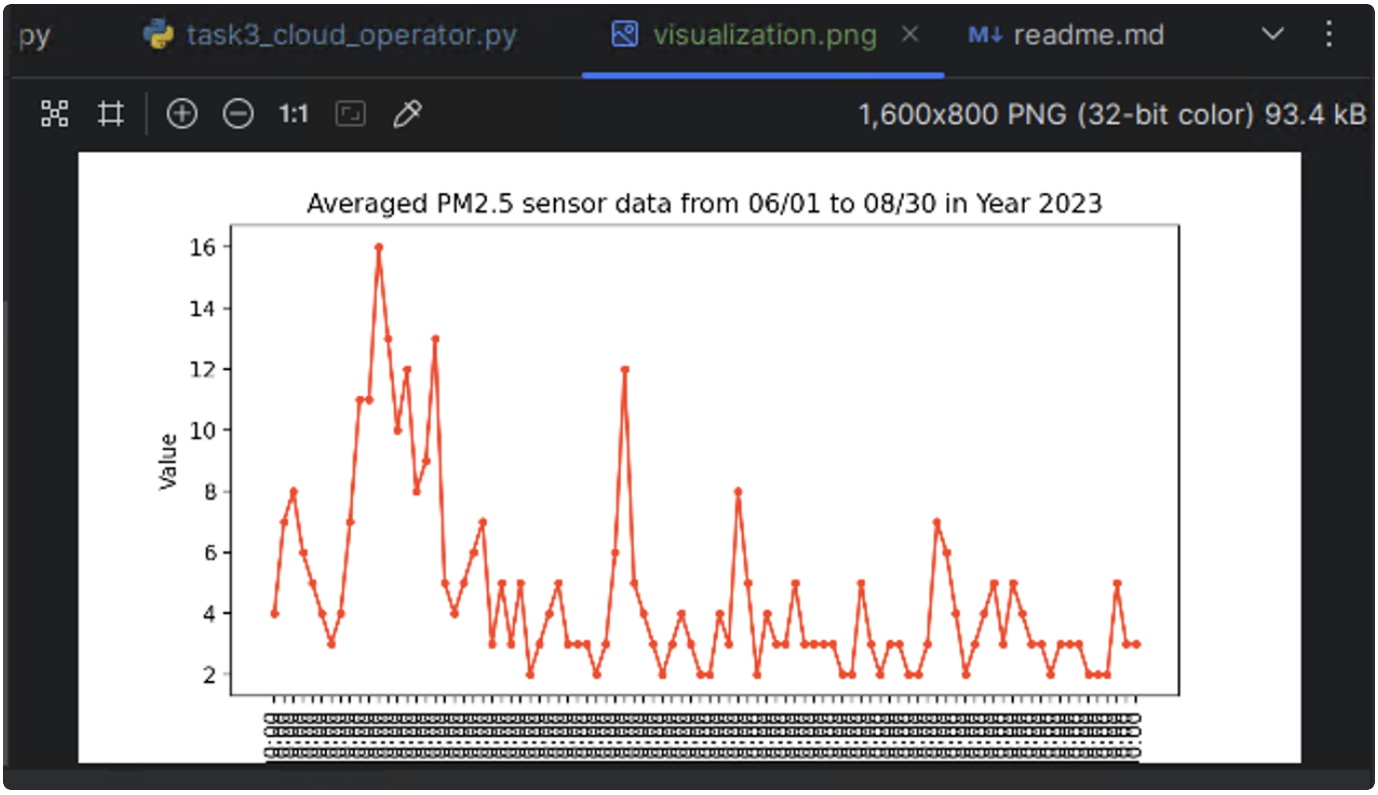
```
def callback(ch, method, properties, body):
    print(f"Got message from producer msg: {json.loads(body)}")
    average_data = json.loads(body)
    Timestamp = []
    Value = []
    for key, value in average_data.items():
        average_value = int(value)
        Value.append(average_value)
        Timestamp.append(f"{key}")
    data = {
        'Timestamp': Timestamp,
        'Value': Value
    }
    print(data)
```

There are two lines of data below in the console. The first line is for "a." and the second line is for "b".



c. Use the line chart component of matplotlib to visualize averaged PM2.5 daily data, directly display the figure or save it as a file.

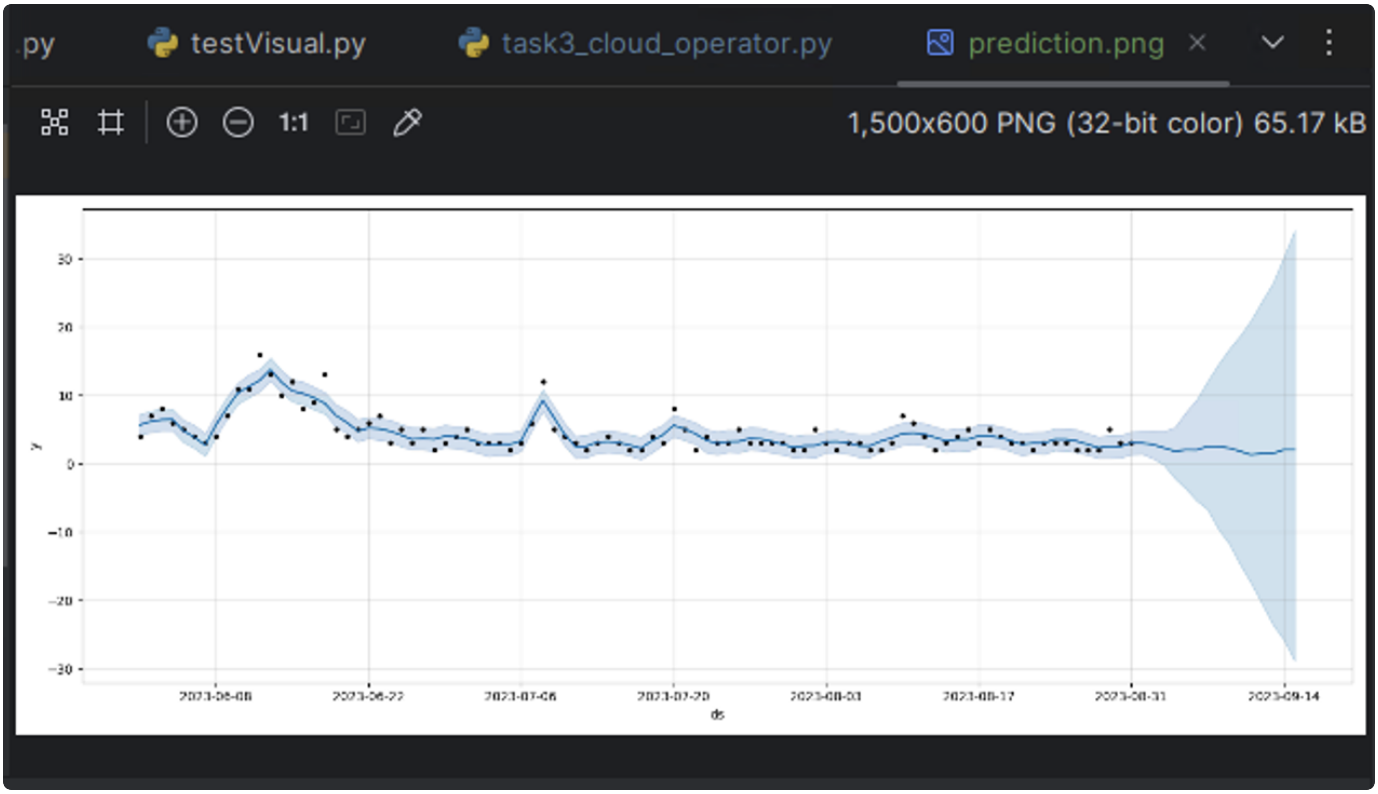
```python
# Visualization
data_df = pd.DataFrame(data)
# Initialize a canvas
plt.figure(figsize=(8, 4), dpi=200)
# Plot data into canvas
plt.plot(data_df["Timestamp"], data_df["Value"], color="#FF3B1D", marker='.', linestyle="-")
plt.title("Averaged PM2.5 sensor data from 06/01 to 08/30 in Year 2023")
plt.xlabel("DateTime")
plt.ylabel("Value")
# Rotate the scale label to display vertically
plt.xticks(Timestamp, rotation='vertical')
# Save as file
plt.savefig("visualization.png")
```

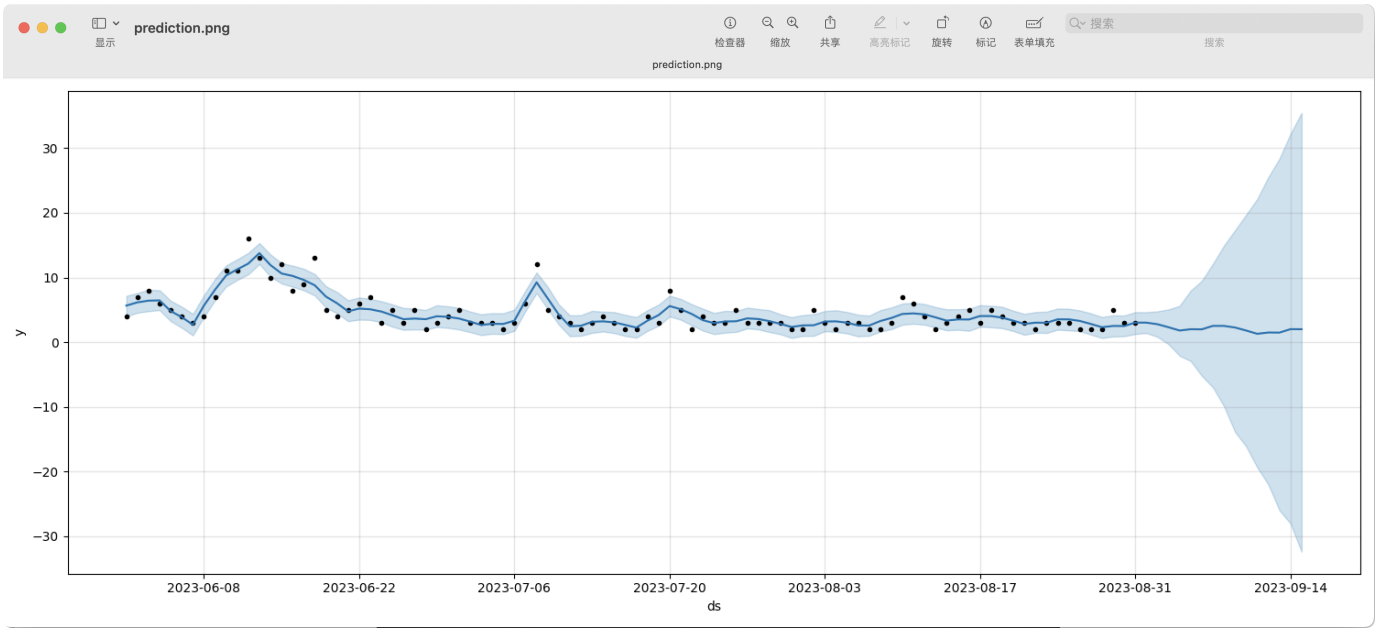


d. Feed averaged PM2.5 data to machine learning model to predict the trend of PM2.5 for the next 15 days (this predicted time period is a default setting of provided machine learning predictor/classifier model).

```
# Format Timestamp to %Y-%m-%d
formatted_Timestamp = [datetime.strptime(timestamp, '%Y-%m-%d %H:%M:%S').strftime('%Y-%m-%d') for timestamp in Timestamp
print(formatted_Timestamp)

# Prepare data
data = {
    'Timestamp': formatted_Timestamp,
    'Value': Value
}
data_df = pd.DataFrame(data)

# Create ML engine predictor object
predictor = MLPredictor(data_df)
# Train ML model
predictor.train()
# Do prediction
forecast = predictor.predict()

# Get canvas
fig = predictor.plot_result(forecast)
fig.savefig("prediction.png")
fig.show()
```

e. Visualize predicted results from Machine Learning predictor/classifier model, directly display the figure or save as it a file (pre-defined in the provided Machine Learning code)





Part B: Analytical discussion of the results and related conclusions.

PM2.5 refers to particulate matter with a diameter of 2.5 microns or less. These particles can penetrate deep into the lungs and even enter the bloodstream, posing a serious threat to human health. PM2.5 can be produced from various sources, including vehicle exhaust, industrial emissions, volcanic eruptions, and dust storms. Due to the significant health concerns associated with PM2.5, forecasting its trends is crucial for issuing health recommendations and taking preventive measures for at-risk populations.

The input dataset has two attributes: a Timestamp and a Value column, as observed from "prediction.png." The axis labeled "ds" corresponds to the "Timestamp," and "y" corresponds to the "Value," where "ds" represents the date/time and "y" is the metric being predicted. In the figure, black dots represent historical data points, the dark blue line indicates the predicted trend, and the light blue shaded area denotes the uncertainty interval.

It is apparent that the prediction model detects change points, indicating significant shifts in the trend of PM2.5 levels at positions corresponding to "2023-06-10" and "2023-07-07." These shifts could be attributed to various factors. The historical input data show that PM2.5 values have remained between 0 and 10, except at the change points. The forecast for the next 15 days suggests that PM2.5 levels are expected to decrease slightly in the short term and then remain relatively stable towards the end of the forecast period.

However, due to the small size of the input dataset and the presence of many unpredictable factors, the uncertainty in the predictions increases rapidly for data points further out in time.