INTERNET OF THINGS
CSC-8112
COURSE-WORK REPORT

NAME: KAILASH BALACHANDIRAN
STUDENT ID: C2024316

AIM: To understand how to process Internet of Things (IoT) sensor data in the edge and cloud setting and able to develop the machine learning-based IoT data processing pipeline (data collection, data preprocessing, prediction and visualization) in the edge cloud setting and be able to use a lightweight virtualization technology stack, such as docker, to implement IoT data processing pipeline in the edge cloud setting.

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

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).

   In the below Figure 1.1 shows that emqx/emqx image is pulled from the docker hub and running in the Azure lab (Edge) using the command "*docker run -p 1883:1883 emqx*" ., By using the docker, EMQX broker is running into the Azure lab (Edge) terminal.

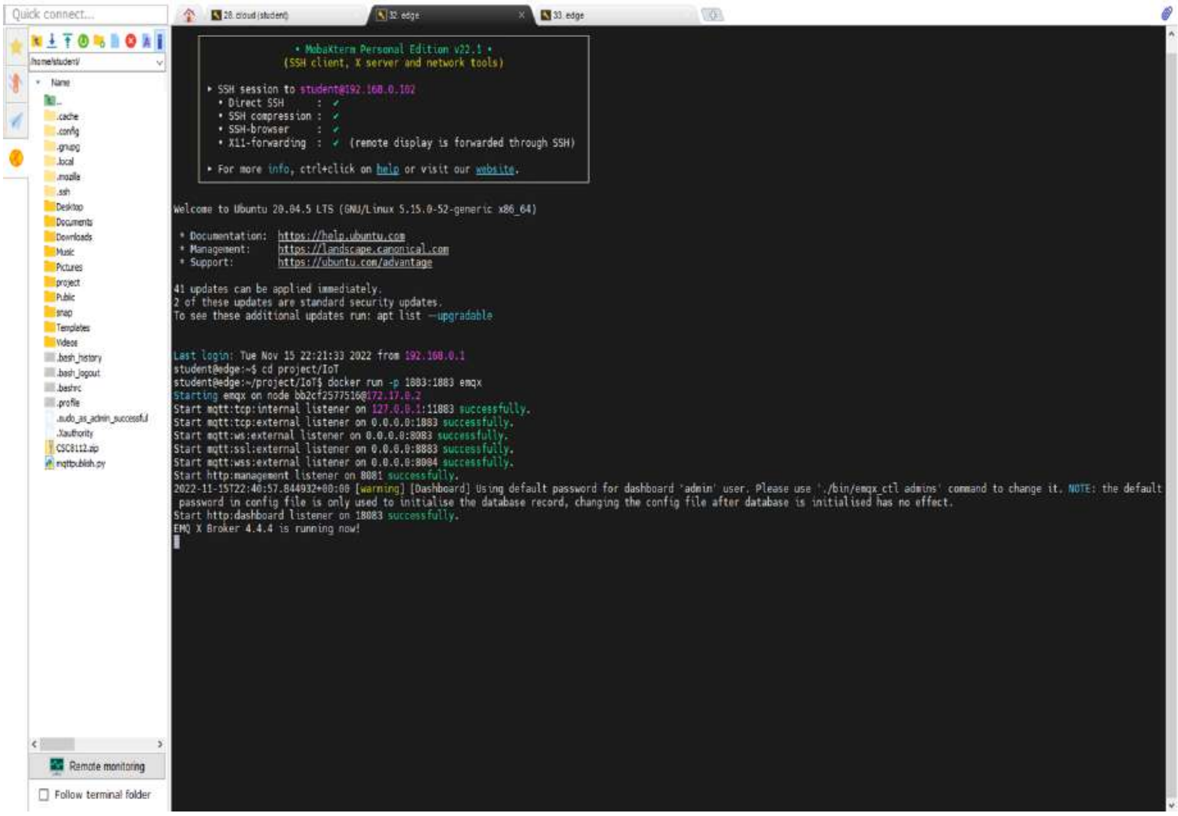


Figure 1.1

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

   In the below MQTT publisher code using the URL, the PM2.5 data is filtered from the urban observatory and sent to the MQTT Subscriber and printing it to the Azure lab (Edge) using the EMQX broker.

MQTT Publisher Code:

```
import json
from paho.mqtt import client as mqtt_client
from urllib.request import urlopen



if __name__ == '__main__':
    mqtt_ip = "192.168.0.102"
```

```
    mqtt_port = 1883
    topic = "CSC8112"

#getting the data from the URL

    url =
"http://uoweb3.ncl.ac.uk/api/v1.1/sensors/PER_AIRMON_MONITOR1135100/data/json/?starttime=20220601
&endtime=20220831"

#Storing the data from the urban observatory into the variable response

    response = urlopen(url)

#data_json variable read the data from the response and stores it to the data_json in the JSON string format

    data_json = json.loads(response.read())

#data_json data further filtered and stores only the PM2.5 variable.

    data = data_json["sensors"][0]["data"]["PM2.5"]

#Using the for loop remaining items are deleted so that only Timestamp and Value will get printed .

    for item in data:
        del item['Sensor Name']
        del item['Flagged as Suspect Reading']
        del item['Units']
        del item['Variable']

    # 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
    msg = json.dumps(data)
    client.publish(topic, msg)
```

MQTT Subscriber Code:

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

rabbitmq_ip = "192.168.0.100"
rabbitmq_port = 5672
# Quseue name
rabbitmq_queque = "CSC8112"


def callback(ch, method, properties, body):
   print(f"Got message from producer msg: {json.loads(body)}")

#The output data of PM2.5 will further be assigned to the msg1 variable.
   msg1 = json.loads(body)

#Output data will be stored in the file called average.json file
   jsonfile = open("average.json", "w")
   jsonfile.write(msg1)
   jsonfile.close()



   # 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)
```

a) Collect data from the Urban Observatory platform by sending an HTTP request to the following URL([http://uoweb3.ncl.ac.uk/api/v1.1/sensors/PER_AIRMON_MONITOR1135100/ data/json/?starttime=20220601&endtime=20220831]). Following that, please print out the raw data streams that you collected on the console.

   Figure 1.2 shows that using the command python3 "*mqttpublish.py*" the raw data is printed in the Azure Lab (Edge)

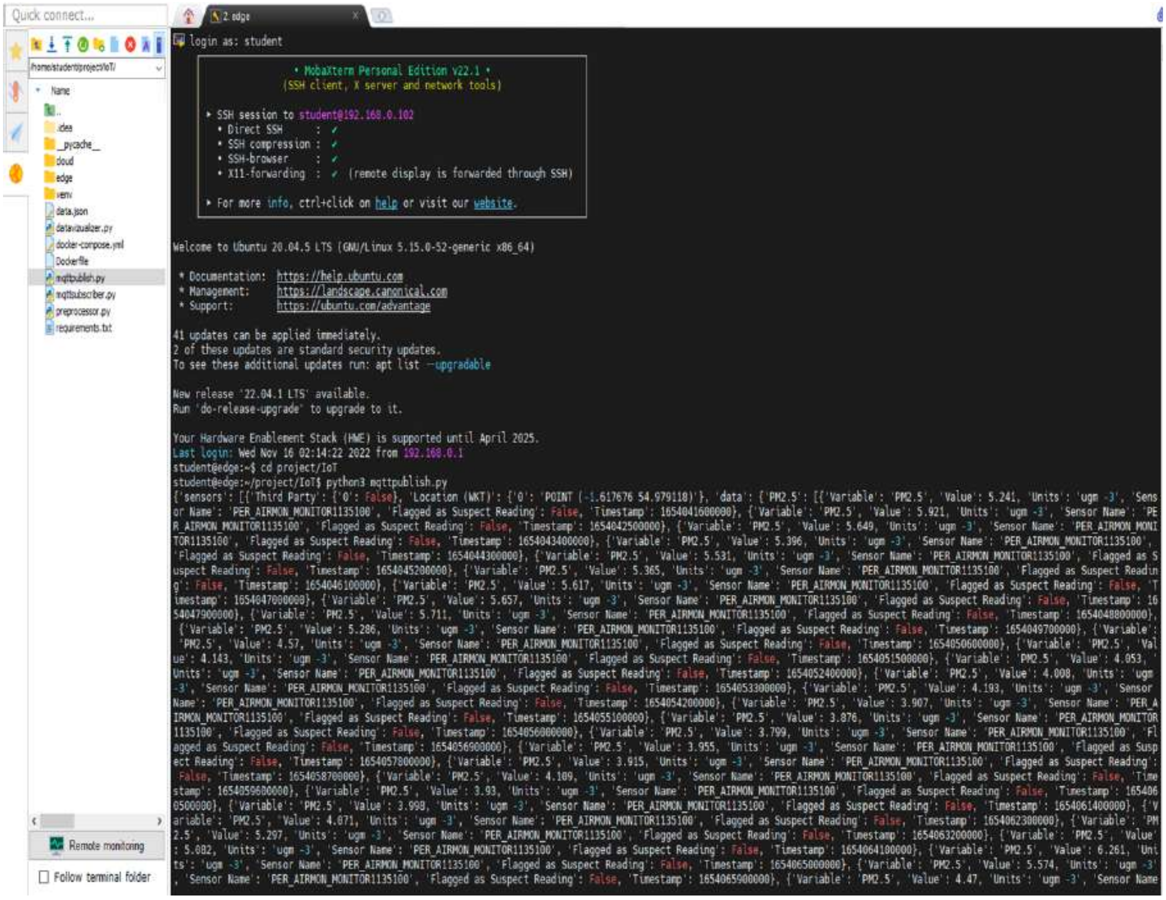


Figure 1.2

b) Although the raw air quality data you collected from the Urban Observatory API contains many metrics including NO2, NO, CO2, 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.

Figure 1.3 shows that using the MQTT Publisher code the raw data is filtered and printed into the Azure Lab (Edge) using the command *"python3 mqttpublish.py"*

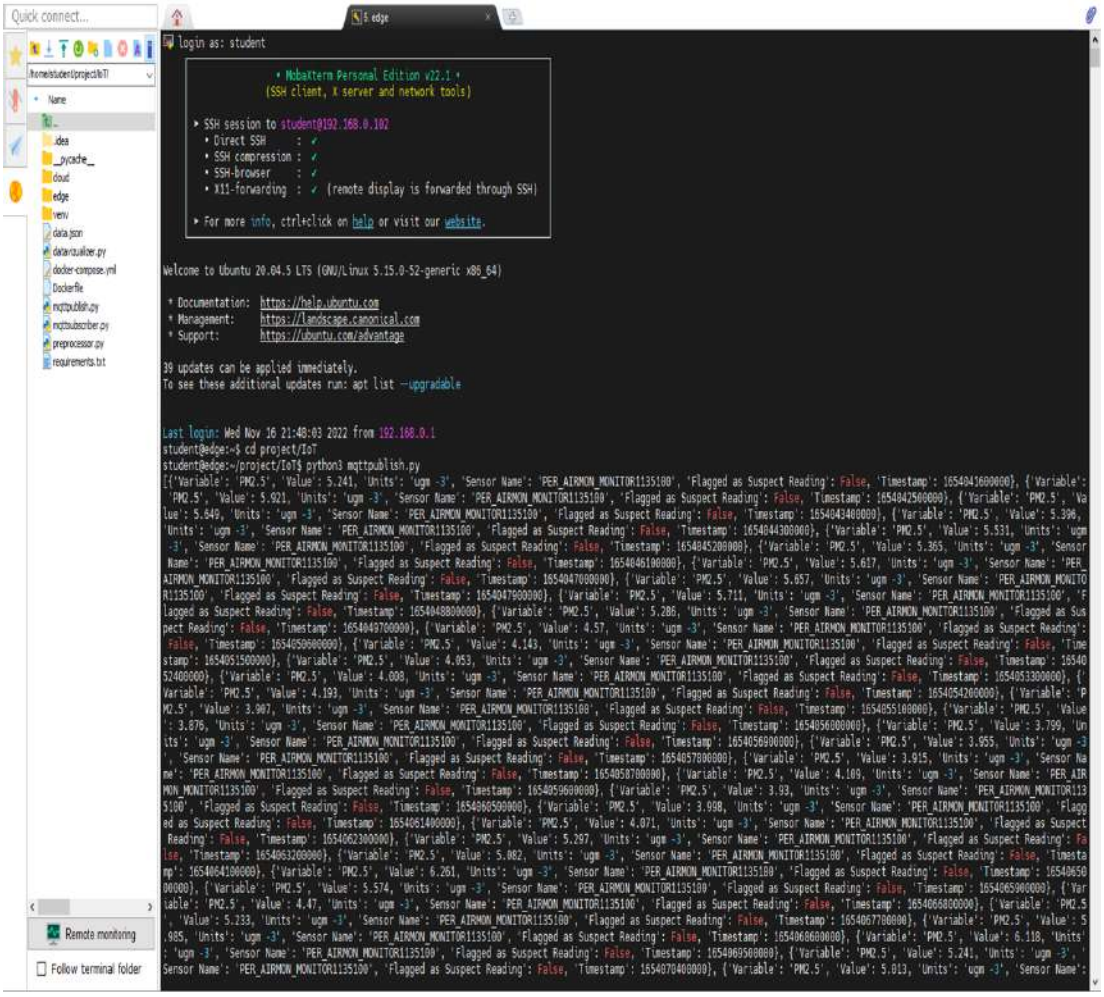


Figure 1.3

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

Figure 1.4 shows that PM2.5 data is further filtered with the Timestamp and Value of PM2.5 and printed into the EMQX service of Azure lab (Edge) using the command "*python3 mqttsubscriber.py*" in the edge.

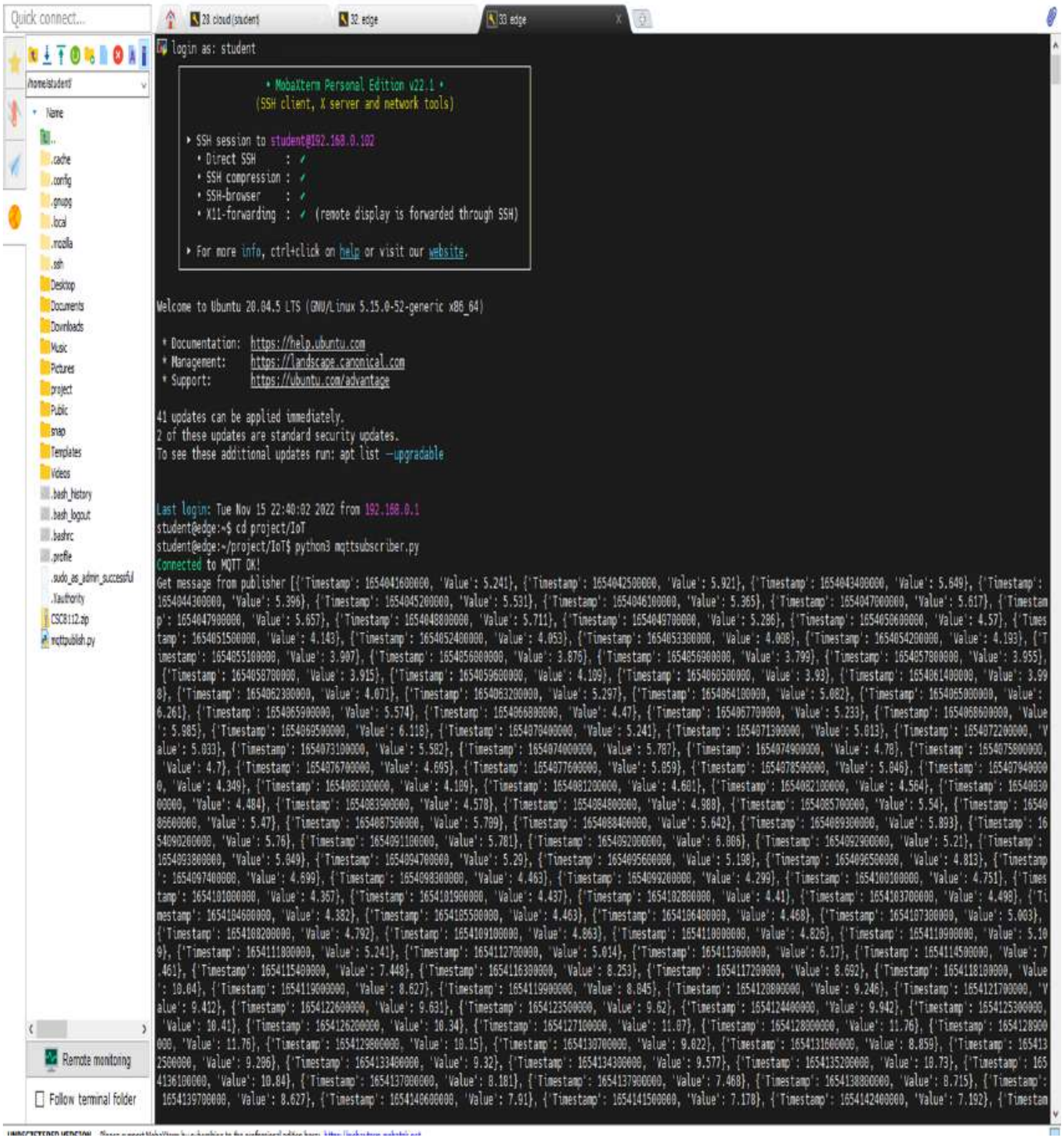


Figure 1.4

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 on Azure lab (Cloud):

   Figure 2.1 shows that the docker image was created for "*data-preprocessing.py*" code which contains the image name, on the Azure lab (Cloud)

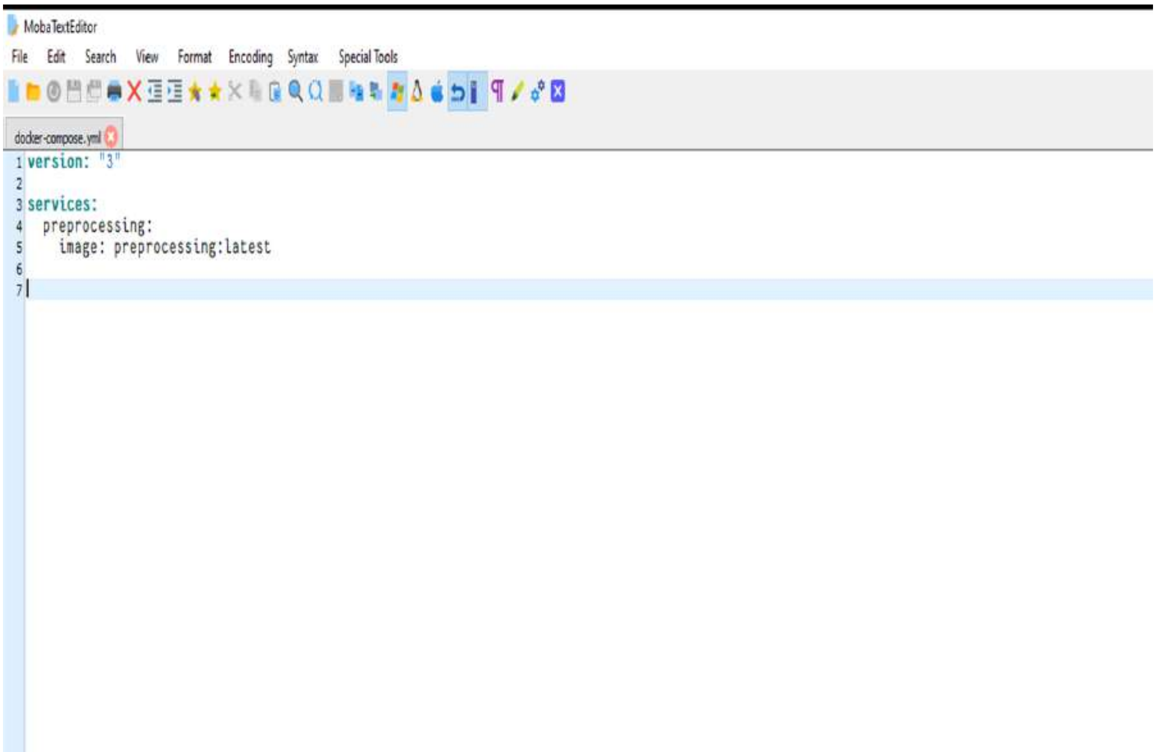


Figure 2.1

Figure 2.2 shows that a docker image was created for the image RabbitMQ, which contains the name of the image and port number of the RabbitMQ on the Azure lab (Cloud)

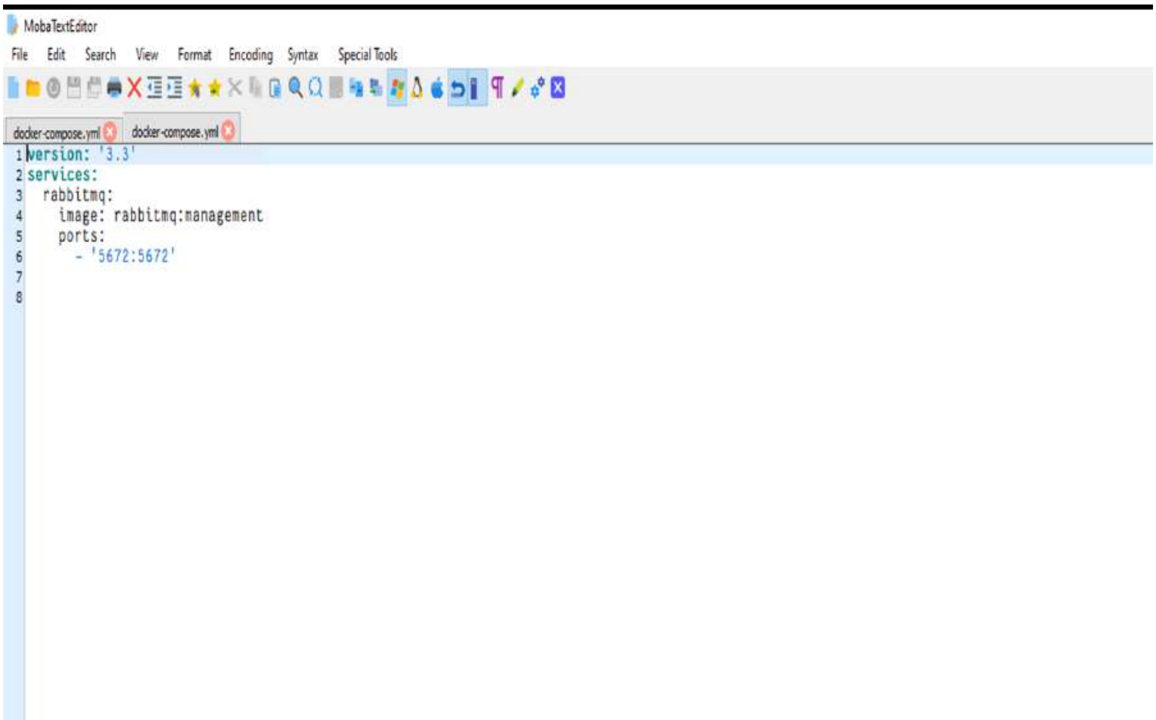
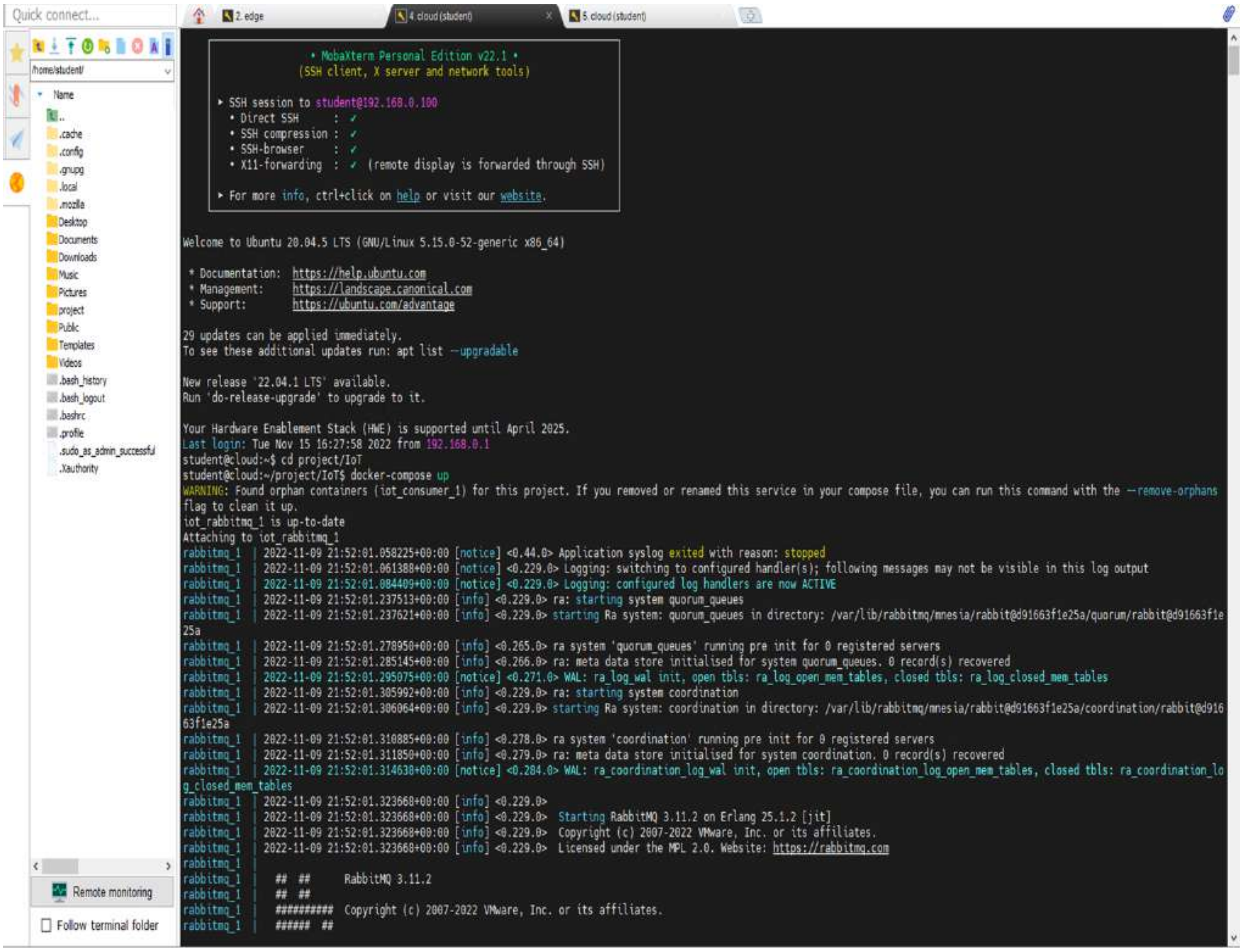


Figure 2.2

a) Download and run the RabbitMQ image (RabbitMQ: management):

Figure 2.3 and 2.4 shows that the RabbitMQ running on the Azure Lab (Cloud) using the command "*docker-compose up*"



• MobaXterm Personal Edition v22.1 •
(SSH client, X server and network tools)
➤ SSH session to student@192.168.0.100
• Direct SSH : ✓
• SSH compression : ✓
• SSH-browser : ✓
• X11-forwarding : ✓ (remote display is forwarded through SSH)
➤ For more info, ctrl+click on help or visit our website.
Welcome to Ubuntu 20.04.5 LTS (GNU/Linux 5.15.0-52-generic x86_64)
* Documentation: https://help.ubuntu.com
* Management: https://landscape.canonical.com
* Support: https://ubuntu.com/advantage
29 updates can be applied immediately.
To see these additional updates run: apt list --upgradable
New release '22.04.1 LTS' available.
Run 'do-release-upgrade' to upgrade to it.
Your Hardware Enablement Stack (HWE) is supported until April 2025.
Last login: Tue Nov 15 16:27:58 2022 from 192.168.0.1
student@cloud:~$ cd project/IoT
student@cloud:~/project/IoT$ docker-compose up
WARNING: Found orphan containers (iot_consumer_1) for this project. If you removed or renamed this service in your compose file, you can run this command with the --remove-orphans flag to clean it up.
iot_rabbitmq_1 is up-to-date
Attaching to iot_rabbitmq_1
rabbitmq_1 | 2022-11-09 21:52:01.058225+00:00 [notice] <0.44.0> Application syslog exited with reason: stopped
rabbitmq_1 | 2022-11-09 21:52:01.061388+00:00 [notice] <0.229.0> Logging: switching to configured handler(s); following messages may not be visible in this log output
rabbitmq_1 | 2022-11-09 21:52:01.084409+00:00 [notice] <0.229.0> Logging: configured log handlers are now ACTIVE
rabbitmq_1 | 2022-11-09 21:52:01.237513+00:00 [info] <0.229.0> ra: starting system quorum_queues
rabbitmq_1 | 2022-11-09 21:52:01.237621+00:00 [info] <0.229.0> starting Ra system: quorum_queues in directory: /var/lib/rabbitmq/mnesia/rabbit@d91663f1e25a/quorum/rabbit@d91663f1e25a
rabbitmq_1 | 2022-11-09 21:52:01.278950+00:00 [info] <0.265.0> ra system 'quorum_queues' running pre init for 0 registered servers
rabbitmq_1 | 2022-11-09 21:52:01.285145+00:00 [info] <0.266.0> ra: meta data store initialised for system quorum_queues. 0 record(s) recovered
rabbitmq_1 | 2022-11-09 21:52:01.295075+00:00 [notice] <0.271.0> WAL: ra_log_wal init, open tbls: ra_log_open_mem_tables, closed tbls: ra_log_closed_mem_tables
rabbitmq_1 | 2022-11-09 21:52:01.305992+00:00 [info] <0.229.0> ra: starting system coordination
rabbitmq_1 | 2022-11-09 21:52:01.306064+00:00 [info] <0.229.0> starting Ra system: coordination in directory: /var/lib/rabbitmq/mnesia/rabbit@d91663f1e25a/coordination/rabbit@d91663f1e25a
rabbitmq_1 | 2022-11-09 21:52:01.310885+00:00 [info] <0.278.0> ra system 'coordination' running pre init for 0 registered servers
rabbitmq_1 | 2022-11-09 21:52:01.311850+00:00 [info] <0.279.0> ra: meta data store initialised for system coordination. 0 record(s) recovered
rabbitmq_1 | 2022-11-09 21:52:01.314638+00:00 [notice] <0.284.0> WAL: ra_coordination_log_wal init, open tbls: ra_coordination_log_open_mem_tables, closed tbls: ra_coordination_log_closed_mem_tables
rabbitmq_1 | 2022-11-09 21:52:01.323668+00:00 [info] <0.229.0>
rabbitmq_1 | 2022-11-09 21:52:01.323668+00:00 [info] <0.229.0> Starting RabbitMQ 3.11.2 on Erlang 25.1.2 [jit]
rabbitmq_1 | 2022-11-09 21:52:01.323668+00:00 [info] <0.229.0> Copyright (c) 2007-2022 VMware, Inc. or its affiliates.
rabbitmq_1 | 2022-11-09 21:52:01.323668+00:00 [info] <0.229.0> Licensed under the MPL 2.0. Website: https://rabbitmq.com
rabbitmq_1 | ## ## RabbitMQ 3.11.2
rabbitmq_1 | ######### Copyright (c) 2007-2022 VMware, Inc. or its affiliates.

Figure 2.3

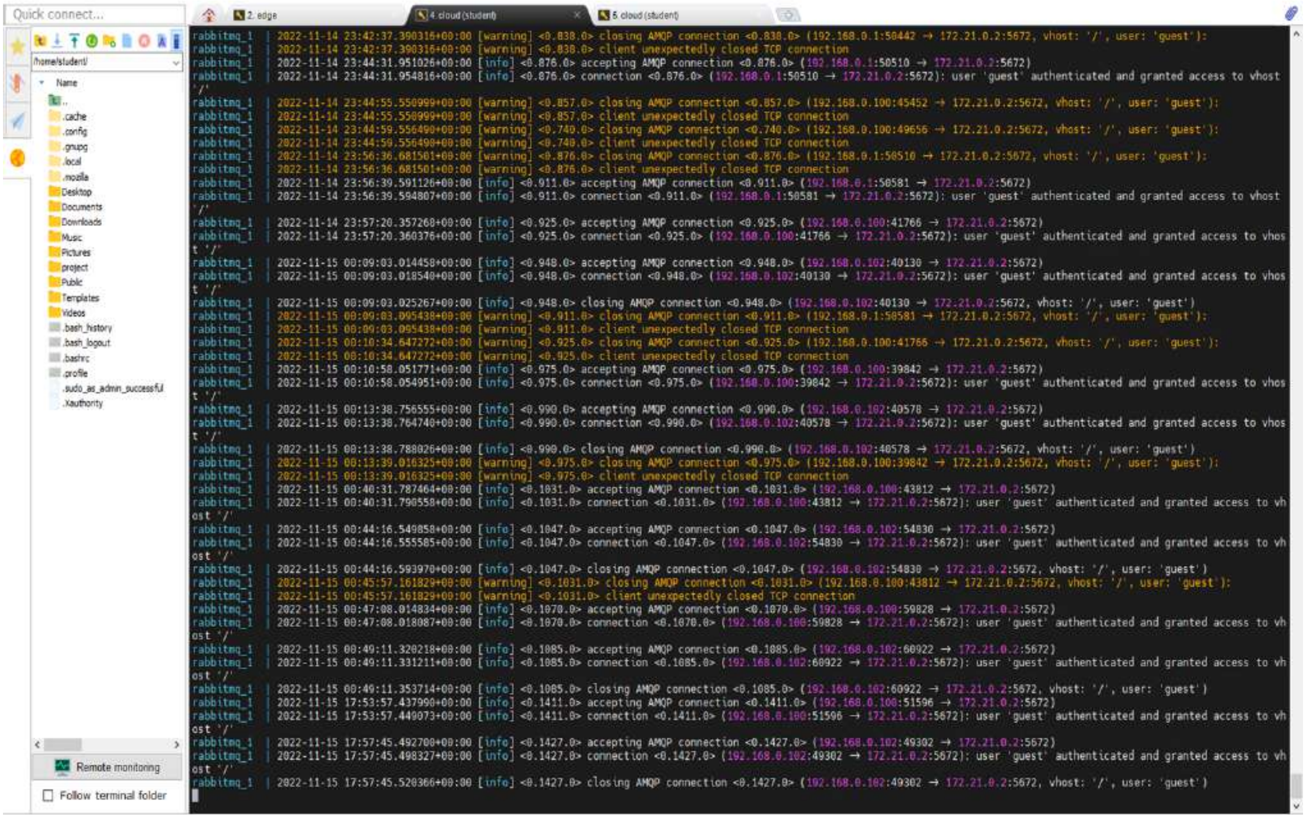


Figure 2.4

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

   In the below "*Data-Pre-processing*" code the data of Timestamp and Value of PM2.5 printed in task 1.2 (c) will be written in the file "*data.json*" from that file input will be taken using the file variable as mentioned below

   There are four functions written in the Data-Pre-processing code

   get_data() method will fetch the data from the "*data.json*" file and print the data in the docker log console of Azure lab (Edge) with the image name "*preprocessing_1*" as shown in Figure 2.5 to get the docker logs in the terminal, "*docker logs preprocessing_1*" is used in the docker terminal.

   get_outliers() method will fetch the data from the "*data.json*" file and create a list to store the value greater than 50, using the "*for*" loop and "*if else*" condition statement the data is filtered and append to the list after converting it into the JSON string format and print the values in the docker logs console as shown in Figure 2.7 to get the docker logs in the terminal use the command "*docker logs preprocessing_1*" in the docker terminal

   get_average() method will fetch the data from the "*data.json*" file and convert and reset that data into a pandas data format for calculating the mean value of each timestamp and convert back into the JSON string format and print the values in the docker logs console as shown in Figure 2.8 to get the docker logs in the terminal use the command *"docker logs preprocessing_1"* is used in the docker terminal.

transfer() method will send the data of average value to the cloud terminal using RabbitMQ container so that the average value will get printed in the Azure Lab (Cloud) as shown in Figure 2.9

After running the RabbitMQ image using the command "*docker-compose up"* in the cloud terminal and running the docker-compose file in the Azure Lab (Edge) using the command *"docker build -t preprocessor"* the docker file will get run in the Azure Lab (Edge) terminal and using docker-compose up *"preprocessing"* image will get run and the data from the producer and the method *get_data(), get_outliers()* and *get_average()* will get printed in the docker logs console of Azure Lab (Edge) and the producer data will be printed in the Azure Lab (Cloud) using the command "*python3 consumer.py"*.

Figure 2.5 shows that the data printed from the "*MQTT Subscriber code"* to the file *"data.json"* which is used for task 2.

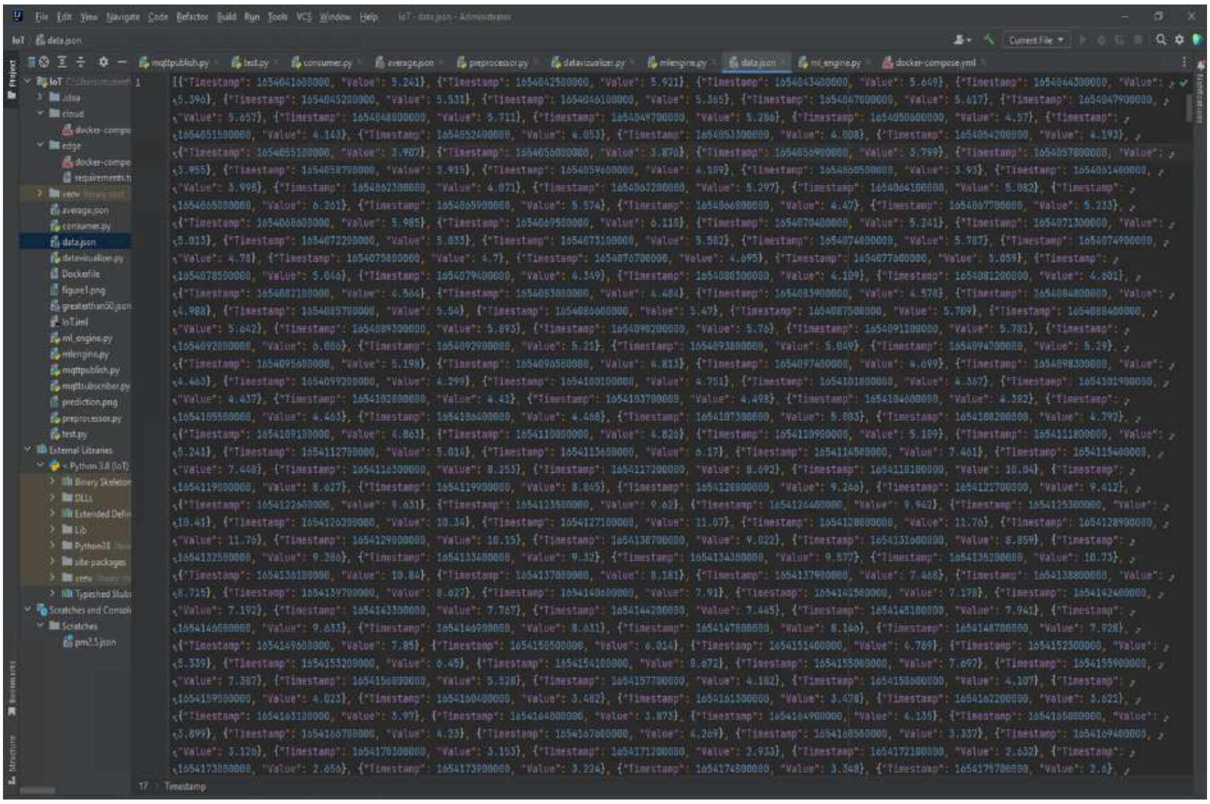


Figure 2.5

Data-Pre-processing code:

```
from datetime import datetime
import datetime
import json
import pandas as pd
from paho.mqtt import client as mqtt_client
import pika
import matplotlib.pyplot as plt

file = open('data.json')
data = json.load(file)
data_str = json.dumps(data)

if __name__ == '__main__':
  def get_data():
    data1 = json.dumps(data)
    print("The PM2.5 data are:", data1)

  def get_outliers():
    list = []
    for item in data:
      if item['Value'] < 50:
        del item['Value']
        del item['Timestamp']
      else:
        list.append(json.dumps(item))
    print("The outliers are:", list)

  def get_average():
    for item in data:
      df = pd.read_json(data_str)
      daily_avg = df.set_index('Timestamp').groupby(pd.Grouper(freq='D')).mean()
      ps = daily_avg.reset_index()
      ps = ps.drop(labels=[72], axis=0)
      df2 = ps.to_json(orient = 'records')
    print("Average of PM2.5 are:", df2)
    return df2

  def transfer():
    rabbitmq_ip = "192.168.0.100"
    rabbitmq_port = 5672
    # Queue name
    rabbitmq_queque = "CSC8112"
    msg = get_average()


    # 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(msg))

        connection.close()

get_data()
get_outliers()
transfer()
```

Rabbit MQ Consumer code:

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

rabbitmq_ip = "192.168.0.100"
rabbitmq_port = 5672
# Quseue name
rabbitmq_queque = "CSC8112"


def callback(ch, method, properties, body):
  print(f"Got message from producer msg: {json.loads(body)}")
  msg1 = json.loads(body)
  jsonfile = open("average.json", "w")
  jsonfile.write(msg1)
  jsonfile.close()



  # 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()
```

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).

Figure 2.6 and 2.7 shows the PM2.5 data printed in the docker logs console by running the docker-compose file using the command *"docker-console up"*.

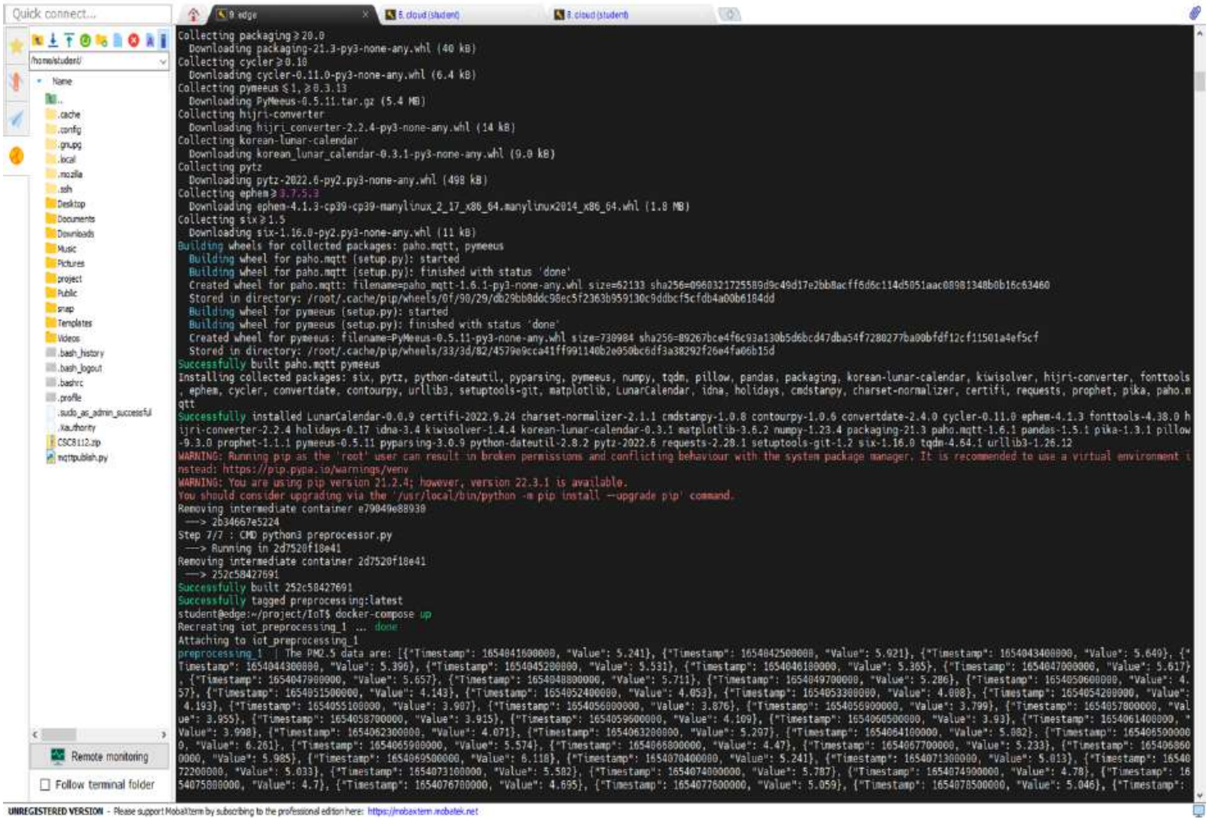


Figure 2.6

Figure 2.7

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

Figure 2.8 shows the outliers of the data PM2.5 printed in the docker log console by running the docker-consumer file using the command *"docker-console up"*.

Figure 2.8

c) 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) and please print out the result to the console (docker logs console).

   Figure 2.9 shows the average of the data PM2.5 printed in the docker log console by running the docker-consumer file using the command *"docker-console up"*.

Figure 2.9

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

   Figure 2.10 shows the averaged data of PM2.5 printed in the Azure Lab (Cloud) by running the command *"python3 consumer.py"*

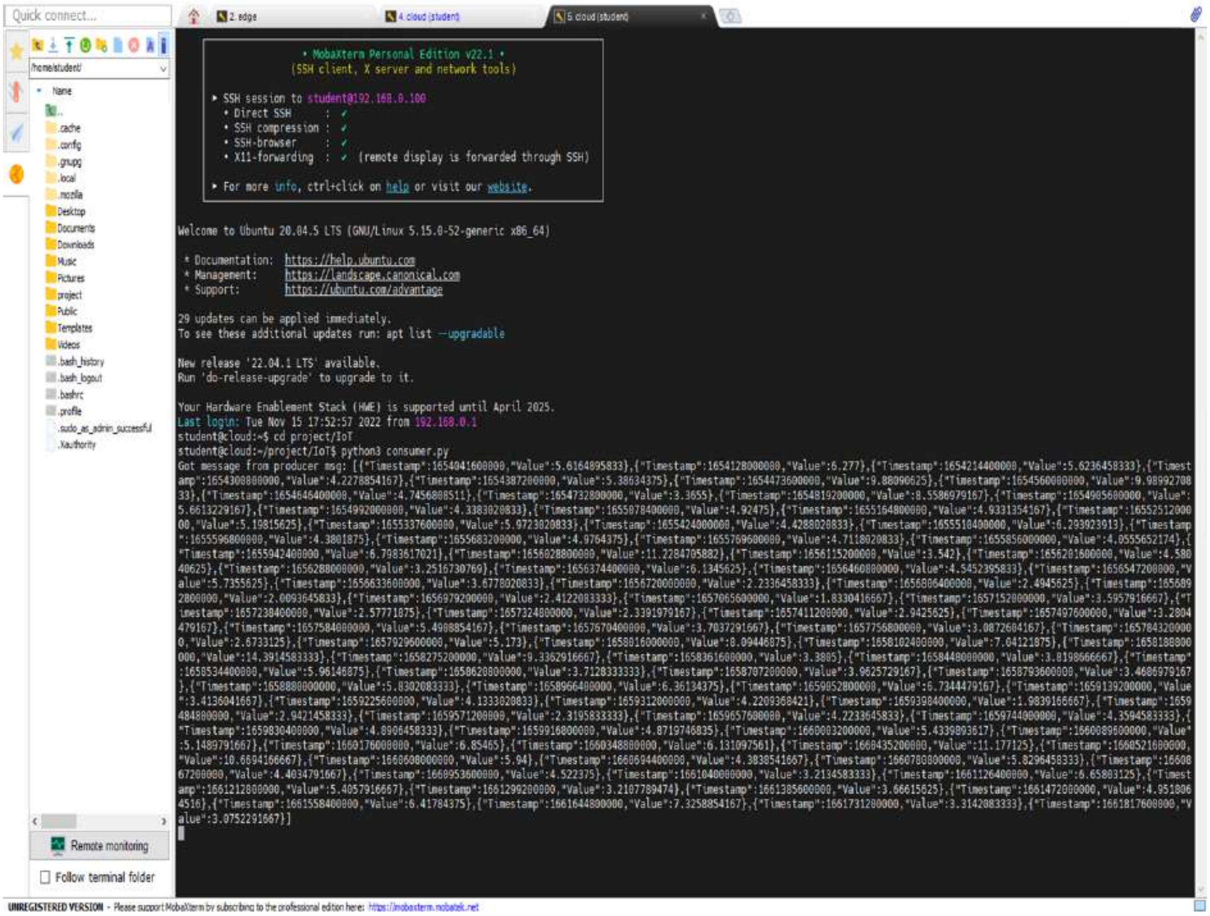


Figure 2.10

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

   Figure 2.11 shows the docker file which includes “*preprocessor.py*” and “*requirements.txt*”

   preprocessor.py acts as a data-pre-processor in task 2 it has four functions which include producer.

   requirements.txt will be having different libraries which are used in the *preprocessor.py*

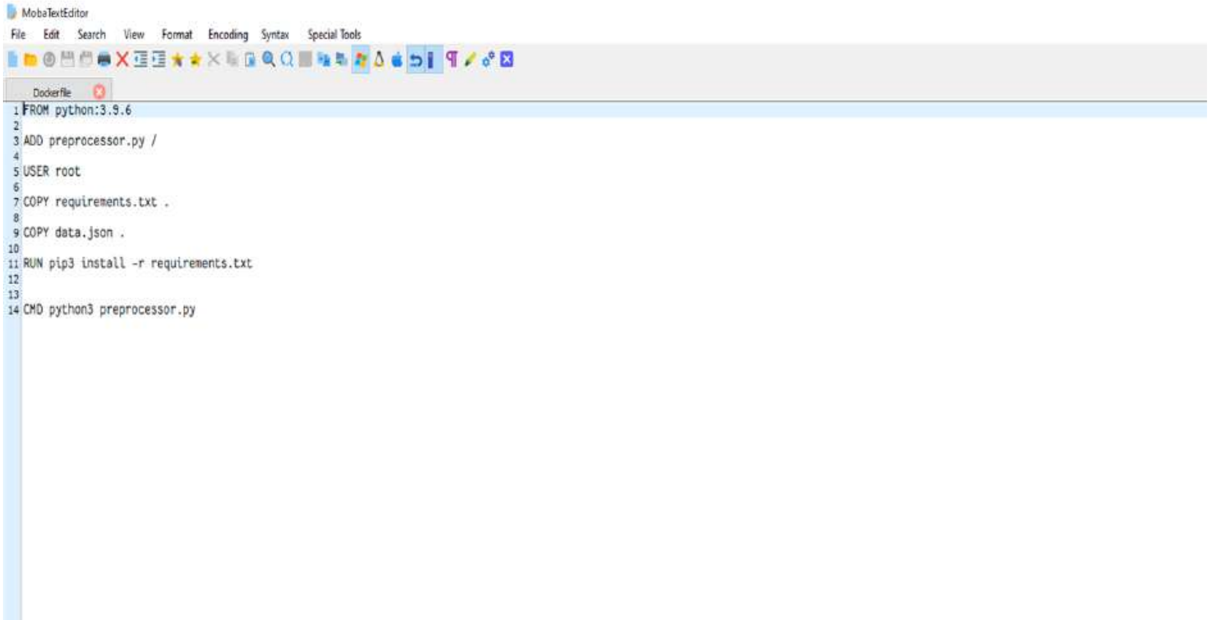
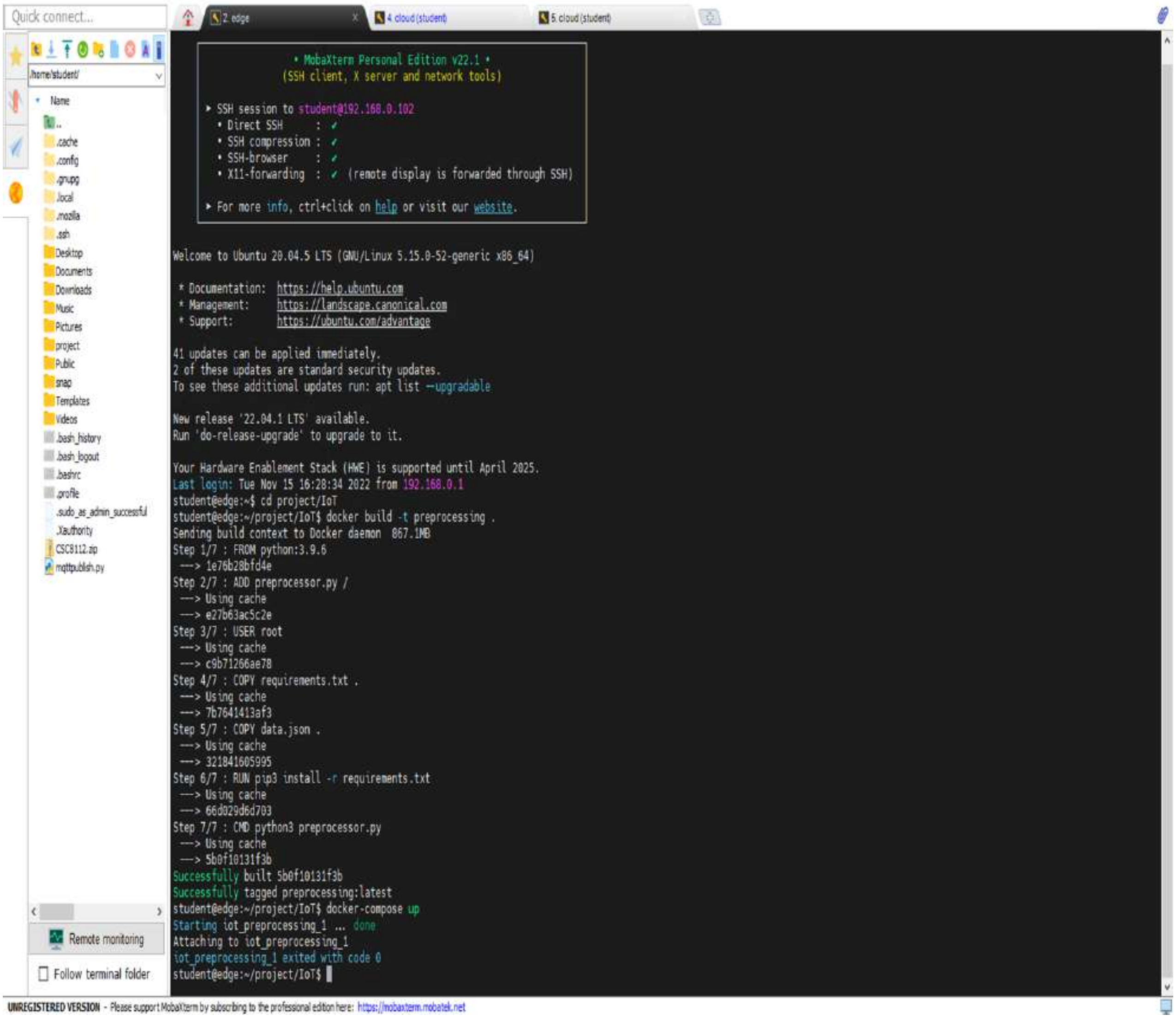


Figure 2.11

Figure 2.12 shows the docker file and image running using the command *"docker build -t preprocessing ."* and *"docker-compose up"*



Quick connect...
2. edge
4. cloud (student)
5. cloud (student)
/home/student/
Name
.cache
.config
.gnupg
.local
.mozilla
.ssh
Desktop
Documents
Downloads
Music
Pictures
project
Public
snap
Templates
Videos
.bash_history
.bash_logout
.bashrc
.profile
.sudo_as_admin_successful
.Xauthority
CSC8112.zip
mqttpublish.py
Remote monitoring
Follow terminal folder
• MobaXterm Personal Edition v22.1 •
(SSH client, X server and network tools)
➤ SSH session to student@192.168.0.102
• Direct SSH : ✓
• SSH compression : ✓
• SSH-browser : ✓
• X11-forwarding : ✓ (remote display is forwarded through SSH)
➤ For more info, ctrl+click on help or visit our website.
Welcome to Ubuntu 20.04.5 LTS (GNU/Linux 5.15.0-52-generic x86_64)
* Documentation: https://help.ubuntu.com
* Management: https://landscape.canonical.com
* Support: https://ubuntu.com/advantage
41 updates can be applied immediately.
2 of these updates are standard security updates.
To see these additional updates run: apt list --upgradable
New release '22.04.1 LTS' available.
Run 'do-release-upgrade' to upgrade to it.
Your Hardware Enablement Stack (HWE) is supported until April 2025.
Last login: Tue Nov 15 16:28:34 2022 from 192.168.0.1
student@edge:~$ cd project/IoT
student@edge:~/project/IoT$ docker build -t preprocessing .
Sending build context to Docker daemon 867.1MB
Step 1/7 : FROM python:3.9.6
---> 1e76b28bfd4e
Step 2/7 : ADD preprocessor.py /
---> Using cache
---> e27b63ac5c2e
Step 3/7 : USER root
---> Using cache
---> c9b71266ae78
Step 4/7 : COPY requirements.txt .
---> Using cache
---> 7b7641413af3
Step 5/7 : COPY data.json .
---> Using cache
---> 321841605995
Step 6/7 : RUN pip3 install -r requirements.txt
---> Using cache
---> 66d029d6d703
Step 7/7 : CMD python3 preprocessor.py
---> Using cache
---> 5b0f10131f3b
Successfully built 5b0f10131f3b
Successfully tagged preprocessing:latest
student@edge:~/project/IoT$ docker-compose up
Starting iot_preprocessing_1 ... done
Attaching to iot_preprocessing_1
iot_preprocessing_1 exited with code 0
student@edge:~/project/IoT$
UNREGISTERED VERSION - Please support MobaXterm by subscribing to the professional edition here: https://mobaxterm.mobatek.net

Figure 2.12

Task 3: Time-series data prediction and visualization

In task 3 the averaged PM2.5 data will be taken from the RabbitMQ consumer in the Azure Lab (Cloud) in Task 2.2 (d) shown in Figure 2.10 will be written in the file *"average.json"* from the file input will be taken for task 3.

1. Download a pre-defined Machine Learning (ML) engine code

    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
```

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

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.

   Figure 3.1 shows the average daily PM2.5 data computed by Task 2.2 (d) printed in the Azure Lab (Cloud).

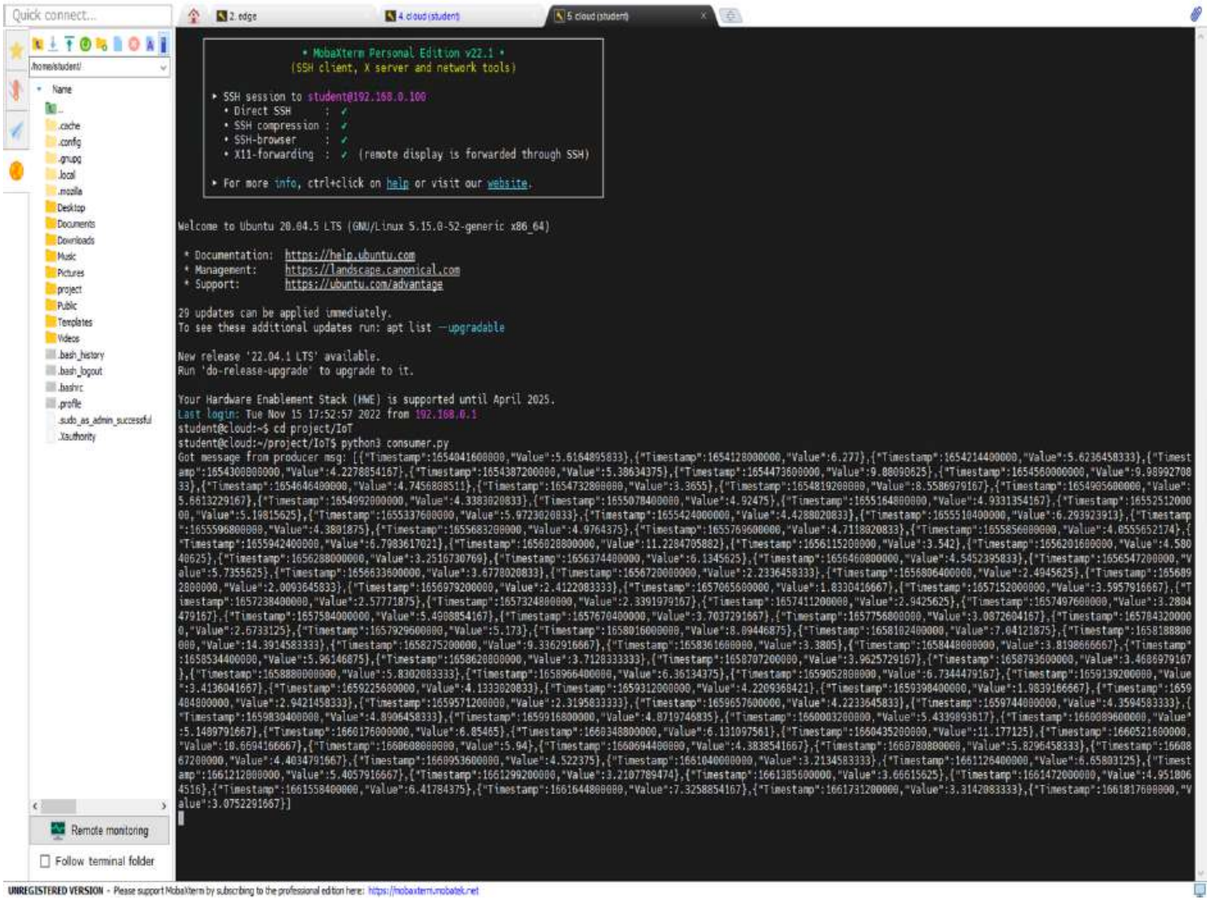


Figure 3.1

Figure 3.2 shows the averaged data written by the *"consumer.py"* printed in the file *"average.json"* it is used for the input of task 3.

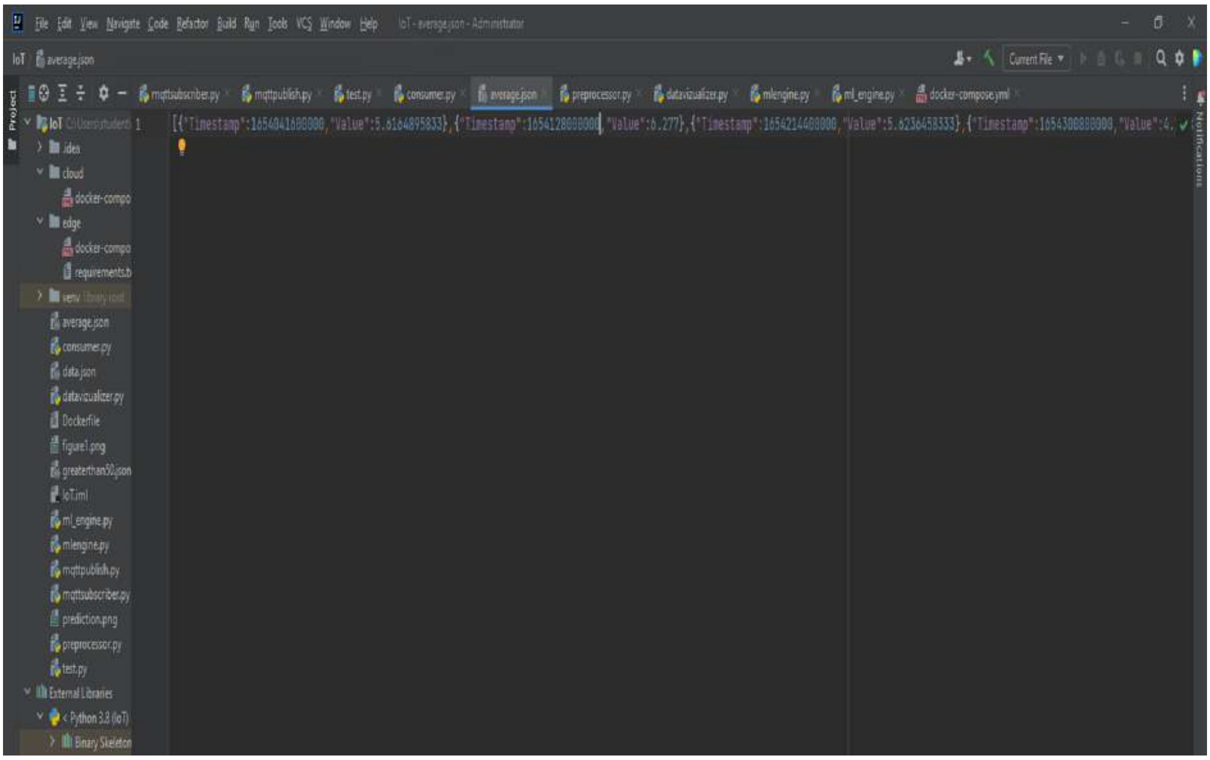


Figure 3.2

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.

Figure 3.3 shows the conversion of the timestamp to date time format (year-month-day hour: minute: second) in the Azure Lab (Cloud) terminal using the command "*python3 datavizualizer.py*"

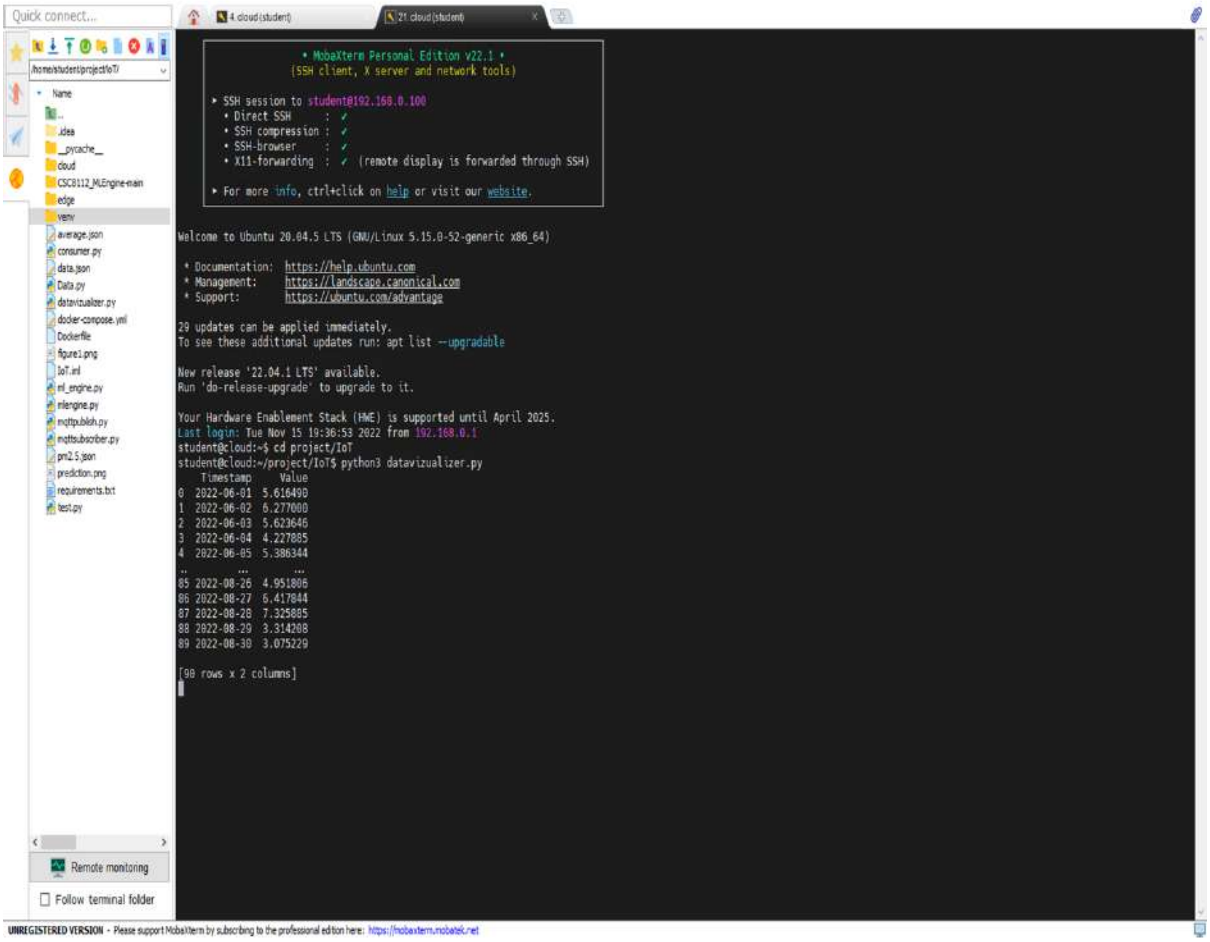


Figure 3.3

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.

Figures 3.4 and 3.5 shows the visualised image of averaged PM2.5 data in the cloud terminal

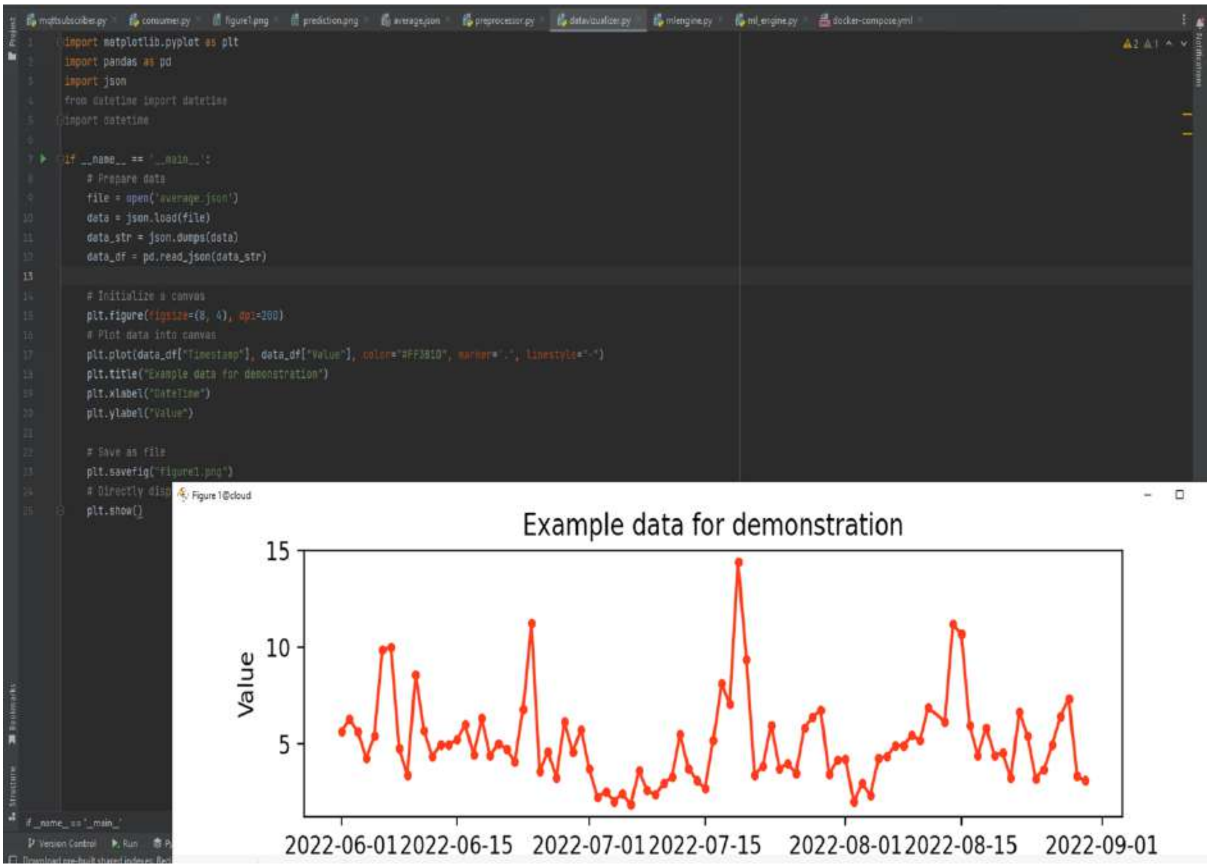


Figure 3.4

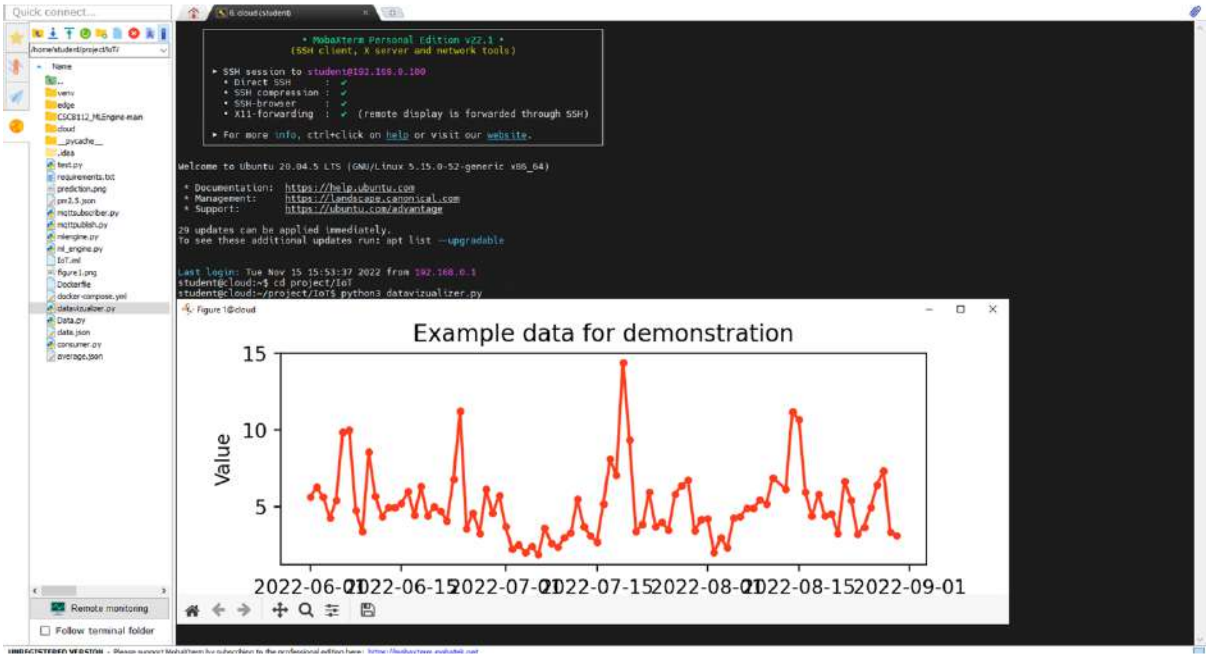


Figure 3.5

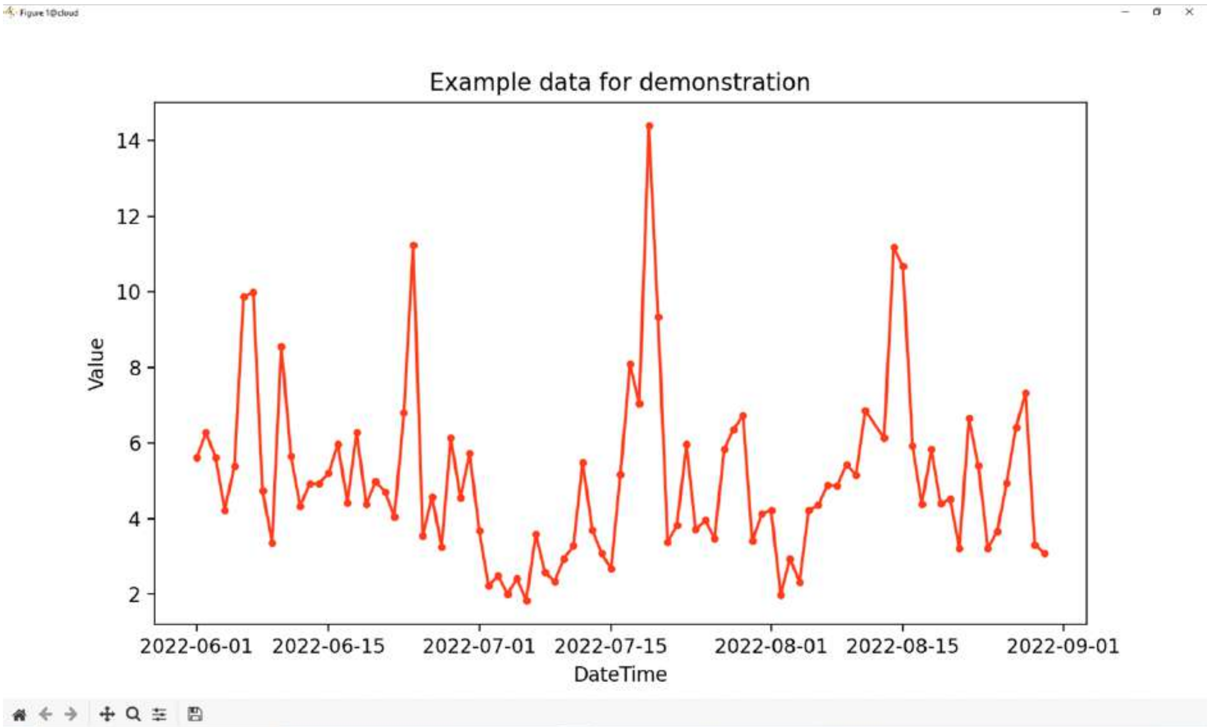


Figure 3.6

d) Feed averaged PM2.5 data to the 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).

```
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()
```

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

Figures 3.4 and 3.5 show the prediction image for the next 15 days using the averaged PM2.5 data in the cloud terminal.

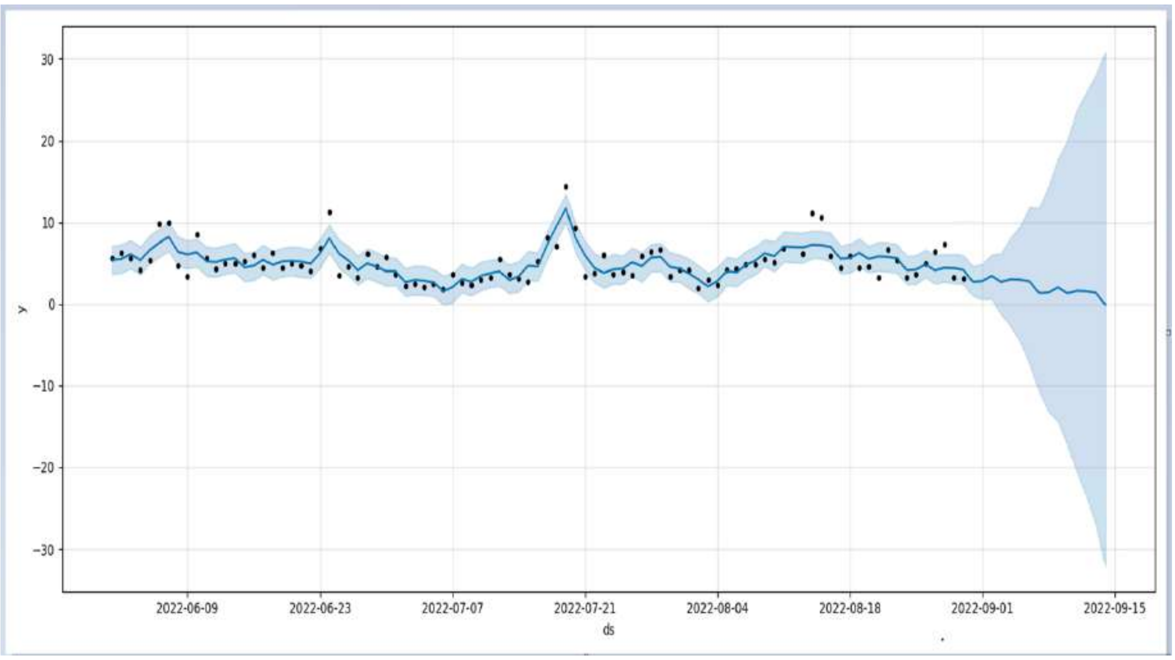


Figure 3.4

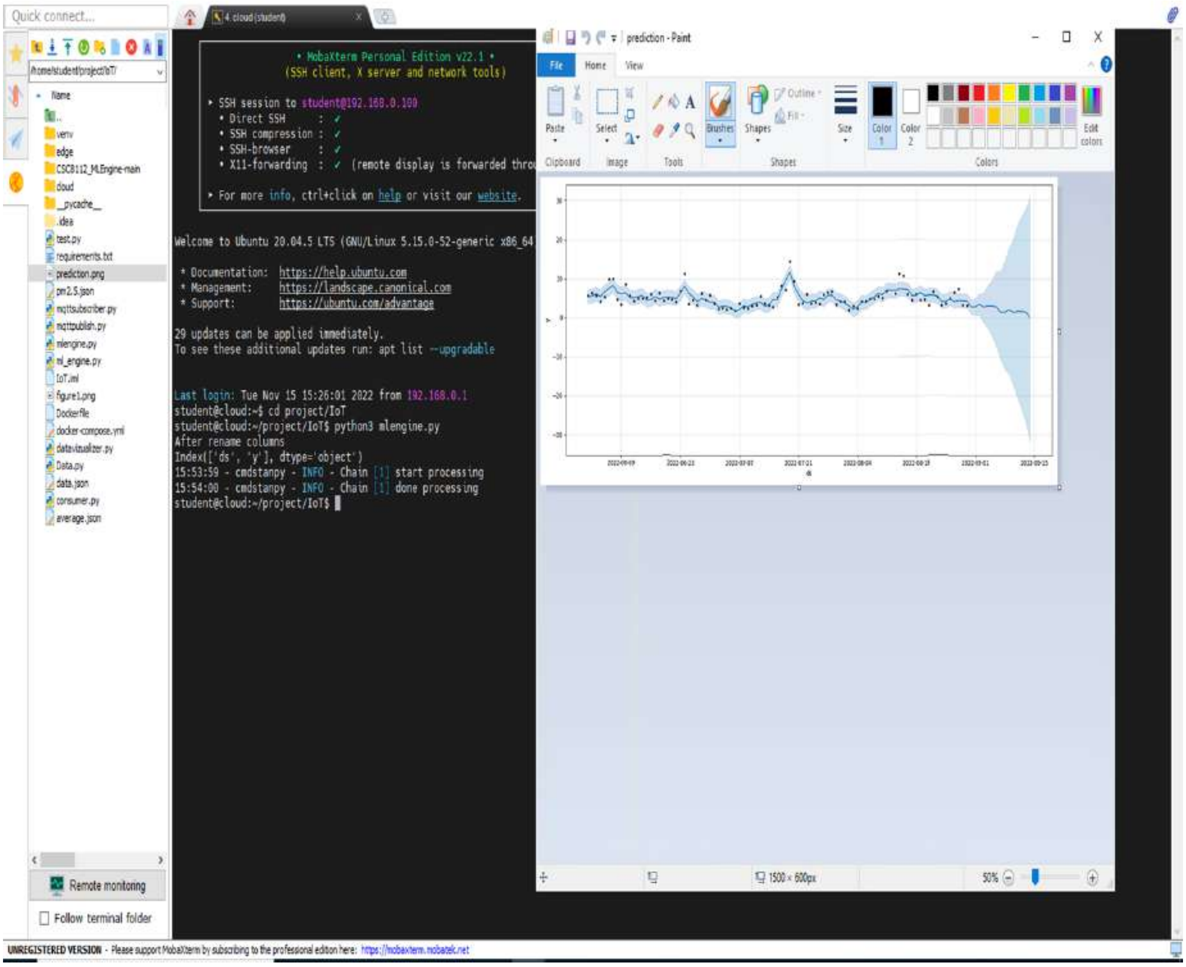


Figure 3.5

Analytical Discussion:

PM2.5 data some particles which are most dangerous than others, such as dust and industry pollution and air pollution e.t.c which cause lung diseases it is invisible to the naked eye it usually comes from the industry and vehicles by plotting the prediction of the PM2.5 graph we can see in Figure 3.4 the value was high in the month of June and July and then it slightly decreased in the month of September.