Task 1: Design a data injector component by leveraging Newcastle Urban Observatory IoT data streams:

- Develop a data Injector Component in Edge Layer
- Collect data from urban observatory platform.
- Send all Pm2.5 data to be used by Task 2.2(a) to EMQX service of azure lab (Edge)

For task 1 Docker compose file is used which has MQTT Publisher ,MQTT Subscriber , and code to pull emqx,emqx image.

Compose.yml

```
version: "3"
services:
  mqtt_service:
    image: emqx/emqx
    container_name: 'task1'
    ports:
      - 1883:1883
      - 8083:8083
      - 8084:8084
      - 8883:8883
      - 18083:18083
  data_injector:
    image: data_injector:latest
    command: python3 mqttPublisher.py
    depends_on:
      mqtt_service:
        condition: service_healthy
        restart: true
  data_pp:
    image: data_pp:latest
    command: python3 mqttSubscriber.py
```

Using the below code we will pull the emqx/emqx image from the docker hub throught the docker compose.

```
    image: emqx/emqx
    container_name: 'task1'
    ports:
      - 1883:1883
      - 8083:8083
      - 8084:8084
      - 8883:8883
      - 18083:18083
```

Using the **docker-compose up** command in the edge layer first emqx/emqx image will be pulled necessary port are written in the docker compose file, then data is collected from the urban observatory and PM 2.5 data is extracted from it. And then send over emqx service which will further used in task 2

MQTT Publisher Code

```
import json
import requests


from paho.mqtt import client as mqtt_client

mqtt_ip = "192.168.0.102"
mqtt_port = 1883
topic = "CSC8112"




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

# Callback function for MQTT connection
def on_connect(client, userdata, flags, rc):
    if rc == 0:
        print("Connected to MQTT OK!")
    else:
        print("Failed to connect, return code %d\n", rc)

# 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
# Target URL
url =
"https://newcastle.urbanobservatory.ac.uk/api/v1.1/sensors/PER_AIRMON_MONITOR1135100/data/json/?star
ttime=20230601&endtime=20230831"


# Request data from Urban Observatory Platform
resp = requests.get(url)

# Convert response(Json) to dictionary format
raw_data_dict = resp.json()

data = raw_data_dict["sensors"][0]["data"]["PM2.5"]
sensordata = list()

for index in data:
    timeStamp = index["Timestamp"]
    value = index["Value"]
    msg = {
        "timestamp" : timeStamp,
        "value" : value
    }
```

```
    sensordata.append(msg)
print(sensordata)
msg = json.dumps(sensordata)

client.publish(topic, msg)
```

MQTT Subscriber

```
import  json
import time
import pika
from paho.mqtt import client as mqtt_client

if __name__ == '__main__':
    mqtt_ip = "192.168.0.102"
    mqtt_port = 1883
    topic = "CSC8112"

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


    # Callback function for MQTT connection
    def on_connect(client, userdata, flags, rc):
        if rc == 0:
            print("Connected to MQTT OK!")
        else:
            print("Failed to connect, return code %d\n", rc)


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


    # Callback function will be triggered
    def on_message(client, userdata, msg):
        data_record = json.loads(msg.payload)
        dayWiseData = dict()
        avgDayWiseData = dict()

        for index in data_record:
            timeStamp = index["timestamp"]
            value = float(index["value"])
            # dateValue = time.strftime("%Y-%m-%d %H:%M:%S", time.localtime(timeStamp / 1000))
            # dateValue = time.strftime("%Y-%m-%d", time.localtime(timeStamp / 1000))
            if value > 50.0:

                # editted
                dateValue = time.strftime("%Y-%m-%d", time.localtime(timeStamp / 1000))
                print(dateValue, "::", value)
                #
                # dayWiseData[dateValue] = dayWiseData.get(dateValue, {"total": 0, "nRecords": 0})
                # dayWiseData[dateValue]["total"] = dayWiseData.get(dateValue, {}).get("total", 0) + value
                # dayWiseData[dateValue]["nRecords"] = dayWiseData.get(dateValue, {}).get("nRecords", 0) + 1
            else:
```

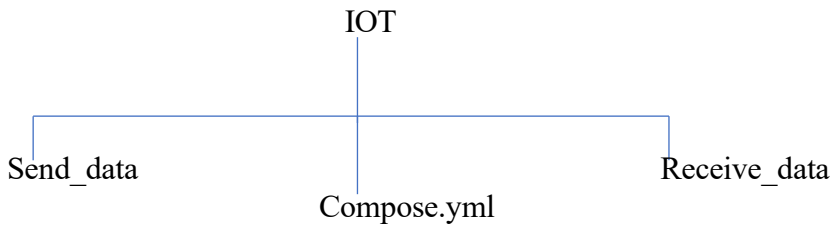
```
            dateValue = time.strftime("%Y-%m-%d 00:00:00", time.localtime(timeStamp / 1000))

            dayWiseData[dateValue] = dayWiseData.get(dateValue, {"total": 0, "nRecords": 0})
            dayWiseData[dateValue]["total"] = dayWiseData.get(dateValue, {}).get("total", 0) + value
            dayWiseData[dateValue]["nRecords"] = dayWiseData.get(dateValue, {}).get("nRecords", 0) + 1
        #     print(value)
    for indx in dayWiseData.items():
        avgDayWiseData[indx[0]] = round(indx[1].get("total") / indx[1].get("nRecords"), 4)

    print(avgDayWiseData)
    print(len(avgDayWiseData))

    print("Sending to rabbit MQ")
    # for index in avgDayWiseData.items():
    #     print(f"Reading sensor data from Urban Observatory: {index}")

    rabbitmq_ip = "192.168.0.100"
    rabbitmq_port = 5672
    # Queue name
    rabbitmq_queque = "CSC8112"
    msg = avgDayWiseData
    # Connect to RabbitMQ service
    connection = pika.BlockingConnection(pika.ConnectionParameters(host=rabbitmq_ip,
port=rabbitmq_port))
    print("Connecting to rabbit MQ")
    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(msg))

    connection.close()


# Subscribe MQTT topic
client.subscribe(topic)
client.on_message = on_message

# Start a thread to monitor message from publisher
client.loop_forever()
```

Raw data from the urban observatory

Figure shows the raw data.

Fig 1.1

Edge Layer has IOT Folder



Send_data directory has.
->Publishermqtt(Publishertask1)
->Dockerfile

Receive_data directory has.
->Subscribermqtt(Subscribertask1)
->Dockerfile

Docker file for Publishertask1 that is Publishermqtt

```
FROM python:3.8.12

USER root

ADD . /usr/local/source

WORKDIR /usr/local/source

RUN pip3 install -r requirements.txt

CMD python3 Publishertask1.py
```

Docker file for Subscribertask1 that is Subscribermqtt

```
FROM python:3.8.12

USER root

ADD . /usr/local/source

WORKDIR /usr/local/source

RUN python3 -m pip install pika --upgrade
RUN pip3 install paho-mqtt


CMD python3 Subscribertask1.py
```

Task 2: Data preprocessing operator design:

- (A)Download and run RabbitMQ image
- (B)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 (so the logs can be seen in the docker.
- (C)Filter out outliers (the value greater than 50), and please print out outliers the console.
- (D)Since the original PM2.5 data readings are collected every 15 mins, so please implement 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

Docker Compose file for Cloud tier which can be executed using ***docker-compose up*** command.

```yaml
version: "3.2"
services:
  rabbitmq:
    image: rabbitmq:management
    container_name: 'task2'
    ports:
      - 5672:5672
      - 15672:15672
  cloudtask3:
    image: cloudtask3:latest
    command: python3 Rabbitrec.py
```

Task 2.(A) Image will be downloaded and running on the cloud tier when docker compose is executed in the cloud tier.

After running the Docker Compose RabbitMQ image will be downloaded.

RabbitMQ Consumer Data

```
import json
import pika
import matplotlib.pyplot as plt import
pandas as pd
from ml_engine import MLPredictor

if __name__ == '__main__': rabbitmq_ip
    = "192.168.0.100"
```

```
    rabbitmq_port = 5672 #
    Queue name
    rabbitmq_queue = "CSC8112"


    def callback(ch, method, properties, body):
        # print(f"Got message from producer msg: {json.loads(body)}") data =
        json.loads(body)
        data_source = {
            'Timestamp': data.keys(), 'Value':
            data.values()
        }
        data_df = pd.DataFrame(data_source)

        # 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("Example data for demonstration")
        plt.xlabel("DateTime") plt.xticks(rotation=90)
        plt.ylabel("Value")

        # Save as file
        plt.savefig("figure1.png") #
        Directly display plt.show()

        # 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()

    # Connect to RabbitMQ service with timeout 1min connection =
    pika.BlockingConnection(
        pika.ConnectionParameters(host=rabbitmq_ip, port=rabbitmq_port,
socket_timeout=60))
    channel = connection.channel() #
    Declare a queue
    channel.queue_declare(queue=rabbitmq_queque)

    channel.basic_consume(queue=rabbitmq_queque,
                          auto_ack=True,
```

```
                                on_message_callback=callback)

    channel.start_consuming()
```

Task 2.(B) Outliner(which are greater than 50) of P.M 2.5 data

```
2023-06-15 :: 52.17
2023-06-15 :: 69.76
2023-06-15 :: 76.49
2023-06-15 :: 67.01
2023-06-15 :: 59.91
```

Fig 2.2

Task2.(C)Data of average calculated value of P.M 2.5 data on daily basis.

```
2023-06-15 :: 52.17
2023-06-15 :: 69.76
2023-06-15 :: 76.49
2023-06-15 :: 67.01
2023-06-15 :: 59.91
{'2023-06-01 00:00:00': 4.3859, '2023-06-02 00:00:00': 7.3664, '2023-06-03 00:00:00': 8.7778, '2023-06-04 00:00:00': 6.6303, '2023-06-05 00:00:00': 5.2609, '2023-06-06 00:00:0
92
```

Fig 2.3

Task 2.(D)Data Pre-Processing part

This code in written in Subscribermqtt and by using Docker-compose up command all the pre-processing will be done.

```
    # Callback function will be triggered def
    on_message(client, userdata, msg):
        data_record = json.loads(msg.payload)
        dayWiseData = dict()
        avgDayWiseData = dict()

        for index in data_record: timeStamp =
            index["timestamp"] value =
            float(index["value"])
```

```
                # dateValue = time.strftime("%Y-%m-%d %H:%M:%S", time.localtime(timeStamp
/ 1000))
                # dateValue = time.strftime("%Y-%m-%d", time.localtime(timeStamp /
1000))
                if value > 50.0:

                    # editted
                    dateValue = time.strftime("%Y-%m-%d", time.localtime(timeStamp /
1000))
                    print(dateValue, "::", value) #
                    # dayWiseData[dateValue] = dayWiseData.get(dateValue, {"total": 0,

"nRecords": 0})    # dayWiseData[dateValue]["total"] = dayWiseData.get(dateValue,
{}).get("total", 0) + value
                    # dayWiseData[dateValue]["nRecords"] = dayWiseData.get(dateValue,
{}).get("nRecords", 0) + 1
                else:
                    dateValue = time.strftime("%Y-%m-%d 00:00:00", time.localtime(timeStamp /
1000))

                    dayWiseData[dateValue] = dayWiseData.get(dateValue, {"total": 0,
"nRecords": 0})
                    dayWiseData[dateValue]["total"] = dayWiseData.get(dateValue,
{}).get("total", 0) + value
                    dayWiseData[dateValue]["nRecords"] = dayWiseData.get(dateValue,
{}).get("nRecords", 0) + 1
                #       print(value)
            for indx in dayWiseData.items():
                avgDayWiseData[indx[0]] = round(indx[1].get("total") / indx[1].get("nRecords"), 4)

            print(avgDayWiseData)
            print(len(avgDayWiseData))

            print("Sending to rabbit MQ")
            # for index in avgDayWiseData.items():
            #       print(f"Reading sensor data from Urban Observatory: {index}")

            rabbitmq_ip = "192.168.0.100"
            rabbitmq_port = 5672 #
            Queue name
            rabbitmq_queque = "CSC8112"
            msg = avgDayWiseData
            # Connect to RabbitMQ service connection
            =
pika.BlockingConnection(pika.ConnectionParameters(host=rabbitmq_ip, port=rabbitmq_port))
            print("Connecting to rabbit MQ") 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(msg))

        connection.close()


    # Subscribe MQTT topic
    client.subscribe(topic) client.on_message
    = on_message

    # Start a thread to monitor message from publisher
    client.loop_forever()
```

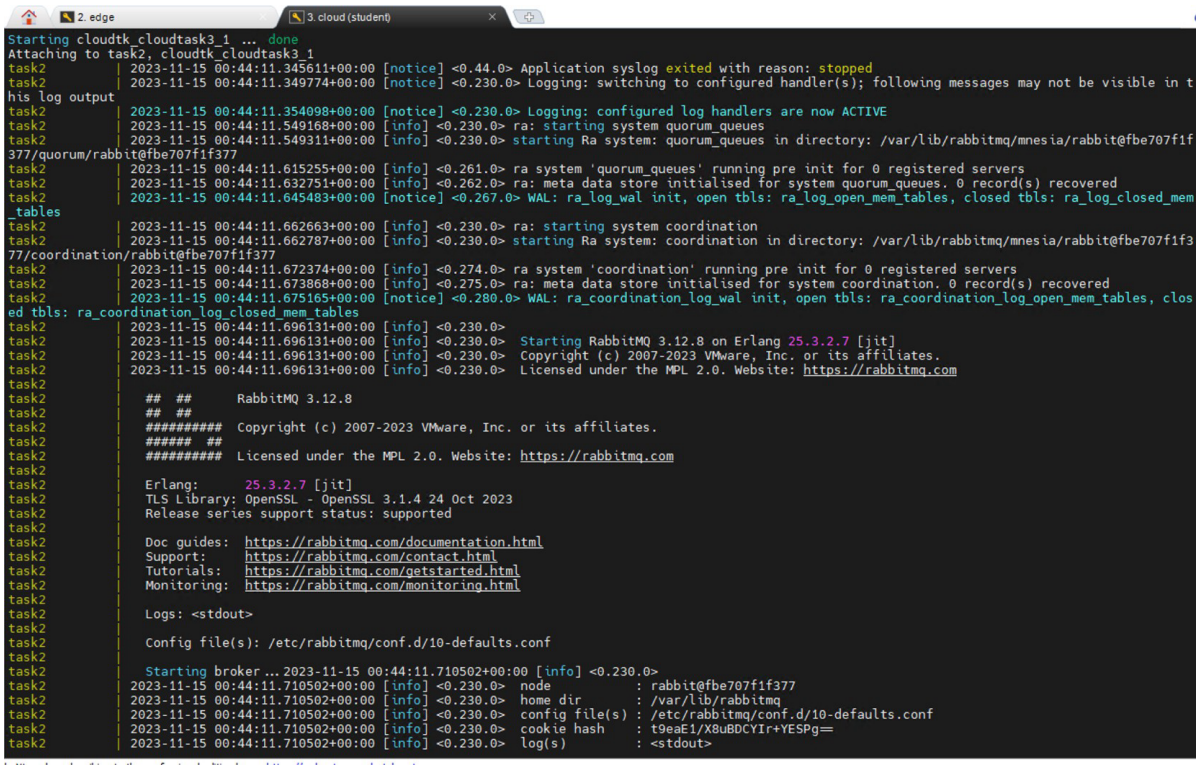


Fig 2.4

logs of docker compose in cloud terminal.

Task 3: Time-series data prediction and visualization

- (A)Download a pre-defined Machine leaning (ML) engine code
- (B) Collect all averaged daily PM2.5 data computed by Task 2.2 (d) from RabbitMQ service, and please print out them to the console.
- (C) 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.
- (D) Use the line chart component of matplotlib to visualize averaged PM2.5 daily data, directly display the figure or save it as a file.
- (E) 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).
- (F) 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

Task 3.(A)shows the Pre-defined Machine Learning code:

```
'''
    Author: Rui Sun
    Date: 2022-09-25

    Predict timeseries data

    Official guide book of Prophet: https://facebook.github.io/prophet/docs/quick_start.html#python-api
'''

from prophet import Prophet


class MLPredictor(object):
    '''
    Example usage method:

        from ml_engine import MLPredictor

        predictor = MLPredictor(pm25_df)
        predictor.train()
        forecast = predictor.predict()


        fig = predictor.plot_result(forecast)
        fig.savefig(os.path.join("Your target dir path", "Your target file name))




'''
```

```
def __init__(self, data_df):
    '''
    :param data_df: Dataframe type dataset
    '''
    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)
    print(f"After rename columns \n{data_df.columns}")
    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
```

Task 3.(B) Timestamp and value


358}, {'timestamp': 1693299600000, 'value': 6.333}, {'timestamp': 1693300500000, 'value': 5.563}, {'timestamp': 1693301400000, 'value': 5.368}
': 1693302300000, 'value': 5.281}, {'timestamp': 1693303200000, 'value': 5.12}, {'timestamp': 1693304100000, 'value': 4.975}, {'timestamp': 16
'value': 5.551}, {'timestamp': 1693305900000, 'value': 5.652}, {'timestamp': 1693306800000, 'value': 5.103}, {'timestamp': 1693307700000, 'val
{'timestamp': 1693308600000, 'value': 4.441}, {'timestamp': 1693309500000, 'value': 4.55}, {'timestamp': 1693310400000, 'value': 3.941}, {'tim
8311300000, 'value': 4.1}, {'timestamp': 1693312200000, 'value': 3.744}, {'timestamp': 1693313100000, 'value': 3.254}, {'timestamp': 1693314000
3.367}, {'timestamp': 1693314900000, 'value': 2.553}, {'timestamp': 1693315800000, 'value': 2.926}, {'timestamp': 1693316700000, 'value': 2.8
amp': 1693317600000, 'value': 3.292}, {'timestamp': 1693318500000, 'value': 4.222}, {'timestamp': 1693319400000, 'value': 4.52}, {'timestamp':
0, 'value': 4.713}, {'timestamp': 1693321200000, 'value': 5.859}, {'timestamp': 1693322100000, 'value': 6.003}, {'timestamp': 1693323000000,
}, {'timestamp': 1693323900000, 'value': 6.617}, {'timestamp': 1693324800000, 'value': 7.044}, {'timestamp': 1693325700000, 'value': 6.475},
1693326600000, 'value': 5.816}, {'timestamp': 1693327500000, 'value': 6.093}, {'timestamp': 1693328400000, 'value': 5.555}, {'timestamp': 1693
alue': 4.755}, {'timestamp': 1693330200000, 'value': 4.125}, {'timestamp': 1693331100000, 'value': 4.279}, {'timestamp': 1693332000000, 'value
timestamp': 1693332900000, 'value': 4.308}, {'timestamp': 1693333800000, 'value': 4.38}, {'timestamp': 1693334700000, 'value': 4.638}, {'times
5600000, 'value': 4.571}, {'timestamp': 1693336500000, 'value': 4.052}, {'timestamp': 1693337400000, 'value': 3.669}, {'timestamp': 1693338300
3.894}, {'timestamp': 1693339200000, 'value': 5.048}, {'timestamp': 1693340100000, 'value': 6.893}, {'timestamp': 1693341000000, 'value': 4.9
amp': 1693341900000, 'value': 3.315}, {'timestamp': 1693342800000, 'value': 3.35}, {'timestamp': 1693343700000, 'value': 3.187}, {'timestamp':
0, 'value': 2.807}, {'timestamp': 1693345500000, 'value': 2.747}, {'timestamp': 1693346400000, 'value': 2.693}, {'timestamp': 1693347300000,
}, {'timestamp': 1693348200000, 'value': 2.936}, {'timestamp': 1693349100000, 'value': 3.132}, {'timestamp': 1693350000000, 'value': 3.281},
1693350900000, 'value': 3.21}, {'timestamp': 1693351800000, 'value': 3.158}, {'timestamp': 1693352700000, 'value': 3.392}, {'timestamp': 1693
alue': 3.171}, {'timestamp': 1693354500000, 'value': 3.183}, {'timestamp': 1693355400000, 'value': 3.334}, {'timestamp': 1693356300000, 'value
imestamp': 1693357200000, 'value': 3.25}, {'timestamp': 1693358100000, 'value': 3.401}, {'timestamp': 1693359000000, 'value': 3.474}, {'timesta
000000, 'value': 3.367}, {'timestamp': 1693360800000, 'value': 3.281}, {'timestamp': 1693361700000, 'value': 3.235}, {'timestamp': 169336260000
3.19}, {'timestamp': 1693363500000, 'value': 3.367}, {'timestamp': 1693364400000, 'value': 3.81}, {'timestamp': 1693365300000, 'value': 3.512},
: 1693366200000, 'value': 3.617}, {'timestamp': 1693367100000, 'value': 3.524}, {'timestamp': 1693368000000, 'value': 3.419}, {'timestamp': 16
'value': 3.579}, {'timestamp': 1693369800000, 'value': 3.69}, {'timestamp': 1693370700000, 'value': 5.248}, {'timestamp': 1693371600000, 'valu
{'timestamp': 1693372500000, 'value': 5.334}, {'timestamp': 1693373400000, 'value': 3.834}, {'timestamp': 1693374300000, 'value': 3.864}, {'tim
375200000, 'value': 4.045}, {'timestamp': 1693376100000, 'value': 4.146}, {'timestamp': 1693377000000, 'value': 3.371}, {'timestamp': 1693377
': 3.438}, {'timestamp': 1693378800000, 'value': 2.943}, {'timestamp': 1693379700000, 'value': 2.954}, {'timestamp': 1693380600000, 'value':
stamp': 1693381500000, 'value': 2.406}, {'timestamp': 1693382400000, 'value': 2.799}, {'timestamp': 1693383300000, 'value': 2.211}, {'timestam
00000, 'value': 2.569}, {'timestamp': 1693385100000, 'value': 2.584}, {'timestamp': 1693386000000, 'value': 2.731}, {'timestamp': 169338690000
995}, {'timestamp': 1693387800000, 'value': 2.373}, {'timestamp': 1693388700000, 'value': 2.741}, {'timestamp': 1693389600000, 'value': 2.907
': 1693390500000, 'value': 2.477}, {'timestamp': 1693391400000, 'value': 2.293}, {'timestamp': 1693392300000, 'value': 2.438}, {'timestamp':
'value': 2.706}, {'timestamp': 1693394100000, 'value': 2.581}, {'timestamp': 1693395000000, 'value': 2.766}, {'timestamp': 1693395900000, 'va
{'timestamp': 1693396800000, 'value': 2.686}, {'timestamp': 1693399500000, 'value': 4.479}, {'timestamp': 1693400400000, 'value': 3.288}, {'t
693401300000, 'value': 3.1}, {'timestamp': 1693402200000, 'value': 3.494}, {'timestamp': 1693403100000, 'value': 3.186}, {'timestamp': 16934040
': 3.36}, {'timestamp': 1693404900000, 'value': 3.644}, {'timestamp': 1693405800000, 'value': 3.457}, {'timestamp': 1693406700000, 'value': 3.
tamp': 1693407600000, 'value': 4.674}, {'timestamp': 1693408500000, 'value': 3.315}, {'timestamp': 1693409400000, 'value': 3.801}, {'timestamp
0000, 'value': 3.685}, {'timestamp': 1693411200000, 'value': 3.245}, {'timestamp': 1693412100000, 'value': 2.773}, {'timestamp': 1693413000000,
865}, {'timestamp': 1693413900000, 'value': 2.849}, {'timestamp': 1693414800000, 'value': 3.078}, {'timestamp': 1693415700000, 'value': 3.111},
: 1693416600000, 'value': 2.737}, {'timestamp': 1693417500000, 'value': 2.841}, {'timestamp': 1693418400000, 'value': 2.911}, {'timestamp': 16
'value': 5.524}, {'timestamp': 1693420200000, 'value': 4.719}, {'timestamp': 1693421100000, 'value': 3.298}, {'timestamp': 1693422000000, 'val
{'timestamp': 1693422900000, 'value': 4.692}, {'timestamp': 1693423800000, 'value': 4.193}, {'timestamp': 1693424700000, 'value': 4.923}, {'ti
93425600000, 'value': 4.787}, {'timestamp': 1693426500000, 'value': 6.005}, {'timestamp': 1693427400000, 'value': 5.039}, {'timestamp': 1693428
e': 5.225}, {'timestamp': 1693429200000, 'value': 5.994}, {'timestamp': 1693430100000, 'value': 6.484}, {'timestamp': 1693431000000, 'value':

Fig 3.1

Task 3.(C)Conversion of timestamp to date time format(years-months-days-hours-minutes=seconds)

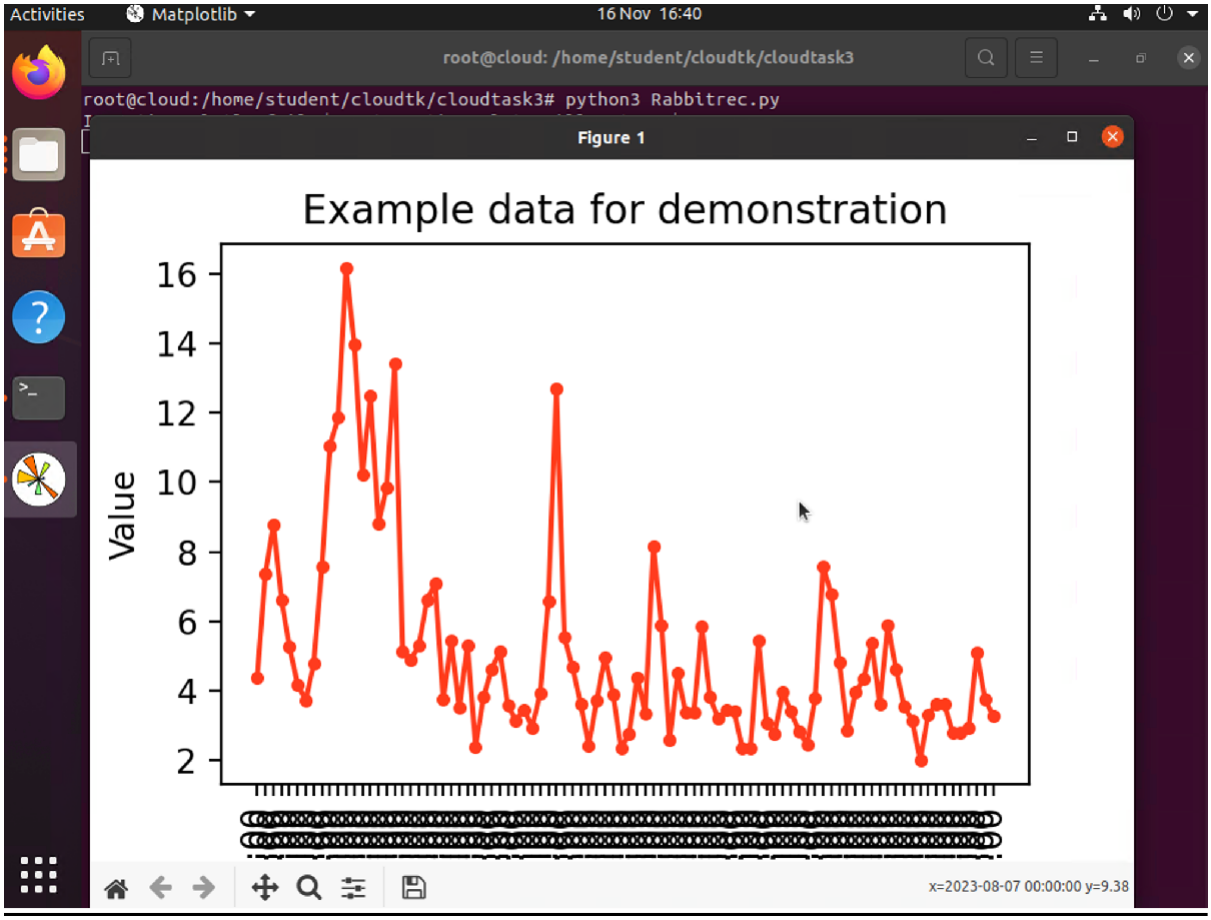

{'2023-06-01 00:00:00': 4.3859, '2023-06-02 00:00:00': 7.3664, '2023-06-03 00:00:00': 8.7778, '2023-06-04 00:00:00': 6.6303, '2023-06-05 00:00:00': 5.2609, '2023-06-06 00:00:0
92

Fig 3.2

Docker file for RabbitMQ(Rabbitrec.py)

```
FROM python:3.8.12

USER root

ADD . /usr/local/source

WORKDIR /usr/local/source

RUN pip3 install pika
RUN pip3 install matplotlib
RUN pip3 install pandas

CMD python3 Rabbitrec.py
```

Task 3(D) Visualised image of P.M 2.5 data in cloud terminal



3.3

Task 3(E)Machine Learning Model to predict trend of PM 2.5 data.

```
from ml_engine import MLPredictor import matplotlib.pyplot as plt import json import pandas as
pd
if __name__ == '__main__':
# Prepare data
file = open('average.json') data =
json.load(file)
data_str = json.dumps(data) data_df = pd.read_json(data_str) #
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()
```

Task 3.(F) The predication graph using the average P.M 2.5 data in the cloud terminal.

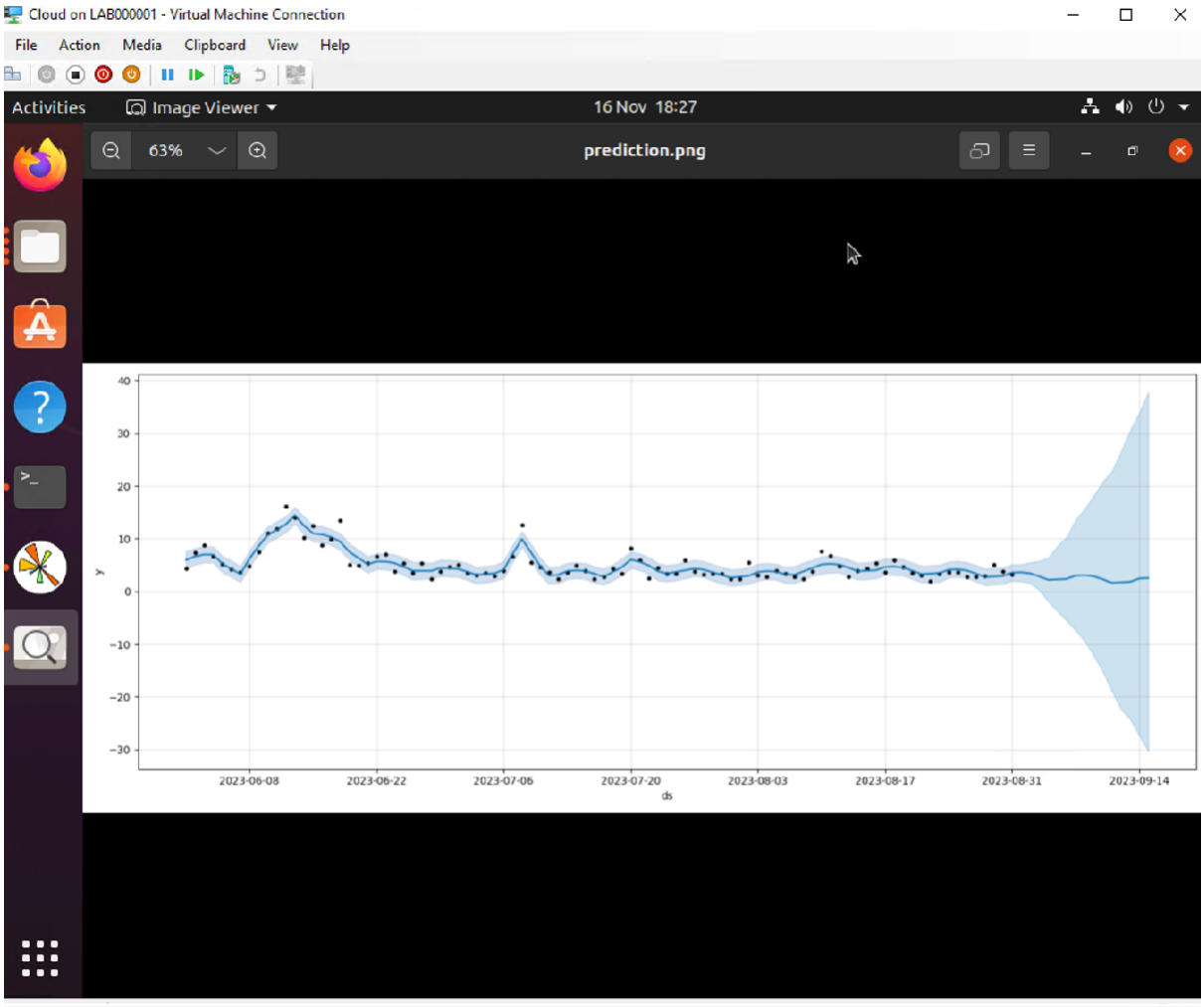

Cloud on LAB000001 - Virtual Machine Connection
File Action Media Clipboard View Help
Activities
Image Viewer
16 Nov 18:27
63%
prediction.png
40
30
20
10
0
−10
−20
−30
y
2023-06-08
2023-06-22
2023-07-06
2023-07-20
2023-08-03
2023-08-17
2023-08-31
2023-09-14
ds

Fig 3.4

Graphs are generated and stored in cloud tier

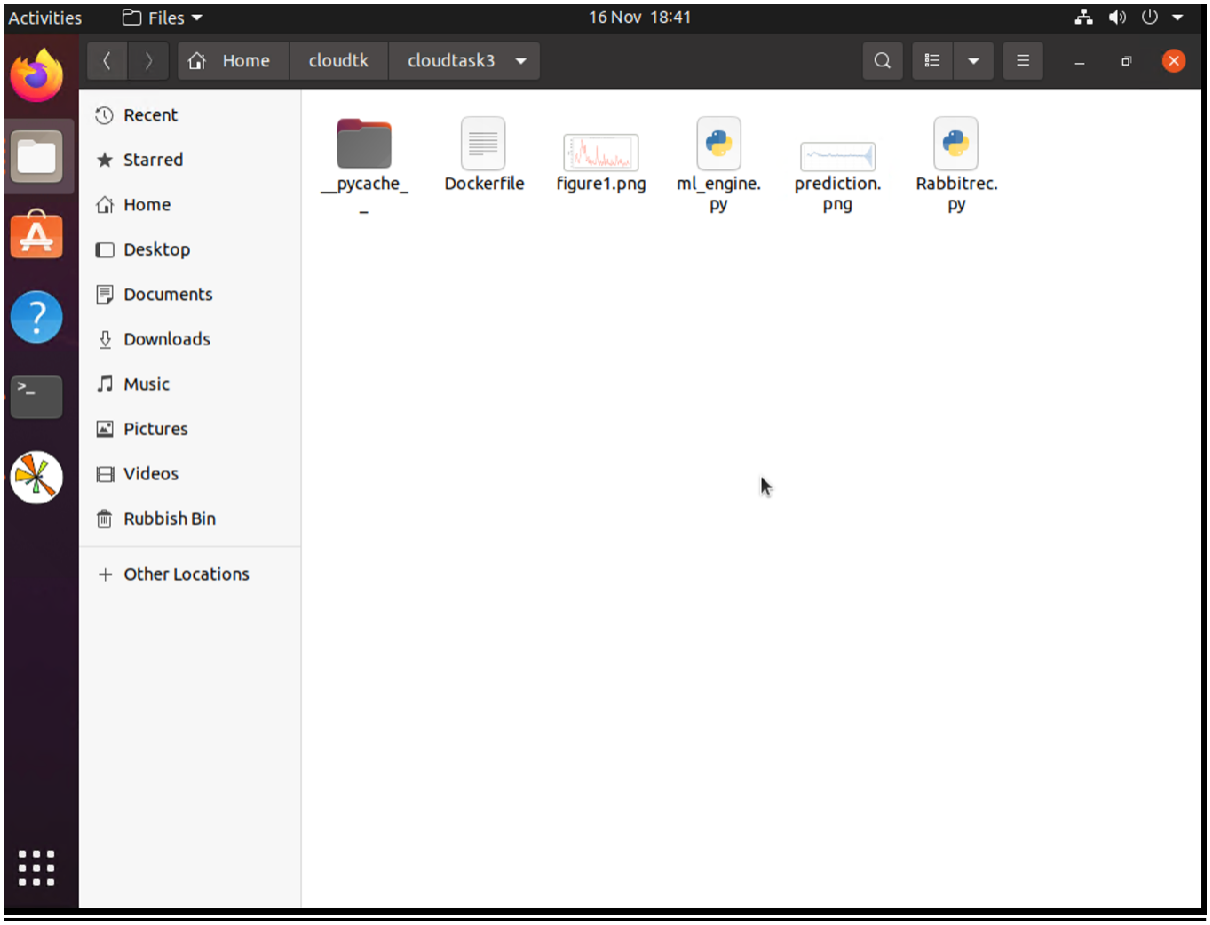


Fig 3.5

Analytical Discussion:

P.M 2.5 data some particles are too dangerous than others, such as industrial pollution and air pollution which can cause many disease to human body which are invisible to human naked eyes. It generally comes from the car bike which we use on daily bases and from industry. By plotting the prediction of P.M2.5 data we can understand that in June and July month the value is high and later it comes down slightly.