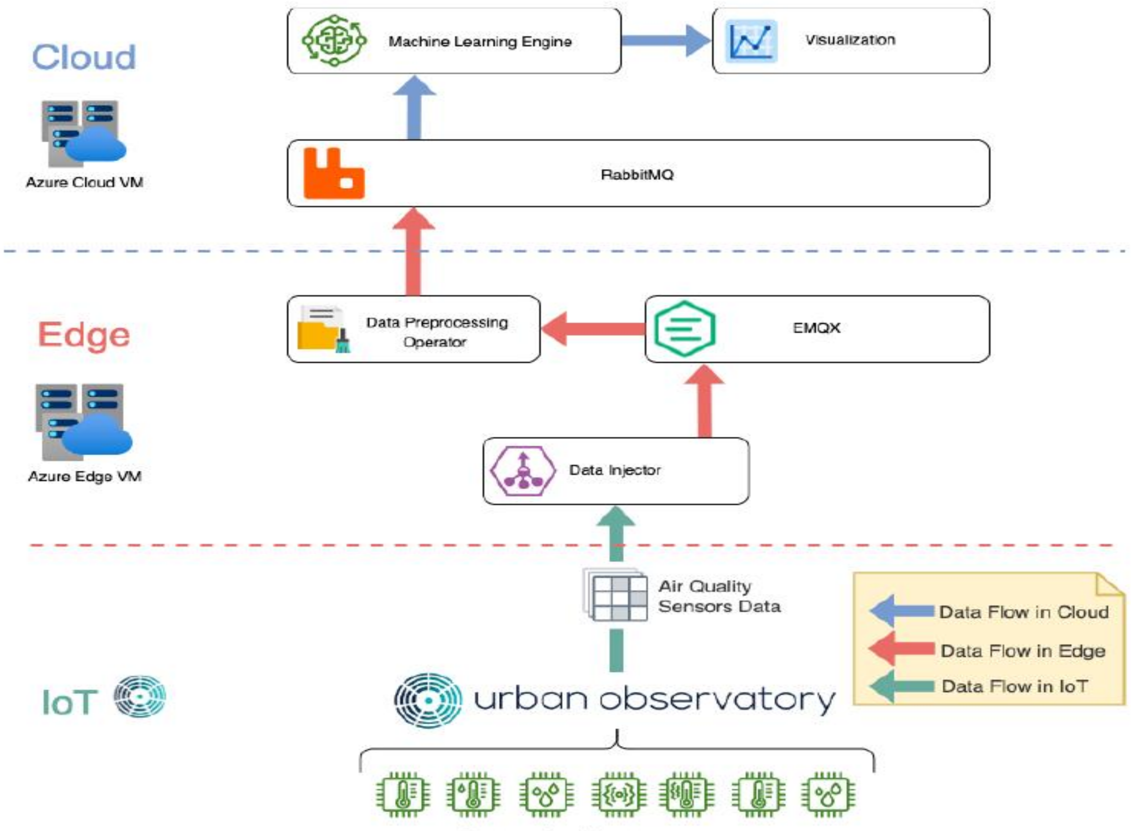
Author: Saivenket Patro K

Prediction of PM2.5 Data Of Newcastle: Documentation

Aim

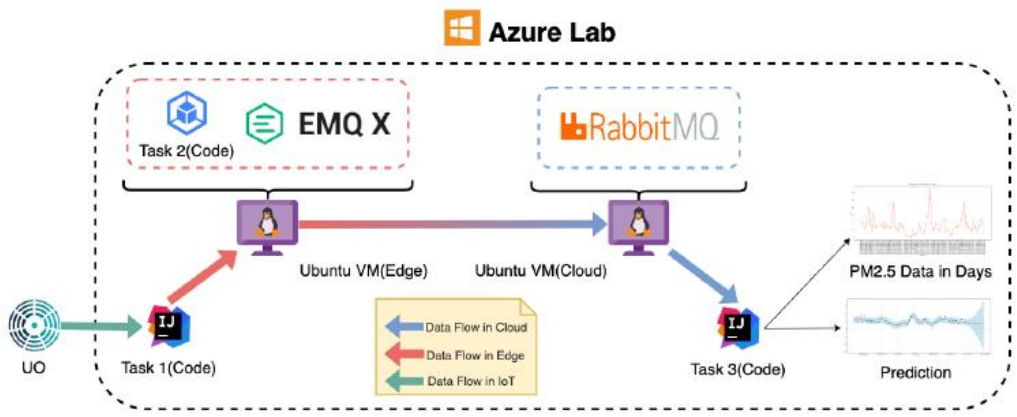
To create an IOT pipeline for predicting PM2.5 data for the next 15days.

Project Architecture



System Architecture

Project Workflow



The Final Workflow Structure

Prerequisites

1. IDE in the local VM
2. Hyper-V Manager to control the Child VMs
3. MobaXterm to directly connect with terminal of the Child VMs and also to upload files from the local VM
4. Download and install Docker program in all the VMs
5. Install docker compose in the Child VMs
6. Create a new virtual environment with specific python dependencies

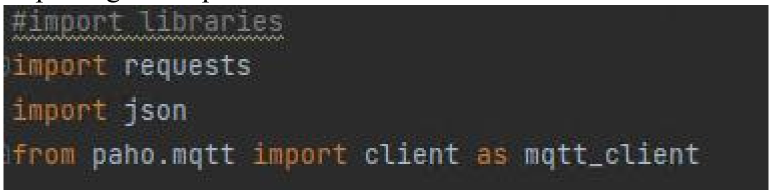
Package Name	Version (== or latest)	Install Command	Task
requests	2.28.1	pip install requests	1
paho.mqtt	1.6.1	pip install paho-mqtt	1 & 2
pika	1.3.0	python -m pip install pika --upgrade	2 & 3
fbprophet	0.7.1	python -m pip install prophet	3
matplotlib	3.6.0	pip install matplotlib	3

While, the python dependencies should be installed based on the task mentioned.

Implementation: Task 1

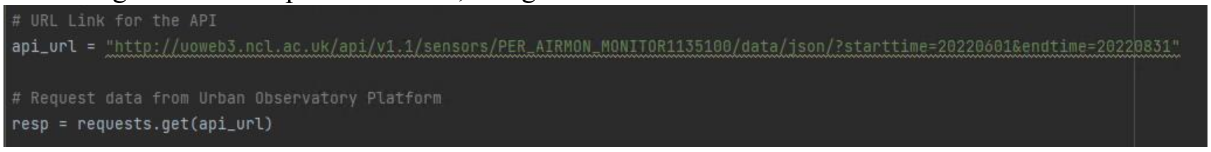
In Task 1 we create a python script which retrieves data from the Newcastle Urban Observatory, and then we pick out the required information of PM2.5 and send it to Edge VM, using the MQTT broker.

1. Importing the required libraries



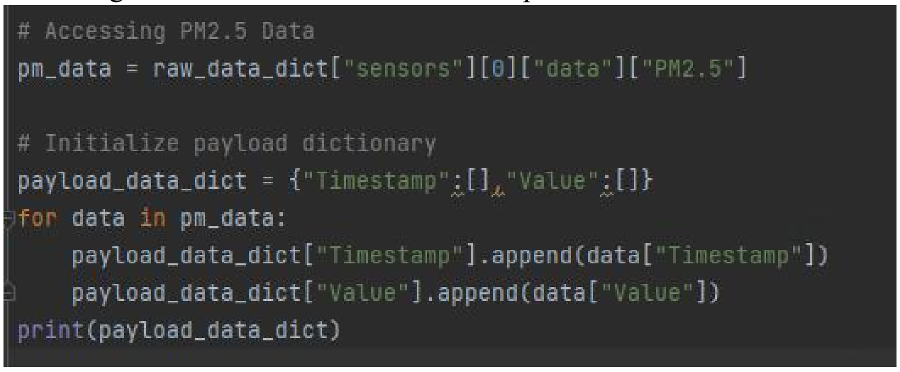
```python
#import libraries
import requests
import json
from paho.mqtt import client as mqtt_client
```

2. Retrieving the data dump from the UO, using an API call.



```python
# URL Link for the API
api_url = "http://uoweb3.ncl.ac.uk/api/v1.1/sensors/PER_AIRMON_MONITOR1135100/data/json/?starttime=20220601&endtime=20220831"

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

3. Accessing the PM2.5 data from the API Response

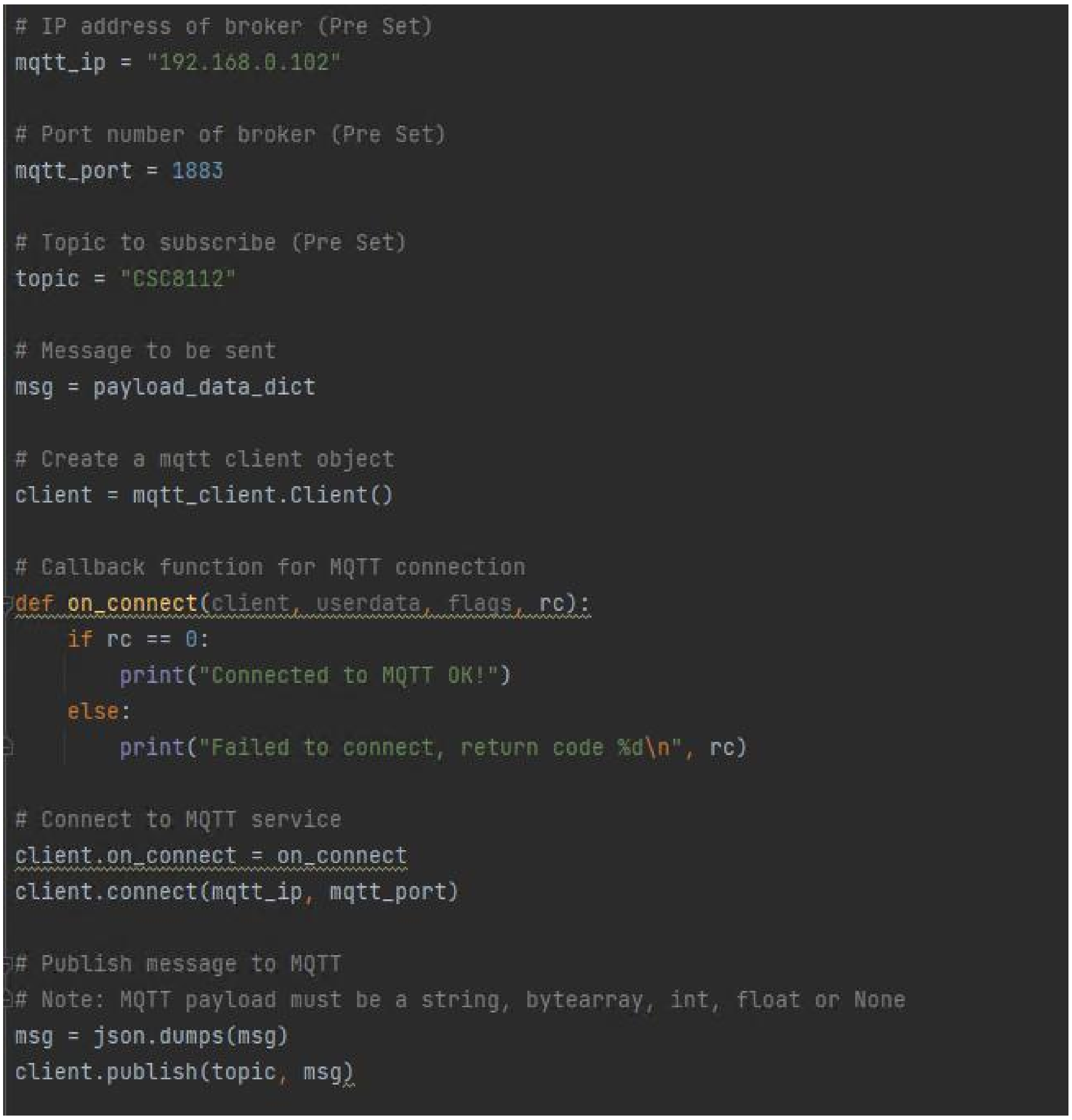


```python
# Accessing PM2.5 Data
pm_data = raw_data_dict["sensors"][0]["data"]["PM2.5"]

# Initialize payload dictionary
payload_data_dict = {"Timestamp":[],"Value":[]}
for data in pm_data:
    payload_data_dict["Timestamp"].append(data["Timestamp"])
    payload_data_dict["Value"].append(data["Value"])
print(payload_data_dict)
```

Understanding the structure of the API response, The PM2.5 data is a key which consists a list of dictionaries containing the required timestamp and value key pairs.

Created a new dictionary to store the payload containing required information of PM2.5, that is Timestamp and Values.

4. Publishing to the MQTT Broker



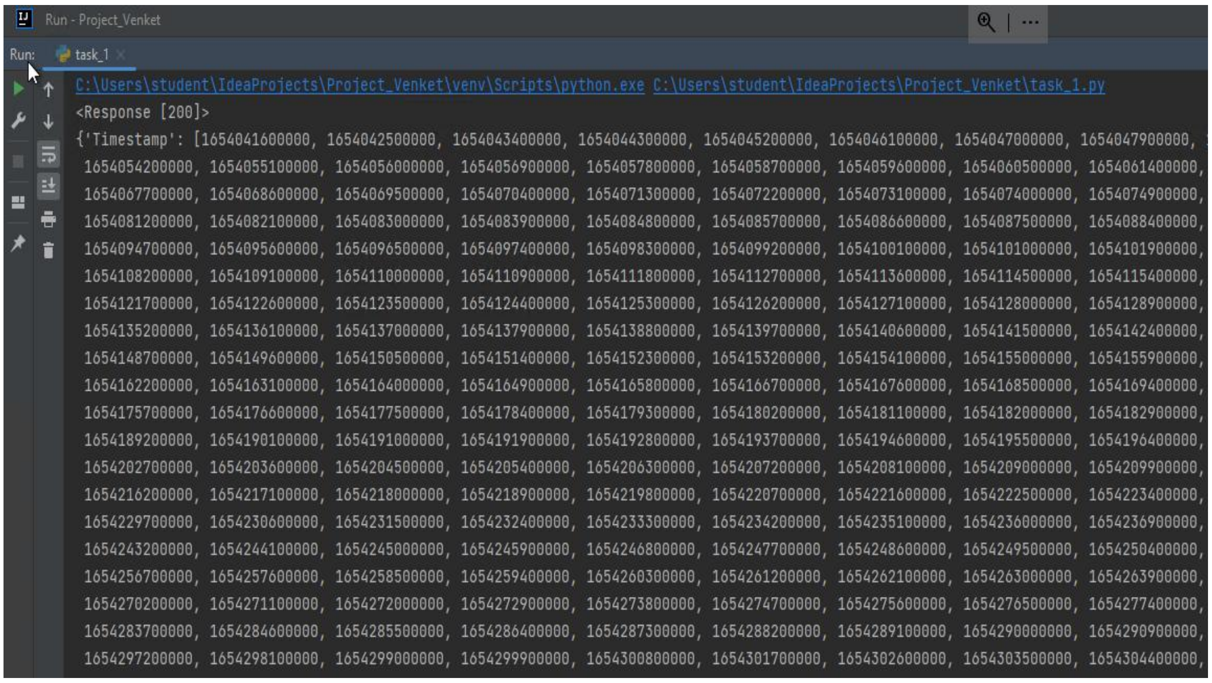
```
# IP address of broker (Pre Set)
mqtt_ip = "192.168.0.102"

# Port number of broker (Pre Set)
mqtt_port = 1883

# Topic to subscribe (Pre Set)
topic = "CSC8112"

# Message to be sent
msg = payload_data_dict

# 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(msg)
client.publish(topic, msg)
```

Defining the IP Address and the port number of the MQTT broker.
Assigning the required payload to the msg variable.

client.connect() – It connects to the required IP and port number of the EMQX Service, which in this case is the EMQX docker service running in the Edge VM.
client.publish() – It publishes the payload along with the topic.

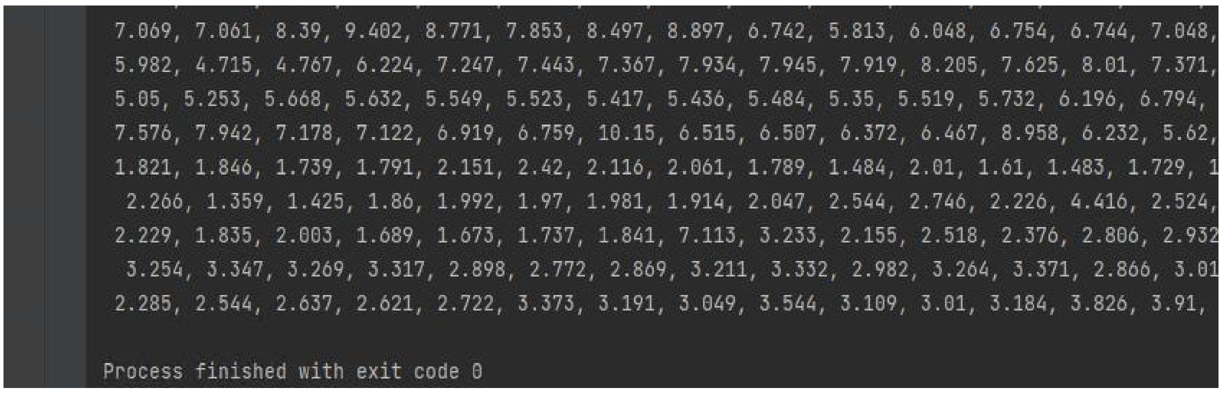
Note: To determine the IP address of Edge VM, you need to run “” command in the terminal of the Ubuntu System.

5. Terminal Output of the Payload
   Note: The output is large and cannot be entirely shown.



```
C:\Users\student\IdeaProjects\Project_Venket\venv\Scripts\python.exe C:\Users\student\IdeaProjects\Project_Venket\task_1.py
<Response [200]>
{'Timestamp': [1654041600000, 1654042500000, 1654043400000, 1654044300000, 1654045200000, 1654046100000, 1654047000000, 1654047900000, 
 1654054200000, 1654055100000, 1654056000000, 1654056900000, 1654057800000, 1654058700000, 1654059600000, 1654060500000, 1654061400000,
 1654067700000, 1654068600000, 1654069500000, 1654070400000, 1654071300000, 1654072200000, 1654073100000, 1654074000000, 1654074900000,
 1654081200000, 1654082100000, 1654083000000, 1654083900000, 1654084800000, 1654085700000, 1654086600000, 1654087500000, 1654088400000,
 1654094700000, 1654095600000, 1654096500000, 1654097400000, 1654098300000, 1654099200000, 1654100100000, 1654101000000, 1654101900000,
 1654108200000, 1654109100000, 1654110000000, 1654110900000, 1654111800000, 1654112700000, 1654113600000, 1654114500000, 1654115400000,
 1654121700000, 1654122600000, 1654123500000, 1654124400000, 1654125300000, 1654126200000, 1654127100000, 1654128000000, 1654128900000,
 1654135200000, 1654136100000, 1654137000000, 1654137900000, 1654138800000, 1654139700000, 1654140600000, 1654141500000, 1654142400000,
 1654148700000, 1654149600000, 1654150500000, 1654151400000, 1654152300000, 1654153200000, 1654154100000, 1654155000000, 1654155900000,
 1654162200000, 1654163100000, 1654164000000, 1654164900000, 1654165800000, 1654166700000, 1654167600000, 1654168500000, 1654169400000,
 1654175700000, 1654176600000, 1654177500000, 1654178400000, 1654179300000, 1654180200000, 1654181100000, 1654182000000, 1654182900000,
 1654189200000, 1654190100000, 1654191000000, 1654191900000, 1654192800000, 1654193700000, 1654194600000, 1654195500000, 1654196400000,
 1654202700000, 1654203600000, 1654204500000, 1654205400000, 1654206300000, 1654207200000, 1654208100000, 1654209000000, 1654209900000,
 1654216200000, 1654217100000, 1654218000000, 1654218900000, 1654219800000, 1654220700000, 1654221600000, 1654222500000, 1654223400000,
 1654229700000, 1654230600000, 1654231500000, 1654232400000, 1654233300000, 1654234200000, 1654235100000, 1654236000000, 1654236900000,
 1654243200000, 1654244100000, 1654245000000, 1654245900000, 1654246800000, 1654247700000, 1654248600000, 1654249500000, 1654250400000,
 1654256700000, 1654257600000, 1654258500000, 1654259400000, 1654260300000, 1654261200000, 1654262100000, 1654263000000, 1654263900000,
 1654270200000, 1654271100000, 1654272000000, 1654272900000, 1654273800000, 1654274700000, 1654275600000, 1654276500000, 1654277400000,
 1654283700000, 1654284600000, 1654285500000, 1654286400000, 1654287300000, 1654288200000, 1654289100000, 1654290000000, 1654290900000,
 1654297200000, 1654298100000, 1654299000000, 1654299900000, 1654300800000, 1654301700000, 1654302600000, 1654303500000, 1654304400000,
```

The output is the payload of type dictionary consisting the Time stamp and Values corresponding to the PM2.5 data attribute.



```
 7.069, 7.061, 8.39, 9.402, 8.771, 7.853, 8.497, 8.897, 6.742, 5.813, 6.048, 6.754, 6.744, 7.048,
 5.982, 4.715, 4.767, 6.224, 7.247, 7.443, 7.367, 7.934, 7.945, 7.919, 8.205, 7.625, 8.01, 7.371,
 5.05, 5.253, 5.668, 5.632, 5.549, 5.523, 5.417, 5.436, 5.484, 5.35, 5.519, 5.732, 6.196, 6.794,
 7.576, 7.942, 7.178, 7.122, 6.919, 6.759, 10.15, 6.515, 6.507, 6.372, 6.467, 8.958, 6.232, 5.62,
 1.821, 1.846, 1.739, 1.791, 2.151, 2.42, 2.116, 2.061, 1.789, 1.484, 2.01, 1.61, 1.483, 1.729, 1
  2.266, 1.359, 1.425, 1.86, 1.992, 1.97, 1.981, 1.914, 2.047, 2.544, 2.746, 2.226, 4.416, 2.524,
 2.229, 1.835, 2.003, 1.689, 1.673, 1.737, 1.841, 7.113, 3.233, 2.155, 2.518, 2.376, 2.806, 2.932
  3.254, 3.347, 3.269, 3.317, 2.898, 2.772, 2.869, 3.211, 3.332, 2.982, 3.264, 3.371, 2.866, 3.01
 2.285, 2.544, 2.637, 2.621, 2.722, 3.373, 3.191, 3.049, 3.544, 3.109, 3.01, 3.184, 3.826, 3.91,

Process finished with exit code 0
```

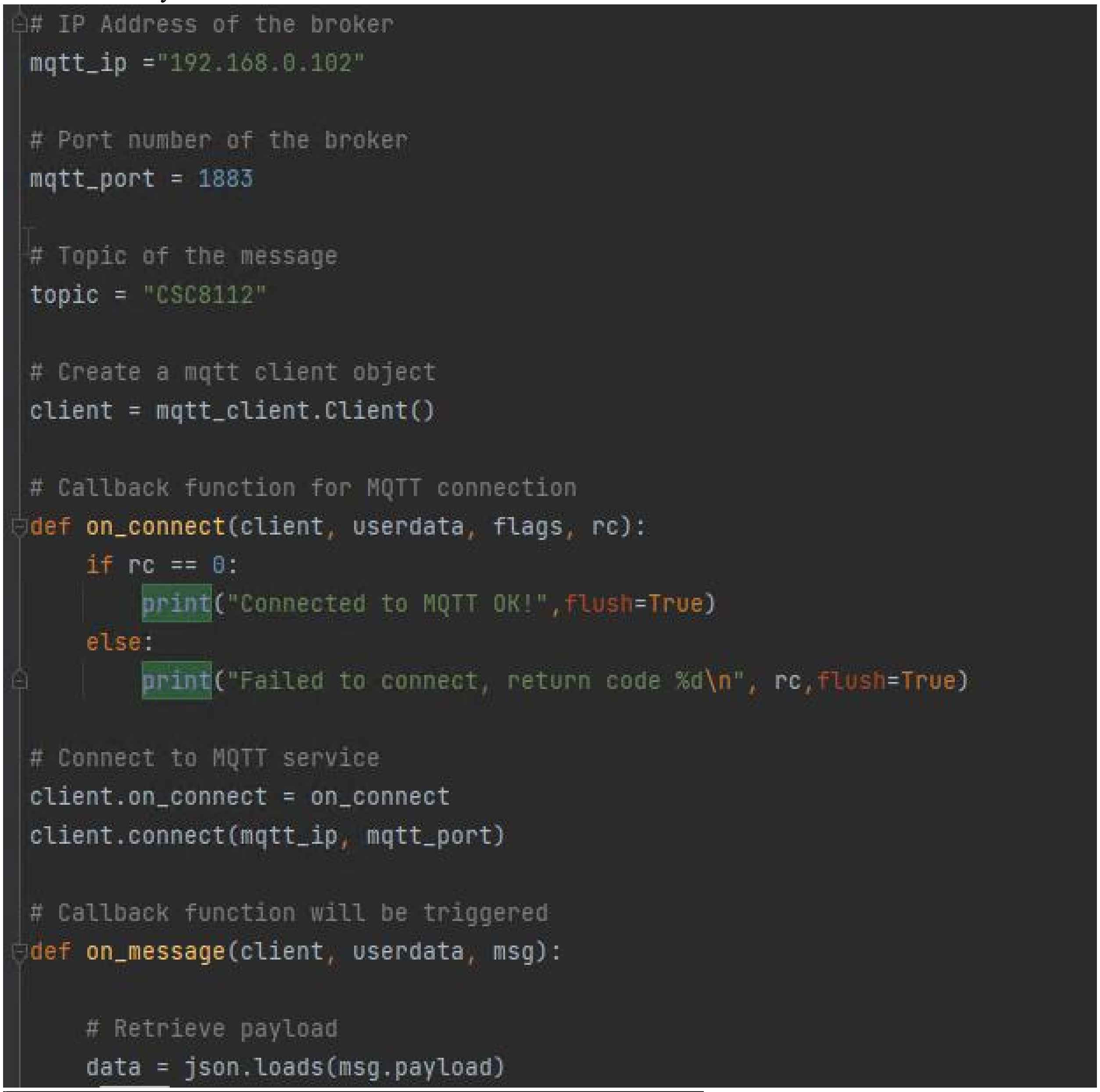
The last line determines that the payload has been successfully sent to the EMQX Service.

Implementation: Task 2

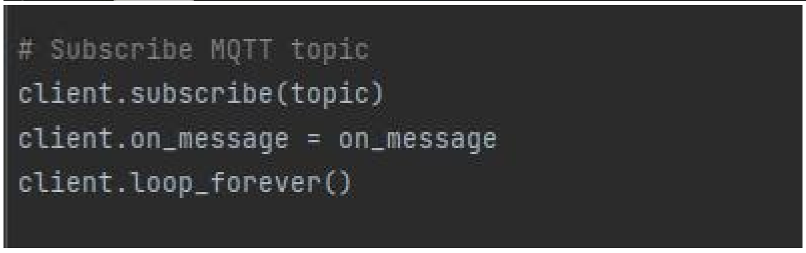
In Task2, We first connect to the MQTT Broker and subscribe to the topic, to receive the payload. Following, we use pre processing methods to clean the data. Finally, we publish the new payload to the Cloud VM via the rabbitmq queuing system.

Python Code Implementation

1. Retrieve the Payload



```python
# IP Address of the broker
mqtt_ip ="192.168.0.102"

# Port number of the broker
mqtt_port = 1883

# Topic of the message
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!",flush=True)
    else:
        print("Failed to connect, return code %d\n", rc,flush=True)

# 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):

    # Retrieve payload
    data = json.loads(msg.payload)
```

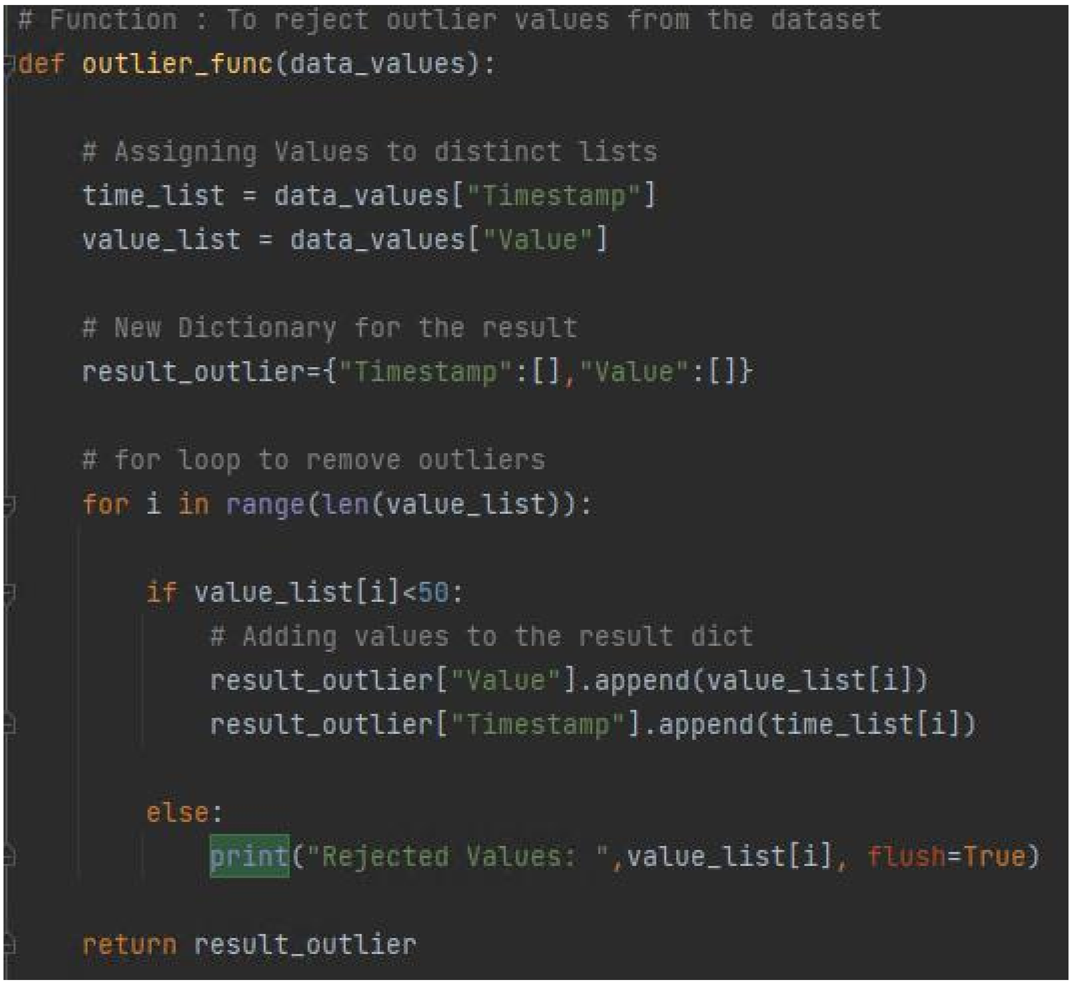


```python
# Subscribe MQTT topic
client.subscribe(topic)
client.on_message = on_message
client.loop_forever()
```

Connecting to the MQTT broker IP address and broker, and then retrieving the message as a payload under the on_message() function.

2. Pre-Processing of the payload

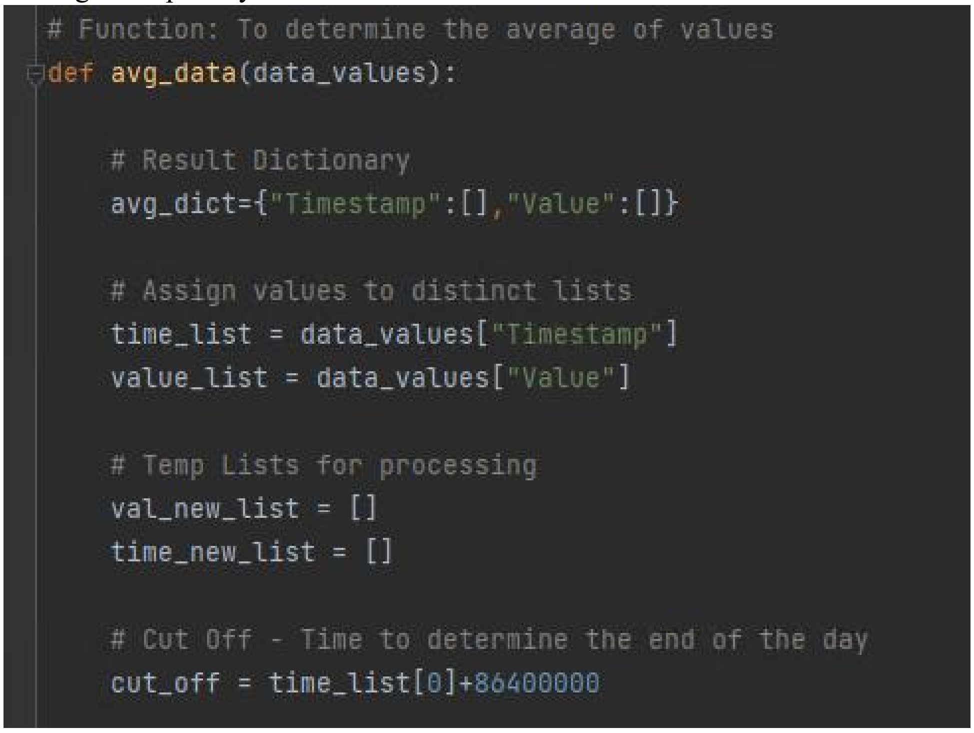
   a. Remove Outliers



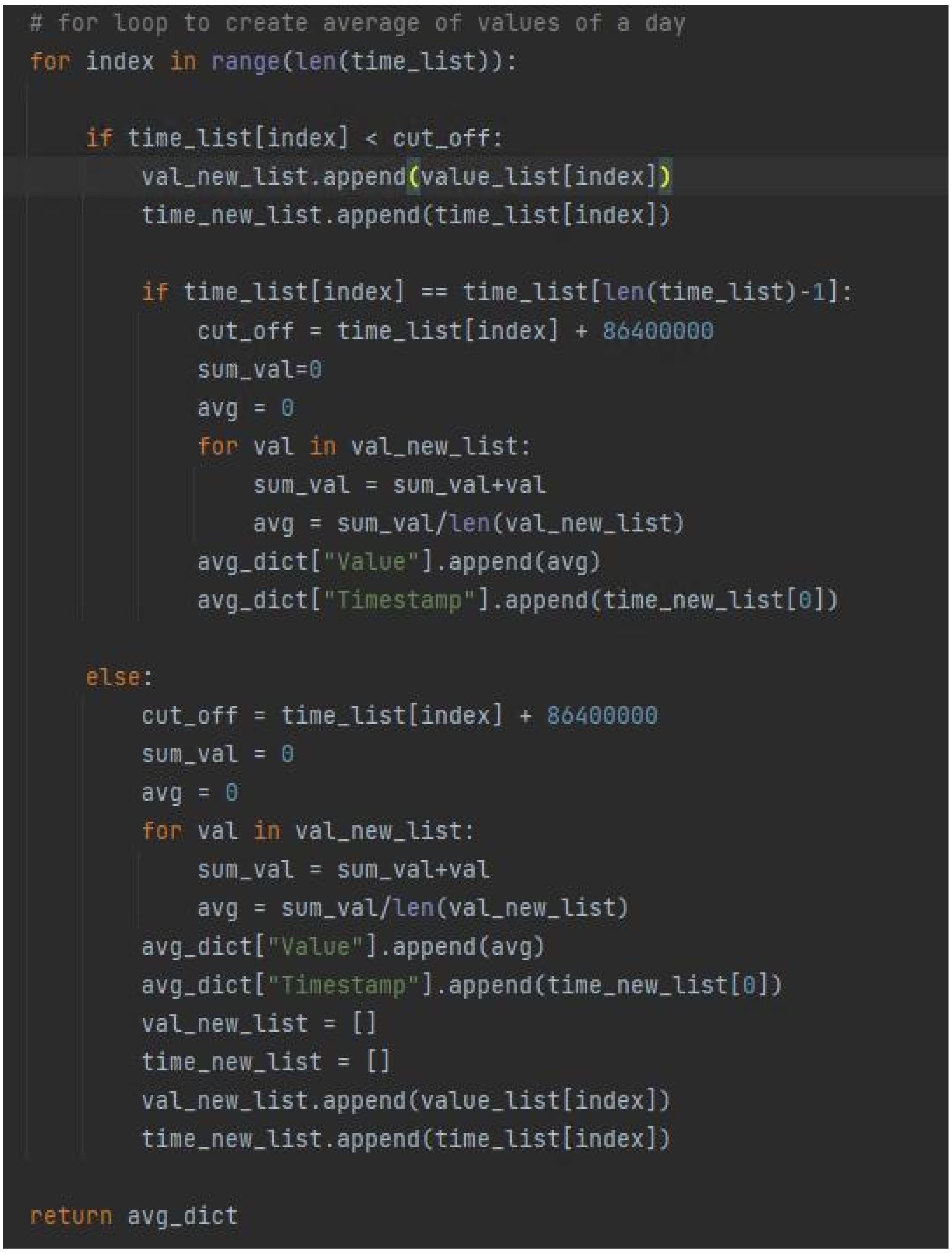
```python
# Function : To reject outlier values from the dataset
def outlier_func(data_values):

    # Assigning Values to distinct lists
    time_list = data_values["Timestamp"]
    value_list = data_values["Value"]

    # New Dictionary for the result
    result_outlier={"Timestamp":[],"Value":[]}

    # for loop to remove outliers
    for i in range(len(value_list)):

        if value_list[i]<50:
            # Adding values to the result dict
            result_outlier["Value"].append(value_list[i])
            result_outlier["Timestamp"].append(time_list[i])

        else:
            print("Rejected Values: ",value_list[i], flush=True)

    return result_outlier
```

The function takes payload as its input and creates a new payload dictionary which consists values less than 50 pertaining to a particular timestamp.
While, if the values are above 50 then those are printed on the terminal.
Additionally, to showcase on the docker logs , flush attribute is added.

   b. Average data per day



```python
# Function: To determine the average of values
def avg_data(data_values):

    # Result Dictionary
    avg_dict={"Timestamp":[],"Value":[]}

    # Assign values to distinct lists
    time_list = data_values["Timestamp"]
    value_list = data_values["Value"]

    # Temp Lists for processing
    val_new_list = []
    time_new_list = []

    # Cut Off - Time to determine the end of the day
    cut_off = time_list[0]+86400000
```

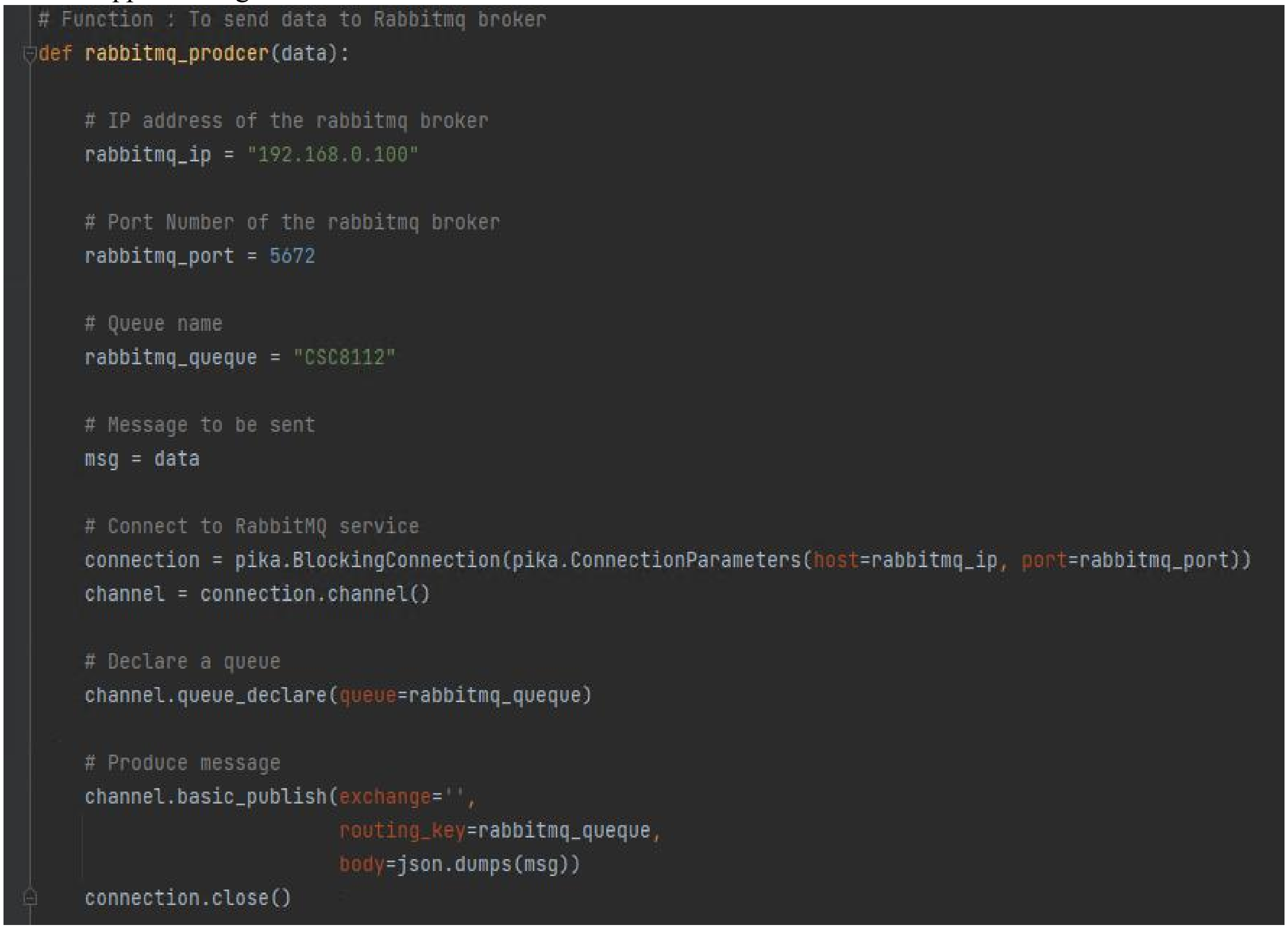


```python
# for loop to create average of values of a day
for index in range(len(time_list)):

    if time_list[index] < cut_off:
        val_new_list.append(value_list[index])
        time_new_list.append(time_list[index])

        if time_list[index] == time_list[len(time_list)-1]:
            cut_off = time_list[index] + 86400000
            sum_val=0
            avg = 0
            for val in val_new_list:
                sum_val = sum_val+val
                avg = sum_val/len(val_new_list)
            avg_dict["Value"].append(avg)
            avg_dict["Timestamp"].append(time_new_list[0])

    else:
        cut_off = time_list[index] + 86400000
        sum_val = 0
        avg = 0
        for val in val_new_list:
            sum_val = sum_val+val
            avg = sum_val/len(val_new_list)
        avg_dict["Value"].append(avg)
        avg_dict["Timestamp"].append(time_new_list[0])
        val_new_list = []
        time_new_list = []
        val_new_list.append(value_list[index])
        time_new_list.append(time_list[index])

return avg_dict
```

This function determines average of values of a day and then returns a dictionary consisting of Timestamps on a daily basis with its value.

Explaining the operation,
The cut off time establishes the amount of time in 24 hours. Hence, after every cut off time, a new day is initialized.
Hence, leveraging the cut off time, Average of values of each day is calculated and store against the timestamp of that particular day.

3. Rabbitmq publishing



```
# Function : To send data to Rabbitmq broker
def rabbitmq_prodcer(data):

    # IP address of the rabbitmq broker
    rabbitmq_ip = "192.168.0.100"

    # Port Number of the rabbitmq broker
    rabbitmq_port = 5672

    # Queue name
    rabbitmq_queque = "CSC8112"

    # Message to be sent
    msg = data

    # 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()
```

The IP address and port is configured inaccordance to where the service is running. In this case, in the Cloud VM.
The payload here is sent after the preprocessing of the data is finished.
It is being sent again with a so called topic, for the consumer to understand the message.

Docker Implementation

Docker in Task2 is used for two main reasons,

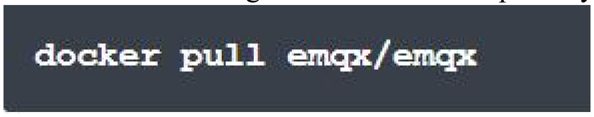
a. To run the EMQX (MQTT Broker) service
b. To run the task2.py python file to execute the edge computing functions

Additional Commands

1. To check the list of docker images present in the system
   docker images
2. To check the list of active containers running in docker
   docker ps
3. To stop a container from running the service
   docker stop image_name/tag_id
4. To remove the container
   docker rm image_name/tag_id
5. To remove a docker image from the system memory
   docker image rm image_name/tag_id
6. To stop a docker compose service
   docker-compose down / Ctrl + c

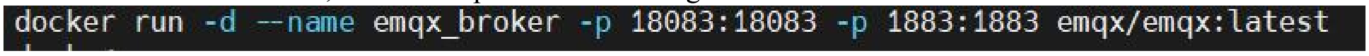
EMQX

1. Pull the official image from the docker repository



```
docker pull emqx/emqx
```

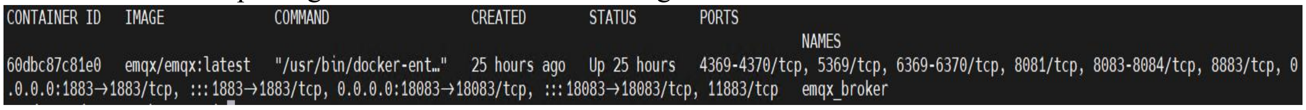
2. After successful download, now the step is to run the image



```
docker run -d --name emqx_broker -p 18083:18083 -p 1883:1883 emqx/emqx:latest
```

--name is the name of the service, in this case emqx_broker
Following that are the port numbers, and finally the image name

3. Command: docker ps - signifies if the service is running or not

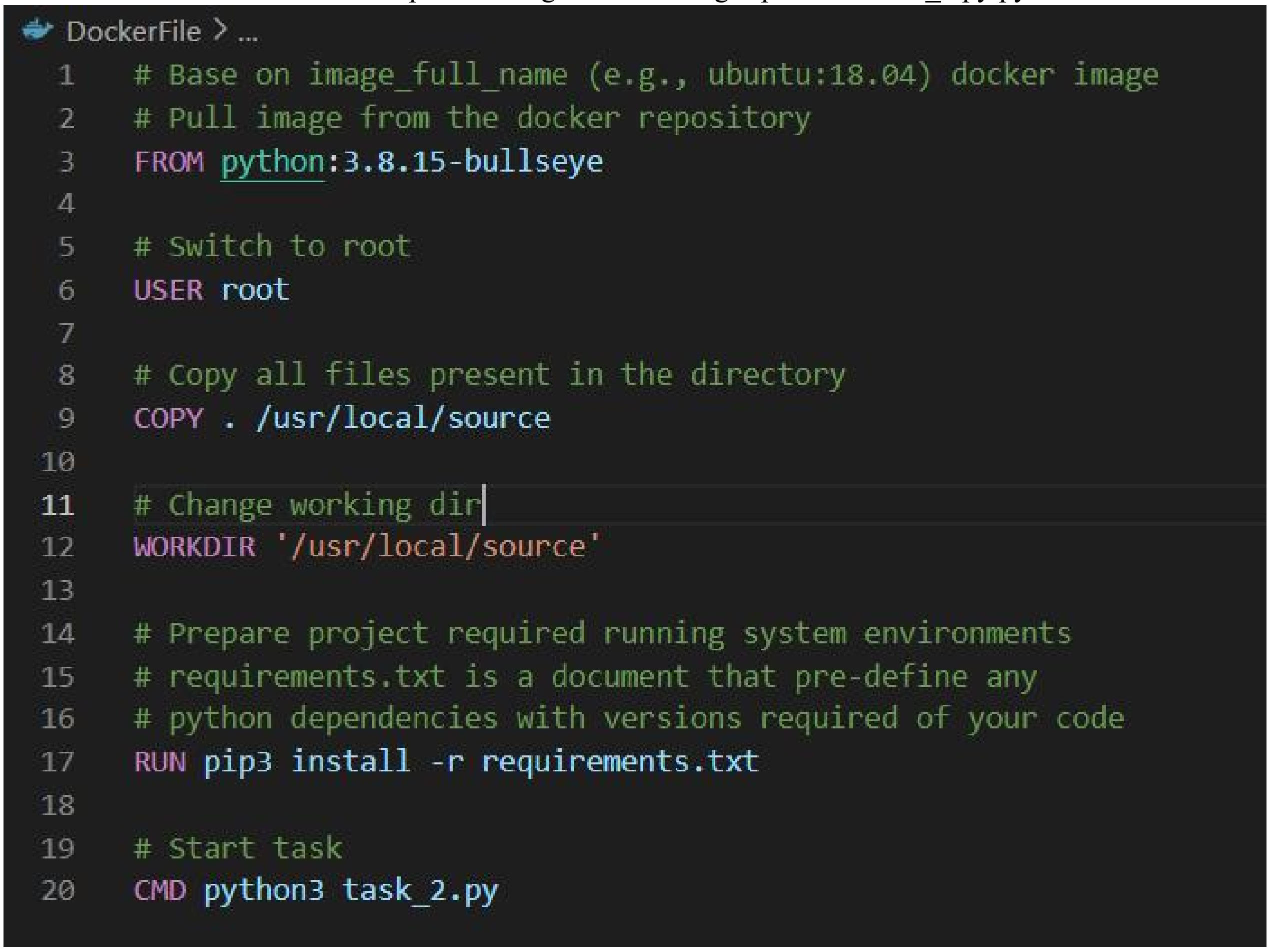


```
CONTAINER ID   IMAGE              COMMAND                  CREATED        STATUS         PORTS
                                                                                                  NAMES
60dbc87c81e0   emqx/emqx:latest   "/usr/bin/docker-ent…"   25 hours ago   Up 25 hours    4369-4370/tcp, 5369/tcp, 6369-6370/tcp, 8081/tcp, 8083-8084/tcp, 8883/tcp, 0
.0.0.0:1883->1883/tcp, :::1883->1883/tcp, 0.0.0.0:18083->18083/tcp, :::18083->18083/tcp, 11883/tcp   emqx_broker
```

Therefore, the EQMX service is running and ready to pass messages upon connection with clients

Task2

1. Create a docker file which will help in creating a docker image specific to task_2.py python file.

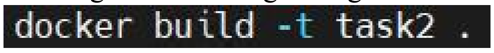


DockerFile > ...

```
# Base on image_full_name (e.g., ubuntu:18.04) docker image
# Pull image from the docker repository
FROM python:3.8.15-bullseye

# Switch to root
USER root

# Copy all files present in the directory
COPY . /usr/local/source

# Change working dir
WORKDIR '/usr/local/source'

# Prepare project required running system environments
# requirements.txt is a document that pre-define any
# python dependencies with versions required of your code
RUN pip3 install -r requirements.txt

# Start task
CMD python3 task_2.py
```

Note: The name of the file should be saved as Dockerfile
Note: Be aware of the location of the file is in correspondence to the directory mentioned in the code.
Explanation of the docker file is in the comments

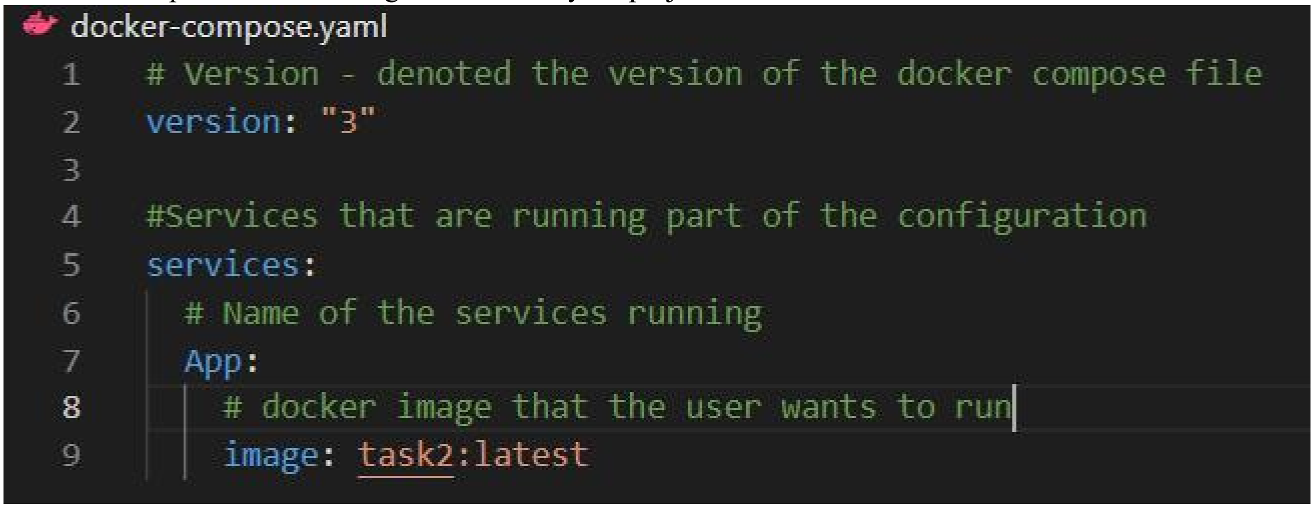
2. Creating a docker image using the docker file



```
docker build -t task2 .
```

This command searches for a file called Dockerfile and then builds a image based on the configurations provided.

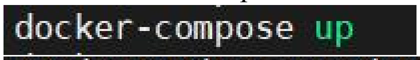
3. Creating a docker compose file
   A docker compose file is a configuration file of your project



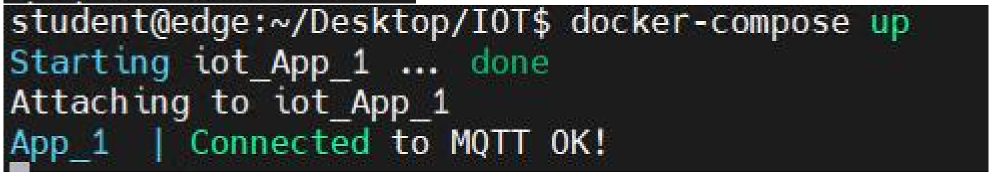
```yaml
docker-compose.yaml
# Version - denoted the version of the docker compose file
version: "3"

#Services that are running part of the configuration
services:
  # Name of the services running
  App:
    # docker image that the user wants to run
    image: task2:latest
```

   Save the configuration file as. yaml in the same directory as the project.

4. Run the docker compose file to start the service (task2)



```
docker-compose up
```

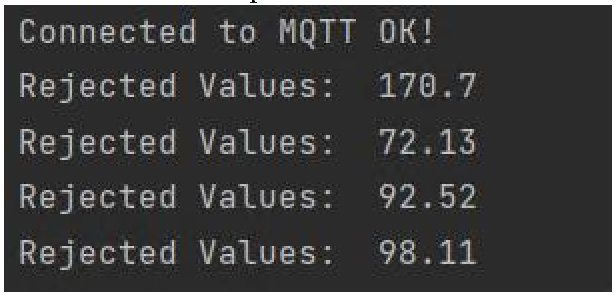


```
student@edge:~/Desktop/IOT$ docker-compose up
Starting iot_App_1 ...  done
Attaching to iot_App_1
App_1  | Connected to MQTT OK!
```

   In conclusion, both the docker images are up and running

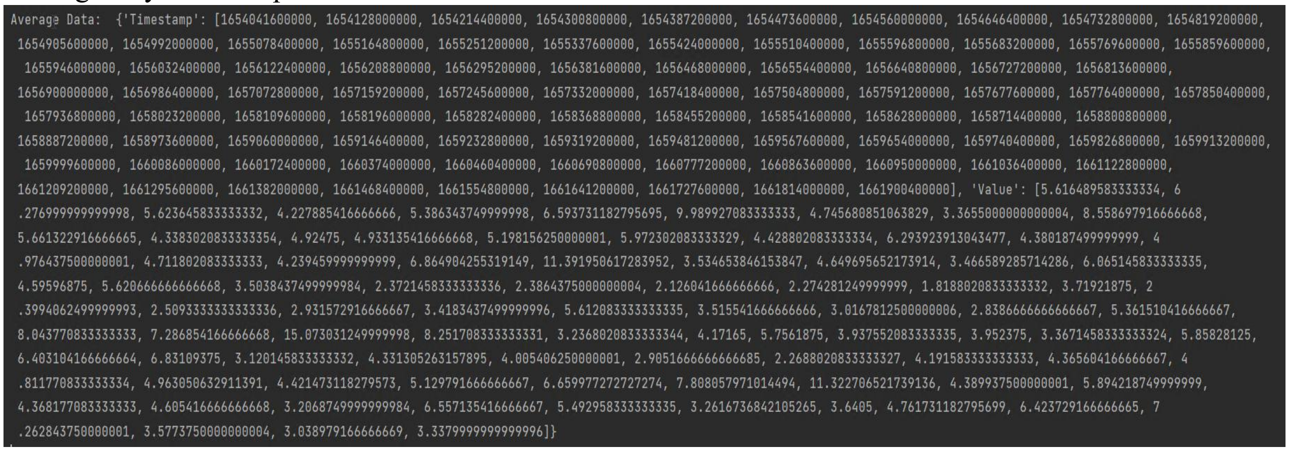
Output Execution

1. Outlier Function Output



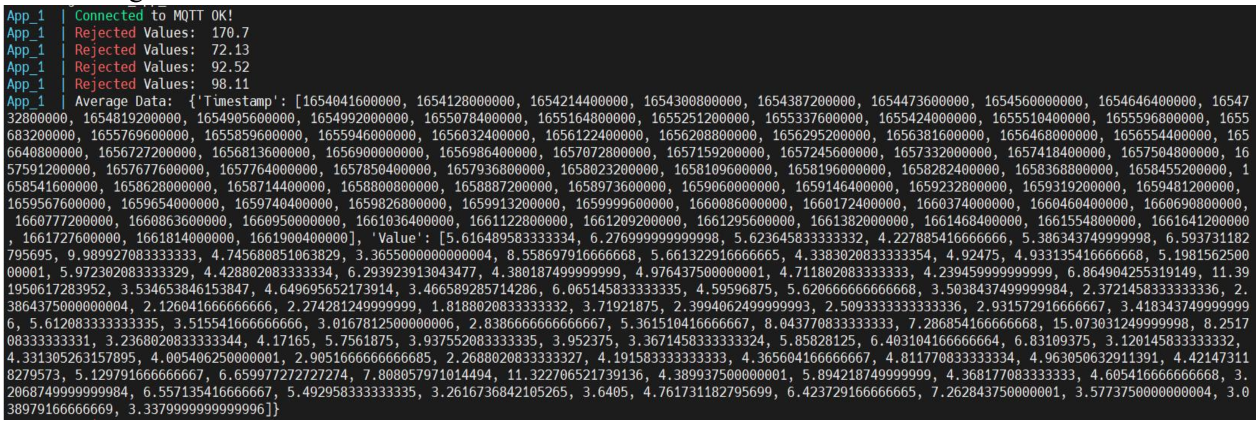
```
Connected to MQTT OK!
Rejected Values:  170.7
Rejected Values:  72.13
Rejected Values:  92.52
Rejected Values:  98.11
```

2. Average Day Data Output



```
Average Data:  {'Timestamp': [1654041600000, 1654128000000, 1654214400000, 1654300800000, 1654387200000, 1654473600000, 1654560000000, 1654646400000, 1654732800000, 1654819200000,
 1654905600000, 1654992000000, 1655078400000, 1655164800000, 1655251200000, 1655337600000, 1655424000000, 1655510400000, 1655596800000, 1655683200000, 1655769600000, 1655859600000,
  1655946000000, 1656032400000, 1656122400000, 1656208800000, 1656295200000, 1656381600000, 1656468000000, 1656554400000, 1656640800000, 1656727200000, 1656813600000,
 1656900000000, 1656986400000, 1657072800000, 1657159200000, 1657245600000, 1657332000000, 1657418400000, 1657504800000, 1657591200000, 1657677600000, 1657764000000, 1657850400000,
  1657936800000, 1658023200000, 1658109600000, 1658196000000, 1658282400000, 1658368800000, 1658455200000, 1658541600000, 1658628000000, 1658714400000, 1658800800000,
 1658887200000, 1658973600000, 1659060000000, 1659146400000, 1659232800000, 1659319200000, 1659481200000, 1659567600000, 1659654000000, 1659740400000, 1659826800000, 1659913200000,
  1659999600000, 1660086000000, 1660172400000, 1660374000000, 1660460400000, 1660690800000, 1660777200000, 1660863600000, 1660950000000, 1661036400000, 1661122800000,
 1661209200000, 1661295600000, 1661382000000, 1661468400000, 1661554800000, 1661641200000, 1661727600000, 1661814000000, 1661900400000], 'Value': [5.616489583333334, 6
 .276999999999998, 5.623645833333332, 4.227885416666666, 5.386343749999998, 6.593731182795695, 9.989927083333333, 4.745680851063829, 3.3655000000000004, 8.558697916666668,
 5.661322916666665, 4.3383020833333354, 4.92475, 4.933135416666668, 5.198156250000001, 5.972302083333329, 4.428802083333334, 6.293923913043477, 4.380187499999999, 4
 .976437500000001, 4.711802083333333, 4.239459999999999, 6.864904255319149, 11.391950617283952, 3.534653846153847, 4.649695652173914, 3.466589285714286, 6.065145833333335,
 4.59596875, 5.620666666666668, 3.5038437499999984, 2.3721458333333336, 2.3864375000000004, 2.126041666666666, 2.274281249999999, 1.8188020833333332, 3.71921875, 2
 .3994062499999993, 2.5093333333333336, 2.931572916666667, 3.4183437499999996, 5.612083333333335, 3.515541666666666, 3.0167812500000006, 2.838666666666667, 5.361510416666667,
 8.043770833333333, 7.286854166666668, 15.073031249999998, 8.251708333333331, 3.2368020833333344, 4.17165, 5.7561875, 3.937552083333335, 3.952375, 3.3671458333333324, 5.85828125,
 6.403104166666664, 6.83109375, 3.120145833333332, 4.331305263157895, 4.005406250000001, 2.9051666666666685, 2.2688020833333327, 4.191583333333333, 4.365604166666667, 4
 .811770833333334, 4.963050632911391, 4.421473118279573, 5.129791666666667, 6.659977272727274, 7.808057971014494, 11.322706521739136, 4.389937500000001, 5.894218749999999,
 4.368177083333333, 4.605416666666668, 3.2068749999999984, 6.557135416666667, 5.492958333333335, 3.261673684210265, 3.6405, 4.761731182795699, 6.423729166666665, 7
 .262843750000001, 3.5773750000000004, 3.038979166666669, 3.3379999999999996]}
```

3. Docker Logs for the same

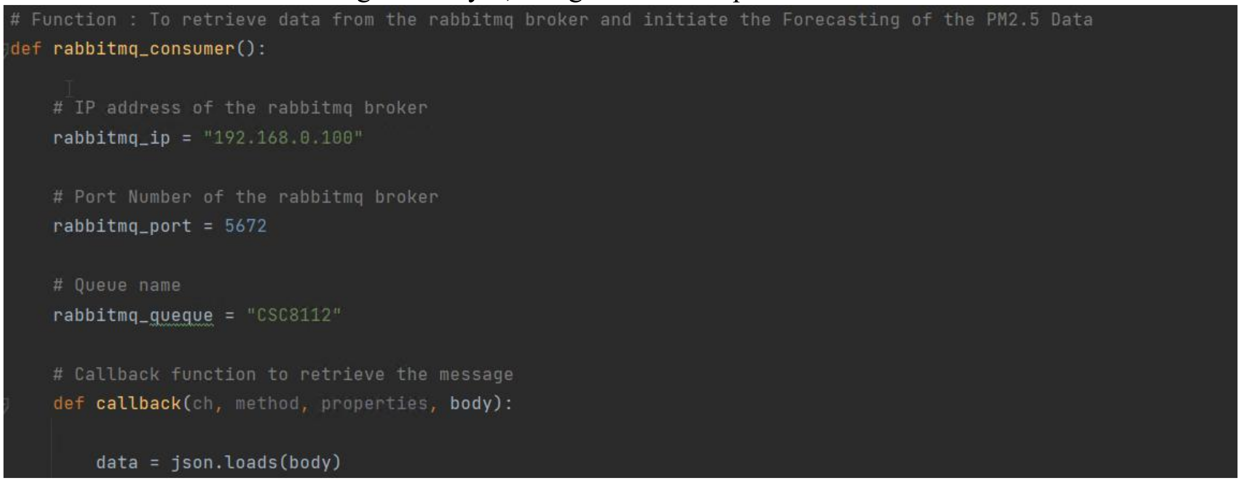


```
App_1  | Connected to MQTT OK!
App_1  | Rejected Values:  170.7
App_1  | Rejected Values:  72.13
App_1  | Rejected Values:  92.52
App_1  | Rejected Values:  98.11
App_1  | Average Data:  {'Timestamp': [1654041600000, 1654128000000, 1654214400000, 1654300800000, 1654387200000, 1654473600000, 1654560000000, 1654646400000, 16547
32800000, 1654819200000, 1654905600000, 1654992000000, 1655078400000, 1655164800000, 1655251200000, 1655337600000, 1655424000000, 1655510400000, 1655596800000, 1655
683200000, 1655769600000, 1655859600000, 1655946000000, 1656032400000, 1656122400000, 1656208800000, 1656295200000, 1656381600000, 1656468000000, 1656554400000, 165
6640800000, 1656727200000, 1656813600000, 1656900000000, 1656986400000, 1657072800000, 1657159200000, 1657245600000, 1657332000000, 1657418400000, 1657504800000, 16
57591200000, 1657677600000, 1657764000000, 1657850400000, 1657936800000, 1658023200000, 1658109600000, 1658196000000, 1658282400000, 1658368800000, 1658455200000, 1
658541600000, 1658628000000, 1658714400000, 1658800800000, 1658887200000, 1658973600000, 1659060000000, 1659146400000, 1659232800000, 1659319200000, 1659481200000,
1659567600000, 1659654000000, 1659740400000, 1659826800000, 1659913200000, 1659999600000, 1660086000000, 1660172400000, 1660374000000, 1660460400000, 1660690800000,
 1660777200000, 1660863600000, 1660950000000, 1661036400000, 1661122800000, 1661209200000, 1661295600000, 1661382000000, 1661468400000, 1661554800000, 1661641200000
, 1661727600000, 1661814000000, 1661900400000], 'Value': [5.616489583333334, 6.276999999999998, 5.623645833333332, 4.227885416666666, 5.386343749999998, 6.593731182
795695, 9.989927083333333, 4.745680851063829, 3.3655000000000004, 8.558697916666668, 5.661322916666665, 4.338302083333354, 4.92475, 4.933135416666668, 5.1981562500
00001, 5.972302083333329, 4.428802083333334, 6.293923913043477, 4.380187499999999, 4.976437500000001, 4.711802083333333, 4.239459999999999, 6.864904255319149, 11.39
1950617283952, 3.534653846153847, 4.649695652173914, 3.466589285714286, 6.065145833333335, 4.59596875, 5.620666666666668, 3.5038437499999984, 2.3721458333333336, 2.
3864375000000004, 2.126041666666666, 2.274281249999999, 1.8188020833333332, 3.71921875, 2.3994062499999993, 2.5093333333333336, 2.931572916666667, 3.418343749999999
6, 5.612083333333335, 3.515541666666666, 3.0167812500000006, 2.838666666666667, 5.361510416666667, 8.043770833333333, 7.286854166666668, 15.073031249999998, 8.2517
08333333331, 3.2368020833333344, 4.17165, 5.7561875, 3.937552083333335, 3.952375, 3.3671458333333324, 5.85828125, 6.403104166666664, 6.83109375, 3.120145833333332,
4.331305263157895, 4.005406250000001, 2.9051666666666685, 2.2688020833333327, 4.191583333333333, 4.365604166666667, 4.811770833333334, 4.963050632911391, 4.42147311
8279573, 5.129791666666667, 6.659977272727274, 7.808057971014494, 11.322706521739136, 4.389937500000001, 5.894218749999999, 4.368177083333333, 4.605416666666668, 3.
2068749999999984, 6.557135416666667, 5.492958333333335, 3.2616736842105265, 3.6405, 4.761731182795699, 6.423729166666665, 7.262843750000001, 3.5773750000000004, 3.0
38979166666669, 3.3379999999999996]}
```

Implementation: Task 3

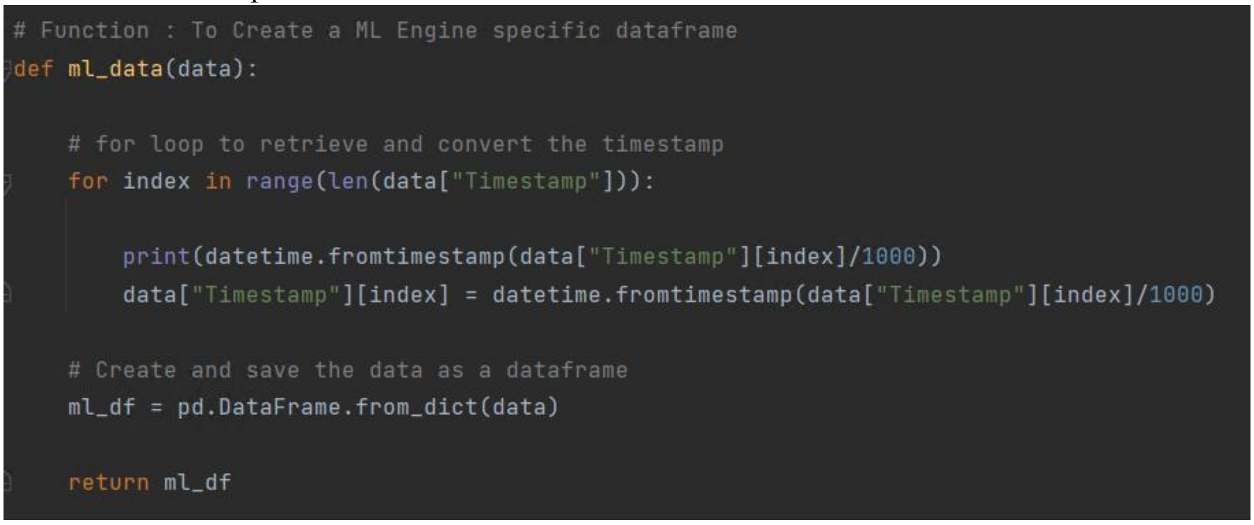
Python Code Implementation

1. Retrieve the data from the Edge VM layer, using the rabbitmq consumer



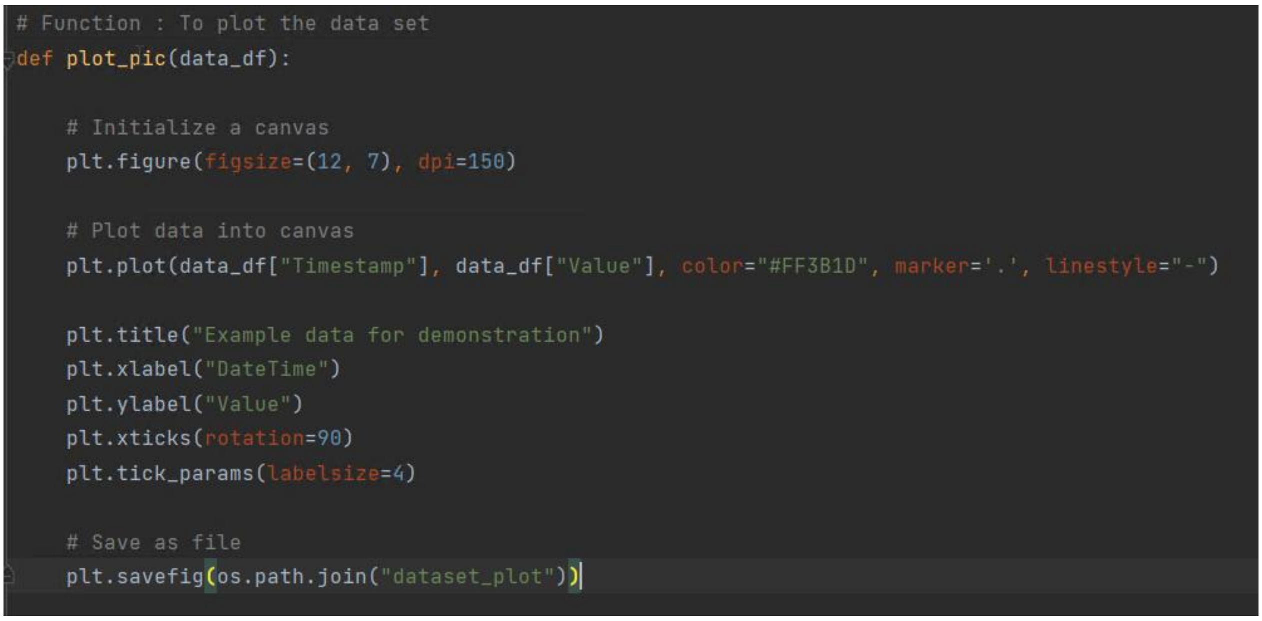
```
# Function : To retrieve data from the rabbitmq broker and initiate the Forecasting of the PM2.5 Data
def rabbitmq_consumer():

    # IP address of the rabbitmq broker
    rabbitmq_ip = "192.168.0.100"

    # Port Number of the rabbitmq broker
    rabbitmq_port = 5672

    # Queue name
    rabbitmq_queque = "CSC8112"

    # Callback function to retrieve the message
    def callback(ch, method, properties, body):

        data = json.loads(body)
```

2. Create a data frame specific to the ML model



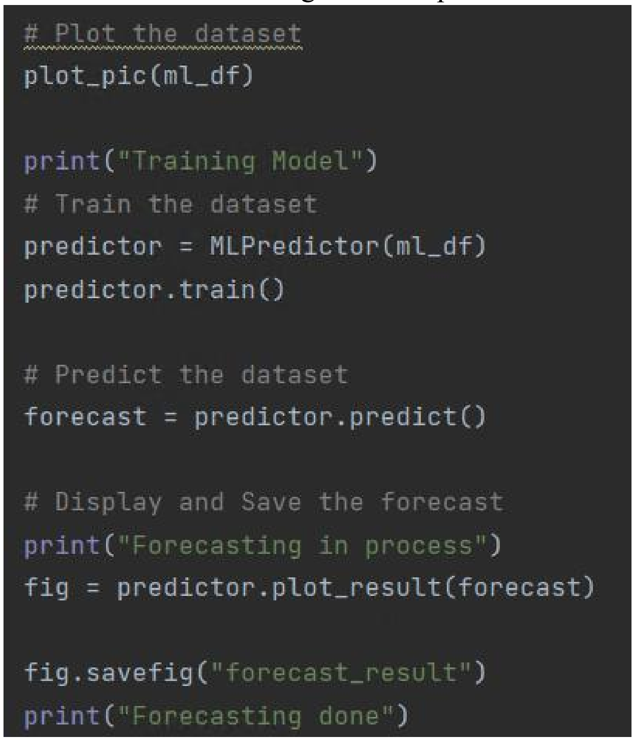
```
# Function : To Create a ML Engine specific dataframe
def ml_data(data):

    # for loop to retrieve and convert the timestamp
    for index in range(len(data["Timestamp"])):

        print(datetime.fromtimestamp(data["Timestamp"][index]/1000))
        data["Timestamp"][index] = datetime.fromtimestamp(data["Timestamp"][index]/1000)

    # Create and save the data as a dataframe
    ml_df = pd.DataFrame.from_dict(data)

    return ml_df
```

3. Plot the Data Set



```
# Function : To plot the data set
def plot_pic(data_df):

    # Initialize a canvas
    plt.figure(figsize=(12, 7), dpi=150)

    # 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.ylabel("Value")
    plt.xticks(rotation=90)
    plt.tick_params(labelsize=4)

    # Save as file
    plt.savefig(os.path.join("dataset_plot"))
```

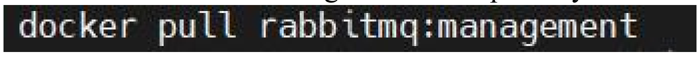
4. Use the Machine Learning Model to predict



```
# Plot the dataset
plot_pic(ml_df)

print("Training Model")
# Train the dataset
predictor = MLPredictor(ml_df)
predictor.train()

# Predict the dataset
forecast = predictor.predict()

# Display and Save the forecast
print("Forecasting in process")
fig = predictor.plot_result(forecast)

fig.savefig("forecast_result")
print("Forecasting done")
```

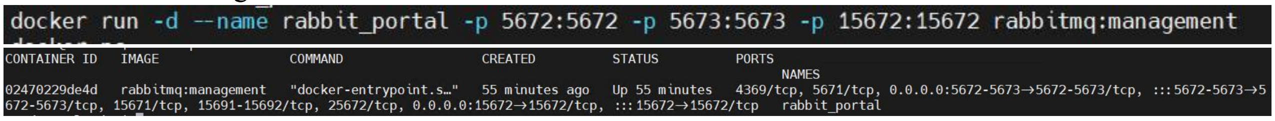
Docker Implementation

1. Pull the official docker image from the repository



```
docker pull rabbitmq:management
```

2. Run the docker image

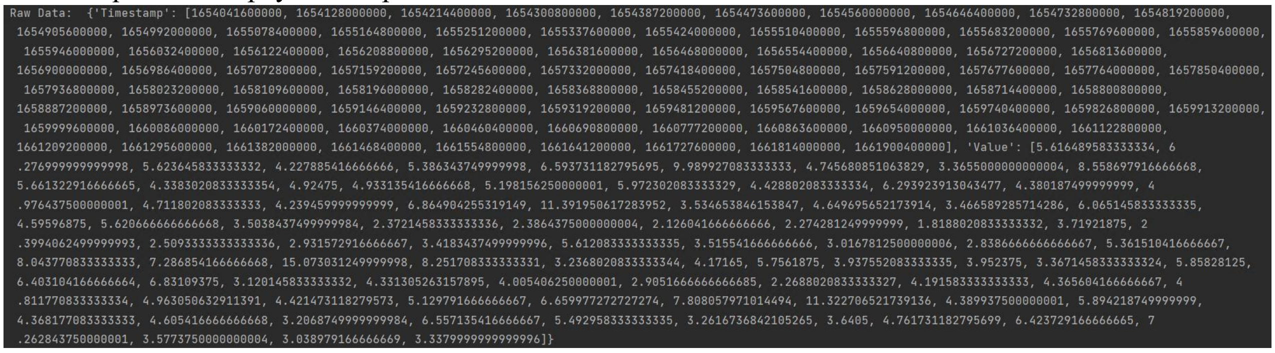


```
docker run -d --name rabbit_portal -p 5672:5672 -p 5673:5673 -p 15672:15672 rabbitmq:management
```

```
CONTAINER ID   IMAGE                 COMMAND                  CREATED          STATUS          PORTS
                                                                                                      NAMES
02470229de4d   rabbitmq:management   "docker-entrypoint.s…"   55 minutes ago   Up 55 minutes   4369/tcp, 5671/tcp, 0.0.0.0:5672-5673->5672-5673/tcp, :::5672-5673->5
672-5673/tcp, 15671/tcp, 15691-15692/tcp, 25672/tcp, 0.0.0.0:15672->15672/tcp, :::15672->15672/tcp   rabbit_portal
```

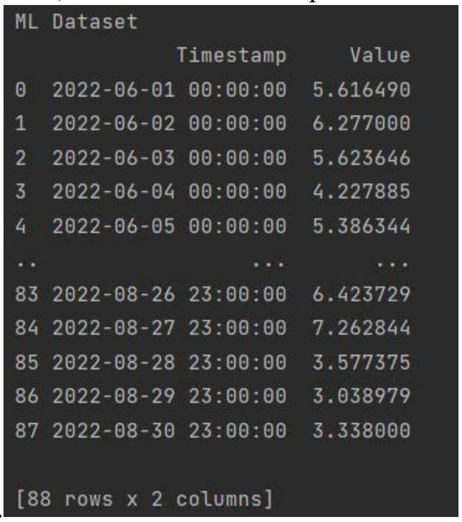
Output Implementation

1. Rabbitmq consumer payload response



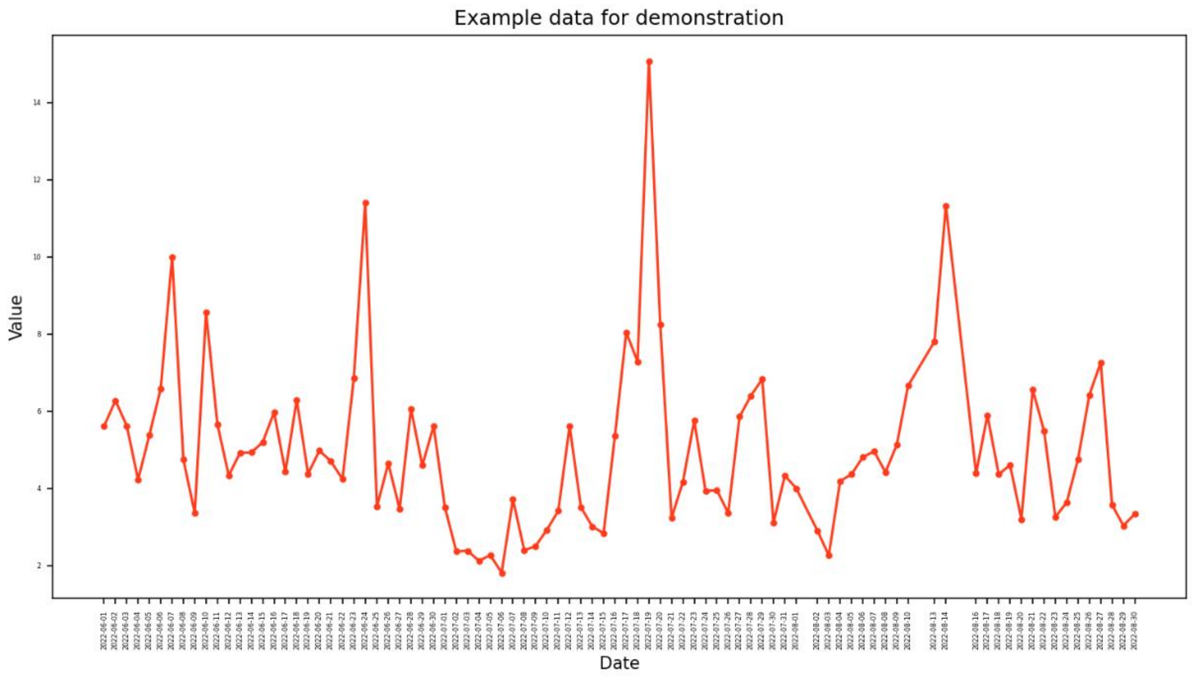
```
Raw Data:  {'Timestamp': [1654041600000, 1654128000000, 1654214400000, 1654300800000, 1654387200000, 1654473600000, 1654560000000, 1654646400000, 1654732800000, 1654819200000,
 1654905600000, 1654992000000, 1655078400000, 1655164800000, 1655251200000, 1655337600000, 1655424000000, 1655510400000, 1655596800000, 1655683200000, 1655769600000, 1655859600000,
  1655946000000, 1656032400000, 1656122400000, 1656208800000, 1656295200000, 1656381600000, 1656468000000, 1656554400000, 1656640800000, 1656727200000, 1656813600000,
 1656900000000, 1656986400000, 1657072800000, 1657159200000, 1657245600000, 1657332000000, 1657418400000, 1657504800000, 1657591200000, 1657677600000, 1657764000000, 1657850400000,
  1657936800000, 1658023200000, 1658109600000, 1658196000000, 1658282400000, 1658368800000, 1658455200000, 1658541600000, 1658628000000, 1658714400000, 1658800800000,
 1658887200000, 1658973600000, 1659060000000, 1659146400000, 1659232800000, 1659319200000, 1659481200000, 1659567600000, 1659654000000, 1659740400000, 1659826800000, 1659913200000,
  1659999600000, 1660086000000, 1660172400000, 1660374000000, 1660460400000, 1660690800000, 1660777200000, 1660863600000, 1660950000000, 1661036400000, 1661122800000,
 1661209200000, 1661295600000, 1661382000000, 1661468400000, 1661554800000, 1661641200000, 1661727600000, 1661814000000, 1661900400000], 'Value': [5.616489583333334, 6
 .276999999999998, 5.623645833333332, 4.227885416666666, 5.386343749999998, 6.593731182795695, 9.989927083333333, 4.745680851063829, 3.3655000000000004, 8.558697916666668,
 5.661322916666665, 4.3383020833333354, 4.92475, 4.933135416666668, 5.198156250000001, 5.972302083333329, 4.428802083333334, 6.293923913043477, 4.380187499999999, 4
 .976437500000001, 4.711802083333333, 4.239459999999999, 6.864904255319149, 11.391950617283952, 3.534653846153847, 4.649695652173914, 3.466589285714286, 6.065145833333335,
 4.59596875, 5.620666666666668, 3.5038437499999984, 2.3721458333333336, 2.3864375000000004, 2.126041666666666, 2.274281249999999, 1.8188020833333332, 3.71921875, 2
 .3994062499999993, 2.5093333333333336, 2.931572916666667, 3.4183437499999996, 5.612083333333335, 3.515541666666666, 3.0167812500000006, 2.838666666666667, 5.361510416666667,
 8.043770833333333, 7.286854166666668, 15.073031249999998, 8.251708333333331, 3.2368020833333344, 4.17165, 5.7561875, 3.937552083333335, 3.952375, 3.3671458333333324, 5.85828125,
 6.403104166666664, 6.83109375, 3.120145833333332, 4.331305263157895, 4.005406250000001, 2.9051666666666685, 2.2688020833333327, 4.191583333333333, 4.365604166666667, 4
 .811770833333334, 4.963050632911391, 4.421473118279573, 5.129791666666667, 6.659977272727274, 7.808057971014494, 11.322706521739136, 4.389937500000001, 5.894218749999999,
 4.368177083333333, 4.605416666666668, 3.2068749999999984, 6.557135416666667, 5.492958333333335, 3.2616736842105265, 3.6405, 4.761731182795699, 6.423729166666665, 7
 .262843750000001, 3.5773750000000004, 3.038979166666669, 3.3379999999999996]}
```

2. ML Dataset, where the time stamp is converted to the required format.

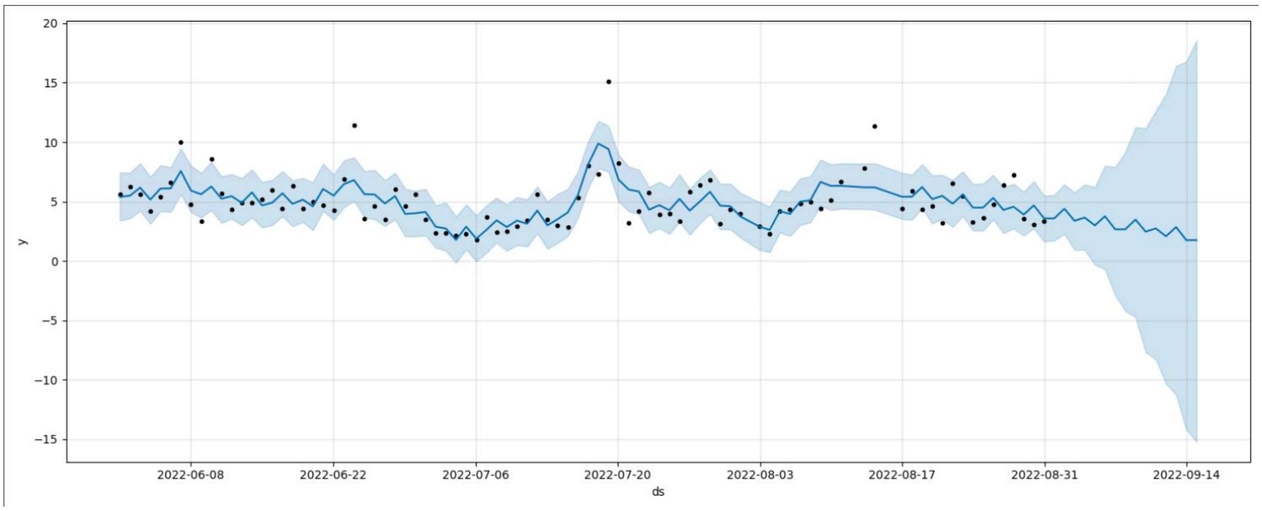


```
ML Dataset
             Timestamp     Value
0  2022-06-01 00:00:00  5.616490
1  2022-06-02 00:00:00  6.277000
2  2022-06-03 00:00:00  5.623646
3  2022-06-04 00:00:00  4.227885
4  2022-06-05 00:00:00  5.386344
..                 ...       ...
83 2022-08-26 23:00:00  6.423729
84 2022-08-27 23:00:00  7.262844
85 2022-08-28 23:00:00  3.577375
86 2022-08-29 23:00:00  3.038979
87 2022-08-30 23:00:00  3.338000

[88 rows x 2 columns]
```

3. Visual Plot of Data Set



4. ML Prediction



Analytical Discussion

The data set plot is used to describe the payload used by the machine learning model. It gives us insights on how the data is changing over time, while the prediction of the forecast is challenged. The plot shows, how there are multiple high points, but the major amount of data points resides on the normal range of 2 to 6. While, there are points which are above 6. Using this as the base of our prime knowledge, the forecast result, presents itself almost accurate to the past data.

The Prediction Plot, describes, after 31st Aug,22 up until 14th Sept, 22, the value of PM2.5 particles in the air is going to reduce slightly over the 15 days. It starts of with close to 5 but then gradually decreases to around 2.5, which is denoted by the dark blue line.

(Note: The black points are the past data set points)

The outer blue lines, leading up to a cone shape horizontally, describes the maximum and minimum range of values that the particle can obtain. The model says that there are chances of it to fall up until in the ranges of 18. While that can be defended, as in the past there have been countable points which are above 12 and thus the model also predicts there is a possibility of it to reach those levels.

Although, the values going below 0 is not considered as that is not emulating the real world scenario.