**TASK-1**

The project on pollution data was done on the data pollutionData209960.csv using R programming language .The initially the data was loaded into memory with parameters :

ozone: measures the concentration of ozone in the air

particulate\_matter: measures the concentration of small particles in the air that can be inhaled

carbon\_monoxide: measures the concentration of carbon monoxide in the air, which can be harmful to human health

sulfur\_dioxide: measures the concentration of sulfur dioxide in the air, which can contribute to acid rain and other environmental problems

nitrogen\_dioxide: measures the concentration of nitrogen dioxide in the air, which can contribute to smog and other environmental problems

longitude: longitude coordinate

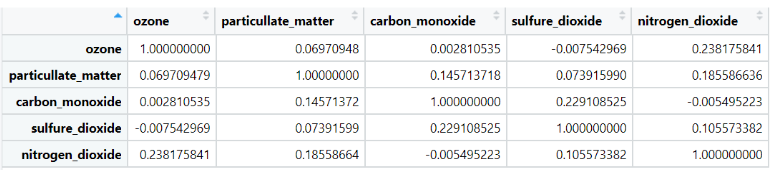
latitude: latitude coordinate

timestamp: date(YYYY/MM/DD) with timing for recorded for every 5 minutes.

The data contained about 17568 rows and 8 columns.To check for null values apply() function is used but it resulted in 0 null values.Data is subseted to find correlation values for numeric features by removing the categorical features using cor() function.We used ggplot for plotting distribution of features.

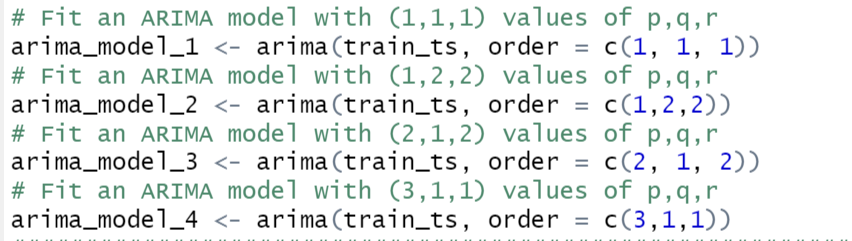
Summary:

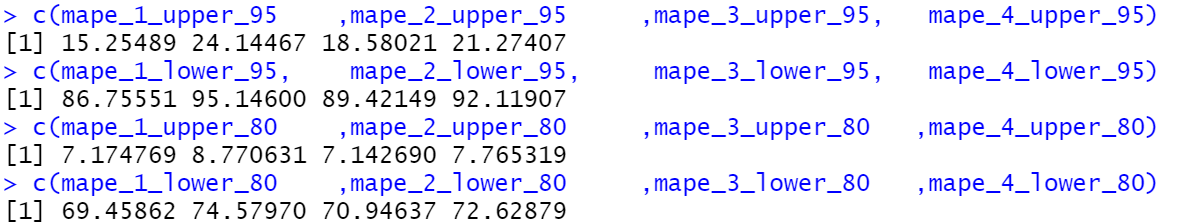
Correlation plot:



**TASK-2**

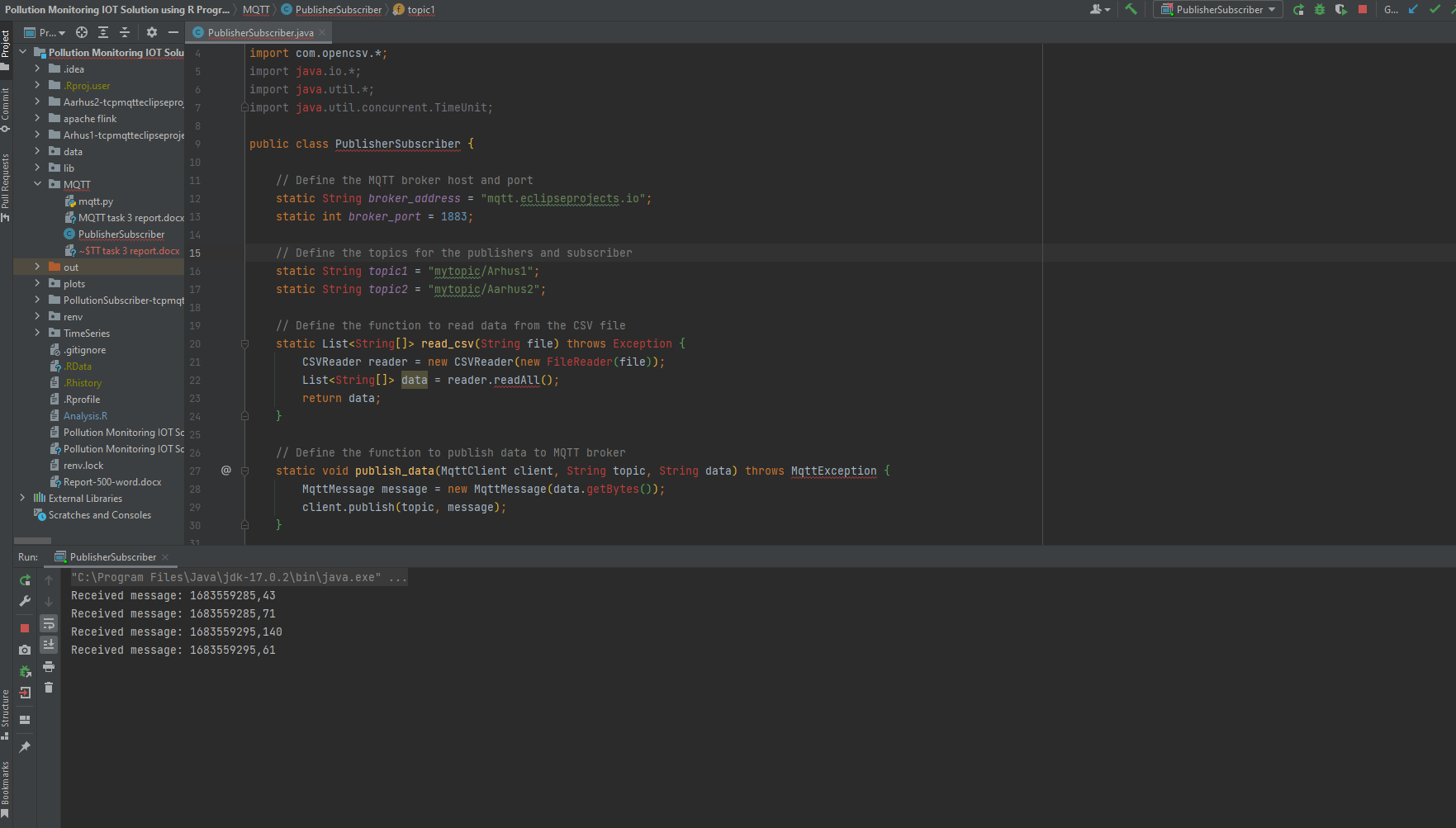
The data was split into training and testing datasets through a manual percentage split, with 97% allocated for training and 3% for testing. The selection of 3% was based on obtaining approximately 527 rows. The attribute "carbon\_monoxide" was used in the ARIMA model, and the frequency was set to 288 using the code "train\_ts <- ts(train\_data, frequency = 288)" to account for the 288 5-minute intervals in every 24 hours. Four types of ARIMA models were considered for training and prediction purposes, with (p,d,q) values of (1,1,1), (1,2,2), (2,1,2), and (3,1,1), selected based on their respective allowable ranges. These models were trained on the training data and tested on the testing data using the forecast function to predict values for the next n intervals. The resulting forecasted values included mean, upper and lower 95% confidence interval bounds, as well as upper and lower 80% confidence interval bounds. Mean Absolute Percentage Error Values were calculated for all four models, and a graph was plotted to identify the best-performing model.





**TASK-3**

The Eclipse Paho MQTT client library is utilized by a Java program to publish and subscribe MQTT messages. The program emulates pollution data from two cities, Aarhus1 and Aarhus2, by reading data from CSV files and publishing it to the MQTT broker at regular 10-second intervals. Two topics, mytopic/Arhus1 and mytopic/Aarhus2, are defined for publishers and subscribers, respectively. Additionally, the MQTT broker host and port are defined as mqtt.eclipseprojects.io and 1883, respectively. The program defines three functions, one to read data from the CSV file, another to publish the data to the MQTT broker, and a third to handle the subscriber's callback function. OpenCSV library is used to read the CSV files, and the publish\_data() function takes the MQTT client, topic, and data as input and publishes the data using the client's publish() method. The subscriber's callback function implements the MqttCallback interface, with messageArrived() method used to print received messages to the console. In the main() method, the program initializes the MQTT clients for publishers and subscribers, connects them to the MQTT broker, and subscribes to the defined topics. Data is then read from the CSV files for Aarhus1 and Aarhus2 publishers, which are then published to their respective topics every 10 seconds in an infinite loop. Thread.sleep() is used to pause the program execution for 10 seconds between publishing cycles.



**TASK-4**

This Python script shows how to subscribe to an MQTT topic and write incoming messages to a Kafka topic using PyFlink. It sets up the MQTT client and Kafka producer, subscribes to the MQTT topic, and assigns the `on\_message()` function to the `on\_message` callback. The script then uses PyFlink to execute the job that listens to incoming MQTT messages and writes them to the Kafka topic. This enables messaging system integration in a distributed system.

