Enhancing Air Quality Prediction in an Intelligent IoT-Powered Application on Google Cloud: A Comparative Study of Machine Learning Approaches

Arjun Paudel (S2271954)

This dissertation is submitted in the partial fulfilment of MSc. Big Data Technologies

School of Computing, Engineering and Built Environment

Glasgow Caledonian University, London

I hereby declare that this paper is all my work, except as indicated in the text:

Data: 10th May 2024

Dr. Hakim Mezali

Project Supervisor

May 2024

Abstract

This paper aims to address the development of IoT-powered applications (Node-Red) on Google Cloud to show the different visualisation of air quality data and implementation of machine learning approaches to predict better air quality index.

The introduction of dissertation gives the general overview of IoT powered application for determining air quality index with the help of Google cloud ecosystem. In traditional methods of air quality monitoring prediction capabilities are often lacking and require expensive equipment's charges. Recent development of IoT technologies to capture the air quality data and support from Cloud based services like Google cloud, helps to predict the air quality index much more accurately. From the literature review it was found that comparison of regression model with neural network model by selecting unique features engineering is limited while predicting air quality index. So, to overcome this issue, this paper aims to predict better air quality index by using machine learning algorithms, one regression (Linear regression) and one neural network (Long Short-Term Memory) model by selecting unique features engineering.

Some methodological process was defined on how IoT application (node-red) can be deploy on Google Cloud to receive data from local server Node-Red by using MQTT protocol which is followed by how machine learning (Linear Regression and Long Short-Term Memory) model framework works to predict air quality data.

This paper's primary body focuses on exploratory data analysis to gain inner insight from the air quality dataset. Furthermore, it elaborates on implementation of regression (Linear Regression) and neural network (Long Short-Term Memory) model by selecting different feature algorithms. From the result, both algorithms give a better result while selecting unique features columns. As Linear regression model give a better predicted AQI while selecting a feature of all the particulate matter (PM) and gases (NO2, O3, CO, SO2) whereas Long Short-Term Memory gives a better result while selecting a features column as both particulate matter (PM2.5, and PM10) along with AQI.

Additionally, this paper suggests for future exploration in this domain where these prediction model could lead to developing a personised air quality alerts application, designing air quality responsive ventilation system, air quality enhanced smart agriculture and pollution sensitive waste management.

Acknowledgement

As an international student doing MSc. Big Data Technologies at Glasgow Caledonian University, London, I have had an amazing and delightful experience. During this period, I have encountered a wide range of individuals who have boosted my confidence and given my life a meaning. This has enabled me to grow on a both personally and academically.

I owe a debt of appreciation to my supervisor, **Dr. Hakim Mezali**, whose advice, perception, and optimistic attitude were invaluable in helping me to complete my master's dissertation. His knowledge and experience in the subject inspired me to further my research.

My heartfelt gratitude goes out to **Prof. Titus Olaniyi**, wonderful program coordinator, for his unwavering support and direction during academic term time at Glasgow Caledonian University. I am profoundly grateful for his mentorship and leadership, which have inspired me to strive for excellence. Additionally, I would like to thank all the lecturers and academic staff of Glasgow Caledonian University, London for their dedication and contribution to my education.

Lastly, I would like to thank my parents, brother and sisters for their unwavering support and understanding, which made it possible for me to continue my studies here in the UK. Because of their encouragement, my time living and studying in the UK has been fantastic.

Table of Content

List of Figures

List of Tables

Chapter One

Introduction

1.1 Introductory Background

The quality of the air we breathe has become an essential concern for public health as well as ecological sustainability in an era filled with increased urbanisation, industry, and vehicle emissions. The intricate combination of pollutants known as air pollution, which includes Particulate Matter (PM), Nitrogen Dioxide (NO2), Sulphur Dioxide (SO2), Carbon Monoxide (CO), and Ozone(O3) poses serious threats to socio-economic growth, environmental integrity, and human health. According to World Health Organisation(WHO) air pollution is one of the worst environmental health hazards, causing an estimated six million seven hundred thousand premature deaths yearly (World Health Organization, 2022).

Negative impact of air pollution on socio-economic areas, environmental, and public health have highlighted how badly we need a reliable air quality prediction system. Traditional approaches often depend on static models unable to consider the dynamic character of environmental elements causing air pollution. The advent of the Internet of Things (IoT) has transformed data gathering capacities, allowing the accumulation of enormous amounts of real-time environmental data with never-before-seen granularity. IoT sensors are outfitted with a variety of environmental sensors and wireless communication capabilities which enable the continuous collection of data related to important air quality parameters such as PM, SO2, CO, O3, and NO2. Making use of this abundance of real-time data might improve the precision and promptness of air quality forecasts, enabling public health professionals, policymakers, and urban planners to carry out focused interventions and mitigation plans.

The research comprises two phases, in the initial phase, a Node-Red application was hosted on both local server and Google Cloud server, facilitating data transmission from the local server to the Google Cloud for visualisation, simulating IoT device functionality. In the subsequent phase, two machine learning models- Linear Regression(LR) and Long Short-Term Memory(LSTM) neural network were implemented, addressing distinct parameters for analysis.

1.2 Problem Statement

In the era of rapidly urbanising cities and growing environmental concerns, it is now critical to estimate air quality accurately. In traditional air quality monitoring methods, real-time data and prediction capabilities are often lacking and required an expensive equipment charge (Sakila and Manohar, 2024). The development of cloud computing platform such as Google Cloud and IoT technologies provides an opportunity to create intelligent system that can provide a more accurate air quality prediction using different machine learning algorithms (Khushalani, 2022).

This paper statement of problem centre on topic of development of intelligent IoT-powered application on Google Cloud while maintaining computational cost, response time and secure system and comparative study of different machine learning approaches for predicting better Air Quality Index (AQI). Despite having the best machine learning model to predict the air quality index, there was a false prediction which constantly misguided the people about AQI. So, to minimize this confusion and predict the more precise AQI, it is necessary to look after every aspect of environmental factor that affects the AQI.

1.3 Research Question

The main purpose of this report is to answer the following research questions.

a) What opportunities and challenges are associated with developing and maintaining an air quality prediction system on Google Cloud Platform, and how may these issues be resolved to guarantee scalability and cost-effectiveness?
b) Which machine learning model exhibits the highest level of accuracy in predicting air quality index?
c) What characteristics have the most considerable influence on the variation of air quality prediction?
d) How distinctive features selection methods be used to improve the accuracy of prediction?

1.4 Aim and Objectives

The research aims to develop an intelligent IoT-Powered application on google cloud platform for predicting air quality index by using Linear Regression and Long Short-Term Memory machine learning approaches.

The objective are as follows:

a) Developing an IoT Simulator in node-red to send the data into Google Cloud Platform.
b) Displaying different visualisation of data that was received from the IoT device (Node-Red PC) on Google Cloud.
c) Evaluate ETL (Extract, Transform, and Load) transformation to gain inner insight from the air quality data.
d) Evaluate different machine learning techniques to predict more precise air quality index.

1.5 Summary of Methodological Approach

The methodology of this paper primarily revolves around two distinct phases: Firstly, it demonstrates the utilisation of IoT sensor devices for data transmission to the Google Cloud Platform, enabling real-time visualization of air quality data. Subsequently, the focus shifts towards the analysis of this data to extract meaningful insight, employing various machine learning algorithms to predict AQI. The whole methodology process is illustrated in ***Figure 1.5.1***.

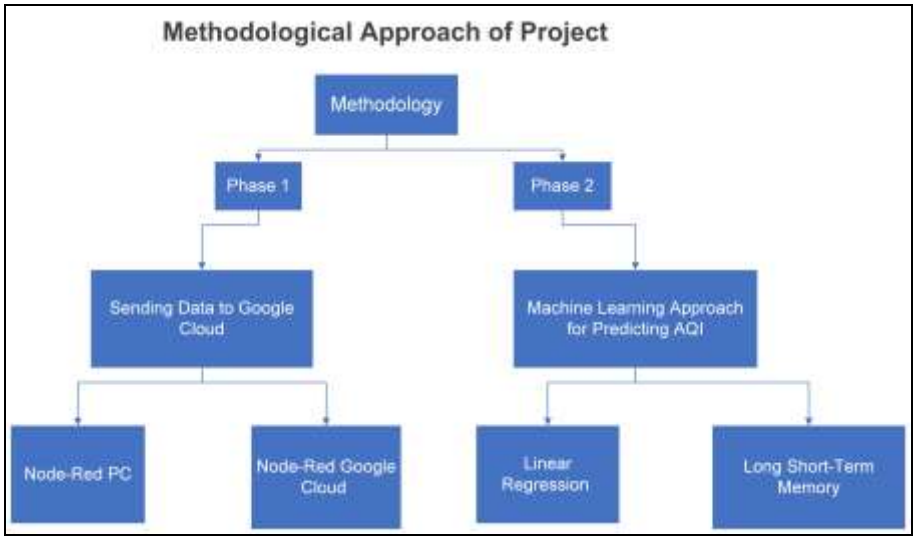


Figure 1.5.1: Methodological Approach of Paper

1.6 Research Contribution

The research contribution of this paper are as follows:

a) Development of intelligent IoT application that shows different visualisation of current air quality data with the help of Node-Red in Google Cloud.
b) Showing the seasonal sequence of air quality data by doing exploratory data analysis.
c) Evaluate the performance of regression (Linear Regression) and Neural Network (Long Short-Term Memory) model in air quality prediction.
d) Evaluate the potential of Long Short-Term Memory networks to capture temporal dependencies in air quality data.
e) Explore how input feature selection and pre-processing technique affect the prediction of air quality index in both Linear regression and Long-Short Term Memory.

1.7 Structure of Report

This paper is divided into five (5) subjects to include, Introduction, Literature Review, Methodological Approach, Discussions and Findings (Implementation), Summary, Conclusion, Recommendation, and Future Work. Chapter One touches on the Introductory Background, Problem Statement, Research Questions, Aim and Objectives, Summary of Methodological Approach, and Research Contributions.

Chapter Two focuses in Literature Review and touches on Introduction, Issue Related to Air Quality Index, IoT Powered Application for Determining Air Quality Index, Uses of Google Cloud Platform in IoT application, Machine Learning Model for Predicting Better Air Quality Index and Summary and Conclusion of Literature Review. Chapter Three focuses on Methodological Approaches and touches on Introduction, sending data to Google Cloud from IoT Device, Machine learning Approaches for Predicting Air Quality Index.

Chapter Four focuses on the Implementation of Discussion and Finding and touches on Introduction, First Phase: IoT Device to Google Cloud, Second Phase: Implementing Machine Learning Model and Key Finding from Implementation. Whereas, Chapter Five focuses on the Summary and Conclusion, Recommendations and Future Work.

Chapter Two

Literature Review

2.1 Introduction

This chapter review literature to develop an artificial intelligent IoT powered applications on Google Cloud platform and predicting air quality index by using different machine learning algorithms. It discusses the issue related to air quality index as reported in Literature. Uses of IoT powered application for determining air quality will be thoroughly explored. Discourse in respect of the uses of Google Cloud platform will be inspected. The various approaches of machine learning model for prediction will be touched. The chapter concludes by summarising the literature with the object of deducing the problem statement and the Research Question as applied in subsequent Chapters.

2.2 Issue Related for Air Quality Index

Government organisations use the Air Quality Index (AQI) to inform the public about the present level of air pollution and the short-term and long-term health implications of such pollution (Hossain *et al.,* 2020). In research published by Lemes (2018) AQI measures air pollution and its potential health impacts. Furthermore, he states that AQI measures the health consequences of breathing in polluted air within a few hours or days. Establishing ambient air quality standard can serve as a foundation for and ensure that ambient air quality is managed to safeguard human heath, preserve natural sustainability, and encourage peaceful environmentally friendly growth that safeguards individuals, communities, and the environment (Fang, Chan and Yao, 2009).

Kumari and Jain (2018) conducted literature research by reviewing and comparing different air quality indexes used globally, and these indexes are from USA, Hong Kong's, and Canada. After this research they come to conclusion that the quantity of index classes (as well as the colours that go with them) and related descriptive words, the pollutants taken into consideration, the class limits, the average times, and the update frequency all varied among the indices. Karavas, Karayannis and Moustakas (2021) do research with a goal of creating a uniform benchmark for evaluating and enhancing air quality throughout Europe. The study contrasts six of AQI already used in the European Union. Due to the inconsistency in their measure, description, and

calculation techniques, it is hard for the public and policymakers to understand air quality and health impact. They suggested that a unified AQI could offer a cogent foundation for dealing with problems related to air pollution.

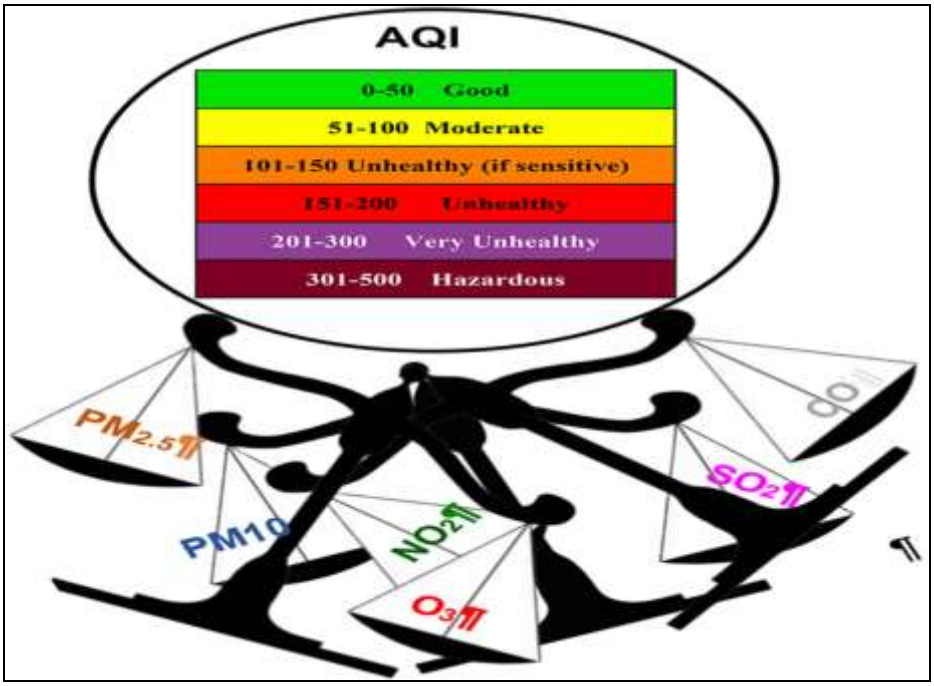


Figure 2.2.1: USA Air Quality Index Level (Horn and Dasgupta, 2023)

Different countries use the different AQI level, ***Figure 2.2.1*** show the AQI level that has been followed by United State of America along with the China, India and many more countries, where air quality level is divided as good, moderate, unhealthy (if sensitive), unhealthy, very unhealthy and hazardous (Shakoor *et al.,* 2020, Janarthanan *et al.,* 2021). ***Figure 2.2.2*** show the AQI level that is current used in United Kingdom, Australia, Hong Kong, South Korea, and Singapore. In United Kingdom AQI is divided into 4 group as low, moderate, high, and extremely high in a scale from 0 to 10 (Jephcote et al., 2021). In Australia AQI is divided into six band as particularly good, good, fair, poor, extremely poor, and hazardous in scale from 0 to 200+. Hong Kong uses the Air Quality Health Index (AQHI) which is divided into 5 bands as low, moderate, high, remarkably high, and hazardous and scale from 1 to 10+ (Tan *et al.,* 2021). South Korea uses the Comprehensive Air quality index (CAI) which is scale from 0 to 500 and divided into 4 band as good, medium, unhealthy, and very unhealthy (Crawford *et al.,* 2021). Whereas in Singapore uses the Pollutant Standards Index where it is divided into 4 bands and scale from 0 to 251+ (Lorenzo, San Tam and Seow, 2021).

Figure 2.2.2: Different Countries AQI Level

2.3 IoT Powered Application for Determining Air Quality

Traditionally, air quality is monitored with immobile reference grade sensor, which are limited by the small area they cover and expensive equipment required to utilise them and both Micro-Balance Particulate Matter (PM) and Portable Light-Scattering PM detectors investigating tools are similarly reliable but very costly ranging from $300 to $2,000 and $50,000 to $100,000 respectively (Sakila and Manohar, 2024). While stationary air quality monitoring sites are also exactly accurate, they can only predict exact results at the specific locations where they are installed and require large initial and ongoing financial investments as well as human resources, and only allow for the hourly collection of data from a restricted number of locations (Golec *et al.,* 2021).

Satellite air quality monitoring also depends on telemetry since it covers a large region, however they do depend heavily on the climate and land-use restrictions (Montrucchio *et al.,* 2020). The precision of air quality measurements can be increased by employing satellite-routed sensors mechanisms, but there are few disadvantages as well, including the potential data collisions, operational delays, and expensive satellite usages (Estoque, 2020). To overcome data collisions issue, (Yang *et al.,* 2017) developed a remarkably accurate three-dimensional AQI mapping method, but its implementation is extremely expensive.

It may be possible to precisely and accurately measure the amounts of PM particle produced by using low-cost IoT devices (Chen et al., 2020). Furthermore, they stated that, low-cost IoT sensor have several benefits, such as being widely available and having the potential to be sent to people living close to industries so that they can check quality of air they consume. ***Figure 2.3.1*** show the decentralized calibration approach in which sensor node can simultaneously display the air quality concentrations both locally and populate on data integration server located at distance (Hashmy *et al.,* 2023).

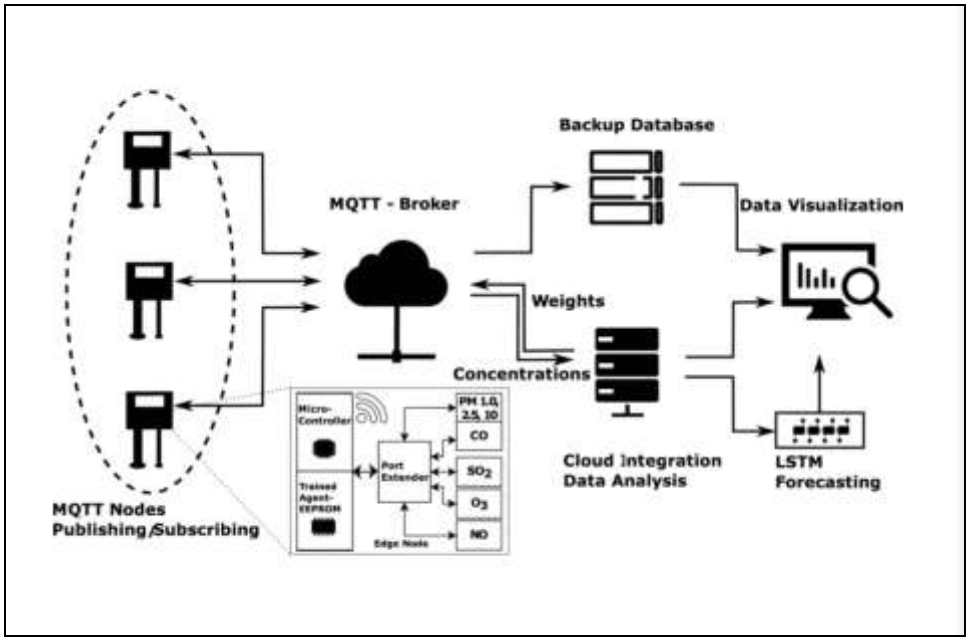


Figure 2.3.1: Mechanism of IoT device to capture and send data to database ((Hashmy et al., 2023)

2.4 Uses of Google Cloud Platform in IoT Application

Having the ability to store and retrieve data and programs online rather than on a hard drive is known as cloud computing (Yin, 2020). Cloud technology is a recent addition to the business skyline, offering several benefits to the majority of business and organisations that now use local data management solutions (Santoso, 2019). Cloud computing, as opposed to traditional hardware and software, allows any organisations to stay on the forefront of technology without having to commit significant funds to acquiring, repairing, and operating infrastructure themselves (Khushalani, 2022). The basis cloud computing model are Infrastructure as a Service (IaaS), Platform as a Service (PaaS), Software as a Service (SaaS) and Recovery as a Service (RaaS) also known as Disaster Recovery as a Service (DRaaS) (Islam *et al*., 2023). **Figure 2.4.1**

show how these models work and different service providers who are based on these models.

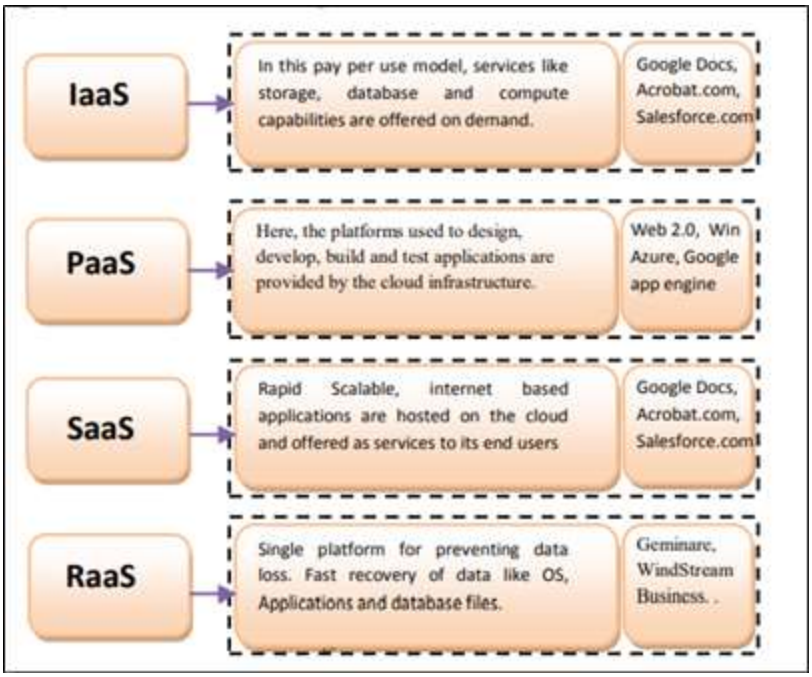


Figure 2.4.1: Basis Cloud Computing Model (Rashid and Chaturvedi, 2019)

Google created the Google Cloud Platform (GCP) in 2011 to provide a cloud-based service to its end-users (Kaushik *et al.,* 2021). Major services that google cloud provides are Storages, Big Data, Internet of Things, Cloud Security, Data Transmission and so on with 121 availability zone around the globe (Mathur, 2024). The two main approach for gathering IoT and Google Cloud is i) Cloud-based IoT which integrates cloud-based IoT features and ii) IoT-Centric Cloud which aims to integrate cloud features with IoT (A. R. Biswas and R. Giaffreda, 2014). **Figure 2.4.2** show the Cloud based IoT architecture which is normally distinguish by three layers that is Perception, Network, and Application layer. Perception layer is responsible for collecting data from sensors, Network layer is responsible for communication whereas Application layer is responsible for user to interact with smart devices and control in accordance with their needs (Rochwerger *et al.,* 2009).

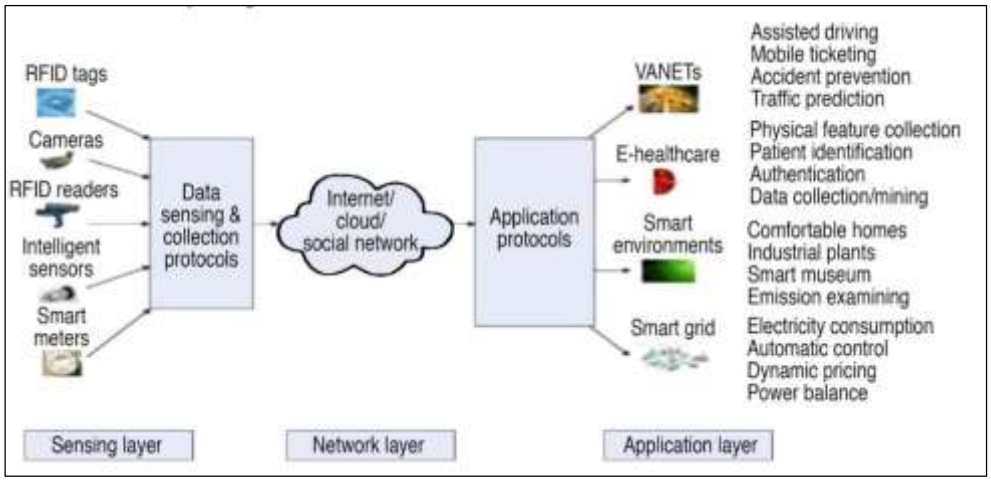


Figure 2.4.2: Cloud Based IoT Architecture (Tyagi and Kumar, 2020)

The Cloud IoT Core (CIC) service and other Google Cloud Platform (GCP) services are part of the GCP-IoT (Google Cloud Platform- Internet of Things) which enables safe connectivity between users, applications, and IoT devices in GCP Cloud (Gupta *et al.,* 2022). Furthermore, the research states that different IoT entities are included in an access control model for GCP-IoT, which should specify how these entities are permitted to efficiently interconnect with each other. **Figure 2.4.3** show Google Cloud Platform IoT Access Control (GCP-IoTAC) model which was designed on top of the Google Cloud access control model and shows the relation between CIC services and other essential services (D. Gupta *et al.,* 2020). The different service components in this model are Cloud IoT Core (CIC), Cloud Pub/Sub (PS), Cloud Function (CF), IoT Devices (IoT), Virtual Devices (VD), Regismes(RG), Topic(T), Function (F), and IoT Operations (OPIoT).

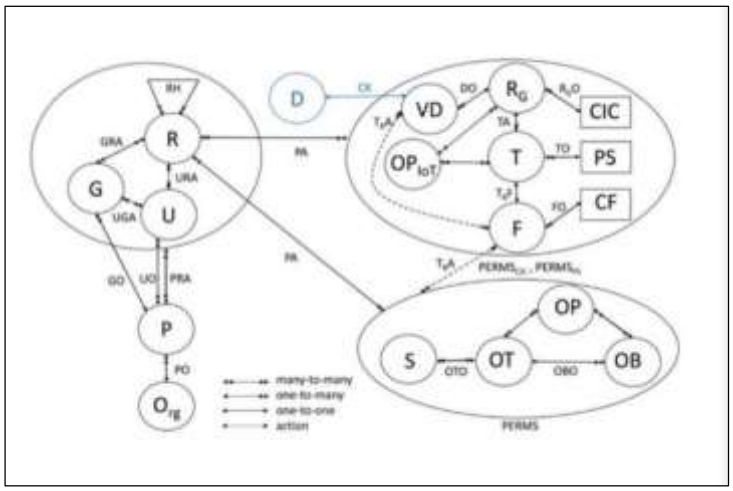


Figure 2.4.3: GCP-IoT Access Control Model within a Single Project (D. Gupta *et al.,* 2020)

2.5 Machine Learning Model for Predicting Better Air Quality Index.

Liu *et al.* (2019) conduct research in two distinct cities: Beijing and an Italian city based on two separated publicly accessible datasets for predicting Nitrogen Oxide (NOx) in an Italian city and AQI for Beijing. For this they employed two machine learning models: Random Forest Regression (RFR), and Support Vector Regression (SVM) Model, where SVM performs more accurate in predicting AQI and RFR performs better in estimating the value of NOx. Aditya et al., (2018) used machine learning algorithms to anticipate and identify the amount of PM2.5 level based in a dataset that included atmospheric variables in a particular city. By using Logistic Regression technique, first they determine if the air is polluted or not and then used Auto Regression model to forecast the quantity of PM2.5 values in future based on historical data. Pasupuleti, Kalyan and Reddy (2020) contrasts Random Forest, Decision Tree, and Linear regression model where with the help of Arduino platform measurements of major air contaminants and meteorological conditions are obtained. The result of this research shows that Random Forest produces the result with more accuracy, but its drawback was shown as higher cost and more memory used.

Du *et al.* (2023) proposed a novel deep learning model named as DeepAir which performs better than other Deep Learning Model to track the impact of metropolitan transport on Shanghai's air quality and forecast surface-level of PM2.5 concentrations. Wu, Song and Peng (2022) used two years of data from the Shanghai roadside station for determining the amount of NO, CO, NO2, and O3 air pollutants in the atmosphere. The study found that the air pollution beneath the elevated highway was worse than then roadside. The Long Short-Term Memory model was suggested by them to determine the air quality where the lowered margin of error was achieved. An integrated Support Vector Regression and Long Short-Term Memory method was introduced by Janarthanan *et al.* (2021) for analysing the air quality index of Chennai India. From this research, the deep learning model gives more accurate and specific values for AQI than any other model.

Shakir and Rakesh (2018) examined the proportions of air pollutants in connection to time of a day and week in the Karnataka city of India, by evaluating environmental parameters including temperature, humidity, and wind speed by using Waikato Environment for Knowledge Analysis (WEKA) tools. Using ZeroR method, results indicated that pollution levels were greater on working days, particularly throughout

peak periods, and lower on weekends and vacations. Furthermore, the research illustrated the relationship between air pollutants and environmental parameters by using a K-means Clustering Method. Random Forest technique was employed by Li *et al.* (2020) to estimate SO2 emission in China, where they contrasted the Random Forest algorithm performance with other machine learning algorithms and found that R2 and RMSE as 0.64 and 17.06 respectively which was more accurate than other machine learning models.

Martínez et al. (2018) found that Random Forest was the most effective method for determining PM10 levels, it is not a reliable indicator of harmful pollutant levels, even when working with partial data sets. Multinomial Logistic Regression and Decision Trees model was employed by Nandini and Fathima (2019) to forecast the number of pollutants in air, first by using K-means Clustering data is clustered into three groups and classify them as low, medium, and high. Then Multinomial Logistic Regression and Decision Tree was applied on training and testing data to predict the future values. From the result Predicted values are much closer to actual values in Multinomial Logistic Regression than in Decision Tree which state that Multinomial Logistic Regression model is consideration for more accurate outcomes.

Wang et al. (2020) compare the Gas Recurrent Neural Network with Support Vector Regression (SVR) and Multilayer Perceptron (MLP) to forecast the degree of air quality and found that because of less sensitive to sensor drift, Gas Recurrent Neural Network executes more effective, but it is more vulnerable to atmospheric humidity.

2.6 Summary and Conclusion of Literature Review

Literature review of this paper conclude that various countries employ different standards of measuring AQI. However, the accompanying ***Figure 2.2.1*** illustrates a commonly adopted standard format utilized by numerous nations, including the United State of America, India, and China. Traditional air quality monitoring methods, including stationary sensors and satellite telemetry, are limited by factors such as inflated costs, location specificity, and data collision issues, while low-cost IoT devices offer a potential solution with benefits including widespread availability and community empowerment. Google Cloud Platform provides numerous services globally, including storage, big data and IoT, and integration of IoT with cloud computing follows two approaches: Cloud-based IoT and IoT-Centric Cloud each with distinct architecture

layer. Numerous studies employ machine learning model to predict AQI in different cities worldwide, with methods including Random Forest Regression, Logistic Regression, Support Vector Regression, LSTM, Gas Recurrent Neural Network, Decision Tree and so on. These approaches offer insight into pollutant levels and aid in prediction of AQI, highlighting the importance of diverse methodologies in addressing environmental concerns.

Chapter Three

Methodological Approach

3.1 Introduction

Quantitative research methodology has been employed to conduct this paper, involving the analysis of numerical data as an integral part of the research process. The study is structured into two phases, delineated in ***Figure 1.5.1*** : In the initial phase, a Node-RED application was hosted on both a local server and a Google Cloud server. This facilitates the transmission of data from the local server to the Google Cloud, effectively simulating IoT device functionality. Subsequently, in the second phase, two machine learning models- regression model (Linear) and a neural network model (Long Short-Term Memory) were implemented, each tailored to address specific parameters for analysis.

3.2 Sending Data to Google Cloud from IoT Device

Smaller computing devices, such as single board embedded computers like Arduino, Raspberry Pi or personalised hardware are typical example of real IoT Devices. Since, it is hard to collect huge amount of data in brief period form the real IoT device to complete This research focus on using local PC (Personal Computer) as a real hardware device which can execute software programs (Node-Red) and connect with cloud-based platform (Google Cloud) as shown in ***Figure 3.2.1***. While the dataset under consideration originates from a separate author who conducted data collection utilising real-time IoT sensor to gather air quality metrics from various locations within Chinese city across distinct temporal intervals.

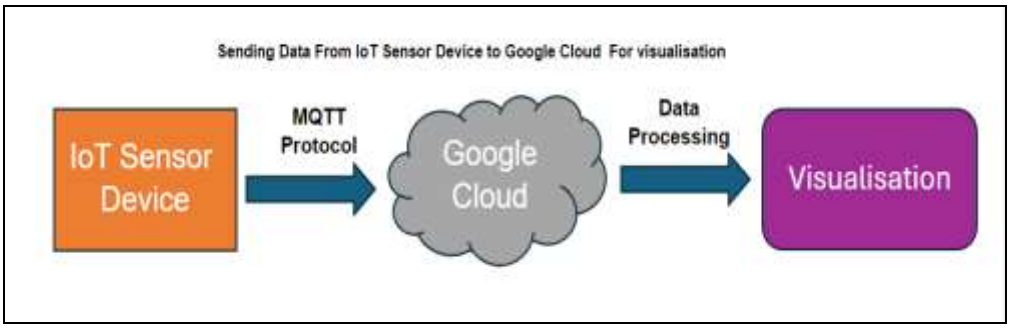


Figure 3.2.1: Sending Data to Google Cloud from IoT Devices for Visualisation

3.2.1 Collecting Data

'Hourly Air Pollution Data in Nine Monitoring stations in Nanjing' dataset which is going to be used on this paper (shao *et al.,* 2022). The dataset holds an hourly air pollution data in nine monitoring stations of China Nanjing city from January 2017 to December 2020. Among these nine stations, ZHM (Zhonghuamen) city data is being used on this project. Datasets contain eight unique features, they are AQI, PM2.5(µg/m3), PM10(µg/m3), CO(µg/m3), NO2(µg/m3), O3(µg/m3), SO2(µg/m3).

Once the data was downloaded, it was slightly changed for the first phase to fit to the methodology process. First, the column name pubtime was changed separated into two columns and named as Data and Time, where data column holds date data and time column hold the time data only instead of common in pubtime. Also, the column name PM2.5 was renamed PM2_5. But it remains the same data from the publisher while implementing machine learning model in phase 2.

3.2.2 Node-Red as IoT Device

An open source, flow-based programming language which make easier for integrating different devices, web services and application programming interface (APIs) is known as Node-Red. It provides a browser-based visual programming interface that allows users to drag and drop nodes into a canvas and link them to form flows. Each node stands for distinct service and function, such as Message Queuing Telemetry Transport (MQTT) messaging, database connections, Hypertext Transfer Protocol (HTTP) endpoints, making dashboard and so on. To further extend Node-Red capabilities, users might install community created custom nodes in addition to the vast library of pre-built nodes that covers a wide variety of operations.

Node-Red is a key player in the IoT space as it provides a platform for easy integration which allows to connect and manage devices, services, and applications. Without having an in-depth programming knowledge, developers can easily create and implement IoT solutions by utilising its user-friendly visual interface. The enormous node library provided by Node-Red makes it easier to communicate with a wide range of IoT devices and protocols, including MQTT, and HTTP. Due to its superior compatibility and adaptability, it is a preferred option for IoT projects. In addition, its event-driven design makes it possible to interpret data streams produced by IoT sensors and actuators in real-time, which makes it easier to create scalable and

responsive IoT applications. Implemented in the cloud, edge devices, or inside IoT routes, Node-Red enables developers to quickly prototype, implement, and refine IoT solutions, speeding up innovation and development in the IoT industry.

Node-Red application will be hosted on the local server (Personal Commuter) and with the help of different node it will read a data from the comma-separated values(csv) file and process it to send the data to external server which will be the prototype of IoT device. MQTT protocol is used to send the data from the local server to external server (Google Cloud) with the help of HiveMQ broker. The main idea behind the messaging in MQTT is publisher and subscriber model where there should be a central broker to process the message and these publisher and subscriber are unknown with each other. First, the publisher publishes the message to a broker, and the broker sends it to the client who has subscribed to receive the data. MQTT never uses addresses like email addresses or phone numbers, instead they use Topic to pass the message which was created by publisher. So, in order to send the data from local server to cloud server this project uses the topic to publish and subscribe to the message with the help of HiveMQ broker.

3.2.3 Google Cloud

Google Cloud is a full range of cloud computing services, which main purpose of which is to give full access to all the tools that are needed to create, implement, and manage cloud-based applications for individual developers to business firms. Since it launched in 2008, Google Cloud became one of the leading global brands in the field of cloud computing while competing with industry giants like Microsoft Azure and Amazon Web Services. Few most of the popular range of service that Google Cloud provides yet to date are big data, machine learning, storage, networking, analytics, database and so on. These services are provided via a worldwide network of data centres which offers scalable, high-performance, and reliable infrastructure to handle a wide range of workload and use cases. Recently, Google Cloud discontinued its most powerful feature Cloud IoT Core services. Though its other features and tie-up with third party application which provides the same features as IoT core, it is still the one of the most popular powerhouses for IoT services.

3.3 Machine Learning Approach for Predicting Air Quality Index

Although there are different machine learning approaches to predict the air quality index, this project focusses on two approaches, Linear Regression (which is used for predicting numerical values based on input features), and Long Short-term Memory (LSTM) (which is used to predicting a sequential time series data) as shown in ***Figure 3.3.1***. All the coding part will be executed on one of the most popular features provided by Google, Google Colab. Google Colab is collaborative Jupyter notebook environment which is integrated with python programming language, provides support to write a code and text cells which allows user to explain, visualise, and document their work. Also, its wide range of libraries support such as TensorFlow and PyTorch, and access to Google Drive directly from Colab notebook, it is most popular among the data scientist and analytics. All the code that is going to run in this project will be run on **Python 3.10.12** version on Google Colab.

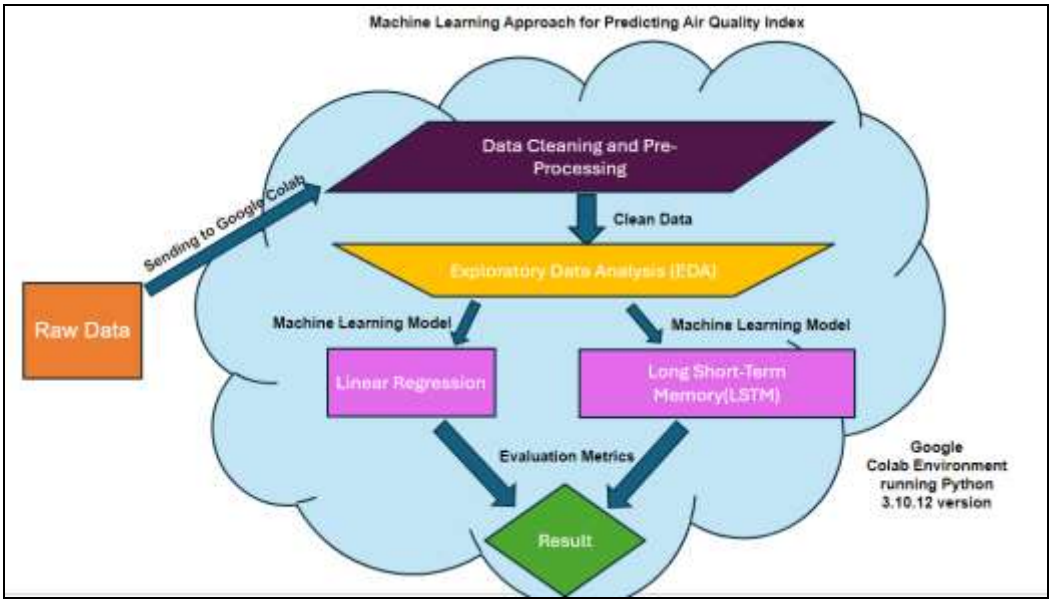


Figure 3.3.1: Machine Leaning Approach for Predicting Air Quality Index

3.3.1 Data Cleaning, Pre-Processing, and Exploratory Data Analysis (EDA)

Quality of input data significantly influences the results, regardless of how complex the algorithm or how strong the processing resources are. This fact emphasises how crucial data cleaning and pre-processing step, commonly the hidden heroes of any analytical project. Data Cleaning and Pre-Processing describe the methodical steps and approaches used to clean raw data, prepare it to structured format for analysis. Hardly, raw data comes in perfect shape, it may include noise, missing values, and

anomalies. This section delivers the approaches, resources, and best practices used to transform raw into clean data, going through the basic principle of data cleaning and pre-processing. Every stage is crucial for maintaining data integrity and consistency, from recognizing and managing missing values to sorting and addressing duplicates.

Exploratory data analysis is a foundational stage in the data analysis process that opens the door to understanding a dataset's complex terrain. EDA helps in the identification of trends, patterns, and anomalies within data. Before taking further action with the data, EDA helps understand data. It is like turning on a torch before searching for anything in a dark room. EDA helps to facilities vision of destination and potential extra gateways during data analysis and machine learning. During the EDA process, most fundamental data points such as range and averages collected at first and with helps of these points' different visualization (graph, charts) will made. These visualisations indicate if the data has any strange trends or clusters. With these pictures, anyone could assume what is going on datasets.

3.3.2 Linear Regression

The linear Regression model is a type of supervised machine learning algorithm that fits a linear equation to observed data to determine the linear relationship among the dependent and independent variables. A simple example of linear regression model is when the researcher is interested in examining the relationship between students' exam grade and the amount of time they spend studying. The linear regression model is called as; Simple Linear Regression when there is only one independent feature and Multiple Linear Regression when there is more than one feature. Similarly, it is called Univariate Linear Regression when there is only one dependent variable whereas, Multivariate Regression Model when there are more than one dependent variables (Weisberg, 2005).

A modeller should first ascertain if there is a link between the variable of interest before trying to fit a linear model to observed data. This implies a considerable correlation between the two variables, but not necessarily that one causes the other. A scatterplot may be a useful tool for figuring out how strongly two variables are related to each other. Fitting a linear regression model to the data will not yield a meaningful model if there does not seem to be any link between the suggested explanatory and dependent variables that is scatterplot does not show up with any decreasing and increasing

pattern. A correlation coefficient of a value ranging from -1 to 1 is a numerical measure of strength of the relationship between two variables based on observable value.

One Significant advantage of linear regression is its interpretability. To help with the deeper comprehension of the underlying dynamics, the model's equations present distinct coefficients that clearly illustrate the effects of each independent variable on the dependent variable. Its simplicity is an asset since linear regression is straightforward, simple to use, and provides the building blocks for more intricate algorithms.

With just one independent variable and one dependent variable, the simplest form of linear regression is known as simple linear regression and equation is:

$$y = \beta_0 + \beta_1 X + \varepsilon \quad (1)$$

In equation (1), $\boldsymbol{y}$ is the dependent variable, $\boldsymbol{X}$ is the independent variable, $\boldsymbol{\beta_0}$ is the intercept, $\boldsymbol{\beta_1}$ is the slope of line, and $\boldsymbol{\varepsilon}$ is the error term representing the difference between observed and predicted values of $\boldsymbol{y}$.

When there is one dependent variable and more than one independent variable then such type of regression is multiple linear regression and the equation is:

$$y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_p X_p + \varepsilon \quad (2)$$

In equation (2) $\boldsymbol{y}$ is the dependent variable, $\boldsymbol{X_1, X_2, \ldots, X_p,}$ are the independent variables, $\boldsymbol{\beta_1}$ is the intercept and $\boldsymbol{\beta_1, \beta_2, \ldots, \beta_p}$ are the slopes of line, and $\boldsymbol{\varepsilon}$ is the error term representing the difference between observed and predicted values of $\boldsymbol{y}$.

3.3.3 Evaluation Metrics for Linear Regression Models

Any machine learning model's strength may be ascertained using a range of evaluation metrics. These evaluation metrics often show how effectively the linear regression model generates observed results. Some of the most common measurements are shown in ***Figure 3.3.2***:

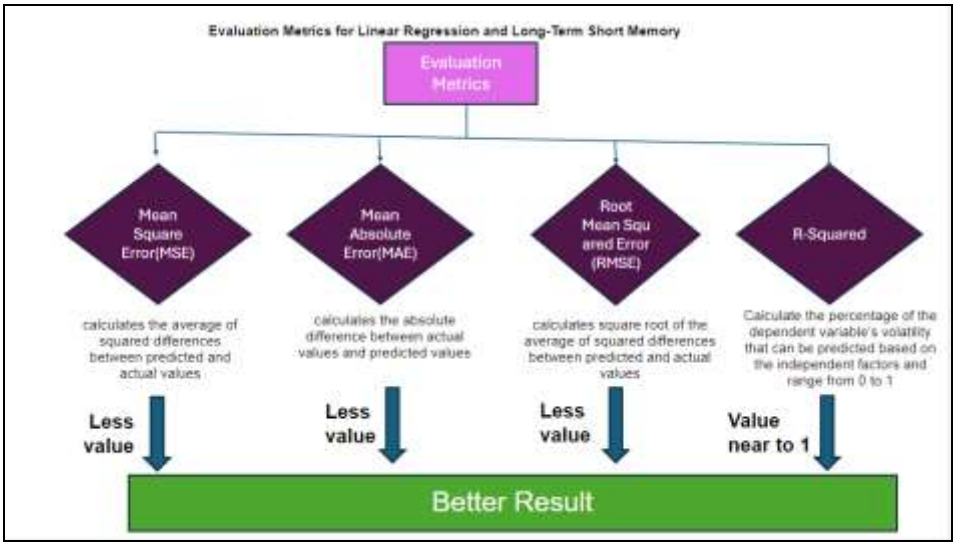


Figure 3.3.2: Evaluation Metrics for Machine Learning

Mean Square Error (MSE)

MSE is an evaluation metric that determines the average of squared deviations between the observed and anticipated values for each data point. The difference is squared to prevent positive and negative errors from voiding each other out. MSE is vulnerable to outliers as substantial errors contribute significant effect on the final score. The equation for calculating MSE is:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \quad (3)$$

In equation (3) , n is the number of observations, y_i is the observed value for ith position data value, $\hat{y}_i$ is the predicted values for ith observation, and $(y_i - \hat{y}_i)^2$ represent the squared difference between observed and predicted values.

Mean Absolute Error (MAE)

MAE metric is used for evaluating the accuracy of regression model. It measures the absolute difference between actual values and predicted values. Lower MAE values represent the improved model performance. As we take absolute differences into consideration, it is not sensitive to the outliers. Equation for calculating MAE is:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \quad (4)$$

In Equation (4), n is the number of observations, y_i is the observed value for i[th] position data value, $\hat{y}_i$ is the predicted values for i[th] observation, and $|y_i - \hat{y}_i|$ represent the absolute difference between actual values and predicted values.

Root Mean Squared Error (RMSE)

RMSE is used to measure the average difference between the anticipated and actual values of the model. Mathematically, it is the standard deviation of residual, where residual means the depth between the data point and regression line. RMSE measures the degree to which these residuals are scattered, providing insight into how well the observed data corresponds to the expected values. The prediction of model is considered better when the value of RMSE is lower. A higher RMSE shows a larger deviation from the predicted values to actual values. Equation for calculating RMSE is:

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \quad (5)$$

In equation (5), n is the number of observations, y_i is the observed value for i[th] position data value, $\hat{y}_i$ is the predicted values for i[th] observation, and $(y_i - \hat{y}_i)^2$ represent the squared difference between observed and predicted values.

R-squared

R-squared is used to evaluating how well regression model fits individual data points. It measures the percentage of the dependent variable's volatility that can be predicted based on the independent factors. R-squared values vary from 0 to 1, the model is supposed to fitted better when the result is near to 1. Equation for calculating R-squared is:

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} \quad (6)$$

In equation (6) numerator is known as residual sum of square which calculate by squared difference of y_i (observed value for ith position data value) and $\hat{y}_i$ (predicted values for ith observation). Denominators represent the total sum of square, which is calculated by squared difference of y_i (observed value for ith position data value) and $\bar{y}$ (mean of the observed values).

3.3.4 Long Short-Term Memory (LSTM)

Hochreiter and Schmidhuber, (1997) designed an improved version of Recurrent Neural Network (RNN) that can detain long term dependencies in sequential data known as Long Short-Term Memory. The main function of LSTM is to process and analyse sequential data such as time series, speech, and text. The power of LSTM is found in its capacity to understand order dependency, which is essential for resolving complex issues like speech recognition and machine translations. Unlike typical RNN, which suffer from the vanishing gradient problem, LSTM utilise a memory cell and gates to regulate the flow of input, enabling them to solely keep or reject information as needed. ***Figure 3.3.3*** Show the architecture of LSTM where three gates-the input, forget and output gate control the memory cell. What data is added to the memory cell is managed by the input gate, forget gate is responsible for what information is removed from the cell and output gate is responsible for what data is output from the memory cell.

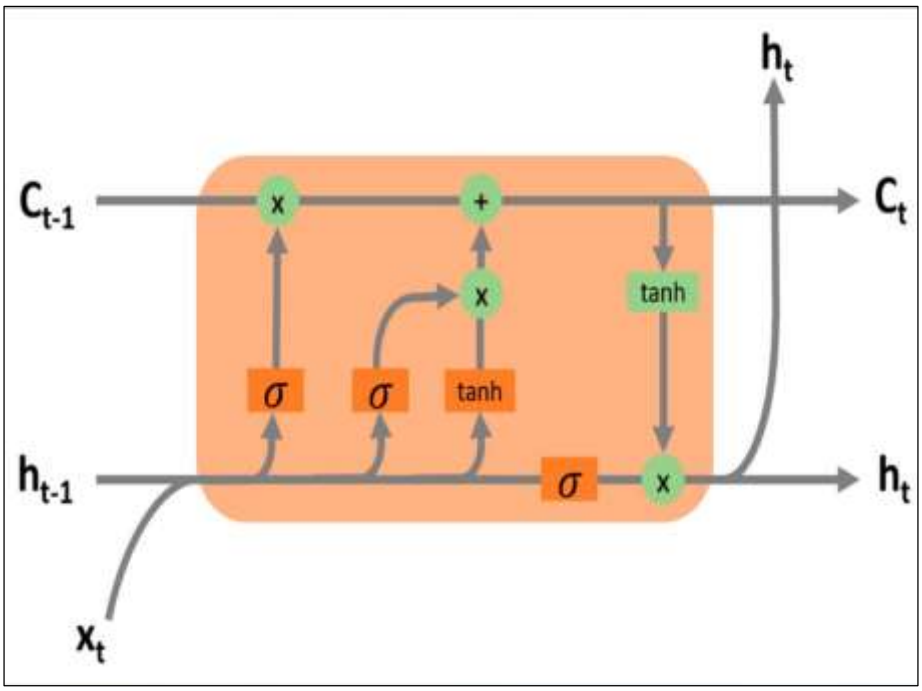


Figure 3.3.3 : Architecture of LSTM (Xayasouk, Lee and Lee, 2020)

LSTM models calculate the hidden state as follows:

$$f_t = \sigma(W_f[h_{t-1}, x_t] + b_f \quad (7)$$

$$i_t = \sigma(W_f[h_{t-1}, x_t] + b_i \quad (8)$$

$$Sigmoid = \frac{1}{1 + e^{-1}} \quad (9)$$

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o \quad (10)$$

$$\widetilde{c_t} = \tan h\,(W_c[h_{t-1}, x_t] + b_c \quad (11)$$

$$c_t = f_t * c_{t-1} + i_t * \widetilde{c_t} \quad (12)$$

$$h_t = o_t * \tan h\,(c_t) \quad (13)$$

In equation (7,8,9,10,11,12,13) $\boldsymbol{\sigma}$ is the logistic sigmoid function. $\boldsymbol{f}$, $\boldsymbol{i}$, and $\boldsymbol{o}$ are the forget, input and output gate, respectively. $\boldsymbol{W}$ is the weight matrix used to convert data from cell vector to gate vector and $\boldsymbol{h}$ is a hidden vector with the same size in every layer. $\boldsymbol{\widetilde{c_t}}$ represent the current input layer, $\boldsymbol{c_t}$ is the calculated internal memory and $\boldsymbol{h_t}$ is the result of a hidden state, obtained by memory multiplication in ***Figure 3.3.3***.

Forget gate is in the position of eliminating information from the cell state. It is fed up with two inputs; the input data for the current time step(x_t), and the state that is hidden output from the previous one(h_{t-1}). A bias is applied after these inputs are multiplied by weight matrices. To determine which values to retain and which to reject, an output vector having values that vary from 0 to 1 is then obtained by applying a sigmoid function. Then input gate forward information to the cell state. A sigmoid function uses a filter for (h_{t-1}) and (x_t), much like the input gate, to provide a vector of appropriate values for the cell state spanning from -1 to 1. Then, values from this vector can be

added to the cell state. The output gate decides which data is output from the memory cell. Mainly in three stage output gates deploy its function; the hyperbolic tangent function $\tan h$ is first applied to the vector in order to scale the values from -1 to 1. Secondly, a filter for values of (h_{t-1}) and (x_t) is created by applying the sigmoid function to the preceding concealed state. LSTM output information is then obtained by multiplying the filtered values by the vector that was formed in the above step.

3.3.5 Evaluation Metrics for Long Short-Term Memory

Evaluation metrics for LSTM models will remain consistent with those outlined in section 3.3.3.

Chapter Four

Implementation of Discussion and Findings

4.1 Introduction

In order to achieve **Aim and Objectives** of this paper, whole implementation process is divided into two phase, first phase show how node-red simulation act as a IoT devices to send data from ground root or local PC(Personal Computer) to Cloud Based Server(Google Cloud) which is followed by setting up Google Cloud parameter to support node-red application, to secure it and visualise different data that is received from the local server. The second phase shows how different machine learning (Linear Regression and Long Short-Term Memory) models can be used to predict accurate air quality index by utilising different parameters of environmental air pollution.

4.2 First Phase: IoT Device to Google Cloud

To achieve the **Aim and Objectives number a and b** of this paper in first phase data was sent to Google Cloud Node-Red from Local PC Node-Red by using a MQTT protocol. Also, different live visualisation was done in Cloud based Node-Red. ***Figure 4.2.1*** show the flow diagram of first phase.

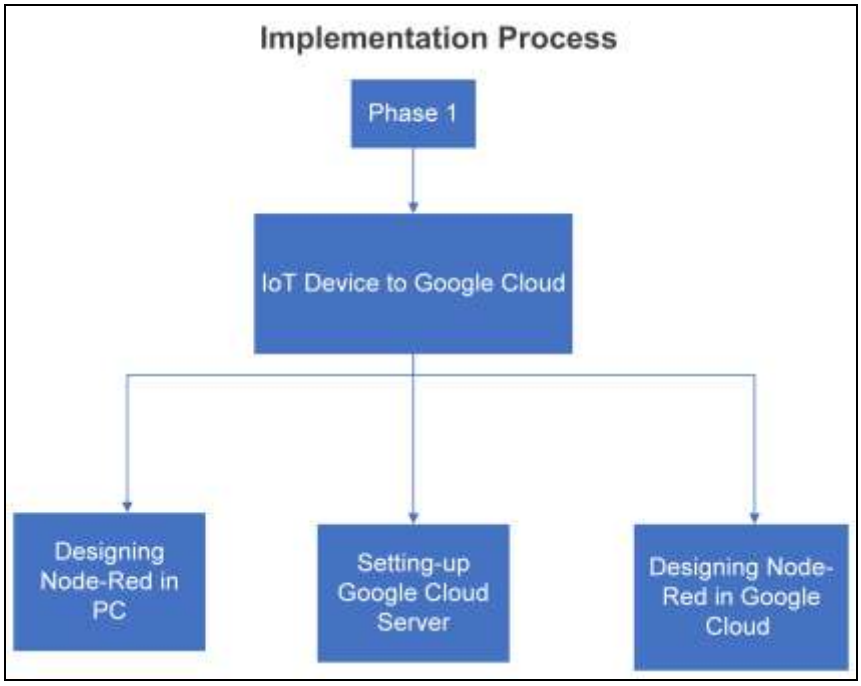


Figure 4.2.1: Flow Diagram of First Phase of Implementation

4.2.1 Designing Node-Red in Local Server

First Node-red application was installed into local PC, and with the help of different node a complete application was developed that send the sensor data to Google Cloud with the help of MQTT out node in order to fulfil the **Aim and Objectives number a: Developing a IoT Simulator in node-red to send the data to Google Cloud Platform**. But before that node.js was installed on the PC.

Once the node-red installation was finished, a command was executed as shown in ***Figure 4.2.2*** to run the node-red in local machine. It will show complete information regarding the node-red version and server for running the node-red. After requesting a URL http://127.0.0.1:1880 via preferred web browser it will show the interface of node-red which will act like as an IoT device that will send the data to Google Cloud with the help of MQTT broker.

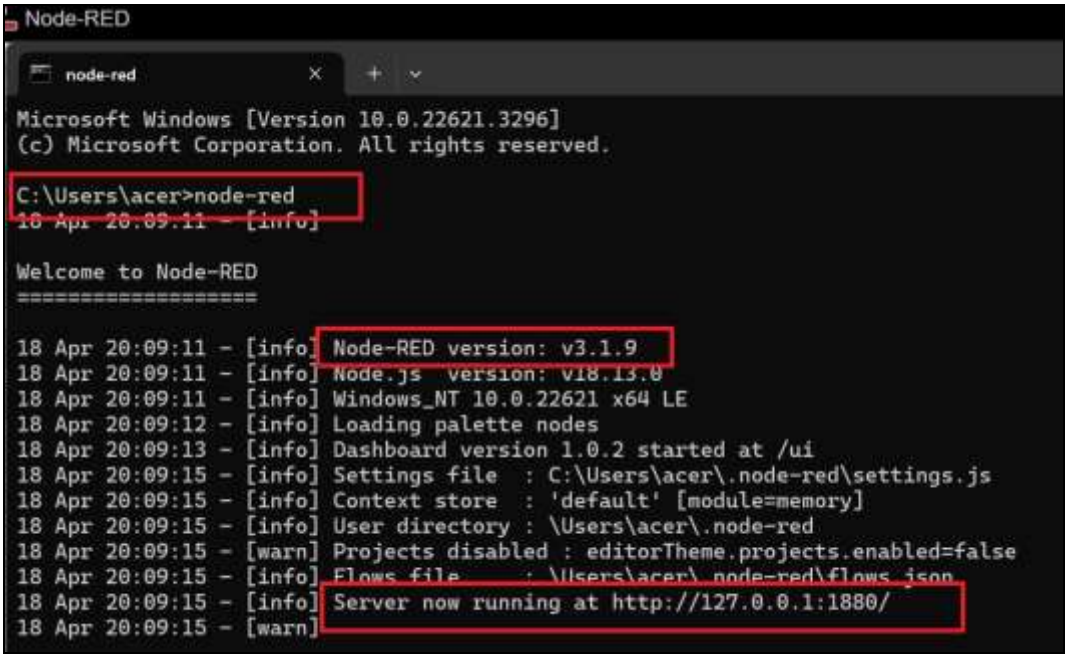


Figure 4.2.2: Running a Node-red Server in Local PC

Figure 4.2.3 show the details interface of node-red where on left side there is different node such as **inject** (which inject the node flow), **debug** (which display the result from the flow), **function** (which help to write a java script code to the message that is passed through this node), and so on. On the middle there is complete space, which is also known as flow space, here different nodes are interconnected with each other through wire to read, write, debug, and communicate with external system (Google Cloud). On the right-hand side, the is a different panel which will show the debug result,

setting of flow, help, and info. On the top right side there is a red deploy button which helps to deploy the flow where there is certain change in flow node, if the system is not deployed after the single minor change in flow, it will not execute the recent flow.

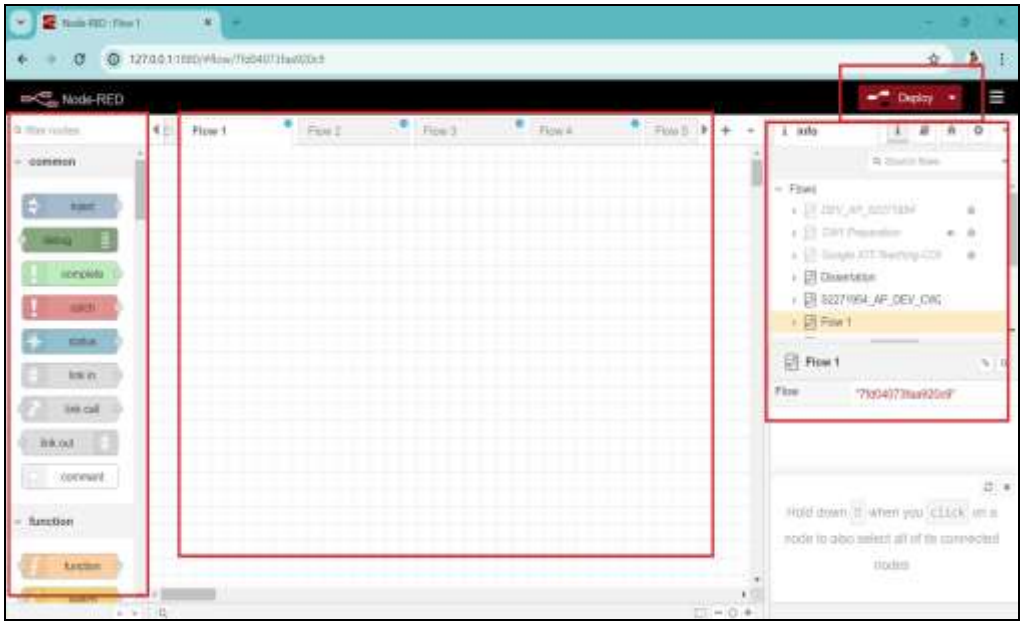


Figure 4.2.3: Node-Red Architecture on Local PC

Node-Red Flow for Publishing Message from Local PC

The Node-Red flow that was used to publish a message from local server (PC) to Google Cloud server is shown in ***Figure 4.2.4***. Different nodes are used to read, split, change, and publish messages to Cloud based Node-Red with the help of MQTT protocol with hivemq broker. First Read node (inject node) was injected to inject an empty message into the flow to trigger the flow. Read a Spreadsheet file (read file) node was connected to inject node which read a csv file from the specified file path that was stored in local pc storage. Then Handle Spreadsheet Files as Book (book) node was connected to the output of read node which converts the contents of spreadsheet file to a workbook object.

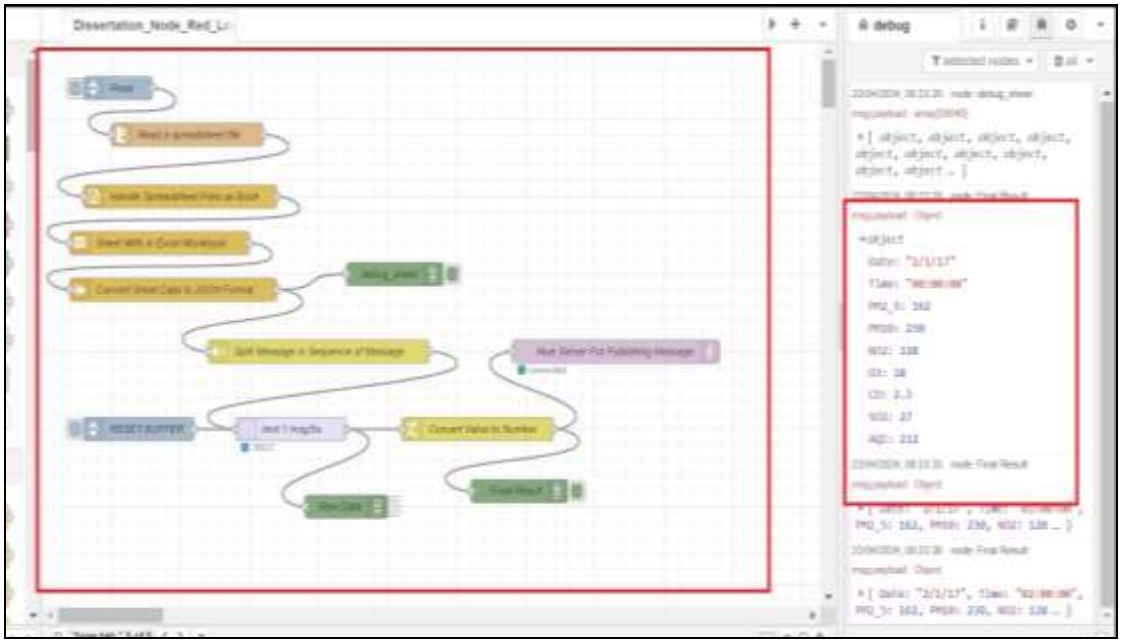


Figure 4.2.4: Node-Red Flow in Local PC

Output of book node was connected to Sheet within Excel Workbook (Sheet) node which picks a sheet object from a workbook object. Convert Sheet Data to JSON Formant (sheet-to-json) node was connect to the output of sheet node which main function is to convert a sheet object to an array of JSON object and which output provide payload message of array with cell values from the csv file. Here the range was set up as A1:I35041, the range of spreadsheet files read the read node. Split Message in Sequence of Message (split) node was connected to the out of sheet-to-json node which main function is to split the whole message into sequence of single message. It split the message by \n character when there was a buffer. Reset Buffer (inject) node which main function is to reset the flow to the subsequent delay node.

The input of delay (limit 1 msg/5s) node is split and reset node; it passed the message on every five second that was received from the split node but when the reset buffer node was executed it will terminate its flow. The output of the delay node was connected towards the Convert Value to Number (change) node, responsible for changing value into number. ***Figure 4.2.5*** show that there are seven rules that are setup for change node where each value of PM2_5, PM10, NO2, O3, CO, SO2, and AQI are set to number format by using a JSON expression. The output of the change node is connected to the Hive Server for Publishing Message (MQTT out) node.

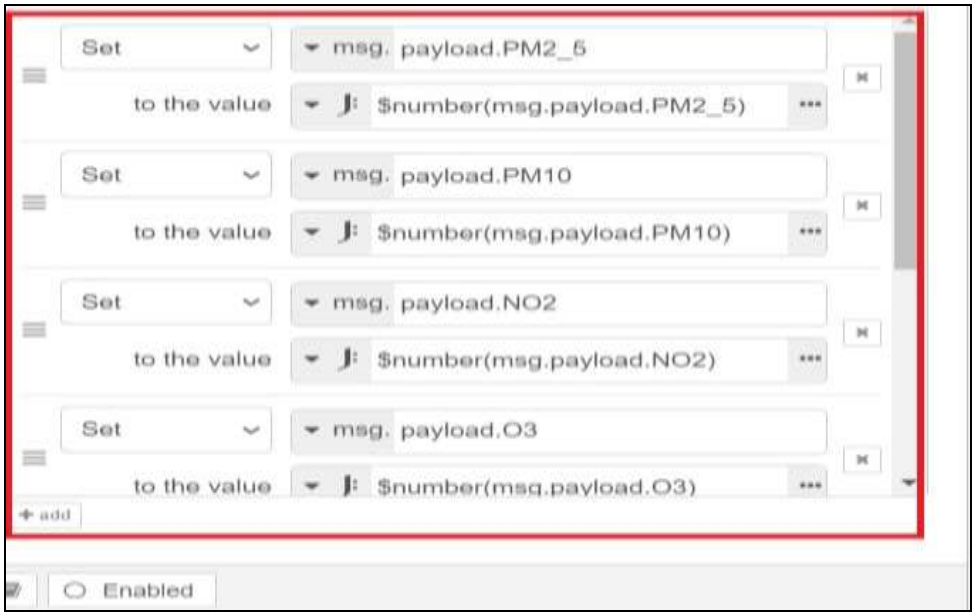


Figure 4.2.5: Changing Node in Node-Red Local PC

The last node of this flow is Hive Server for Publishing Message (MQTT out) node which main function of which is to build a connection to an external device with the help of topic, server, and port. ***Figure 4.2.6*** show the complete configuration of this node where the message was sent using a topic as 'pub99932/dev-node3213/telemetry' with Quality of Service (QoS) as level 2 and through the hive server. Hive server uses the port 1883 with the 'broker.hivemq.com' as address with a protocol of MQTT V3.1.1 which will connect automatically.

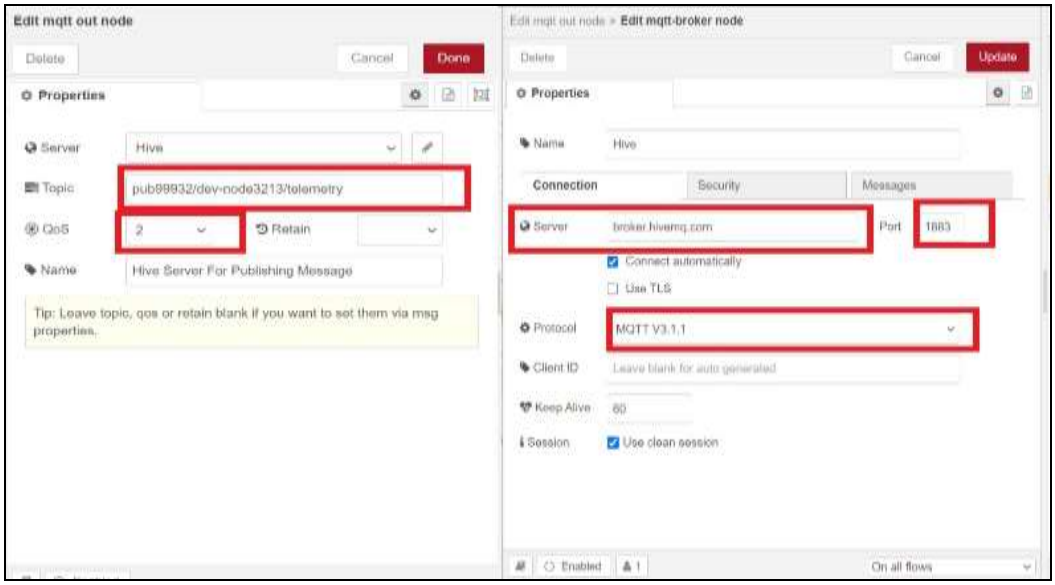


Figure 4.2.6: Publishing Message from Local PC Node-Red

4.2.2 Setting-up Google Cloud Server for Node-Red

This section shows all the requirements that are needed to host Node-Red on Google Cloud. First A Project was created on Google Cloud Console that host all the necessary items that are needed to host the Node-Red. Then a virtual machine was created and setup the setting file that were needed to support Node-Red. After that Docker was installed into VM, and with the help of Docker container a newly fresh Node-Red was hosted on it. Once the Node-Red was installed, it was accessed after setting up firewall rules, and Node-Red is secured with unique password. Then at last a static IP address was set up to access the Node-Red.

Creating a Project

The first project was created on Google Cloud. While creating a project there are few things that need to be consider, project name should always be unique, once the project is created it is impossible to change its name, and also it will be more easy if the project name is created on lowercase format because when project is created project id will automatically create as lowercase so there will not be any miscommunication. “Iot-s2271954” was created as a project name.

Creating a VM (Virtual Machine) Instance for Hosting a Node-Red in Google Cloud

There are several ways of hosting a node-red in Google Cloud. Among them virtual machine is one of them and it was created to host the node-red. ‘**vm-node-red-iot-s2271954**’ was the name of virtual machine which was hosted on a region of ‘europe-west2(London)’, with a machine type E2 (having 2 vCPU, 1 core, 4 GB Memory with 10 GB balanced persistent disk).

Setting Up a Files to Support Node-Red

Once the VM was created, a setting file was uploaded to support the Node-Red in Cloud through SSH (Secure Shell) terminals. First, ‘**node-red**’ folder was created on the home directory of virtual machine that is **‘/home/arjunofficial446’**. Then ‘**settings.js’** file was uploaded through upload files which will store the files in home directory of virtual machine. Then that file was moved to a newly created folder that is ‘**node-red**’ which will provide all the settings to host the node-red in Docker.

Installing Docker on the VM

Next Docker was installed on VM instance which will host the node-red application. ***Code 4.2.1*** shows the seven steps that were taken while installing Docker in VM. After the installation of Docker Node-Red was hosted on the Docker container.

Step 1: Open SSH and go to the home directory.

cd ~

Step 2: Download script that will install Docker.

curl -fsSL https://get.docker.com -o install-docker.sh

Step 3: See the script content to make sure that the right file was downloaded.

cat install-docker.sh

Step 4: Dry run of script (not the exact installation process) which will show the exact number of steps that need to be executed.

sh install-docker.sh --dry-run

Step 5: Finally, this code was run to install the Docker.

sudo sh install-docker.sh

Step 6: Once the Docker is installed current user was added to the Docker group which is used for managing container.

sudo usermod -aG docker $USER

newgrp docker

Step 7: Docker installation was checked by running following code.

sudo docker run hello-world

Code 4.2.1: Step on Installing Docker (Docker , 2024)

Hosting Node-Red in Docker Container

Before hosting node-red in Docker container the ownership of .**node-red** directory was set. So, for that **sudo chown -R 1000:1000 .node-red/** code was run from the home directory. Once the home directory was set, a new Docker container was created using a Docker Node-Red image. For that ***Code 4.2.2*** was run. After completing this process, a new Docker container was formed with a Docker name as **'mynodered'**.

```
docker run -d --restart=always -it -p 1880:1880 -v /home/arjunofficial446/.node-red:/data --name mynodered nodered/node-red
```

Code 4.2.2: Code for Creating Docker Ned-Red Image

Accessing Google Cloud Node-Red

Next step was taken as to connect across from any browser to the VM that host the node-red, which is delivered on port 1880. So, for this copied the VM external IP address (**34.89.31.99**) and browse in a browser as (**http://34.89.31.99:1880**). First it shows that the site cannot be reached but when the same external IP address was pinging from the personal computer it shows that the IP address is pinging. That means there was some problem which was encountered while running the node-red on browser. So, after some investigation it was found that, Google Cloud VM instances set up default firewall rules to determine what request can be accepted. So, to access this IP address, a new firewall rule was set up.

Setting Up VM New Firewall Rules

To set up the new firewall rules, first a network tag (node-red-1) was added into by editing the VM instance. This network tag helps when accessing the machine through IP address with port number, it will check the VPC network firewall with the same network tags and allow its connection to the VM otherwise not.

Once the network tag was assigned to VM, new firewall rules were set up from VPC (Virtual Private Cloud) Network Firewall. '**node-red-1880**' was named as a new firewall, where target tag was entered as '**node-red-1**' that was created from VM, specified protocol and port with TCP and1880, and last source IPV4 range was changed to '**0.0.0.0/0**'. Here source IPV4 play a significant role to allow access to whom. Though, we can only put our device IP address to access this VM only through these devices, but it has been made just for demo so IPV4 range was put as '**0.0.0.0/0**'. After the firewall was created, any IP address on the internet can connect to this VM through Port 1880.

Once the firewall was updated, the node-red that was hosted on google cloud was accessed through the external IP address of VM with the port number 1880 which is show on ***Figure 4.2.7***.

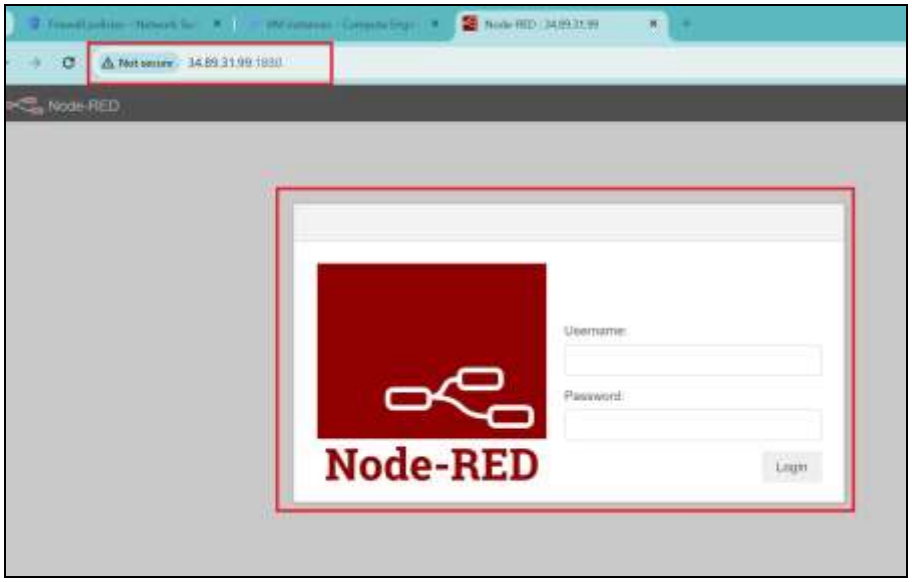


Figure 4.2.7: Node-Red Hosted on Google Cloud Access Through External IP Address with Port 1880

Securing Node-Red

After accessing the node-red through IP address, node-red was made secured with the unique password. First a new hash password was created from the SSH terminal with the help of **Code *4.2.3***.Then, this hash password was overwriting in 'settings.js' files in local device as shown in ***Figure 4.2.8*** and again uploaded to the VM and move it into the node-red directory as did in previous steps.

```
docker exec mynodered node -e
"console.log(require('bcryptjs').hashSync(process.argv[1], 8));" S2271954
```

Code 4.2.3: Creating A Hash function for New Password

Figure 4.2.8: Updating Hash Password in Settings.js File

So, the login credential of the cloud Node-Red was set up. Again, there was another problem encounter, when the virtual machine was stopped and restarted, the external IP address was changed every time, and it was harder to gain the node-red access from browser without opening Google Cloud. To encounter this problem, a static IP address was issued to this VM instance.

Finally, the node-red was successfully hosted on google cloud. ***Figure 4.2.9*** show the node-red that was hosted on Google Cloud with its unique URL which can be accessed across the globe.

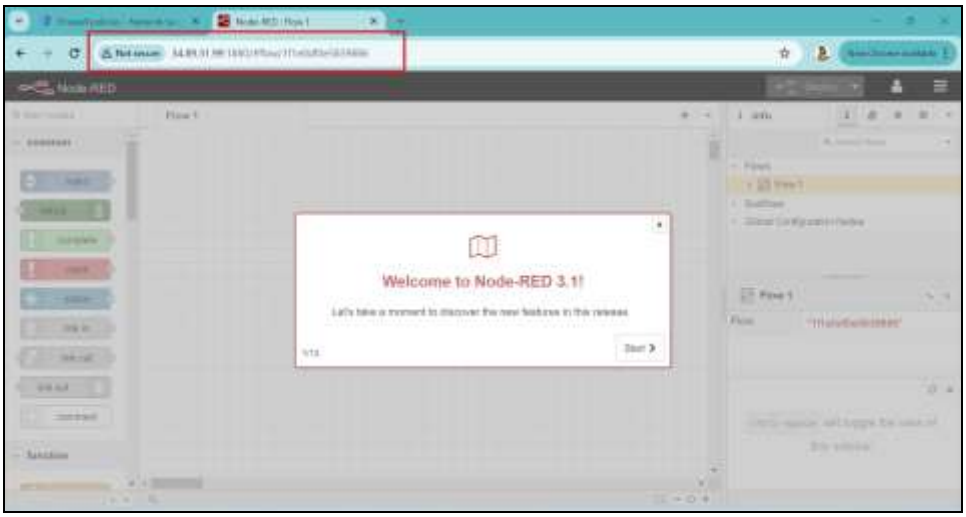


Figure 4.2.9: Newly Installed Node-Red in Google Cloud

4.2.3 Designing Node-Red Flow in Google Server

After the successful installation and figure out all security feature of Node-Red in Google Cloud, a flow was designed as shown in ***Figure 4.2.10*** to fulfil the **Aim and Objectives number b,** displaying different visualisation of data that was received from the IoT device (Node-Red PC) on Google Cloud. First data is received from local Node-Red with the help of MQTT broker and changed that data into favourable format by using different function node and at last, some Node-Red dashboard node were used to visualise different charts.

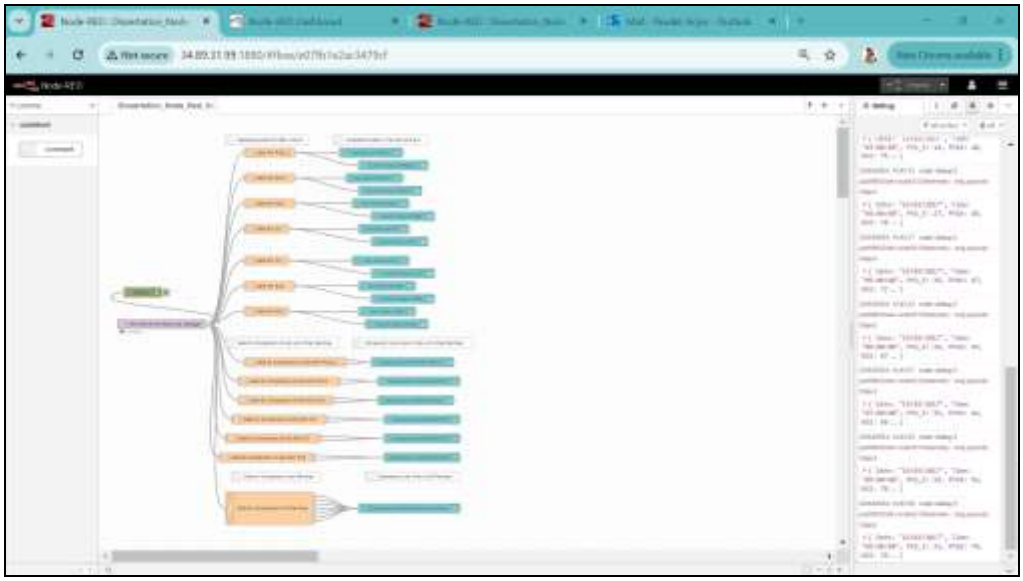


Figure 4.2.10: Node-Red Flow in Google Cloud

The Hive Server for Receiving Message (MQTT) node was used to receive the message from the PC Node-Red with the same setting used to send the data. ***Figure 4.2.11*** show the setting that was used, Topic for subscribing message is 'pub99932/dev-node3213/telemetry' with the QoS level 2 through the MQTT V3.1.1 protocol having a server address as 'broker.hivemq.com' with port number 1883. The data was received in the form of an object which is then passed through different function nodes to extract the desired data.

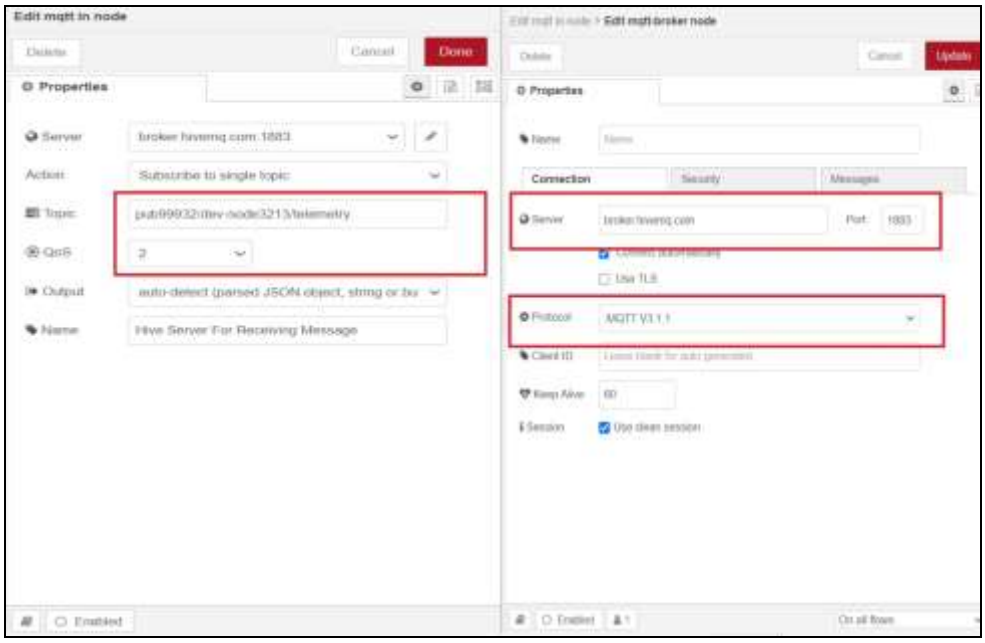


Figure 4.2.11: Subscribing Data from Node-Red PC by Using MQTT Server

Showing Different Visualisation Chart of Current Data

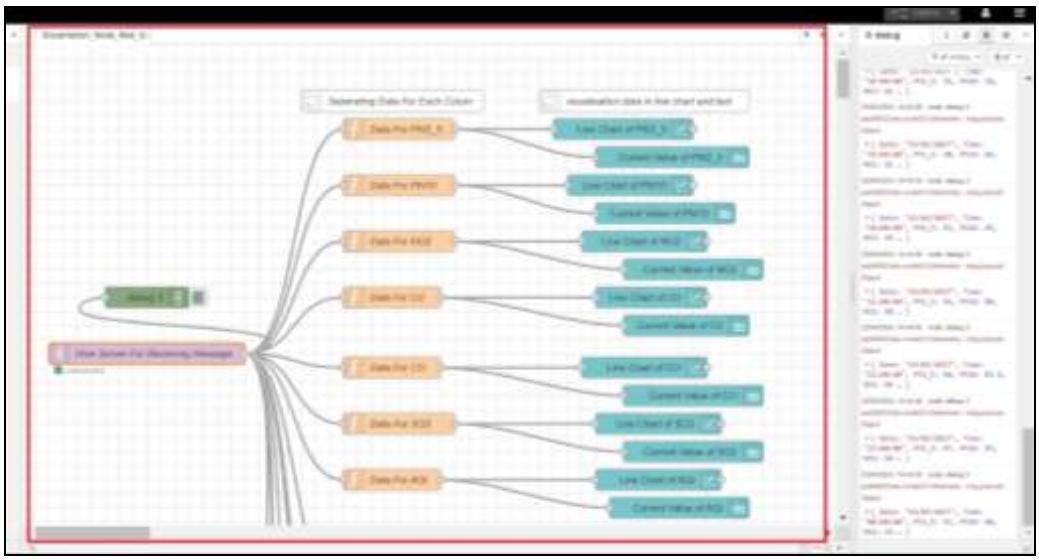


Figure 4.2.12: Showing Flow of Different Visualisation Chart of Current Sensor Data

Figure 4.2.12 show the Node-Red flow for showing different charts and current values of data. Once the data is received by MQTT in a node it is connected to a different function node. These seven different function nodes extract the data value of PM2_5, PM10, NO2, O3, CO, SO2, and AQI. **Code 4.2.4** is an example of PM2_5 function node which helps to extract the data for PM2_5 chart and text. Each Chart node receives the message for each particle and visualizes it in a line chart and shows the data from the last 1 hour. Each text node shows the current value of sensor for each particle, respectively. ***Figure 4.2.13*** show the dashboard of Node-Red cloud which shows all the current data that was received from the local PC Node-red and all these dashboard groups were grouped in same tab under Current Sensor data. Dashboard can be accessed by any person around the globe by using this URL(http://34.89.31.99:1880/ui).

```
PM2_5 = msg.payload.PM2_5;
msg = {topic: 'PM2_5', payload: PM2_5};
return msg
```

Code 4.2.4: Function Node Code for Executing Single Output

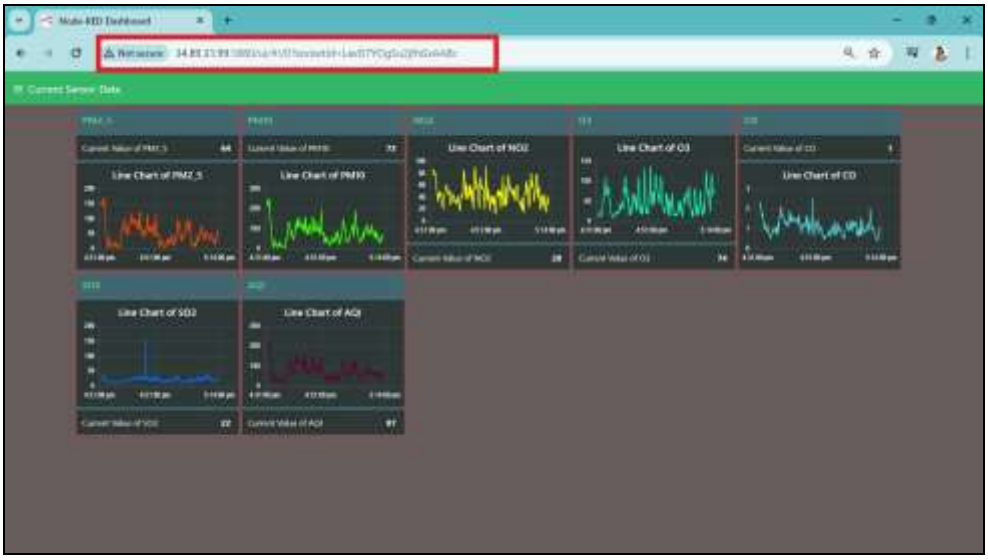


Figure 4.2.13: Current Sensor Data in Dashboard

Comparing AQI Data with Other Particles

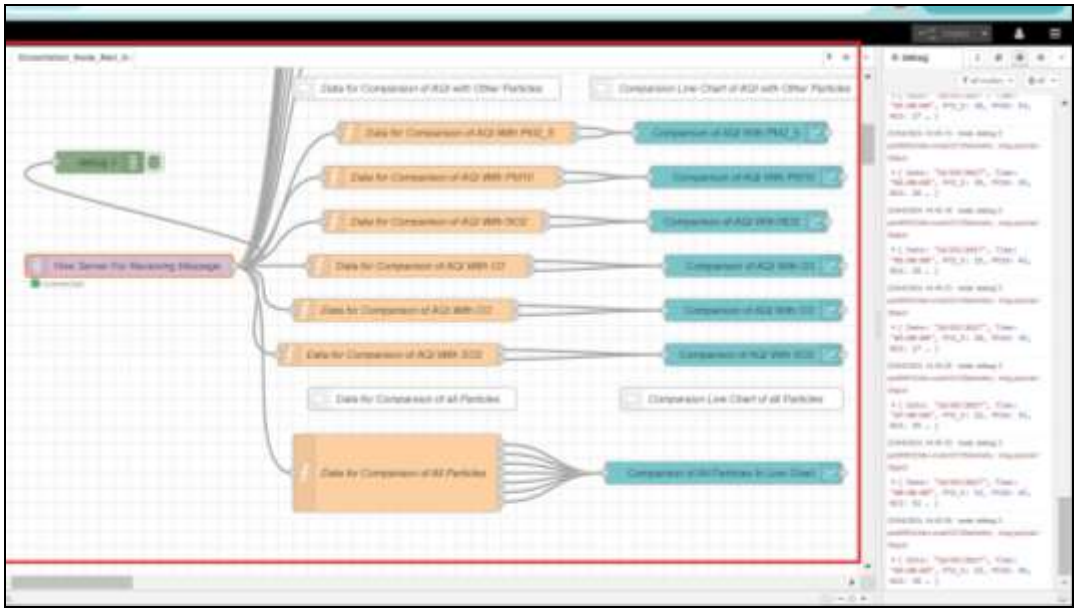


Figure 4.2.14: Node-Red Flow of Comparing AQI Data with Other Particles

Figure 4.2.14 show the Node-Red flow for showing the comparison of AQI with different particles (PM2_5, PM10, NO2, O3, CO, and SO2). Function node helps to extract the data of AQI with related particles each and that data is sent to chart node. ***Code 4.2.5*** shows the function node code that was used for getting the value for AQI with PM2_5. Here, the output of function node was set to two outputs as there were two payload messages passed through it. Once the chart node received the data it visualises up to a time interval of one hours.

```
ValueToLine1 = msg.payload.AQI;
ValueToLine2 = msg.payload.PM2_5;

line1 = {topic: 'AQI', payload: ValueToLine1};
line2 = {topic: 'PM2_5', payload: ValueToLine2};

return [line1, line2]
```

Code 4.2.5: Passing AQI Data with PM2_5 Together

Last two node, Data for Comparison of All Particles(function) and Comparison of All Particles in Line Chart (Chart) node is used to compare and visualise all the seven component that was received from the local Node-Red. **Code 4.2.6** is used to convert the message into the readable format for the chart node. The output of function node is set seven outputs as it provides the seven different messages, and each output port carries one message. Once it was connected to the chart node, it visualise the data as shown in ***Figure 4.2.15 Figure 4.2.14***and the chart show the data for last one hour.

```
ValueToLine1 = msg.payload.AQI;
ValueToLine2 = msg.payload.PM2_5;
ValueToLine3 = msg.payload.PM10;
ValueToLine4 = msg.payload.NO2;
ValueToLine5 = msg.payload.O3;
ValueToLine6 = msg.payload.CO;
ValueToLine7 = msg.payload.SO2;
line1 = {topic: 'AQI', payload: ValueToLine1};
line2 = {topic: 'PM2_5', payload: ValueToLine2};
line3 = {topic: 'PM10', payload: ValueToLine3};
line4 = {topic: 'NO2', payload: ValueToLine4};
line5 = {topic: 'O3', payload: ValueToLine5};
line6 = {topic: 'CO', payload: ValueToLine6};
line7 = {topic: 'SO2', payload: ValueToLine7};
return [line1, line2, line3, line4, line5, line6, line7]
```

Code 4.2.6: Function Node Code for Generating Seven Output Message

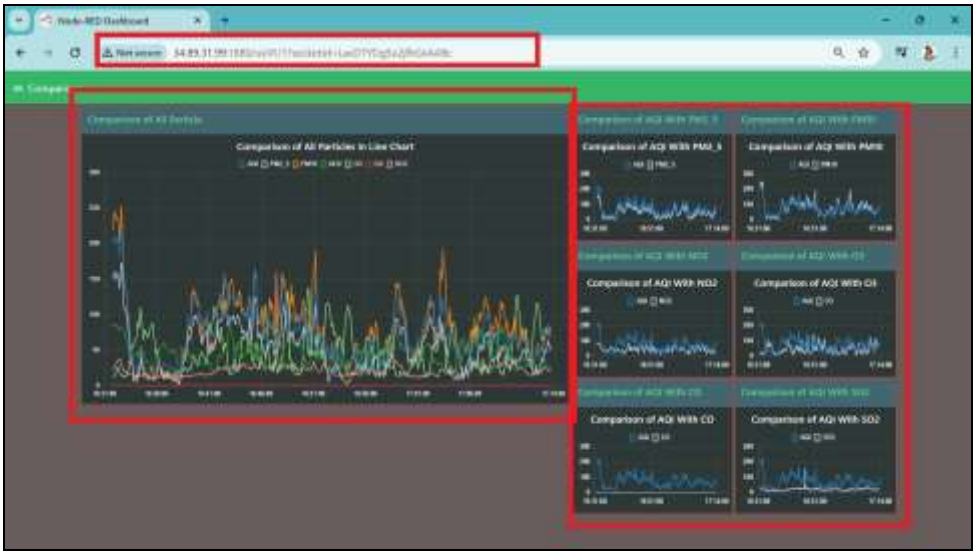


Figure 4.2.15: Comparison of AQI with Other Particles in Chart

4.3 Second Phase: Implementing Machine Learning Model

To achieve the **Aim and Objectives number c and d** of this paper in second phase, exploratory data analysis and different machine learning algorithms were implemented. Before that, data cleaning and pre-processing was done to predict more precise results. Two different machine learning algorithms (Linear Regression, and Long Short-Term Memory) were implemented, with each model further implemented four times by selecting a different feature column that affects the air quality. In Linear regression model; model-1 is used to predict AQI with the help of feature column PM2.5 value only, model-2 is used to predict AQI with the help of feature column PM2.5, and PM10 values, model-3 is used to predict AQI with the help of feature column NO2, O3, CO, and SO2 and in last mode-4 is used to predict AQI with the help of feature column PM2.5, PM10, NO2, O3, CO, and SO2. Whereas, in LSTM model, model-1 is used to predict AQI with the help of univariate LSTM which mean predicting AQI with the help of pre-existing AQI values, in model-2 AQI is predict with the help of Multivariate LSTM (Using PM2.5, and PM10), in model-3 AQI is predict with the help of Multivariate LSTM (Using NO2, O3, CO, SO2), and in model-4 AQI is predicted with the help of Multivariate LSTM (by using all variable: PM2.5, PM10, NO2, O3, CO, SO2). ***Figure 4.3.1*** show the overall flow diagram of phase two of this project.

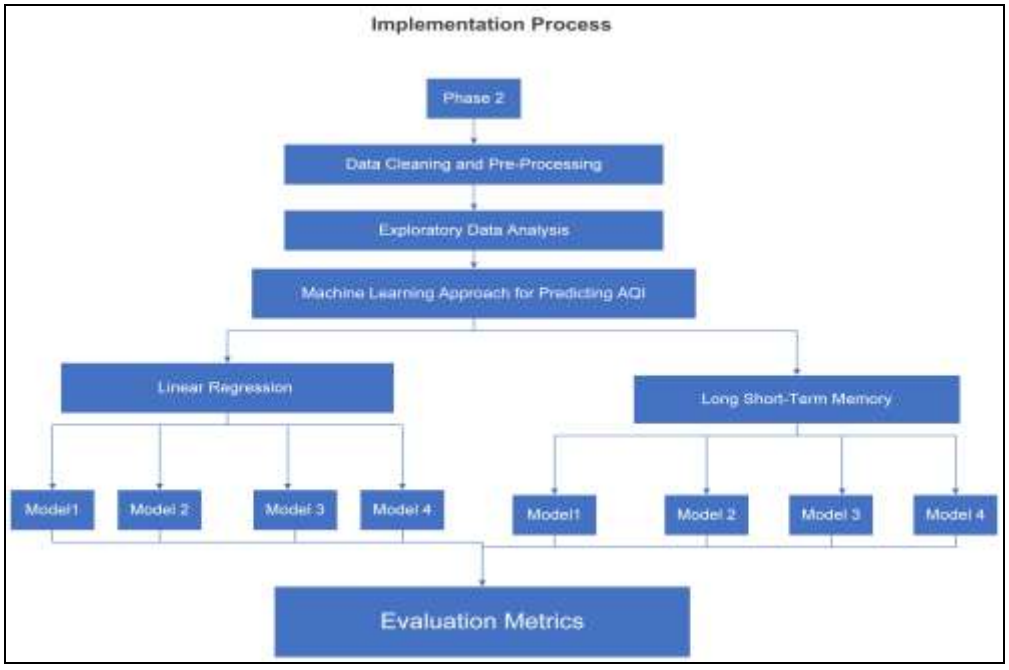


Figure 4.3.1: Flow Diagram of Second Phase of Implementation

4.3.1 Data Cleaning and Pre-Processing

Different data cleaning and pre-processing processes were done to transform raw data into structured and usable format, to perform the exploratory data analysis and machine learning technique. First all the necessary python libraries were loaded into the Google Colab notebook and then the dataset was loaded from Google Drive. Once the data is available in notebook, data cleaning and pre-processing step was carried out and ***Figure 4.3.2*** show the first few rows of the datasets. Where the dataset's shape is (35040, 8), 35040 different observations between January 2017 to December 2020, and 8 is the different variable where data was observed. These variables are 'pubtime', 'PM2.5', 'PM10','NO2','O3','CO','SO2', and 'AQI'. This dataset is a time-series data where every data point was collected in an hour.

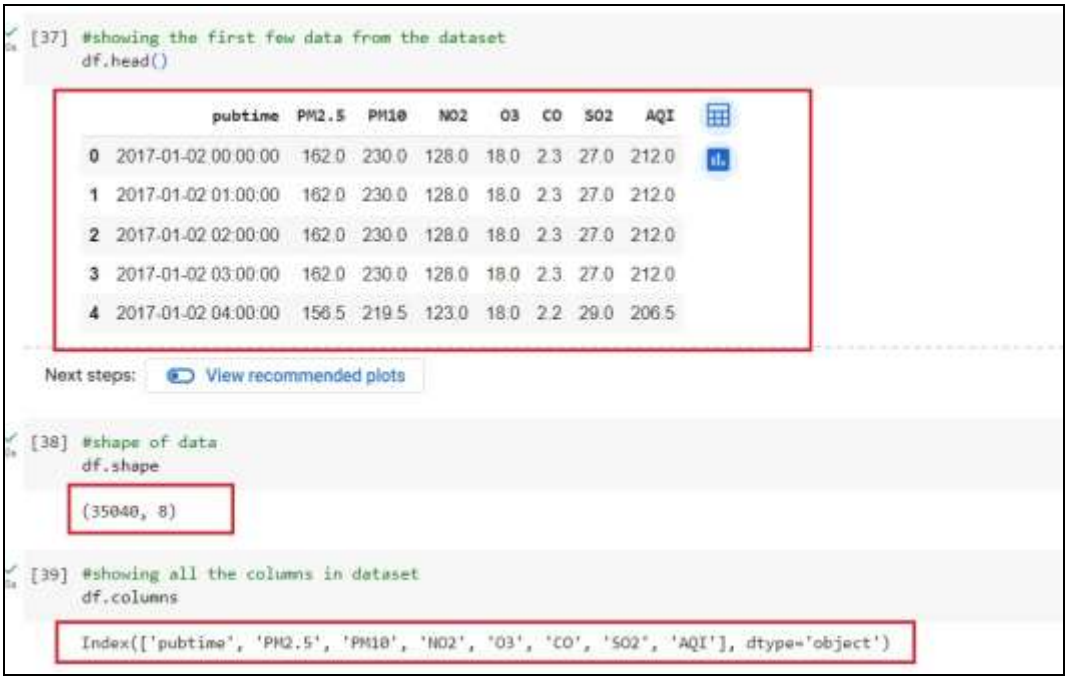


Figure 4.3.2: First Few Data from Dataset with Shape and Variable

From the analysis it shows that the datatype of pubtime is object and rest of the other variables are float. As the dataset has the value from the last four years it might not give the proper output when analysed the whole dataset. So, the dataset was filtered with six months of data that is from thirty of June 2020 to thirty-first of December 2020. ***Figure 4.3.2*** show the dataset that will be used to explore. The total number of observations observed is 4417 with the same 8 variables. Once the data was filtered, data was re-indexed in chronological order.

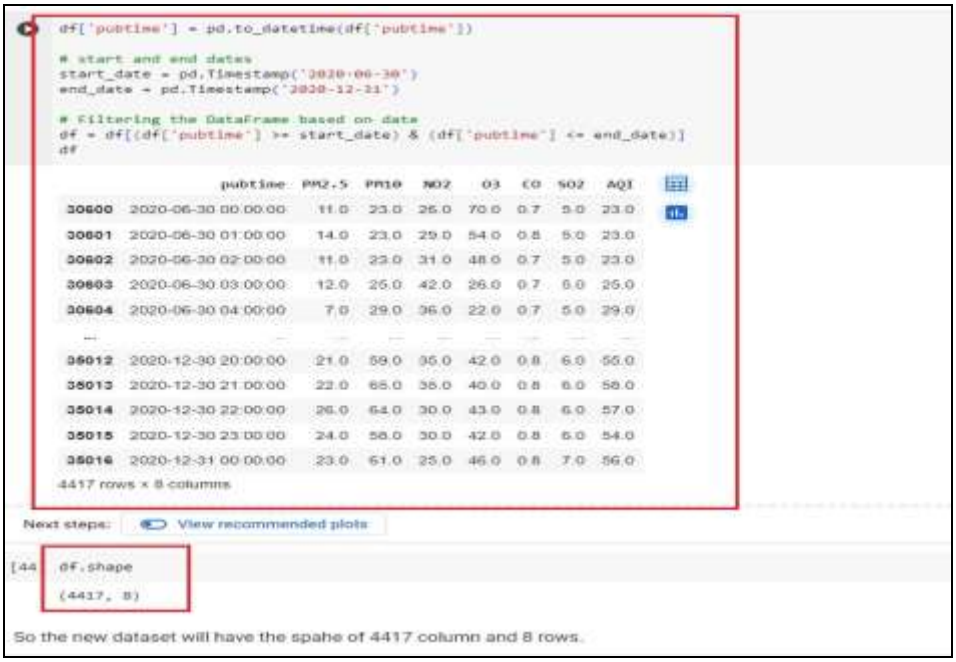


Figure 4.3.3: Filter Data to be used in EDA and Machine Learning

After the data was ordered in chronological order null values and duplicate values were checked. From the analysis it shows that there were no null values and duplicate values which can be stated from the ***Figure 4.3.4***.

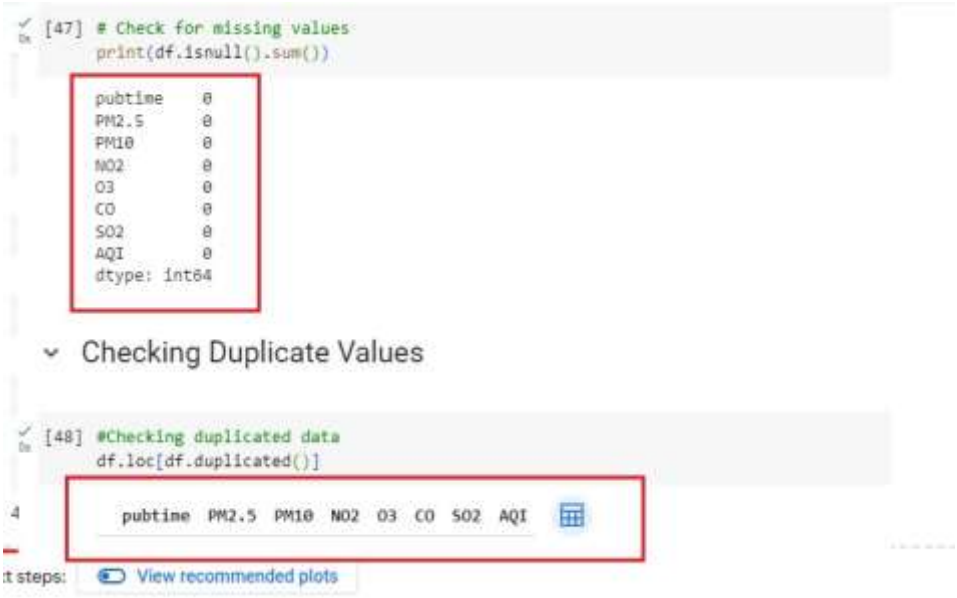


Figure 4.3.4: Checking Missing and duplicate values

4.3.2 Exploratory Data Analysis of Air Quality Data

Once the data was cleaned, different exploratory analysis was done. For that different question was set up to gain more insight from that dataset to fulfil the **Aim and Objectives number c:** Evaluate ETL (Extract, Transform, and Load) transformation to gain the inner insight from the air quality data.

Question 1: How does PM2.5 concentration vary over time?

Figure 4.3.5 show the PM2.5 concentrations over the period of each month. During the mid of December, it shows that the PM2.5 concentrations was recorded highest which is almost 175 and in whole December month it was recorded high as compared to the previous months.

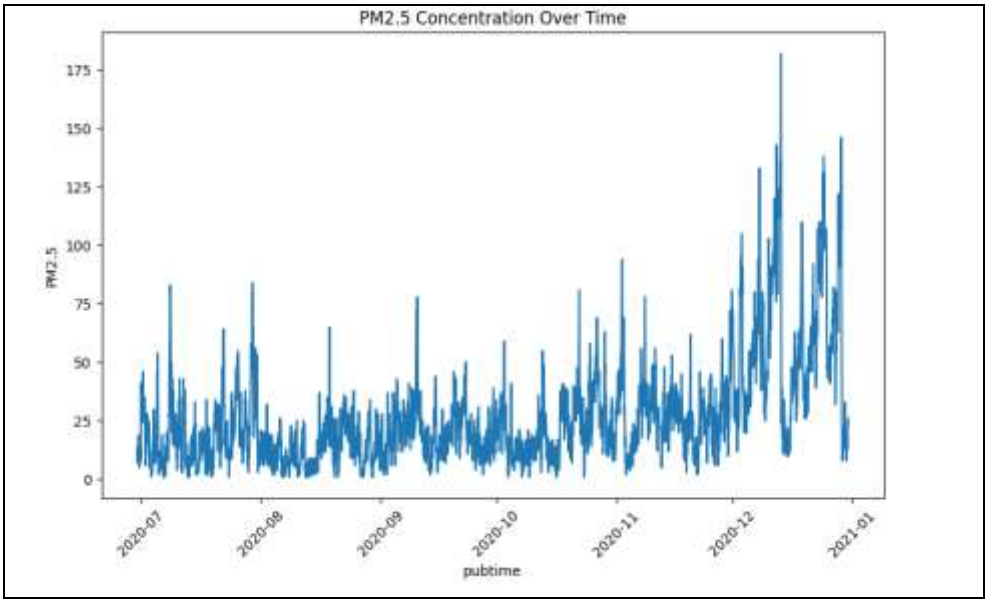


Figure 4.3.5: PM2.5 Concentration Over Time

Question 2: What is the distribution of PM10 Concentrations?

Figure 4.3.6 show the distribution of PM10 concentrations. Histogram plots show the PM10 concentrations are skewed to right, where lower concentrations are occurring more frequently than higher concentrations. This suggests most observations fall in lower concentration range as long tail extends towards higher values.

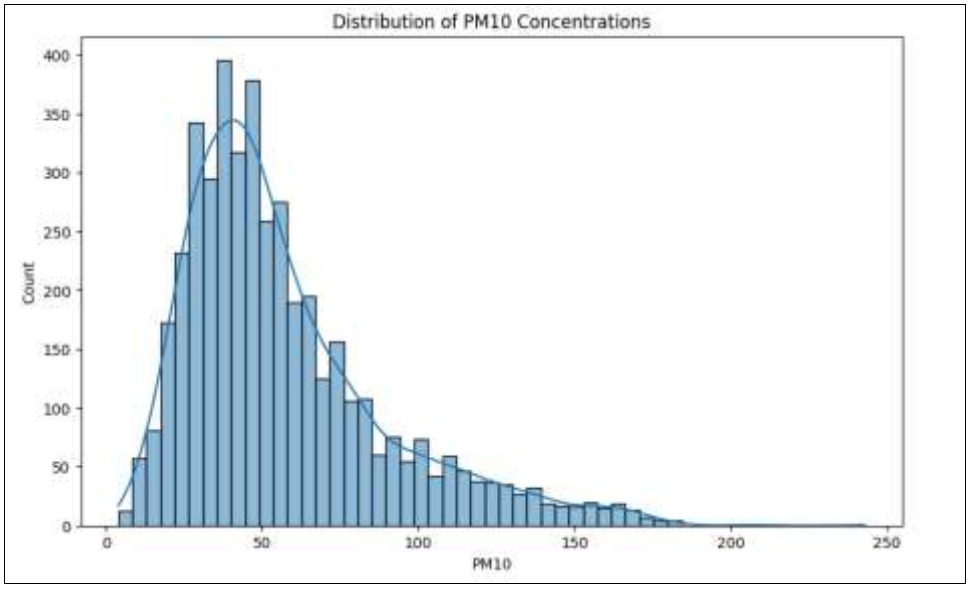


Figure 4.3.6: Distribution of PM10 Concentrations

Question 3: How does the NO2 level change over time?

Figure 4.3.7 show the NO2 level changes over time. The line plot shows that the level of NO2 fluctuated throughout the different months. Sometimes it goes to maximum and sometimes it goes to minimum.

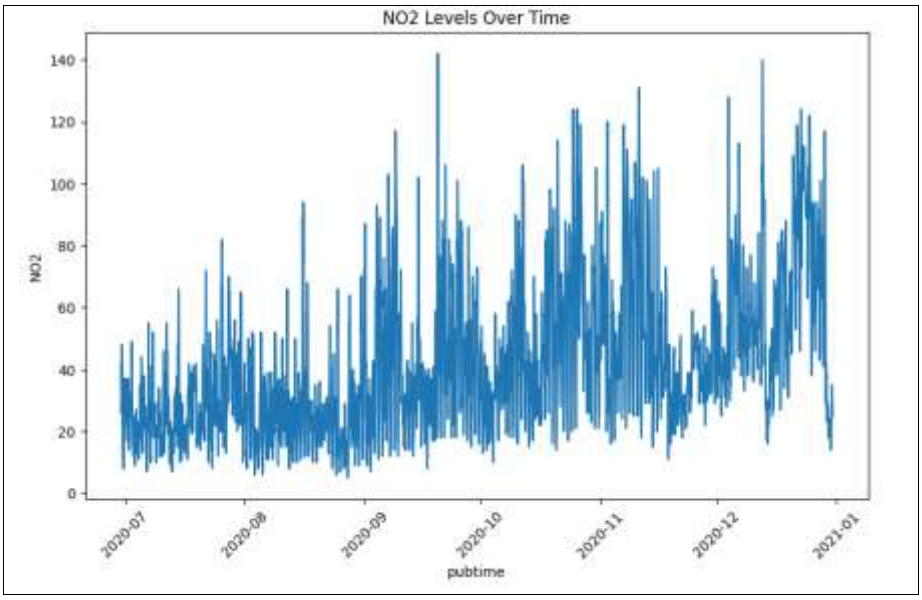


Figure 4.3.7: NO2 Level over Time Period

Question 4: Is there any correlation between O3 and CO levels?

Figure 4.3.8 show correlation between O3 and CO levels through scatter plot where, each point on plot represents the observed data having X-axis as O3 level and Y-axis as CO level. It shows a weak negative correlation and due to the point spread around the trends line, the relationship between CO and O3 is not particularly strong.

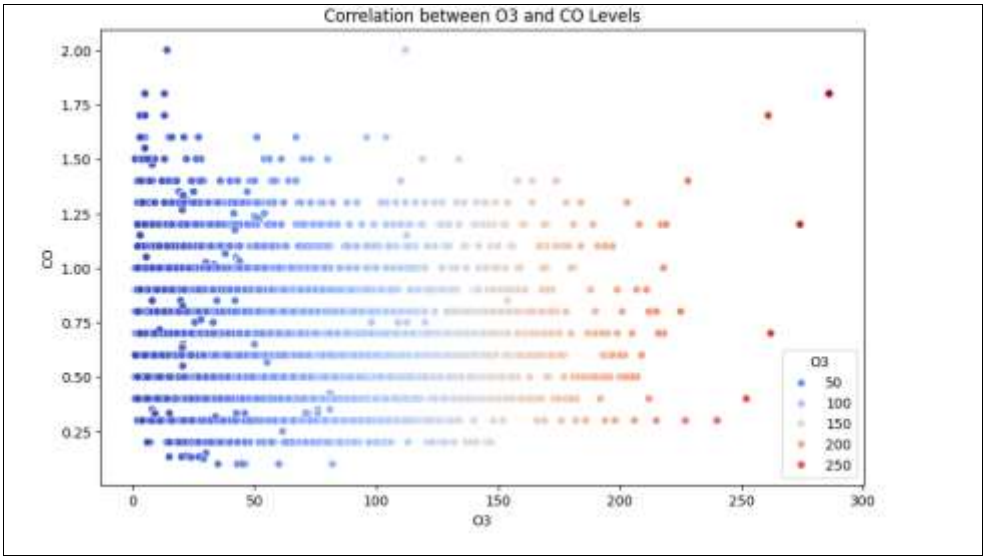


Figure 4.3.8: Correlation between O3 and CO Levels

Question 5: How does AQI vary across various times?

Figure 4.3.9 show the AQI over the period of each month. During the mid of December, it shows that the AQI was recorded highest which is more than 200 and in whole December month it was recorded high as compared to the previous months.

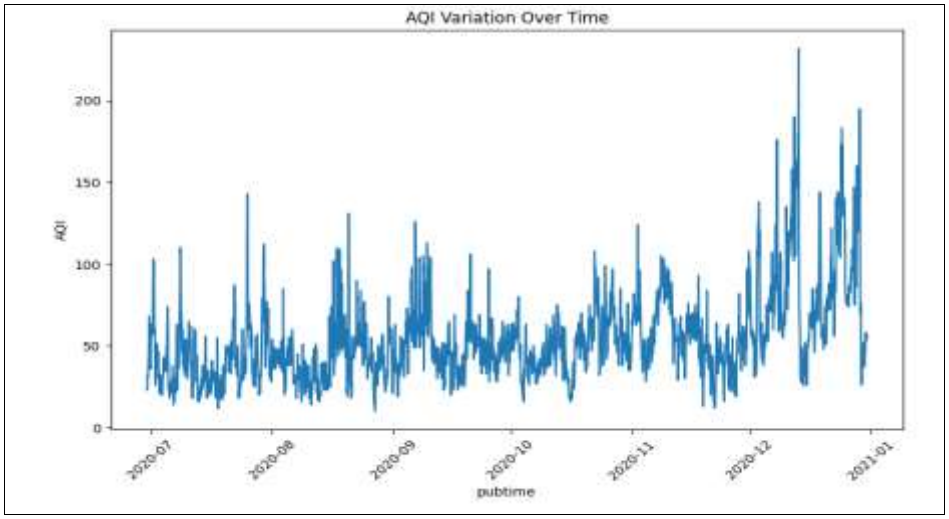


Figure 4.3.9: AQI variation Over Time Period

Question 6: What is the distribution of SO2 Concentrations?

Figure 4.3.10 show the distribution of SO2 concentrations. Histogram plots show the SO2 concentration is skewed to the right, where lower concentrations are occurring more frequently than higher concentrations. This suggests most observations fall in lower concentration range as long tail extends towards higher values.

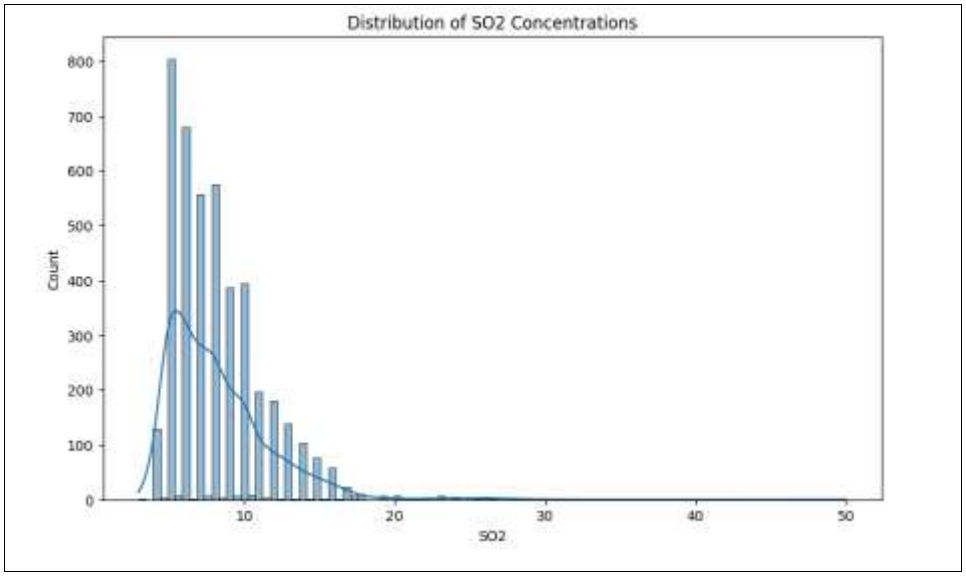


Figure 4.3.10: Distribution of SO2 Concentrations

Question 7: How do PM2.5 and PM10 levels compare?

Figure 4.3.11 show the comparison of PM2.5 and PM10 through violin plot. Each violin represents the observed data with particles where width of each violin shows the frequency of observation at different concentrations level. Each quartile is represented by the horizontal line drawn inside the violin plot.

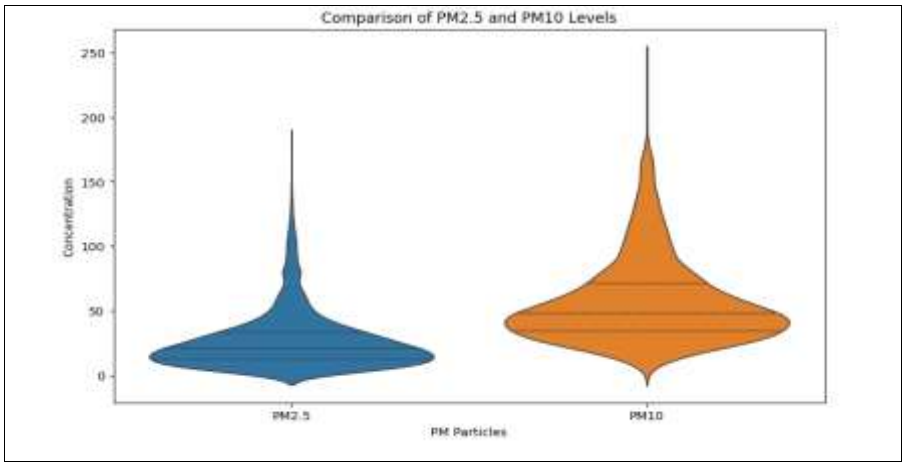


Figure 4.3.11: Comparison of PM2.5 and PM10 Levels

Question 8: How do the NO2 and CO levels vary spatially?

Figure 4.3.12 show the correlation Heatmap of NO2 and CO levels where correlations coefficient is 0.35 between these two gases which can be state as moderate positive correlation. This suggests that with higher concentration of CO tend to exhibit higher concentration of NO2 and vice versa.

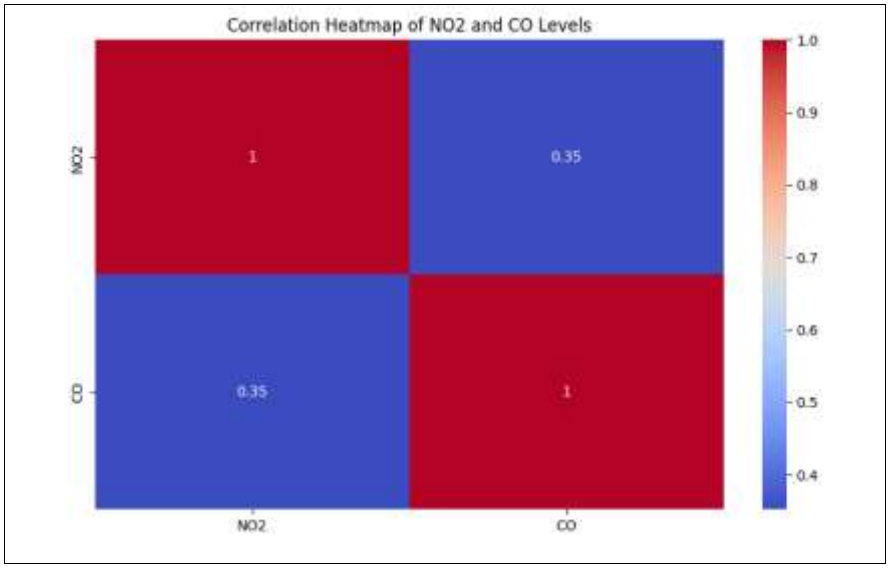


Figure 4.3.12: Correlation Heatmap of NO2 and CO levels

Question 9: Are there any seasonal patterns in AQI concentrations?

Figure 4.3.13 show the seasonal patterns in AQI concentrations through the line plot. X-axis show the all the month of the year, where Y-axis show the meaning of AQI. From the plot it can be stated that AQI slightly increased from June to November and after that it skyrocketed and reached the highest level.

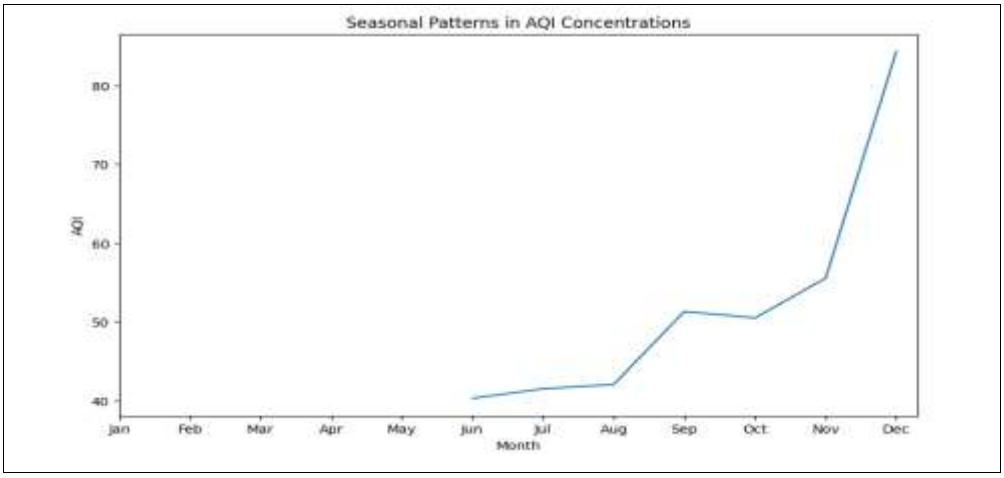


Figure 4.3.13: Seasonal Patterns in AQI Concentrations

Question 10: Are there any seasonal patterns in all variables?

Figure 4.3.14 show the seasonal patterns in all seven variables (PM22.5, PM10, NO2, O3, CO, SO2, AQI) with the help of line graph distributed all the month on X-axis and particles concentration on Y-axis. From the graph it can be stated that concentration of AQI, PM2.5, PM10, and NO2 was increased in each month towards the end of year, and O3 concentration was moving downwards in each month towards the end of year. Whereas the concentration of CO and SO2 was the same throughout each month.

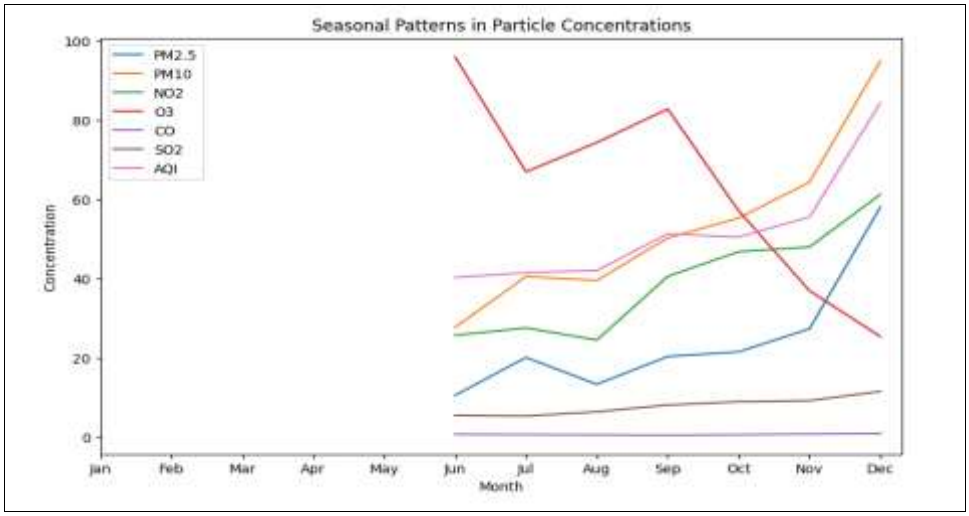


Figure 4.3.14: Seasonal patterns in all particle concentrations

Question 11: How do all variables vary with each other?

Figure 4.3.15 show the correlation Heatmap of all variable's levels. Positive correlations are indicated by warmer color red whereas, negative correlations are indicated by cooler color blue. Correlation coefficient closer to 1 indicates the stronger correlation between two variables whereas negative or coefficient close to 0 indicates the weak relationship between two variables.

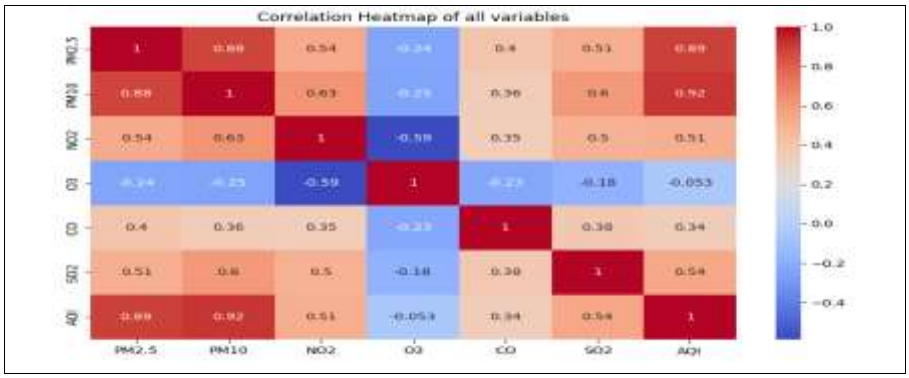


Figure 4.3.15: Correlation Heatmap of all variables

With this exploratory data analysis of air quality data, inner insight was gained from the data. At last, the clean data was extracted and saved it into Google Drive by renaming it to “clean_data.csv” which will be used in the machine leaning model in section 4.3.3 and 4.3.4. ***Figure 4.3.16*** show the shape of data that will be used to perform machine learning model to find the better air quality index predicting model.

Figure 4.3.16: Saving Clean data to Google Drive

4.3.3 Linear Regression

LR Model-1 (Predicting AQI with the help of PM2.5)

In model-1, AQI values were predicted with the help of only one feature column PM2.5 through Linear Regression machine learning model. First data was loaded into cloud notebook from Google Drive, then feature variable (PM2.5) was store on x and target column (AQI) was store in y. With the help of train_test_split libraries of sklearn, data was split into x_train, y_train, x_test, and y_test. Test_size was set to 0.2 which is the ratio of training and testing data, that means the whole dataset is divided 80% to training set and 20% to testing. Once the data is ready a linear regression model was fitted into training data as shown in ***Code 4.3.1***.

```
#dropping all column except feature column PM2.5 and store in x
x= df.drop(columns = ['pubtime','AQI','PM10','NO2','O3','CO','SO2'])
#storing target column in y
y = df['AQI']
#splitting test train data by using train_test_split libraries from
sklearn
x_train, x_test, y_train, y_test= train_test_split(x, y, test_size=
0.2,random_state=0)
#fitting the model
lr= LinearRegression()
lr.fit(x_train,y_train)
```

Code 4.3.1: Fitting of Model 1 Linear Regression

Once the model was created, intercept was calculated, also known as value of dependent(target) variable when independent variable(features) is zero. The value of intercept for the first model was '26.340297259337067'. Then coefficient value was calculated for the feature column which indicates the strength and direction of relationship between each feature and target variables. The value of coefficient for the first model was '1.04081298'. Then training values for y (y_pred_train) was predicted with the help of model by passing x_train values on it. Once the training values for y was predicted, scatter plot was plotted to compare the actual and predicted training AQI values as shown in ***Figure 4.3.17***.

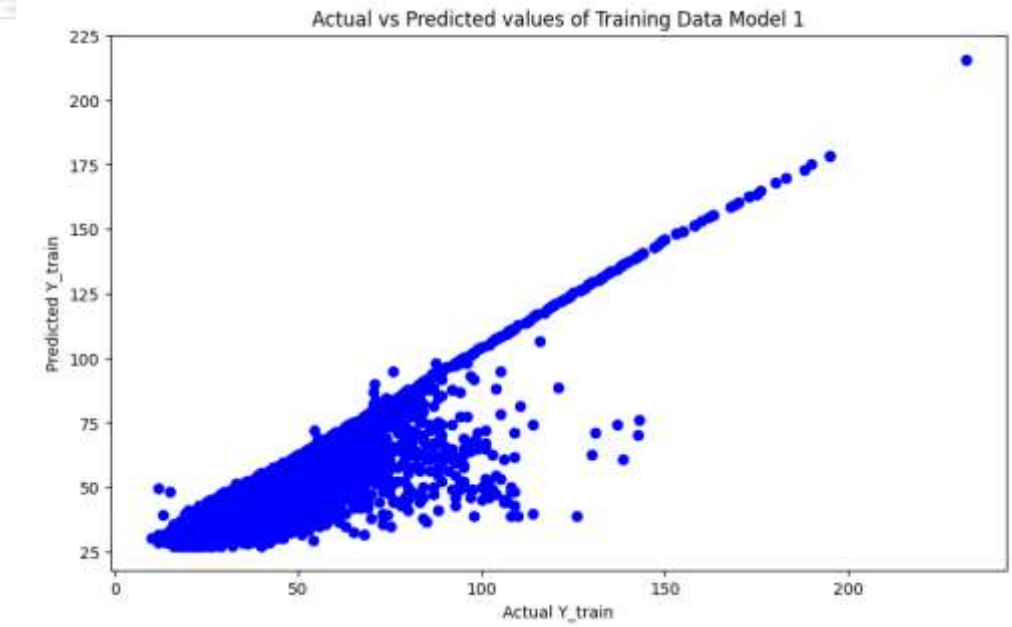


Figure 4.3.17: Comparison of Actual and Predicted training AQI Values of LR Model 1

Then testing values for y (y_pred_test) was predicted with the help of model by passing x_test values on it. Once the testing values for y was predicted, scatter plot was plotted to compare the actual AQI (y_test) with predicted AQI (y_pred_test) testing values as shown in ***Figure 4.3.18***.

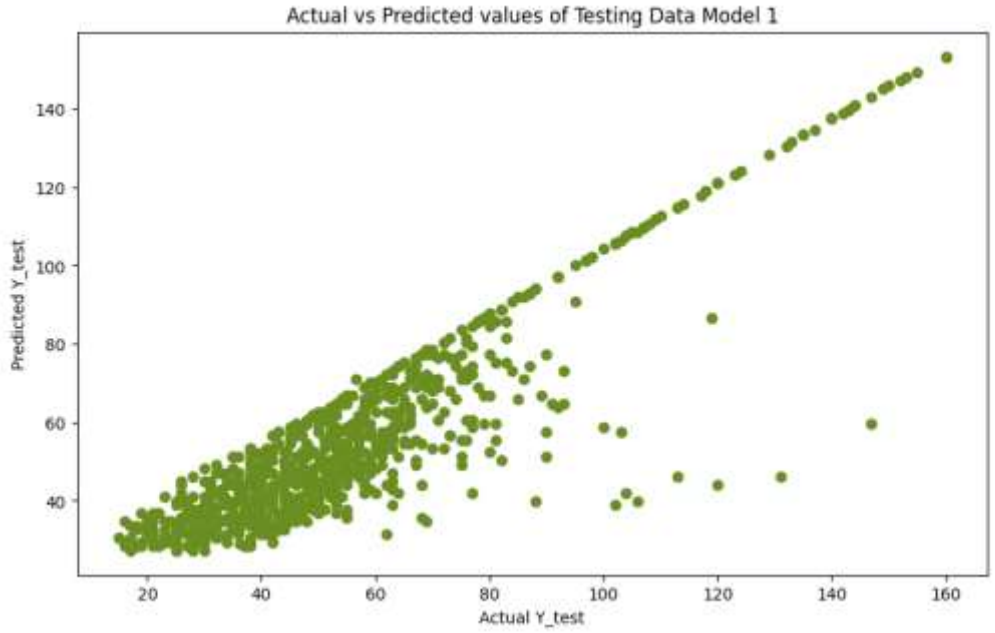


Figure 4.3.18: Comparison of Actual and Predicted Testing AQI Values of LR Model 1

An error term for training data was calculated by subtracting y_pred_train data from y_train data and store it into res_train. ***Figure 4.3.19*** (a) show the distribution of error

terms on training data which shows many data points falls around -20 to 20 on x-axis, that indicate model predictions is far to the actual values for sizable portion of datasets. Then error term for testing data was calculated by subtracting y_pred_test data from y_test data and store it into res_test. ***Figure 4.3.19*** (b) shows the distribution of error terms on testing data which shows many data points falls around -20 to 20 on x-axis, that indicate model predictions is far to the actual values for sizable portion of datasets.

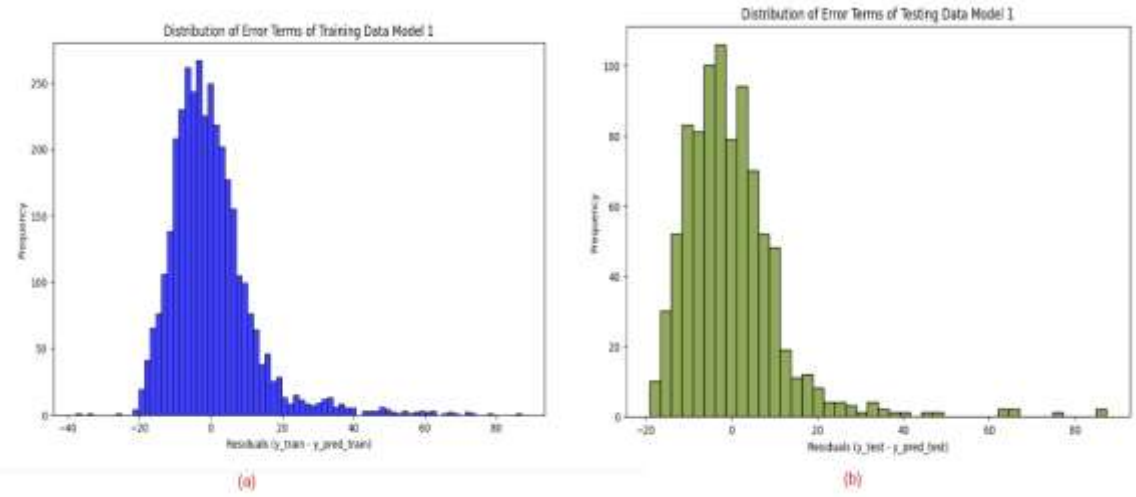


Figure 4.3.19: Distribution of Error Terms in Training and Testing Data of LR Model 1

After figuring out each aspect of model, an evaluation of linear regression was done using different parameters in training and testing data. ***Code 4.3.2*** shows the calculation of these parameters and ***Table 4.3.1*** show the result that was obtained through the linear regression model 1 that is predicting AQI with one feature column PM2.5 on both training and testing data.

```
#Training Data
#Calculating Mean Squared Error of Training Data
mse = mean_squared_error(y_train, y_pred_train)
print("Mean Squared Error (MSE) of Training Data:", mse)
#calculating Root Mean Squared Error of Training Data
rmse = np.sqrt(mse)
print("Root Mean Squared Error (RMSE) of Training Data:", rmse)
#Calculating Mean Absolute Error of Training Data
mae = mean_absolute_error(y_train, y_pred_train)
print("Mean Absolute Error (MAE) of Training Data:", mae)
#Calculating R-Squared of Training Data
rsquare =  r2_score (y_train, y_pred_train)
```

```
print("R-sqaured of Training Data:", rsquare)

#Testing Data
#Calculating Mean Squared Error
mse = mean_squared_error(y_test, y_pred_test)
print ("Mean Squared Error (MSE) of Testing Data:", mse)
#calculating Root Mean Squared Error
rmse = np.sqrt(mse)
print("Root Mean Squared Error (RMSE) of Testing Data:", rmse)
#Calculating Mean Absolute Error
mae = mean_absolute_error(y_test, y_pred_test)
print("Mean Absolute Error (MAE) of Testing Data:", mae)
#Calculating R-Squared of Testing Data
rsquare = r2_score(y_test, y_pred_test)
print("R-sqaured of Testing Data:", rsquare)
```

Code 4.3.2: Evaluation of LR Model 1 on both training and testing data

Table 4.3.1 Evaluation of Linear Regression Model 1

	Training Data	Testing Data
MSE	137.09	132.22
RMSE	11.71	11.49
MAE	8.18	7.90
R-Squared	0.79	0.79

LR Model-2 (Predicting AQI with the help of PM2.5 and PM10)

In model-2, AQI values were predicted with the help of two feature columns PM2.5 and PM10 through Linear Regression machine learning model. Once the data is ready a linear regression model was fitted into training data as shown in ***Code 4.3.3***.

```
#dropping all column except feature column PM2.5 and PM10 and store
in x
x= df.drop(columns = ['pubtime','AQI','NO2','O3','CO','SO2'])
#storing target column in y
y = df['AQI']
#splitting test train data by using train_test_split libraries from
sklearn
x_train, x_test, y_train, y_test= train_test_split(x, y, test_size=
0.2,random_state=0)
#fitting the model
lr= LinearRegression()
lr.fit(x_train,y_train)
```

Code 4.3.3: Fitting of Model 2 Linear Regression

Once the model was created then intercept was calculated. The value of intercept for the second model was '15.780111766846716'. The values of coefficient for the second model were '[0.43999371, 0.46367613]'. Then training values for y (y_pred_train) was predicted with the help of model by passing x_train values on it. Once the training values for y was predicted, scatter plot was plotted to compare the actual AQI (y_train) with predicted AQI (y_pred_train) training values for model 2 as shown in ***Figure 4.3.20***.

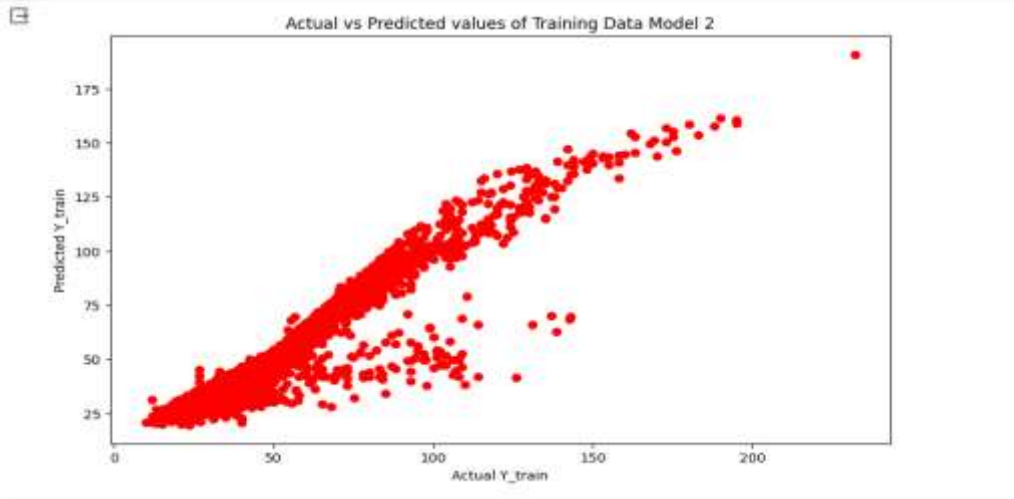


Figure 4.3.20: Comparison of Actual and Predicted training AQI Values of LR Model 2

Then testing values for y (y_pred_test) was predicted with the help of model by passing x_test values on it. Once the testing values for y was predicted, scatter plot was plotted to compare the actual AQI (y_test) with predicted AQI (y_pred_test) testing values as shown in ***Figure 4.3.21***.

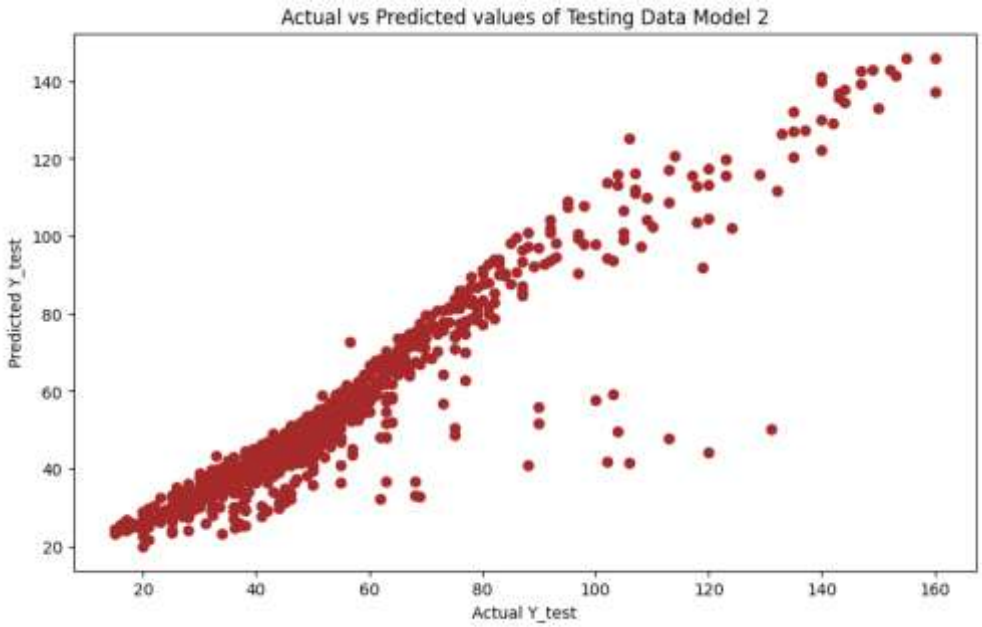


Figure 4.3.21: Comparison of Actual and Predicted Testing AQI Values of LR Model 2

An error term for training data and testing data was calculated. ***Figure 4.3.22*** (a) show the distribution of error terms on training data which shows many data points falls around -10 to 10 on x-axis, that indicate model predictions is near to the actual values for sizable portion of datasets. ***Figure 4.3.22*** (b) shows the distribution of error terms on testing data which shows many data points falls around -10 to 10 on x-axis,

that indicates model 2 prediction is close to the actual values for a sizable portion of datasets.

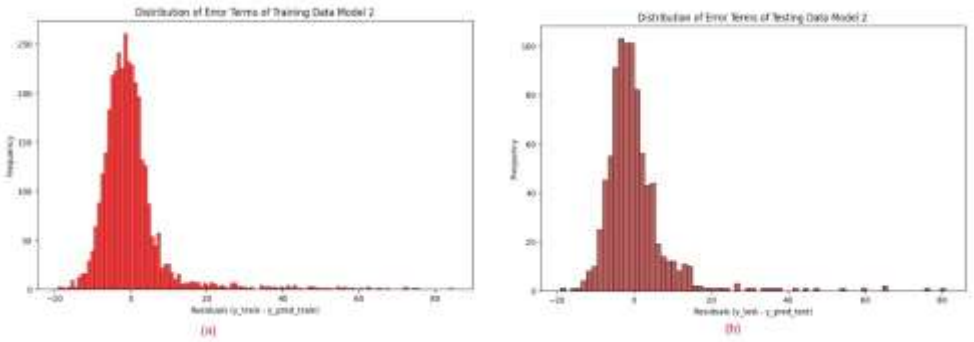


Figure 4.3.22: Distribution of Error Terms in Training and Testing Data of LR Model 2

After figuring out each aspect of model 2, an evaluation of linear regression was done using different parameters in training and testing data. ***Table 4.3.2*** show the result that was obtained through the linear regression model 2 that is predicting AQI with two feature columns PM2.5 and PM 10 on both training and testing data.

Table 4.3.2 Evaluation of Linear Regression Model 2

	Training Data	Testing Data
MSE	83.78	79.53
RMSE	9.15	8.91
MAE	5.30	5.21
R-Squared	0.87	0.87

LR Model-3 (Predicting AQI with the help of NO2, O3, CO, and SO2)

In model-3, AQI values were predicted with the help of four feature columns (all gases) NO2, O3, CO, and SO2 through Linear Regression machine learning model. Once the data is ready a linear regression model was fitted into training data as shown in ***Code 4.3.4***.

```
#dropping all column except feature column NO2, O3, CO, and SO2 and
store in x
x= df.drop(columns = ['pubtime','AQI','PM2.5','PM10'])
#storing target column in y
y = df['AQI']
#splitting test train data by using train_test_split libraries from
sklearn
x_train, x_test, y_train, y_test= train_test_split(x, y, test_size=
0.2,random_state=0)
#fitting the model
lr= LinearRegression()
lr.fit(x_train,y_train)
```

Code 4.3.4: Fitting of Model 3 Linear Regression

Once the model was created then intercept was calculated, value of intercept for the third model was ‘ -6.101049643142872’. The values of coefficient for the third model were ‘[0.52839932, 0.17889162,10.62499604, 2.40649915]’. Then training values for y (y_pred_train) was predicted with the help of model by passing x_train values on it. Once the training values for y was predicted, scatter plot was plotted to compare the actual AQI (y_train) with predicted AQI (y_pred_train) training values as shown in ***Figure 4.3.23***.

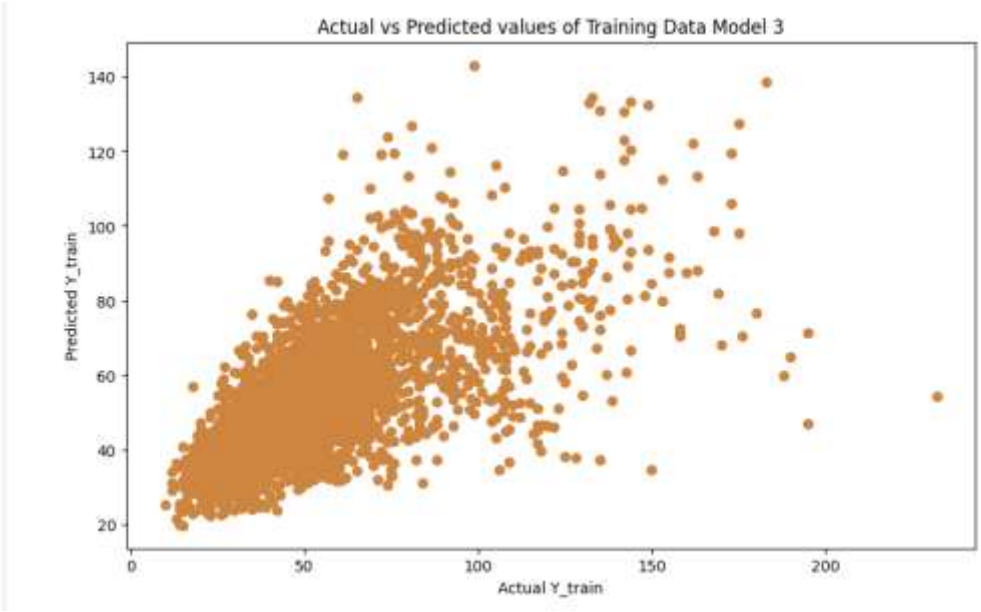


Figure 4.3.23: Comparison of Actual and Predicted Training AQI Values of LR Model 3

Then testing values for y (y_pred_test) was predicted with the help of model by passing x_test values on it. Once the testing values for y was predicted, scatter plot was plotted to compare the actual AQI (y_test) with predicted AQI (y_pred_test) testing values as shown in ***Figure 4.3.24***.

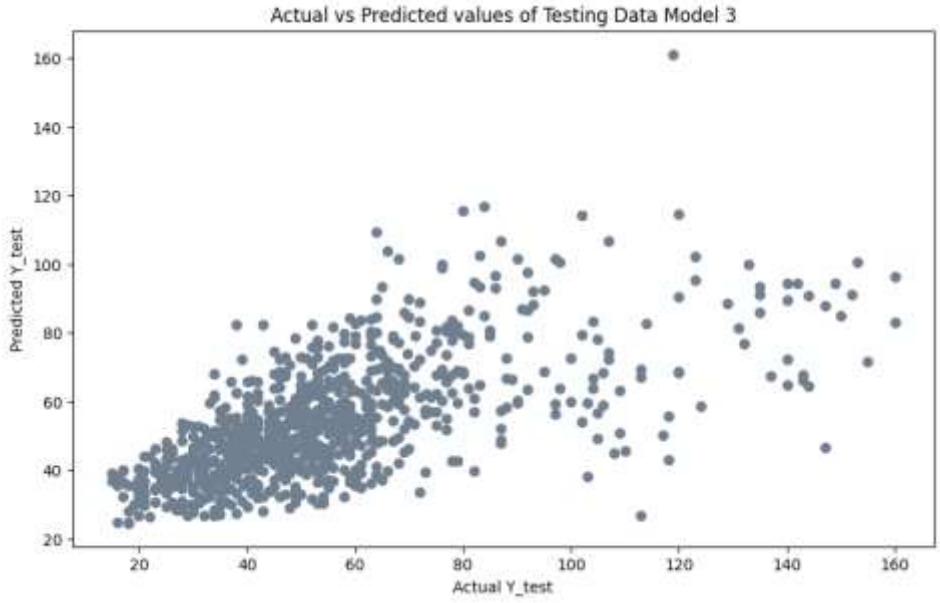


Figure 4.3.24: Comparison of Actual and Predicted Testing AQI Values of LR Model 3

An error term for training and testing data was calculated. ***Figure 4.3.25*** (a) show the distribution of error terms on training data which shows many data points fall around -50 to 50 on x-axis, that indicate model predictions are too far to the actual values for

sizable portion of datasets. ***Figure 4.3.25*** (b) shows the distribution of error terms on testing data which shows many data points falls around -20 to 20 on x-axis, that indicate model 3 prediction is too far to the actual values for sizable portion of datasets.

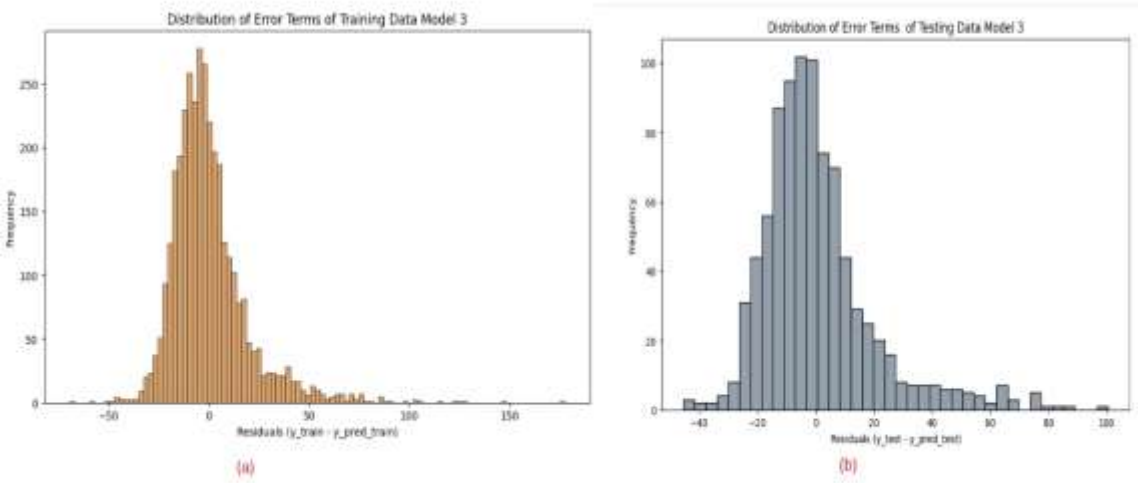


Figure 4.3.25: Distribution of Error Terms in Training and Testing Data of LR Model 3

After figuring out each aspect of model 3, an evaluation of linear regression was done using different parameters in training and testing data. ***Table 4.3.3*** show the result that was obtained through the linear regression model 3 that is predicting AQI with four feature columns NO2, O3, CO, and SO2 on both training and testing data.

Table 4.3.3 Evaluation of Linear Regression Model 3

	Training Data	**Testing Data**
MSE	367.51	381.11
RMSE	19.17	19.52
MAE	13.35	13.71
R-Squared	0.45	0.42

LR Model-4 (Predicting AQI with the help of PM2.5, PM10, NO2, O3, CO, and SO2)

In model-4 AQI values were predicted with the help of six feature columns (all particle and gases) PM2.5, PM10, NO2, O3, CO, and SO2 through Linear Regression machine learning model. Once the data is ready a linear regression model was fitted into training data as shown in ***Code 4.3.5***.

```
#dropping all column except feature column PM2.5, PM10, No2, O3, CO
and SO2 and store in x
x= df.drop(columns = ['pubtime','AQI'])

#storing target column in y
y = df['AQI']
#splitting test train data by using train_test_split libraries from
sklearn
x_train, x_test, y_train, y_test= train_test_split(x, y, test_size=
0.2,random_state=0)
#fitting the model
lr= LinearRegression()
lr.fit(x_train,y_train)
```

Code 4.3.5: Fitting of Model 4 Linear Regression

Once the model was created then intercept was calculated, which was '5.5155327552498505'. The values of coefficient for the fourth model were [0.47231332, 0.46027295, 0.07279498, 0.12498361, 0.45818509, -0.11613176]. Then training values for y (y_pred_train) was predicted with the help of model by passing x_train values on it. Once the training values for y was predicted, scatter plot was plotted to compare the actual AQI (y_train) with predicted AQI (y_pred_train) training values as shown in ***Figure 4.3.26***.

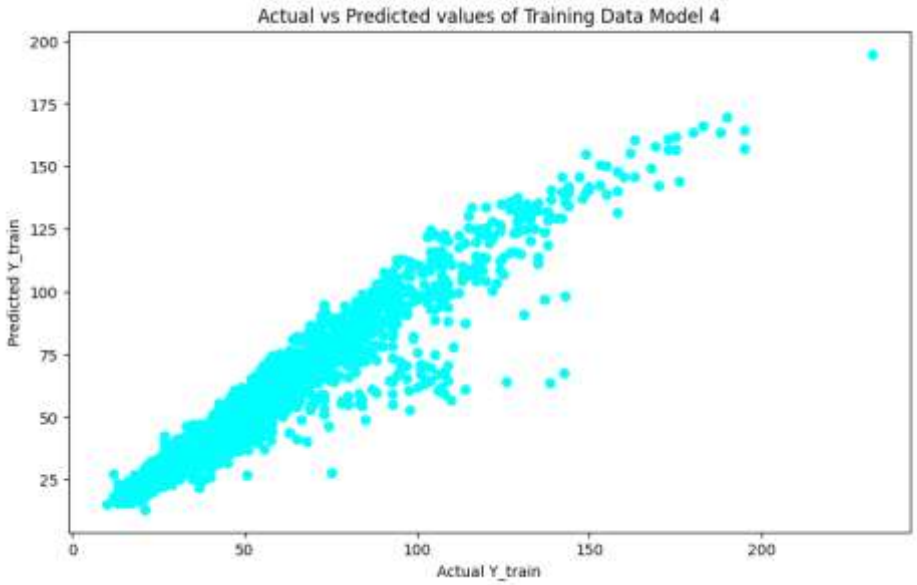


Figure 4.3.26: Comparison of Actual and Predicted Testing AQI Values of LR Model 4

Then testing values for y (y_pred_test) was predicted with the help of model by passing x_test values on it. Once the testing values for y was predicted, scatter plot was plotted to compare the actual AQI (y_test) with predicted AQI (y_pred_test) testing values as shown in***Figure 4.3.27***.

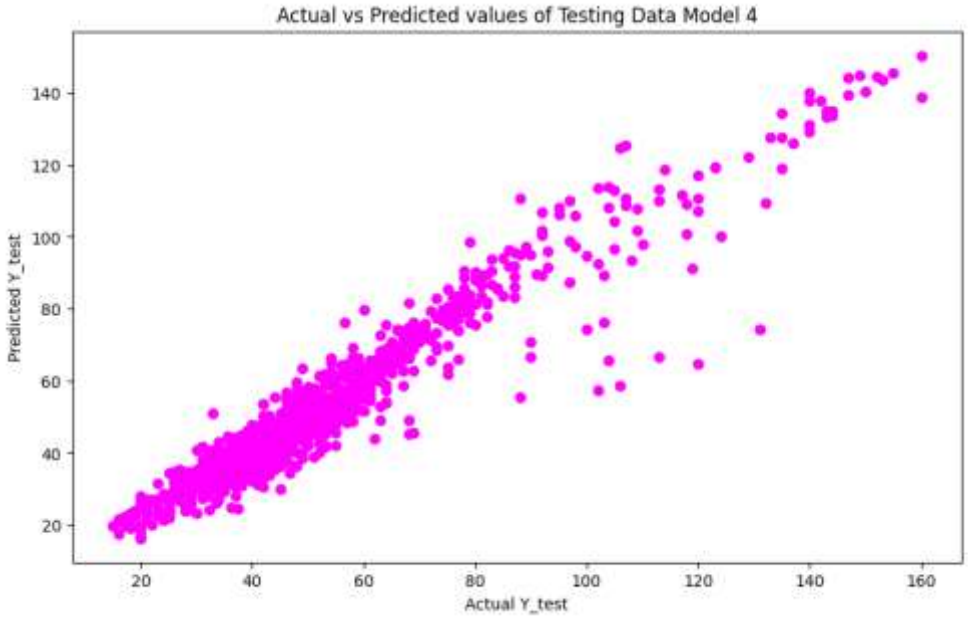


Figure 4.3.27: Comparison of Actual and Predicted Testing AQI Values of LR Model 4

An error term for training and testing data was calculated. ***Figure 4.3.28*** (a) show the distribution of error terms on training data which shows many data points falls around

zero on x-axis, that indicate model 4 prediction is closer to the actual values for sizable portion of datasets. ***Figure 4.3.28*** (b)shows the distribution of error terms on testing data which shows many data points falls around zero on x-axis, that indicates model 4 prediction is closer to the actual values for sizable portion of datasets.

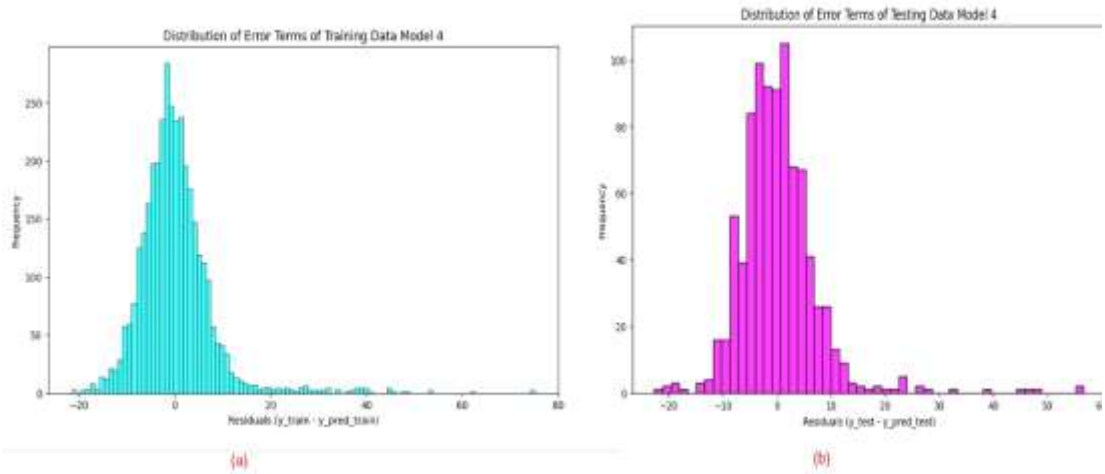


Figure 4.3.28: Distribution of Error Terms in Testing Data of LR Model 4

After figuring out each aspect of model 4, an evaluation of linear regression was done using different parameters in training and testing data. ***Table 4.3.4*** show the result that was obtained through the linear regression model 4 that is predicting AQI with six feature columns PM2.5, PM10, NO2, O3, CO, and SO2 on both training and testing data.

Table 4.3.4 Evaluation of Linear Regression Model 4

	Training Data	**Testing Data**
MSE	58.33	55.27
RMSE	7.63	7.43
MAE	5.09	5.01
R-Squared	0.91	0.91

4.3.4 Long Short-Term Memory

LSTM Model-1 (Predicting AQI with Univariate LSTM)

In model-1 of LSTM model, AQI values were predicted by using univariate column (AQI) (analysis of single column in a dataset to predict the target column). First data was loaded into Google Colab Notebook from Google drive and select a just one column AQI to predict the AQI. LSTM is a time-series forecasting method, so while splitting data the first 80% of data is used for training and the remaining 20% is used for testing. After that the whole dataset was scaled from 0 to 1, because this scaling ensures that all the input features lie within a range of 0 to 1 to improve the convergence and performance of the model. Once the training and testing data was scaled, a new function 'create_sequences' was created as shown on ***Code 4.3.6***. This function helps to create the data by dividing the dataset into the sequence of length(input_size). Input_size was setup as 24, while collecting the data it was collected in an interval of 1 hour. So, this function creates a sequence of 24 data points which is equal to one day. With the help of this function LSTM model uses each sequence as an input, and the subsequent element in dataset as the matching output. As neural network layer always expect dataset into three-dimensional format, that is data, timestamp, and feature column. So, data was sent to three-dimensional format by creating sequence on both X_train and X_test data with help of create_sequences function where three dimensional is each data point, timestamp as 24, and feature column as 1(AQI itself). Resulting the shape of X_train, and X_test as (3509, 24, 1), and (860,24,1) respectively and y_train, and y_test is (3509,1), and (860,1) respectively.

```
#Selecting only one column to predict AQI from AQI previous data
df= df[['AQI']]
df.head()
#spliting dataset for training and testing phase
train_size = int(len(df) * 0.8)
train_data = df.iloc[:train_size]
test_data = df.iloc[train_size:]
#scaling dataset to o and 1 by using MinMaxScaler
scaler = MinMaxScaler(feature_range= (0, 1))
train_scaled = scaler.fit_transform(train_data)
test_scaled = scaler.transform(test_data)
```

```
#Creating input-output pairs by splitting dataset into a sequence of
input_size
def create_sequences(data, input_size):
    X, y = [], []

    for i in range(len(data) - input_size):
        X.append(data[i:i + input_size])
        y.append(data[i + input_size])
    return np.array(X), np.array(y)

input_size = 24
X_train, y_train = create_sequences(train_scaled, input_size)
X_test, y_test = create_sequences(test_scaled, input_size)

#creating a LSTM model
model = Sequential()
model.add(LSTM(100, activation='relu',input_shape=(X_train.shape[1],
X_train.shape[2])))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse')

#fitting the model
history = model.fit(X_train, y_train, epochs=50,
batch_size=1,validation_split=0.2)
```

Code 4.3.6: Creating A LSTM Model 1

Once the training and testing data was ready, an LSTM model was created as shown in ***Code 4.3.6***. 100 represent the number of neurons used in LSTM model which helps to adjust the complexity of problem, activation =’relu’ represent Rectified Linear Unit functions which helps to learn the complex patterns in the data, input_shape represent the X_train.shape[1] which is 24 as timestamp, and X_train.shape[2] which is 1 as feature column. Dense layer was used as 1 which normally carried out regression task to predict the one continuous value. Then the model was compiled as adam as optimizer and mse as a loss. At last LSTM model was trained on prepared training data where epoch is set to 50, which means the total number of times entire dataset pass forward and backward through neural network for training, batch_size was set as

'1' which helps model to update its parameter after process each sample, and validation_split as 0.2 where 20% of training data will be used for validation and rest of 80% for actual training. ***Figure 4.3.29*** show the training and validation loss over each epoch.

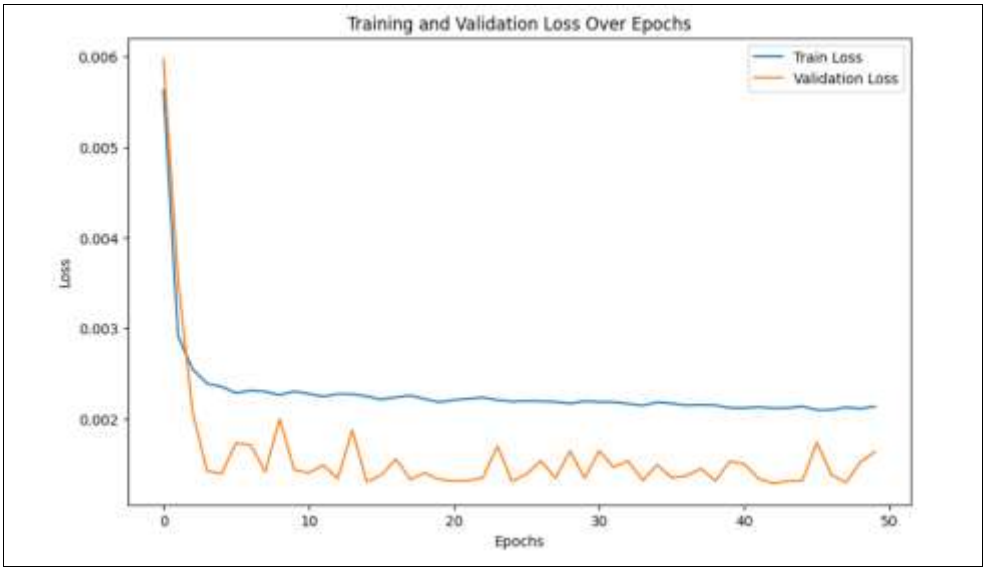


Figure 4.3.29: Training and Validation Loss over Epoch of LSTM Model 1

Once the machine learned from the training data, train and test data was predicted by using LSTM model 1. To evaluate the model performance, both predicated and actual data was scaled back to original shape with the help of inverse_transform as shown in ***Code 4.3.7***.

```
#predicting train and test data from x_train and x_test
train_predicted = model.predict(X_train.reshape(X_train.shape))
test_predicted = model.predict(X_test.reshape(X_test.shape))

#Scale back to original scale for both train and test predicted data
train_predicted = scaler.inverse_transform(train_predicted)
test_predicted = scaler.inverse_transform(test_predicted)

#Scale back to original scale for both train and test actual data
train_actual = scaler.inverse_transform(y_train)
test_actual = scaler.inverse_transform(y_test)


#creating numpy array data for plotting actual and predicted data
x1 = np.arange(0, len(train_actual))
```

```
x2 = np.arange(len(train_actual), len(train_actual)+len(test_actual))

# Plotting Both Train and Test Actual Data
plt.figure(figsize=(10, 6))
plt.plot(x1, train_actual)
plt.plot(x1, train_predicted)
plt.plot(x2, test_actual)
plt.plot(x2, test_predicted)
plt.legend(['Train Actual', 'Train Predicted', 'Test Actual', 'Test
Predicted'])
plt.xlabel('Data Points')
plt.ylabel('Values')
plt.title('Comparison of Train, Test Actual and Predicted  Data')
plt.vlines(x=len(train_actual), color='r', linestyles='dashed',
ymin=0, ymax=max(test_actual))
plt.show()
```

Code 4.3.7: Inverse Scaling and Plotting LSTM Model 1 Actual and Predicted Data

Two numpy arrays were created x1 store data point from index 0 to the length of actual train data and x2 store the last of actual train data to last of actual test data. Then plot was plotted to compare the comparison of actual train and test data with predicted train and testing data as shown in ***Figure 4.3.30***. The red line in the figure dived the training and testing data used in this model.

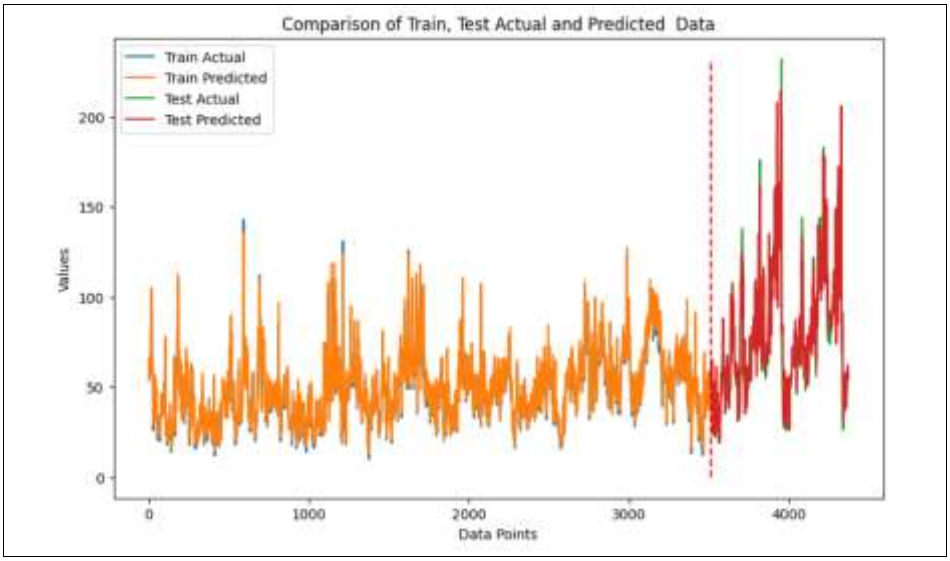


Figure 4.3.30: Comparison of Train, Test Actual and Predicted Data of LSTM Model 1

At last, with the help of sklearn metrics libraries, LSTM model 1 performance was evaluated based on MSE, RMSE, MAE and R-Squared which is show on ***Table 4.3.5***.

Table 4.3.5: Evaluation of LSTM Model 1

	Training Data	**Testing Data**
MSE	33.63	171.98
RMSE	5.79	13.11
MAE	3.90	8.16
R-Squared	0.90	0.86

LSTM Model -2 (Predicting AQI with Multivariate LSTM PM2.5, and PM10)

In model-2 of LSTM model, AQI values were predicted by using multivariate method where the feature column was set to PM2.5, PM10 and AQI itself. All the processes were used as same for splitting data and scaling the dataset as in LSTM Model 1. But in 'create_sequence' function there was a slight change which is shown in ***Code 4.3.8***. As there are three feature column that was passed through the training and testing phase, but the target column should only be one that is AQI, that's why in code '**y.append(data i+input_size, -1)**' -1 was added as it state that it will take only last element from row, which AQI values. As the create_sequence function was changed as there are three features column, so shape of X_train and X_test became (3509, 24, 3) and (860, 24, 3) respectively.

```
#Creating input-output pairs by splitting dataset into a sequence of
input_size

def create_sequences(data, input_size):
    X, y = [], []
    for i in range(len(data) - input_size):
        X.append(data[i:i + input_size, :])
        y.append(data[i + input_size, -1])
    return np.array(X), np.array(y)

input_size = 24

X_train, y_train = create_sequences(train_scaled, input_size)

X_test, y_test = create_sequences(test_scaled, input_size)
```

Code 4.3.8: Creating Input-Output Pairs for LSTM Model 2

Once the training and testing data was ready, LSTM model was created and fit the model as same as in **LSTM Model-1 (Predicting AQI with Univariate** LSTM). ***Figure 4.3.31*** show the training and validation loss over each epoch in LSTM Model 2.

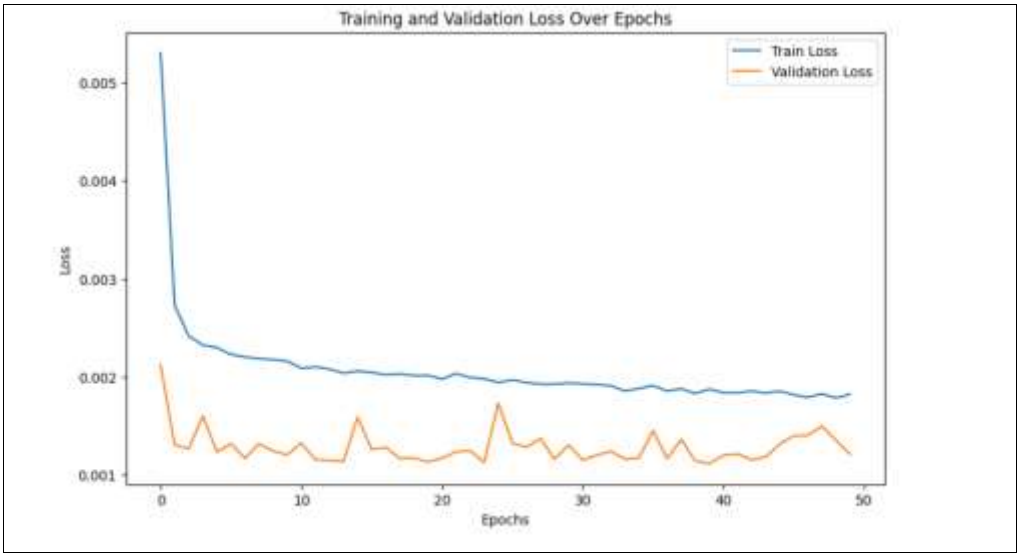


Figure 4.3.31: Training and Validation Loss over Epoch of LSTM Model 2

Once the model learns from the training data, train and test data was predicted by using LSTM model 2. To evaluate the model performance, both predicated and actual data was scaled back to original shape with the help of inverse_transform as shown in ***Code 4.3.9***. But before applying inverse transform on train_predicted, and test_predicted must make a same shape of size as previous that was done while scaling. While scaling, there were three columns for each row, but now there is only one, the target column. To solve this problem, single predicted values are copied 2 times which make the shape similar. Also train_actual, and test_actual values also scale at first, so we repeat the same process of copying the same data 2 times to make the shape same. At last inverse_transform function was applied on both predicted and actual data.

```
#predicting train and test data from x_train and x_test
train_predicted = model.predict(X_train.reshape(X_train.shape))
test_predicted = model.predict(X_test.reshape(X_test.shape))
#Repeating three column of same data to use inverse transform
train_prediction_copies = np.repeat(train_predicted, 3, axis = -1  )
test_prediction_copies = np.repeat(test_predicted, 3, axis = -1  )
train_actual = np.repeat(y_train, 3, axis = -1  )
test_actual = np.repeat(y_test, 3, axis = -1  )
```

```
#Scale back to original scale for both train and test predicted data
by selecting first column from repeated cloumn
train_predicted =
scaler.inverse_transform(np.reshape(train_prediction_copies,(len(trai
n_predicted),3)))[:,0]
test_predicted =
scaler.inverse_transform(np.reshape(test_prediction_copies,(len(test_
predicted),3)))[:,0]

#Scale back to orginal scale for both train and test actual databy
selecting first column from repeated cloumn
train_actual =
scaler.inverse_transform(np.reshape(train_actual,(len(X_train),3)))[:
,0]
test_actual =
scaler.inverse_transform(np.reshape(test_actual,(len(y_test),3)))[:,0
]
```

Code 4.3.9: Inverse Scaling on LSTM Model 2

Two numpy arrays were created x1 store data point from index 0 to the length of actual train data and x2 store the last of actual train data to last of actual test data. Then plot was plotted to compare the comparison of actual train and test data with predicted train and testing data in LSTM model-2 as shown in ***Figure 4.3.32***.

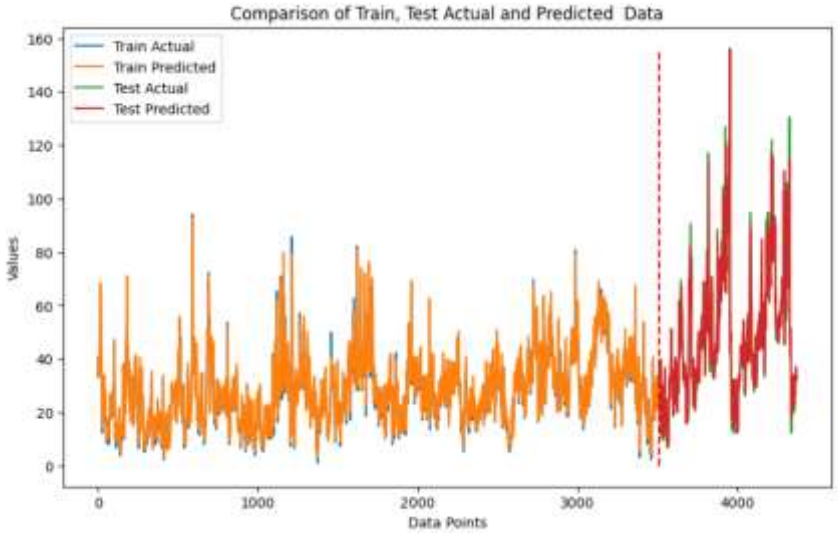


Figure 4.3.32: Comparison of Train, Test Actual and Predicted Data of LSTM Model 2

At last, LSTM model 2 performance was evaluated based on MSE, RMSE, MAE and R-Squared which is show on ***Table 4.3.6***.

Table 4.3.6: Evaluation of LSTM Model 2

	Training Data	**Testing Data**
MSE	14.24	62.09
RMSE	3.77	7.87
MAE	2.62	4.93
R-Squared	0.91	0.90

LSTM Model -3 (Predicting AQI with Multivariate LSTM NO2, O3, CO, SO2)

In model-3 of LSTM model, AQI values were predicted by using multivariate method where the feature column was set to NO2, O3, CO, SO2 and AQI itself. All the processes were used as same for splitting data, scaling the dataset, and creating sequence as in **LSTM Model -2 (Predicting AQI with Multivariate LSTM PM2.5, and** PM10). As there are five features column, shape of X_train and X_test was (3509, 24, 5) and (860, 24, 5) respectively. Once the training and testing data was ready, LSTM model was created and fit the model as same as in **LSTM Model-1 (Predicting AQI with Univariate** LSTM). ***Figure 4.3.33*** show the training and validation loss over each epoch in LSTM Model 3.

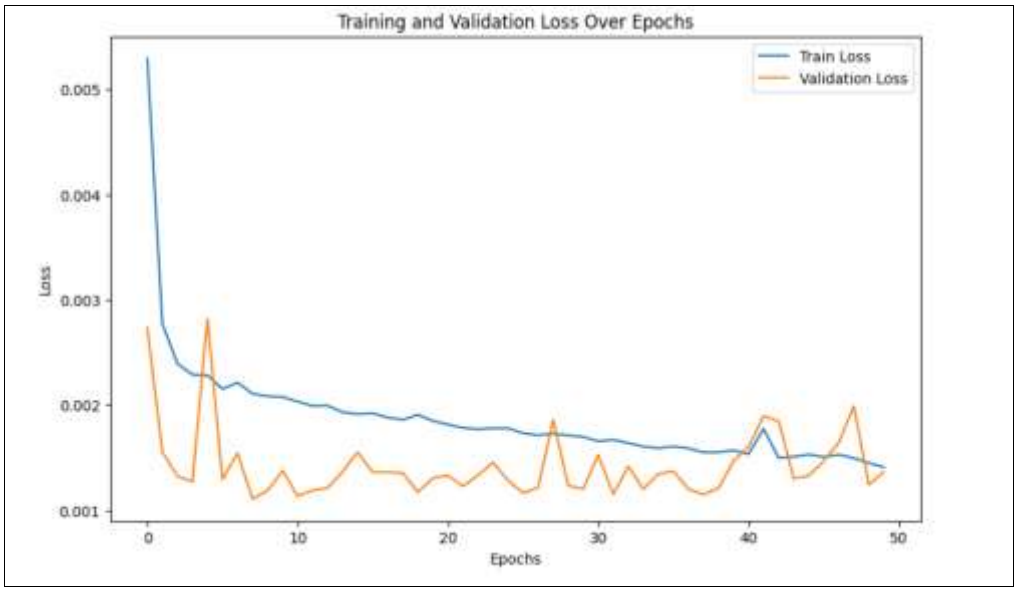


Figure 4.3.33: Training and Validation Loss over Epoch of LSTM Model 3

Once the model learns from the training data, train and test data was predicted by using LSTM model 3. To evaluate the model performance, both predicated and actual data was scaled back to original shape as same as in **LSTM Model -2 (Predicting AQI with Multivariate LSTM PM2.5, and** PM10) . But in model 3 there are five feature columns, so train_predicted, test_predicted, train_actual and test_actual data was copied five times to make the shape same to perform inverse transform as shown in ***Code 4.3.10***.

```
#predicting train and test data from x_train and x_test
train_predicted = model.predict(X_train.reshape(X_train.shape))
test_predicted = model.predict(X_test.reshape(X_test.shape))

#Repeating three column of same data to use inverse transform
train_prediction_copies = np.repeat(train_predicted, 5, axis = -1  )
test_prediction_copies = np.repeat(test_predicted, 5, axis = -1  )
train_actual = np.repeat(y_train, 5, axis = -1  )
test_actual = np.repeat(y_test, 5, axis = -1  )

#Scale back to original scale for both train and test predicted data
by selecting first column from repeated cloumn
train_predicted =
scaler.inverse_transform(np.reshape(train_prediction_copies,(len(trai
n_predicted),5)))[:,0]
test_predicted =
scaler.inverse_transform(np.reshape(test_prediction_copies,(len(test_
predicted),5)))[:,0]

#Scale back to orginal scale for both train and test actual databy
selecting first Colum from repeated Cloum
train_actual =
scaler.inverse_transform(np.reshape(train_actual,(len(X_train),5)))[:
,0]
test_actual =
scaler.inverse_transform(np.reshape(test_actual,(len(y_test),5)))[:,0
]
```

Code 4.3.10: Inverse Scaling on LSTM Model 3

Two numpy arrays were created x1 and x2. Then plot was plotted to compare the comparison of actual train and test data with predicted train and testing data in LSTM model-3 as shown in ***Figure 4.3.34***.

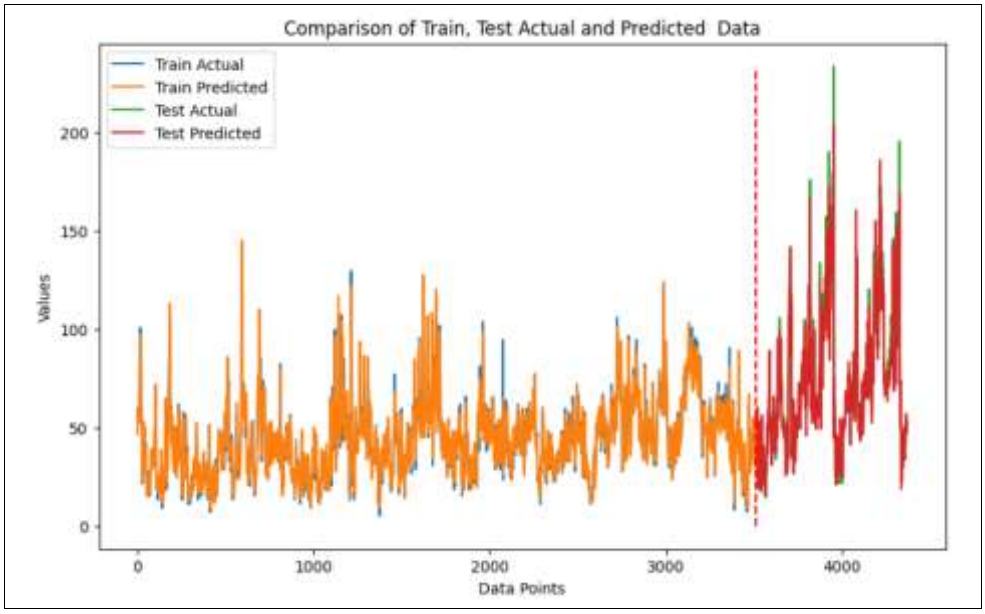


Figure 4.3.34: Comparison of Train, Test Actual and Predicted Data of LSTM Model 3

At last, LSTM model 3 performance was evaluated based on MSE, RMSE, MAE and R-Squared which is show on ***Table 4.3.7***.

Table 4.3.7: Evaluation of LSTM Model 3

	Training Data	**Testing Data**
MSE	25.05	150.88
RMSE	5.005	12.28
MAE	3.51	7.94
R-Squared	0.93	0.88

LSTM Model -4 (Predicting AQI with Multivariate LSTM (Including All Variable))

In model-4 of LSTM model, AQI values were predicted by using multivariate method where the feature column was set to all column variable (PM2.5, PM10, NO2, O3, CO, SO2, and AQI itself). All the processes were used as same for splitting data, scaling the dataset, and creating sequence as in **LSTM Model -2 (Predicting AQI with Multivariate LSTM PM2.5, and** PM10). As there are seven features' columns, the shape of X_train and X_test was (3509, 24, 7) and (860, 24, 7) respectively. Once the training and testing data was ready, LSTM model was created and fit the model as same as in **LSTM Model-1 (Predicting AQI with Univariate** LSTM). ***Figure 4.3.35*** show the training and validation loss over each epoch in LSTM Model 4.

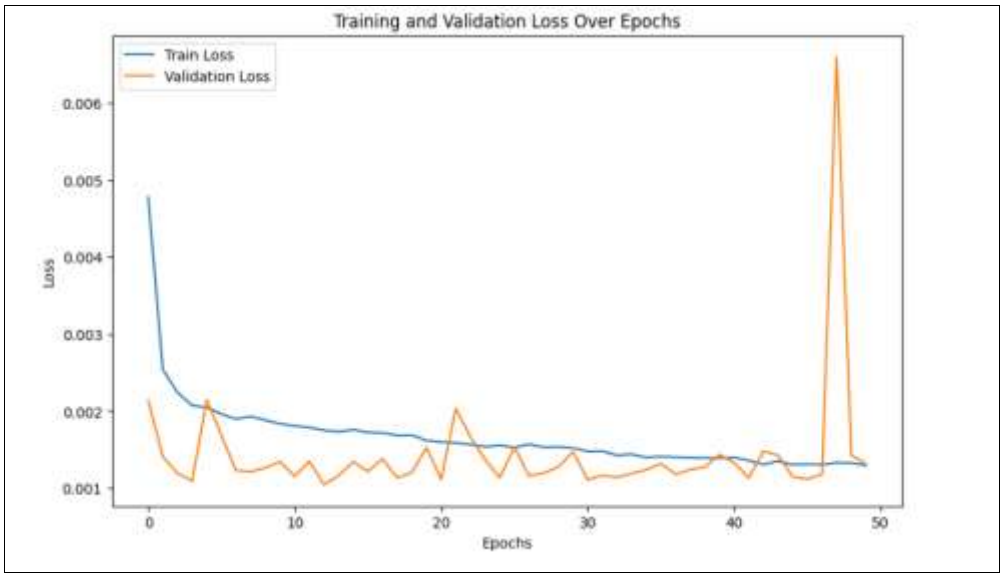


Figure 4.3.35: Training and Validation Loss over Epoch of LSTM Model 4

After the model learns from the training data, train and test data was predicted by using LSTM model 4. To evaluate the model performance, both predicated and actual data was scaled back to original shape as same as in **LSTM Model -2 (Predicting AQI with Multivariate LSTM PM2.5, and** PM10) . But in model 4 there are seven feature columns, so train_predicted, test_predicted, train_actual and test_actual data was copied seven times to make the shape same to perform inverse transform as shown in ***Code 4.3.11***.

```
#predicting train and test data from x_train and x_test
train_predicted = model.predict(X_train.reshape(X_train.shape))
test_predicted = model.predict(X_test.reshape(X_test.shape))
#Repeating three column of same data to use inverse transform
train_prediction_copies = np.repeat(train_predicted, 7, axis = -1  )
test_prediction_copies = np.repeat(test_predicted, 7, axis = -1  )
train_actual = np.repeat(y_train, 7, axis = -1  )
test_actual = np.repeat(y_test, 7, axis = -1  )
#Scale back to original scale for both train and test predicted data
by selecting first column from repeated cloumn
train_predicted =
scaler.inverse_transform(np.reshape(train_prediction_copies,(len(trai
n_predicted),7)))[:,0]
test_predicted =
scaler.inverse_transform(np.reshape(test_prediction_copies,(len(test_
predicted),7)))[:,0]
#Scale back to orginal scale for both train and test actual databy
selecting first column from repeated cloumn
train_actual =
scaler.inverse_transform(np.reshape(train_actual,(len(X_train),7)))[:
,0]
test_actual =
scaler.inverse_transform(np.reshape(test_actual,(len(y_test),7)))[:,0
]
```

Code 4.3.11: Inverse Scaling on LSTM Model 4

Two numpy arrays were created x1 and x2.Then plot was plotted to compare the comparison of actual train and test data with predicted train and testing data in LSTM Model-4 as shown in ***Figure 4.3.36***.

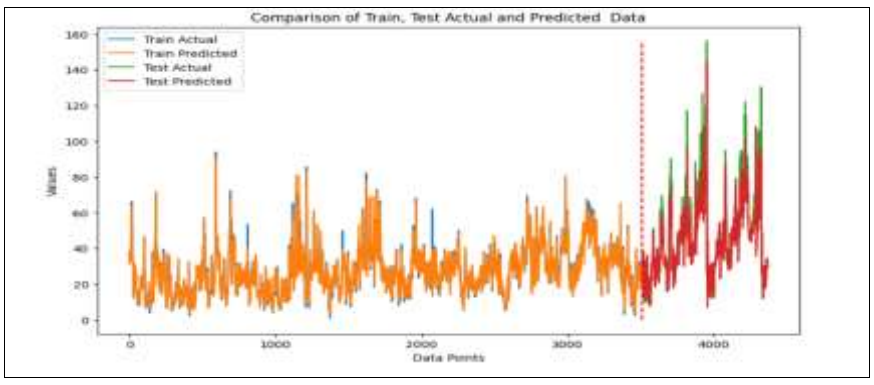


Figure 4.3.36: Comparison of Train, Test Actual and Predicted Data of LSTM Model 4

At last, LSTM model 4 performance was evaluated based on MSE, RMSE, MAE and R-Squared which is show on ***Table 4.3.8***.

Table 4.3.8: Evaluation of LSTM Model 4

	Training Data	**Testing Data**
MSE	9.99	113.46
RMSE	3.16	10.65
MAE	2.21	6.51
R-Squared	0.94	0.81

4.4 Key Finding From Implementation

Key findings of this paper from the first phase of implementation; IoT device to Google Cloud is that, even though Google Cloud discontinued its IoT core service recently, data can still be send from IoT device to Google Cloud by using MQTT protocol with the help of HiveMQ broker. The main idea behind the messaging in MQTT is publisher and subscriber model where there should be a central broker to process the message and these publisher and subscriber are unknown with each other. MQTT never uses addresses like email address, phone number, instead they use Topic to pass the message which was created by publisher. Another crucial point while creating a VM instance on Google Cloud, VM itself decide its own firewall services. So, to access a particular application that was hosted on Google VM, a new firewall services need to be setup. These firewall service can be made to access application publicly or can only be accessed through IP address which was setup in firewall rules. Another key finding is that every time when Google Cloud VM was turn off, it generates a new IP address to access the application that was hosted on VM which will make more difficulties to the end user to know the updated IP address. To overcome this issue a static IP address, need to be issued to a VM during its creation time.

In the second phase of Implementation, air quality data was cleaned, analysed, and used machine learning model (Linear Regression and Long Short-Term Memory) to predict accurate air quality index. From the exploratory data analysis, it can be stated that AQI values have directly impacted when any one harmful gas along with the particulate matter in the environment increase. From the seasonal patterns analysis of all variables as show in **Figure *4.3.14***, AQI level has been dramatically increase in the end of the year in Zhonghuamen city of China.

In the Linear Regression model four different machine learning methods were implemented. In LR model- 1, AQI was predicted by using single feature column which is PM2.5 where value of intercept was found as ‘26.340297259337067’ and value of coefficient was ‘1.04081298’. And when error term was visualising in graph, LR model 1 show that both training and testing data point fall inside -20 to 20 range on x-axis, which show that model prediction is far to actual values. When the model was evaluated in testing data it shows a value of (MSE as 132.22, RMSE as 11.49, MAE as 7.90, and R-squared as 0.79).

In LR model- 2, AQI was predicted by using two feature columns which is PM2.5 and PM10 where value of intercept was found as ‘15.780111766846716’ and value of coefficient was ‘[0.43999371, 0.46367613]’. And when error term was visualising in graph, LR model 2 show that both training and testing data point fall inside -10 to 10 range on x-axis, which show that model prediction is near to actual values. When the model was evaluated in testing data it shows a value of (MSE as 79.53, RMSE as 8.91, MAE as 5.21, and R-squared as 0.87).

In LR model- 3, AQI was predicted by using four feature columns which is NO2, O3, CO, and SO2 where value of intercept was found as ‘-6.101049643142872’ and value of coefficient was ‘[0.52839932, 0.17889162,10.62499604, 2.40649915]’. And when error term was visualising in graph, LR model 3 show that both training and testing data point fall inside -50 to 50 and -20 to 20 range on x-axis, which show that model prediction is too far from actual values. When the model was evaluated in testing data it shows a value of (MSE as 381.11, RMSE as 19.52, MAE as 13.71, and R-squared as 0.42).

In LR model- 4, AQI was predicted by using all feature columns which is PM2.5, PM10 NO2, O3, CO, and SO2 where value of intercept was found as ‘5.5155327552498505’ and value of coefficient was’ [0.47231332, 0.46027295, 0.07279498, 0.12498361, 0.45818509, -0.11613176]’. And when error term was visualising in graph, LR model 4 show that both training and testing data point fall inside zero range on x-axis, which show that model prediction is closer to actual values. When the model was evaluated in testing data it shows a value of (MSE as 55.27, RMSE as 7.43, MAE as 5.01, and R-squared as 0.91).

Table *4.4.1* show the evaluation metric for all four linear regression models on both training and testing dataset. From the table it can be concluded that a better AQI can be predicted from Linear regression model when all the particulate matter (PM) and gases were passed as a feature column. Also, AQI can predict more accurately when both particulate matter (PM2.5, and PM10) was passed as feature columns in linear regression than the gases (NO2, O3, CO, and SO2). At last, linear regression model predicts worst AQI when only gases were passed as feature columns.

Table 4.4.1: Evaluation of all four LR Model Performance

	Linear Regression							
	Model 1 (Predicting AQI with PM2.5)		**Model 2** (Predicting AQI with PM2.5, and PM10)		**Model 3** (Predicting AQI with NO2, O3, CO, and SO2)		**Model 4** (Predicting AQI with PM2.5, PM10, NO2, O3, CO, and SO2)	
	Training Data	Testing Data	Training Data	Testing Data	Training Data	Testing Data	Training Data	Testing Data
MSE	137.09	132.22	83.78	79.53	367.51	381.11	58.33	**55.27**
RMSE	11.71	11.49	9.51	8.91	19.17	19.52	7.63	**7.43**
MAE	8.18	7.90	5.30	5.21	13.35	13.71	5.09	**5.01**
R-Squared	0.79	0.79	0.87	0.87	0.45	0.42	0.91	**0.91**

In Long Short-Term Memory model four different machine learning method was implemented where in all four model timestamp(sequence) was set as 24 and model was predicting using 100 of neurons, activation as relu, dense layer as 1, optimizer as adma and at last model was fitted using 50 epochs, having a batch size of 1 and validation split as 0.2.

In LSTM model- 1, AQI was predicted by using univariate LSTM where AQI was predicted by using AQI itself. When the model was evaluated in testing data it shows a value of (MSE as 171.98, RMSE as 13.11, MAE as 8.16, and R-squared as 0.86).

In LSTM model- 2, AQI was predicted by using multivariate LSTM where AQI was predicted by using PM2.5, and PM10. When the model was evaluated in testing data it shows a value of (MSE as 62.09, RMSE as 7.87, MAE as 4.93, and R-squared as 0.90).

In LSTM model- 3, AQI was predicted by using multivariate LSTM where AQI was predicted by using NO2, O3, CO and SO2. When the model was evaluated in testing data it shows a value of (MSE as 150.88, RMSE as 12.28, MAE as 7.94, and R-squared as 0.88).

In LSTM model- 4, AQI was predicted by using multivariate LSTM where AQI was predicted by using PM2.5, PM10, NO2, O3, CO and SO2. When the model was evaluated in testing data it shows a value of (MSE as 113.46, RMSE as 10.65, MAE as 6.51, and R-squared as 0.81).

Table 4.4.2 show the evaluation metric for all four LSTM models on both training and testing dataset. From the table it can be concluded that AQI can predict accurately when both particulate matter (PM2.5, and PM10) was passed as a feature column. Also, a good AQI can be predicted from LSTM model when both particulate matter (PM2.5, and PM10) and gases (NO2, O3, CO, and SO2) was passed as a feature column.

Table 4.4.2: Evaluation of all four LSTM Model Performance

	LSTM							
	Model 1 (Predicting AQI with Univariate LSTM (AQI itself))		**Model 2** (Predicting AQI with Multivariate LSTM (PM2.5, and PM10))		**Model 3** (Predicting AQI with Multivariate LSTM (NO2, O3, CO, and SO2)		**Model 4** (Predicting AQI with Multivariate LSTM (PM2.5, PM10, NO2, O3, CO, and SO2))	
	Training Data	Testing Data	Training Data	Testing Data	Training Data	Testing Data	Training Data	Testing Data
MSE	33.63	171.98	14.24	**62.09**	25.05	150.88	9.99	113.46
RMSE	5.79	13.11	3.77	**7.87**	5.005	12.28	3.16	10.65
MAE	3.90	8.16	2.62	**4.93**	3.51	7.94	2.21	6.51
R-Squared	0.90	0.86	0.91	**0.90**	0.93	0.88	0.94	0.81

Chapter Five

Summary, Conclusion, Recommendations and Future Work

5.1 Summary and Conclusion

In conclusion, the purpose of dissertation was to examine the development of IoT-powered application (Node-Red) on Google Cloud to show the different visualisation of air quality data and implementation of machine learning approaches to predict better air quality index. In traditional methods of air quality monitoring prediction capabilities are often lacking and require expensive equipment's charges. Recent development of IoT technologies to capture the air quality data and support from Cloud based services like Google cloud, helps to predict the air quality index data much more accurately.

According to the World Health Organization (2022) air pollution is one the worst environmental health hazard causing an estimated seven million of premature death every year. This paper aims to address implementation of machine learning approaches to predict better air quality index by using unique features engineering which enables public health professionals, policymakers, and urban planner to carry out focus interventions and mitigation plans so that these number might be reduce in future.

The introduction of dissertation gives the general overview of IoT powered application for determining air quality index with the help of Google cloud ecosystem. From the literature review it was found that comparison of regression model with neural network model by selecting unique features engineering is limited while predicting air quality index. So, to overcome this issue, this paper aims to predict better air quality index by using different machine learning algorithms; one regression (Linear regression) and one neural network (Long Short-Term Memory) model by selecting unique features engineering.

Some methodological process was defined on how IoT application (node-red) can be deploy on Google Cloud to receive data from local server Node-Red by using MQTT protocol which is followed by machine learning (Linear Regression and Long Short-Term Memory) model to predict air quality data.

The research comprises two phases, in the initial phase, a Node-Red application is hosted on both local server and Google Cloud server, facilitating data transmission from the local server to the Google Cloud for visualisation, simulating IoT device functionality to inform public about the pollution level. Key finding from the first phase is that even though recently google discontinued it IoT Core Service, data can still be send from IoT devices to cloud by using MQTT protocol. The main idea behind the messaging in MQTT is publisher and subscriber model where there should be a central broker to process the message and these publisher and subscriber are unknown with each other. MQTT never uses addresses like email address, phone number, instead they use Topic to pass the message which was created by publisher.

In the subsequent phase, two machine learning models- Linear Regression and Long-Short Term Memory neural network were implemented, addressing distinct parameters for analysis. From the exploratory data analysis, it can be state that AQI values has directly impacted when any one harmful gas along with the particulate matter in the environment increase.

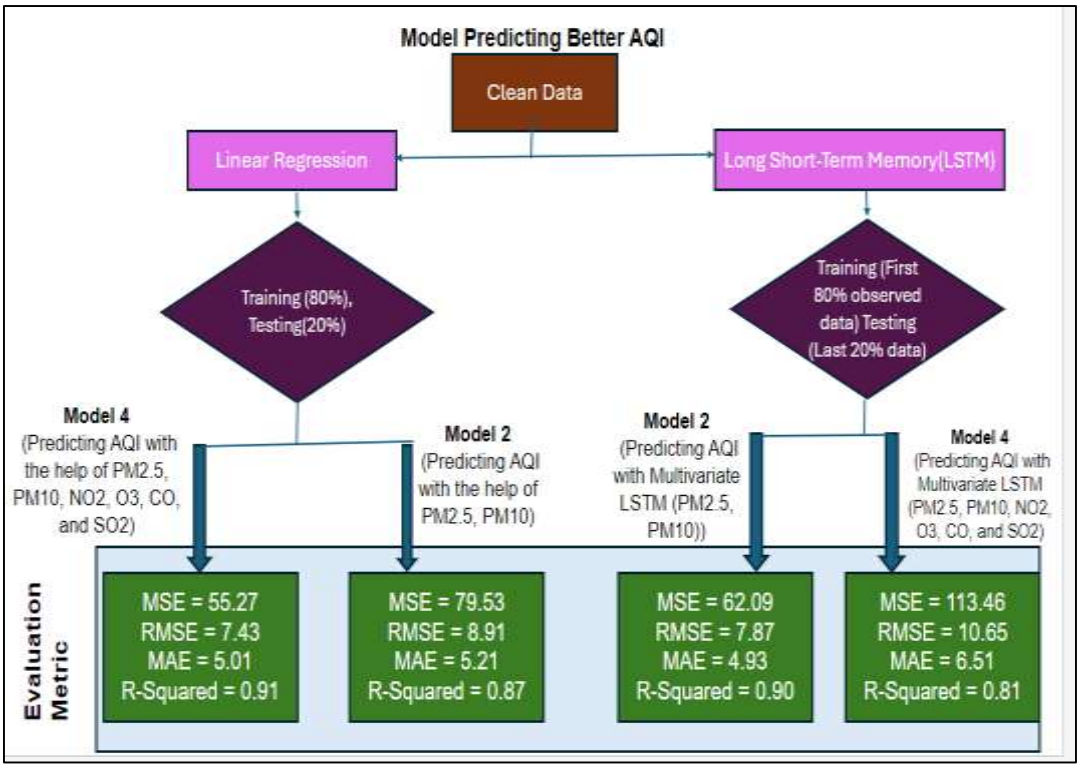


Figure 5.1.1: Best Performing Model for Predicting AQI

In linear regression four models were designed with different air quality feature selection, and all models were based on same training (80%) and testing (20%) data. The linear regression model gives more precise air quality prediction when all particulate matters (PM2.5, and PM10) and gases (NO2, O3, CO, and SO2) as a feature column were passed as show in ***Figure 5.1.1***. Evaluation metric for this model on testing data shows as (MSE as 55.27, RMSE as 7.43, MAE as 5.01, and R squared as 0.91). Whereas Linear Regression gives least predicted values when all gases (NO2, O3, CO, and SO2) were used as a feature column where evaluation metric for this model on testing data shows as (MSE as 381.11, RMSE as 19.52, MAE as 13.71, and R-squared as 0.42). But when model was used to predict only on particulate matter (PM2.5, PM10) as a feature column it gives the better result than the model that was used to predict only using gases feature. Evaluation metric for this model on testing data shows as (MSE as 79.53, RMSE as 8.91, MAE as 5.21, and R-squared as 0.87).

In LSTM four models were designed with different air quality feature selection, and all models were based on same training (first 80% of whole dataset) and testing (last 20% of whole dataset) data. Moreover, all four-model timestamp(sequence) was set as 24 and model was predicting using 100 of neurons, activation as relu, dense layer as 1, optimizer as adma and at last model was fitted using 50 epochs, having a batch size of 1 and validation split as 0.2. A better AQI can be predicted from LSTM model when both particulate matter (PM2.5, and PM10) was passed as a feature column where evaluation metric of this model show as (MSE as 62.09, RMSE as 7.87, MAE as 4.93, and R-squared as 0.90 on testing data as shown in ***Figure 5.1.1***. When all particulate matter (PM2.5, and PM10) along with the gases (NO2, O3, CO, and SO2) were selected, LSTM prediction was a little bit less. Whereas LSTM give the lowest value of evaluation metric when it was predicted using univariate LSTM that is passing AQI values as a feature column to predict the AQI values which evaluation metric on testing data as (MSE as 171.98, RMSE as 13.11, MAE as 8.16, and R-squared as 0.86).

In conclusion Linear Regression model, show a better prediction when all particulate matter (PM2.5 and PM10) and gases (NO2, O3, CO, and SO2) were passed as a feature column, whereas LSTM show a better prediction when only particulate matter (PM2.5 and PM10) were passed as a features column. In overall, this paper

underscore how unique features selection and different machine learning model predict the different air quality values.

5.2 Recommendations

This research intends to contribute to the body of knowledge on visualising raw data that was received from the IoT device in the Google Cloud with the help of Node-Red to inform the public about the current situation of environmental air pollution. Furthermore, it focuses on predicting air quality index with two machine learning models by selecting unique features of air quality parameters.

Apart from these valuable insights, there are few recommendations that might expend the applicability and importance of this research, some of them are:

a) **Collecting More Data**: Whole machine learning process was done by using six months of data, might be it gives more accurate prediction if the data volume was increased.
b) **Splitting Training and Testing Dataset in LSTM Model**: Further research can be done by splitting training and testing dataset in another format. As LSTM is time-series model, it learns from the sequence so while feeding the model if the train and test dataset split into a format of train: [1,2,3] test: [4], train: [5,6,7] test: [8], then it might give a better result.
c) **Adding More layer in LSTM Model**: During LSTM model implementation, model was made using 100 of neurons, activation as relu, dense layer as 1, optimizer as adma and last model was fitted using 50 epochs, having a batch size of 1 and validation split as 0.2. This model can be implemented by adding more layers such as dense layer, Convolutional Layer and so.
d) **Creating different Sequence**: In this paper sequence of timestamps for LSTM model was made as 24 which make a sequence of data from last 24 rows as the dataset was recorded in hourly basis, so we made a sequence of data for one day. But changing sequence data might give better results for LSTM models.

5.3 Future Work

Beside the research contribution of this paper as state in section 1.6 there are number future work that might expend the applicability and importance of this research, some of them are:

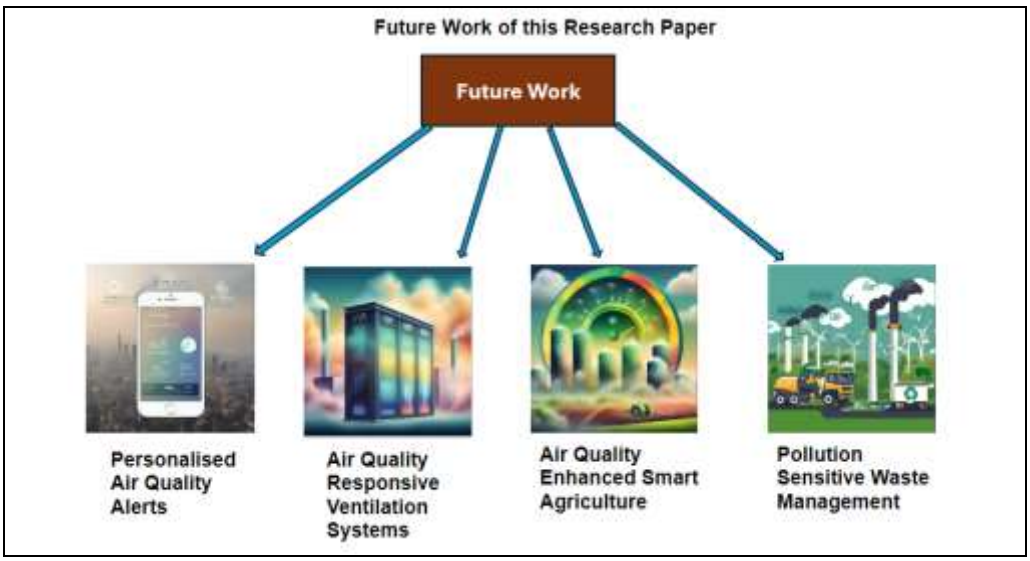


Figure 5.3.1: Future Work of this Research Paper

a) **Personalised Air Quality Alerts**: Developing a mobile application that works with wearable technology to deliver suggestions and alerts about the quality of the air that are specific to each user based on their activity levels, location history, and personal health profiles. This enables user information they need to know what to do outside and how to modify their behaviour to reduce their exposure to pollution.
b) **Air Quality Responsive Ventilation Systems**: Creating a ventilation system that is sensitive to changes in air quality by integrating these models with building management systems. These systems will automatically modify airflow and filtration rate in response to predicted level of pollution.
c) **Air Quality Enhanced Smart Agriculture**: Smart agricultural systems can be benefited from the integration of these air quality prediction model, which can optimise crop choice, planting schedules, and irrigation technique by taking pollution level into account.
d) **Pollution Sensitive Waste Management**: Implementing garbage collection systems that modify pickup times, routes, and destroying time in response to prediction of air quality index by these models.

References

Aditya, C.R., Deshmukh, C.R., Nayana, D.K. and Vidyavastu, P.G. (2018) 'Detection and prediction of air pollution using machine learning models', *International journal of engineering trends and technology (IJETT),* 59(4), pp. 204-207.

Biswas, A.R. and Giaffreda, R. (2014) 'IoT and cloud convergence: Opportunities and challenges', *2014 IEEE World Forum on Internet of Things (WF-IoT),* pp. 375-376.

Docker (2024) *Use the Docker command line.* Available at: https://docs.docker.com/engine/reference/commandline/cli/ (Accessed: 18/04/2024).

Du, W., Chen, L., Wang, H., Shan, Z., Zhou, Z., Li, W. and Wang, Y. (2023) 'Deciphering urban traffic impacts on air quality by deep learning and emission inventory', *Journal of Environmental Sciences,* 124, pp. 745-757.

Estoque, R.C. (2020) 'A review of the sustainability concept and the state of SDG monitoring using remote sensing', *Remote Sensing,* 12(11), pp. 1770.

Fang, M., Chan, C.K. and Yao, X. (2009) 'Managing air quality in a rapidly developing nation: China', *Atmospheric Environment,* 43(1), pp. 79-86.

Golec, M., Ozturac, R., Pooranian, Z., Gill, S.S. and Buyya, R. (2021) 'IFaaSBus: A security-and privacy-based lightweight framework for serverless computing using IoT and machine learning', *IEEE Transactions on Industrial Informatics,* 18(5), pp. 3522-3529.

Gupta, D., Bhatt, S., Gupta, M., Kayode, O. and Tosun, A.S. (2020) 'Access control model for google cloud iot', *2020 IEEE 6th Intl conference on big data security on cloud (BigDataSecurity), IEEE Intl conference on high performance and smart computing,(HPSC) and IEEE Intl conference on intelligent data and security (IDS),* pp. 198-208.

Gupta, M., Bhatt, S., Alshehri, A.H. and Sandhu, R. (2022) 'Access control models in cloud iot services'*Access Control Models and Architectures For IoT and Cyber Physical Systems* Springer, pp. 63-96.

Hashmy, Y., Khan, Z.U., Ilyas, F., Hafiz, R., Younis, U. and Tauqeer, T. (2023) 'Modular air quality calibration and forecasting method for low-cost sensor nodes', *IEEE Sensors Journal,* 23(4), pp. 4193-4203.

Hochreiter, S. and Schmidhuber, J. (1997) 'Long short-term memory', *Neural computation,* 9(8), pp. 1735-1780.

Horn, S.A. and Dasgupta, P.K. (2023) 'The Air Quality Index (AQI) in historical and analytical perspective a tutorial review', *Talanta,* , pp. 125260.

Hossain, E., Shariff, M.A.U., Hossain, M.S. and Andersson, K. (2020) *A novel deep learning approach to predict air quality index.* Proceedings of International Conference on Trends in Computational and Cognitive Engineering: Proceedings of TCCE 2020, pp. 367-381.

Islam, R., Patamsetti, V., Gadhi, A., Gondu, R.M., Bandaru, C.M., Kesani, S.C. and Abiona, O. (2023) 'The future of cloud computing: benefits and challenges', *International Journal of Communications, Network and System Sciences,* 16(4), pp. 53-65.

Janarthanan, R., Partheeban, P., Somasundaram, K. and Elamparithi, P.N. (2021) 'A deep learning approach for prediction of air quality index in a metropolitan city', *Sustainable Cities and Society,* 67, pp. 102720.

Jephcote, C., Hansell, A.L., Adams, K. and Gulliver, J. (2021) 'Changes in air quality during COVID-19 ‘lockdown’in the United Kingdom', *Environmental Pollution,* 272, pp. 116011.

Karavas, Z., Karayannis, V. and Moustakas, K. (2021) 'Comparative study of air quality indices in the European Union towards adopting a common air quality index', *Energy & Environment,* 32(6), pp. 959-980.

Kaushik, P., Rao, A.M., Singh, D.P., Vashisht, S. and Gupta, S. (2021) *Cloud computing and comparison based on service and performance between amazon*

AWS, microsoft azure, and google cloud. 2021 International Conference on Technological Advancements and Innovations (ICTAI), pp. 268-273.

Khushalani, P. (2022) 'What Is Cloud Computing?'*Kubernetes Application Developer: Develop Microservices and Design a Software Solution on the Cloud* Springer, pp. 1-20.

Kumari, S. and Jain, M.K. (2018) 'A critical review on air quality index', *Environmental Pollution: Select Proceedings of ICWEES-2016,* pp. 87-102.

Lemeš, S. (2018) *Air quality index (AQI)—comparative study and assesment of an appropriate model for B&H.* 2th Scientific/Research Symposium with International Participation 'Metallic And Nonmetallic Materials, pp. 282-291.

Li, R., Cui, L., Liang, J., Zhao, Y., Zhang, Z. and Fu, H. (2020) 'Estimating historical SO2 level across the whole China during 1973–2014 using random forest model', *Chemosphere,* 247, pp. 125839.

Liu, H., Li, Q., Yu, D. and Gu, Y. (2019) 'Air quality index and air pollutant concentration prediction based on machine learning algorithms', *Applied sciences,* 9(19), pp. 4069.

Lorenzo, J.S.L., San Tam, W.W. and Seow, W.J. (2021) 'Association between air quality, meteorological factors and COVID-19 infection case numbers', *Environmental research,* 197, pp. 111024.

Martínez, N.M., Montes, L.M., Mura, I. and Franco, J.F. (2018) *Machine learning techniques for PM 10 levels forecast in bogotá.* 2018 ICAI Workshops (ICAIW), pp. 1-7.

Mathur, P. (2024) 'Cloud Computing Infrastructure, Platforms, and Software for Scientific Research', *High Performance Computing in Biomimetics: Modeling, Architecture and Applications*, pp. 89-127.

Montrucchio, B., Giusto, E., Vakili, M.G., Quer, S., Ferrero, R. and Fornaro, C. (2020) 'A densely-deployed, high sampling rate, open-source air pollution monitoring WSN', *IEEE Transactions on Vehicular Technology,* 69(12), pp. 15786-15799.

Nandini, K. and Fathima, G. (2019) *Urban air quality analysis and prediction using machine learning.* 2019 1st International Conference on Advanced Technologies in

Intelligent Control, Environment, Computing & Communication Engineering (ICATIECE), pp. 98-102.

Pasupuleti, V.R., Kalyan, P. and Reddy, H.K. (2020) *Air quality prediction of data log by machine learning.* 2020 6th International Conference on Advanced Computing and Communication Systems (ICACCS), pp. 1395-1399.

Rashid, A. and Chaturvedi, A. (2019) 'Cloud computing characteristics and services: a brief review', *International Journal of Computer Sciences and Engineering,* 7(2), pp. 421-426.

Rochwerger, B., Breitgand, D., Levy, E., Galis, A., Nagin, K., Llorente, I.M., Montero, R., Wolfsthal, Y., Elmroth, E. and Caceres, J. (2009) 'The reservoir model and architecture for open federated cloud computing', *IBM Journal of Research and Development,* 53(4), pp. 4: 1-4: 11.

Sakila, V.S. and Manohar, S. (2024) 'Real-time air quality monitoring in Bull Trench Kiln-based Brick industry by calibrating sensor readings and utilizing the Serverless Computing', *Expert Systems with Applications,* 237, pp. 121397.

Santoso, L.W. (2019) *Cloud technology: Opportunities for cybercriminals and security challenges.* 2019 Twelfth International Conference on Ubi-Media Computing (Ubi-Media), pp. 18-23.

Shakir, M. and Rakesh, N. (2018) *Investigation on air pollutant data sets using data mining tool.* 2018 2nd International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud)(I-SMAC) I-SMAC (IoT in Social, Mobile, Analytics and Cloud)(I-SMAC), 2018 2nd International Conference on, pp. 480-485.

Shakoor, A., Chen, X., Farooq, T.H., Shahzad, U., Ashraf, F., Rehman, A., Sahar, N.E. and Yan, W. (2020) 'Fluctuations in environmental pollutants and air quality during the lockdown in the USA and China: two sides of COVID-19 pandemic', *Air Quality, Atmosphere & Health,* 13, pp. 1335-1342.

shao, m., chen, x., li, y. and xu, x. (2022) 'Hourly Air Pollution Data in Nine monitoring stations in Nanjing', V1(Mendeley Data) Available at: 10.17632/kvgwcrbjm3.1 .

Tyagi, H. and Kumar, R. (2020) 'Cloud computing for iot', *Internet of Things (IoT) Concepts and Applications,* pp. 25-41.

Wang, S., Hu, Y., Burgués, J., Marco, S. and Liu, S. (2020) *Prediction of gas concentration using gated recurrent neural networks.* 2020 2nd IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS), pp. 178-182.

Weisberg, S. (2005) *Applied linear regression* John Wiley & Sons.

World Heath Organiztion (2022) *Ambient (outdoor) air pollution.* Available at: https://www.who.int/news-room/fact-sheets/detail/ambient-(outdoor)-air-quality-and-health (Accessed: 15/04/2024).

Wu, C., Song, R. and Peng, Z. (2022) 'Prediction of air pollutants on roadside of the elevated roads with combination of pollutants periodicity and deep learning method', *Building and Environment,* 207, pp. 108436.

Xayasouk, T., Lee, H. and Lee, G. (2020) 'Air pollution prediction using long short-term memory (LSTM) and deep autoencoder (DAE) models', *Sustainability,* 12(6), pp. 2570.

Yang, Y., Zheng, Z., Bian, K., Jiang, Y., Song, L. and Han, Z. (2017) 'Arms: A fine-grained 3D AQI realtime monitoring system by UAV', *GLOBECOM 2017-2017 IEEE Global Communications Conference,* pp. 1-6.

Yin, K. (2020) 'Cloud computing: Concept, model, and key technologies', *ZTE Communications,* 8(4), pp. 21-26.

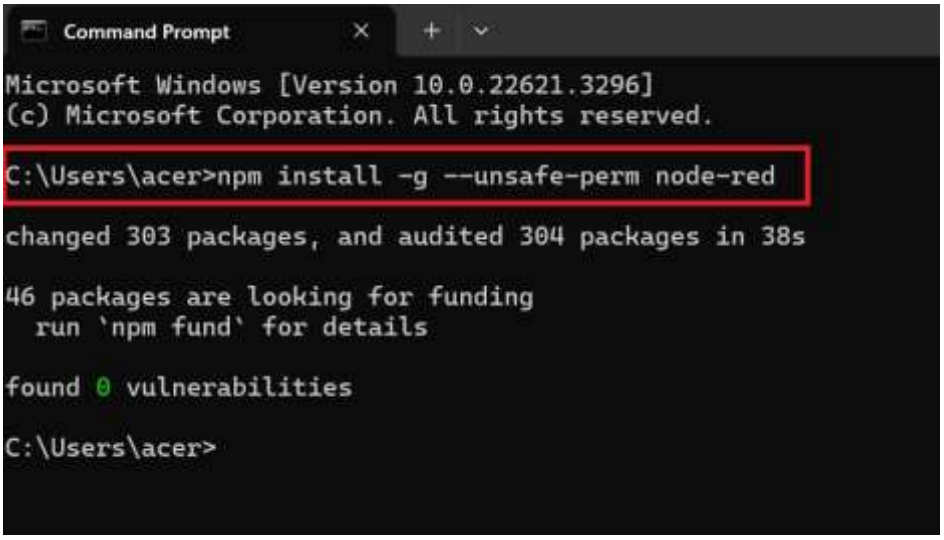
Appendix

Link to the all the files that was store in Glasgow Caledonian University Drive

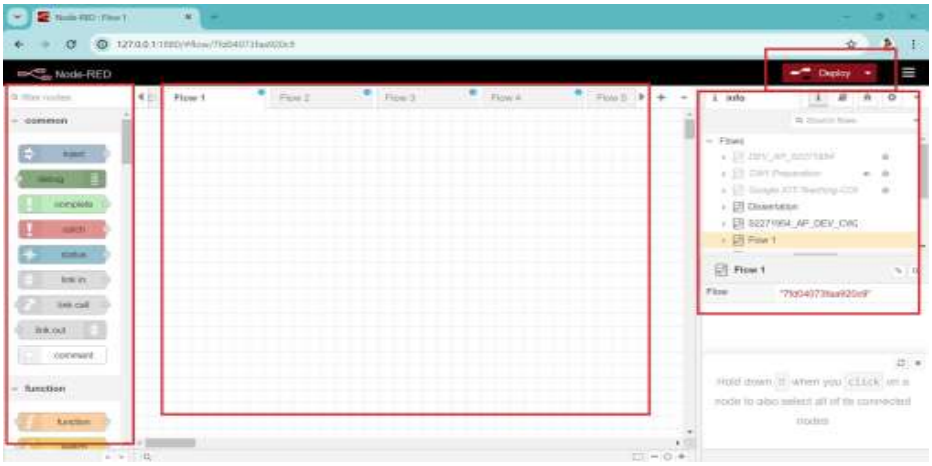
https://caledonianac-my.sharepoint.com/:f:/r/personal/apaude300_caledonian_ac_uk/Documents/All%20Code?csf=1&web=1&e=Ld9FvK

APPENDIX 1: Setting Up Node-Red

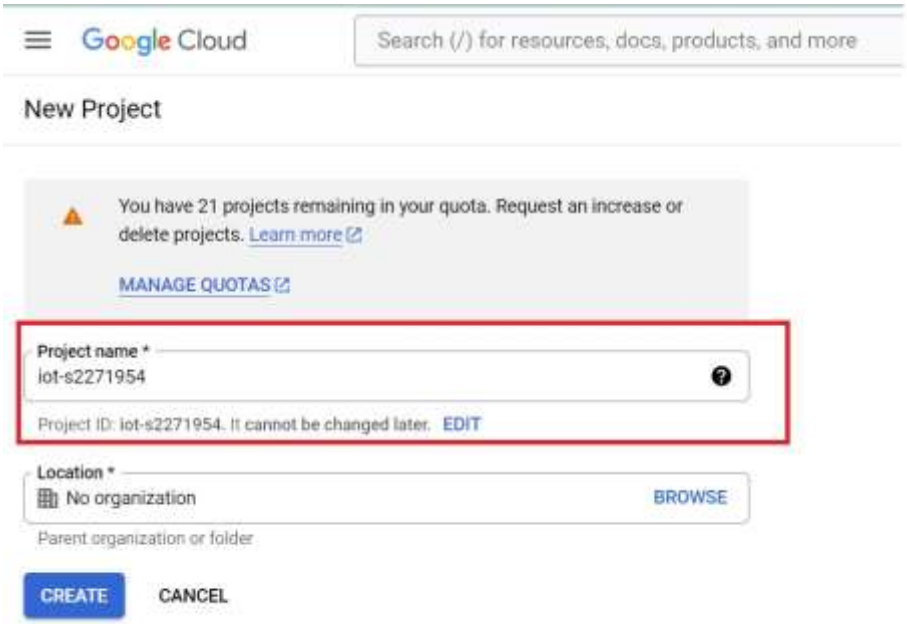
Node-Red Diagram



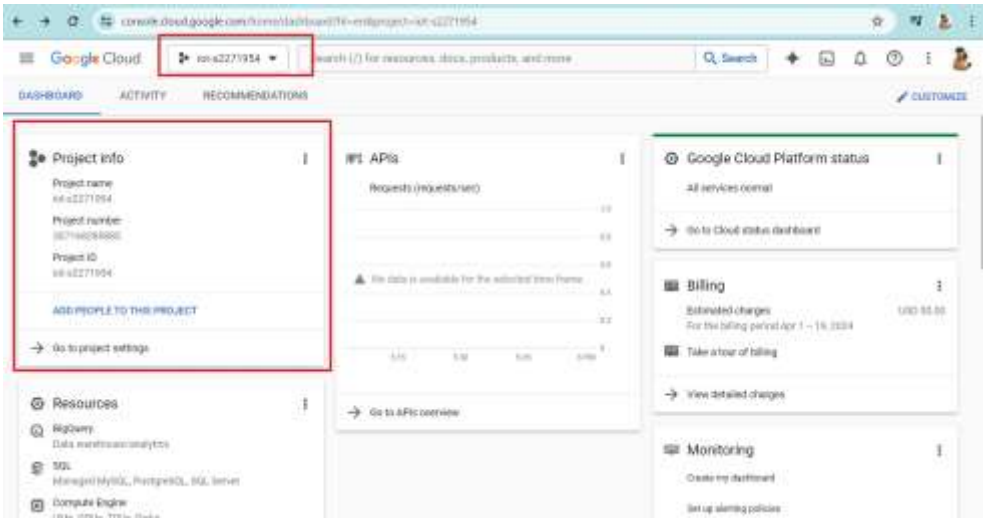
Appendix Figure 1: Installation of Node-Red in Local PC



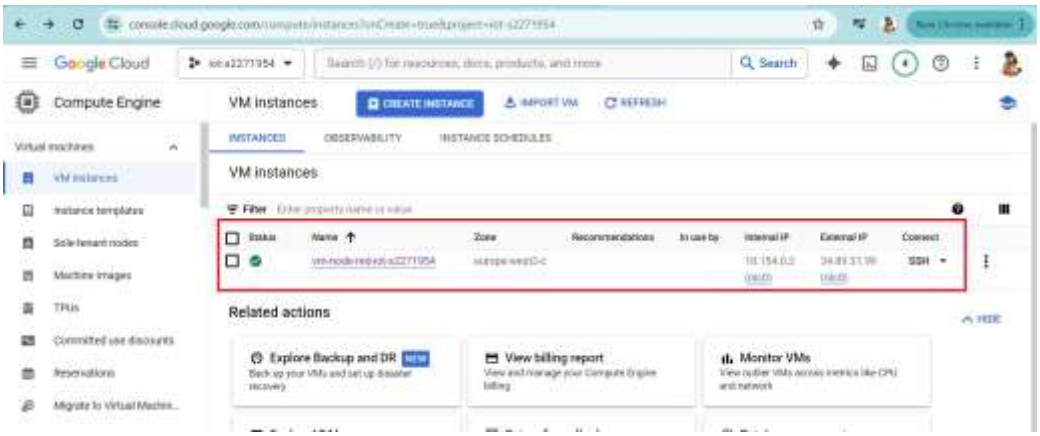
Appendix Figure 2: Node-Red Architecture on Local PC



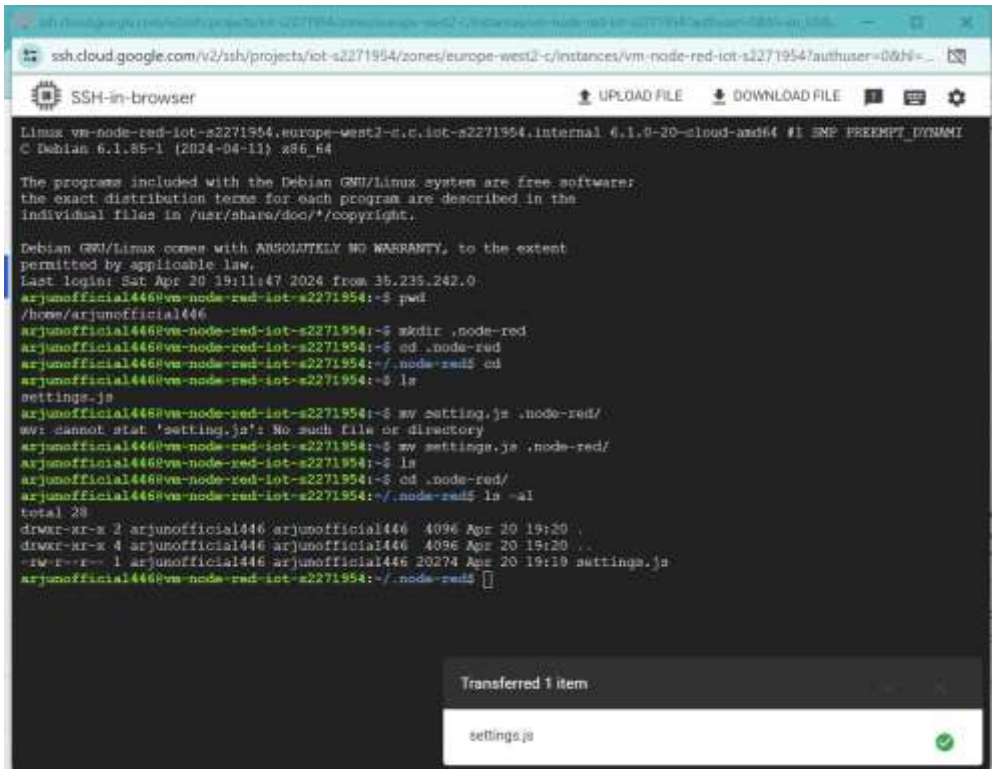
Appendix Figure 3: Creating a Project on Google Cloud



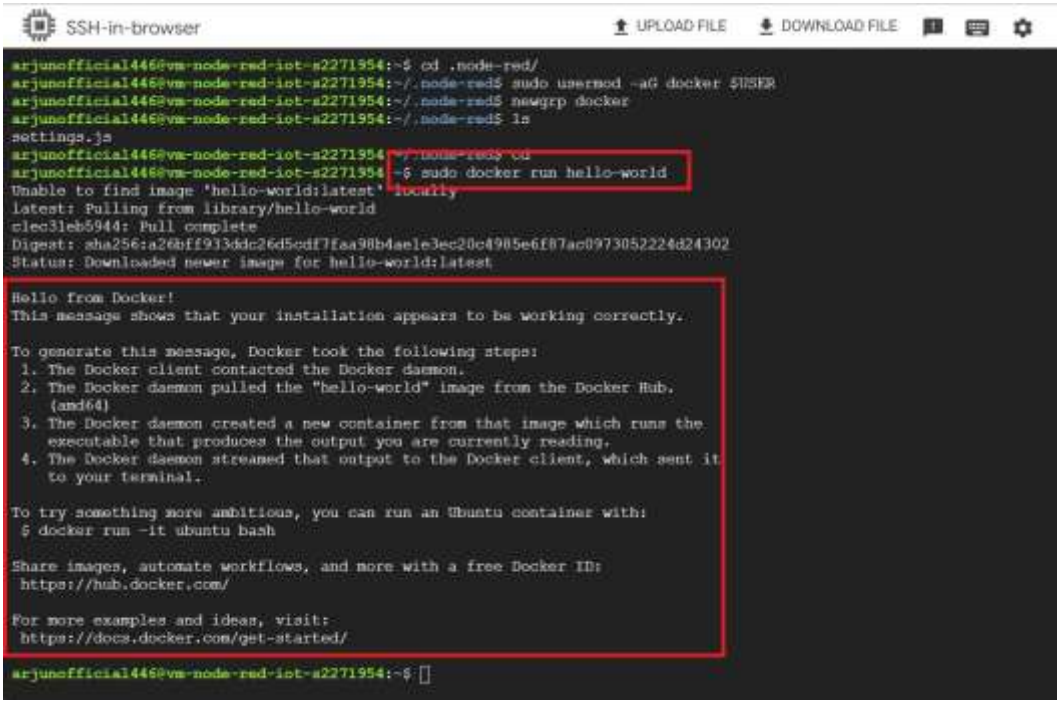
Appendix Figure 4: Google Cloud Project Dashboard



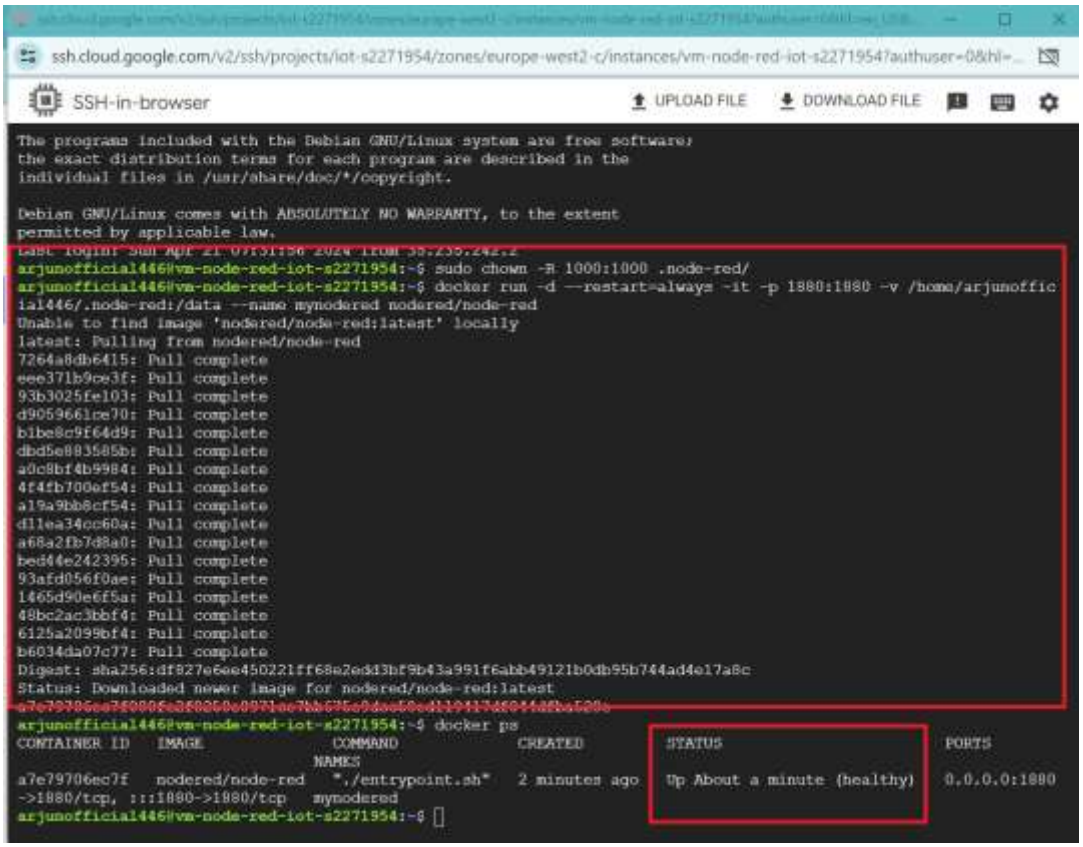
Appendix Figure 5: Creation of Virtual Machine in Google Cloud to Host Node-Red



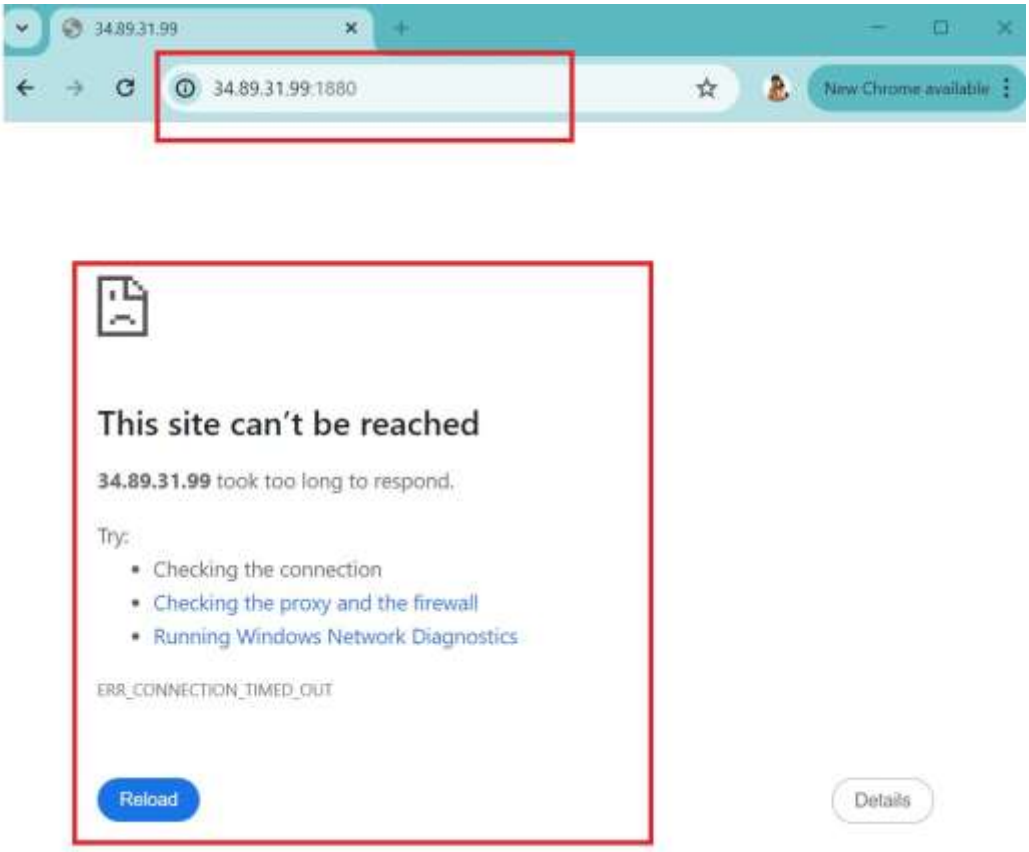
Appendix Figure 6: Uploading a Setting Files to Support Node-Red



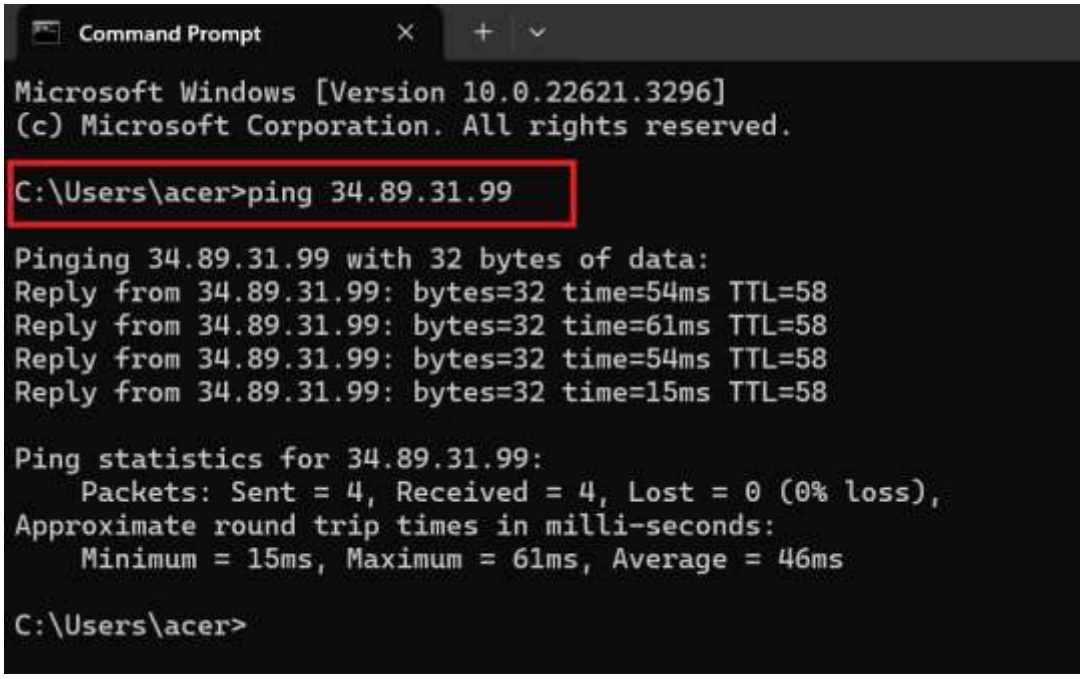
Appendix Figure 7: Successful installation of Docker



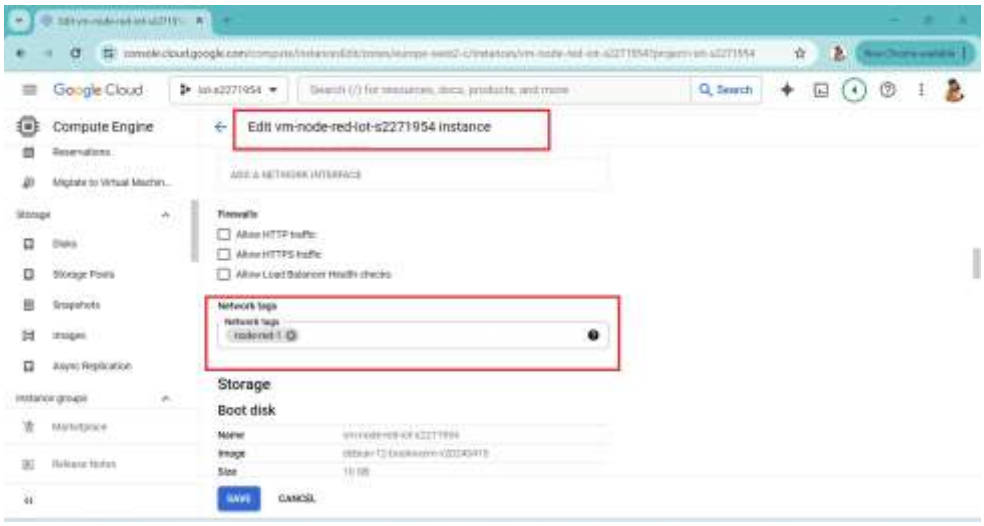
Appendix Figure 8: Successful creation of Docker Container with Docker Node-Red Image



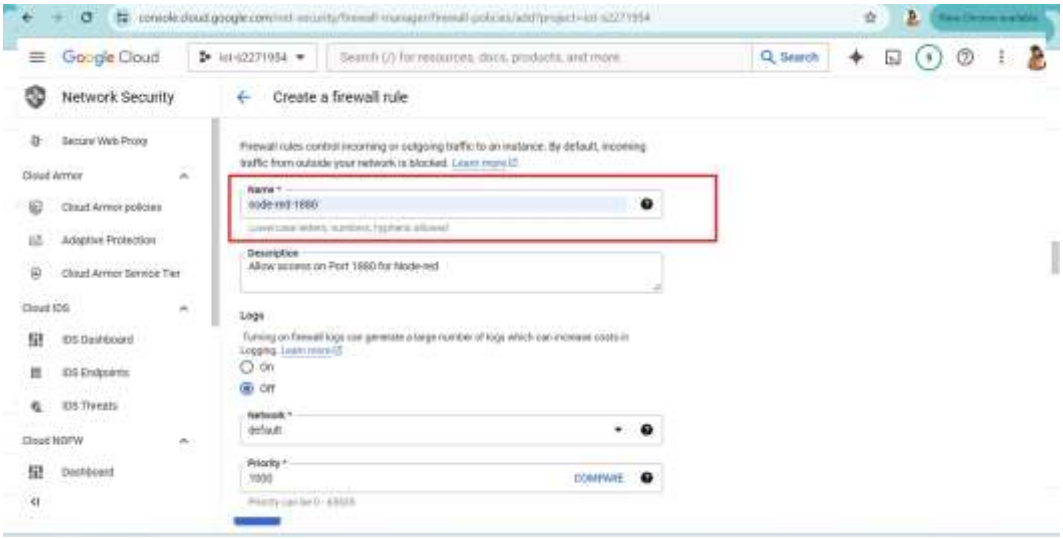
Appendix Figure 9: External IP with port 1880 not Responding in Browser



Appendix Figure 10: External IP Pinging from Local Device



Appendix Figure 11: Assigning Network Tags in VM Instance



Appendix Figure 12: Creation of New Firewall Rule

Appendix Figure 13: Creation of Static IP Address

Local PC Node-Red Flow Code


```
[
    {
        "id": "8aa436dcdc93598b",
        "type": "tab",
        "label": "Dissertation Node Red Local PC",
        "disabled": false,
        "info": "",
        "env": []
    },
    {
        "id": "2fe96b4324e638c6",
        "type": "inject",
        "z": "8aa436dcdc93598b",
        "name": "Read",
        "props": [
            {
                "p": "payload"
            },
            {
                "p": "topic",
                "vt": "str"
            }
        ],
        "repeat": "",
        "crontab": "",
        "once": false,
        "onceDelay": "",
        "topic": "",
        "payload": "",
        "payloadType": "str",
        "x": 110,
        "y": 60,
        "wires": [
            [
                "d8f2394c7f207349"
            ]
        ]
    },
    {
        "id": "d8f2394c7f207349",
        "type": "file in",
        "z": "8aa436dcdc93598b",
        "name": "Read a spreadsheet file ",
```

```
        "filename": "D:\\MSc Big Data Technologies\\Dissertation
\\kvgwcrbjm3-1\\kvgwcrbjm3-1\\pollutant\\pollutant\\ZHM.csv",
        "filenameType": "str",
        "format": "",
        "chunk": false,
        "sendError": false,
        "allProps": false,
        "x": 210,
        "y": 120,
        "wires": [
            [
                "7ea86178751666a4"
            ]
        ]
    },
    {
        "id": "7ea86178751666a4",
        "type": "book",
        "z": "8aa436dcdc93598b",
        "name": "Handle Spreadsheet Files as Book",
        "raw": false,
        "x": 200,
        "y": 200,
        "wires": [
            [
                "af99aa9187ed8f02"
            ]
        ]
    },
    {
        "id": "af99aa9187ed8f02",
        "type": "sheet",
        "z": "8aa436dcdc93598b",
        "name": "Sheet With in Excel Workbook",
        "sheetName": "Sheet1",
        "x": 170,
        "y": 260,
        "wires": [
            [
                "82513e1e8eb20fd0"
            ]
        ]
    },
```

```
    {
        "id": "82513e1e8eb20fd0",
        "type": "sheet-to-json",
        "z": "8aa436dcdc93598b",
        "name": "Convert Sheet Data to JSON Format",
        "raw": "false",
        "range": "A1:I35041",
        "header": "default",
        "blankrows": false,
        "x": 190,
        "y": 320,
        "wires": [
            [
                "faf80ba4090503f3",
                "1768848c051bafdd"
            ]
        ]
    },
    {
        "id": "1768848c051bafdd",
        "type": "debug",
        "z": "8aa436dcdc93598b",
        "name": "debug sheet",
        "active": true,
        "tosidebar": true,
        "console": false,
        "tostatus": false,
        "complete": "payload",
        "targetType": "msg",
        "statusVal": "",
        "statusType": "auto",
        "x": 510,
        "y": 300,
        "wires": []
    },
    {
        "id": "faf80ba4090503f3",
        "type": "split",
        "z": "8aa436dcdc93598b",
        "name": "Split Message in Sequence of Message",
        "splt": "\\n",
        "spltType": "str",
        "arraySplt": "1",
        "arraySpltType": "len",
```

```
        "stream": false,
        "addname": "",
        "x": 400,
        "y": 400,
        "wires": [
            [
                "0bd7eb41a57472ae"
            ]
        ]
    },
    {
        "id": "acff4d44afa9e1d7",
        "type": "inject",
        "z": "8aa436dcdc93598b",
        "name": "RESET BUFFER",
        "props": [
            {
                "p": "reset",
                "v": "true",
                "vt": "bool"
            },
            {
                "p": "topic",
                "vt": "str"
            }
        ],
        "repeat": "",
        "crontab": "",
        "once": false,
        "onceDelay": 0.1,
        "topic": "",
        "x": 140,
        "y": 500,
        "wires": [
            [
                "0bd7eb41a57472ae"
            ]
        ]
    },
    {
        "id": "0bd7eb41a57472ae",
        "type": "delay",
        "z": "8aa436dcdc93598b",
        "name": "",
```

```
        "pauseType": "rate",
        "timeout": "2",
        "timeoutUnits": "seconds",
        "rate": "1",
        "nbRateUnits": "5",
        "rateUnits": "second",
        "randomFirst": "1",
        "randomLast": "5",
        "randomUnits": "seconds",
        "drop": false,
        "allowrate": false,
        "outputs": 1,
        "x": 360,
        "y": 500,
        "wires": [
            [
                "ce975de0fd53dbd8",
                "f3b2a17c80e95d72"
            ]
        ]
    },
    {
        "id": "ce975de0fd53dbd8",
        "type": "debug",
        "z": "8aa436dcdc93598b",
        "name": "Raw Data",
        "active": false,
        "tosidebar": true,
        "console": false,
        "tostatus": false,
        "complete": "payload",
        "targetType": "msg",
        "statusVal": "",
        "statusType": "auto",
        "x": 440,
        "y": 600,
        "wires": []
    },
    {
        "id": "f3b2a17c80e95d72",
        "type": "change",
        "z": "8aa436dcdc93598b",
        "name": "Convert Value to Number",
```

```
"rules": [
    {
        "t": "set",
        "p": "payload.PM2_5",
        "pt": "msg",
        "to": "$number(msg.payload.PM2_5)\t",
        "tot": "jsonata"
    },
    {
        "t": "set",
        "p": "payload.PM10",
        "pt": "msg",
        "to": "$number(msg.payload.PM10)\t",
        "tot": "jsonata"
    },
    {
        "t": "set",
        "p": "payload.NO2",
        "pt": "msg",
        "to": "$number(msg.payload.NO2)",
        "tot": "jsonata"
    },
    {
        "t": "set",
        "p": "payload.O3",
        "pt": "msg",
        "to": "$number(msg.payload.O3)\t",
        "tot": "jsonata"
    },
    {
        "t": "set",
        "p": "payload.CO",
        "pt": "msg",
        "to": "$number(msg.payload.CO)",
        "tot": "jsonata"
    },
    {
        "t": "set",
        "p": "payload.SO2",
        "pt": "msg",
        "to": "$number(msg.payload.SO2)",
        "tot": "jsonata"
    },
    {
```

```
            {
                "t": "set",
                "p": "payload.AOI",
                "pt": "msg",
                "to": "$number(msg.payload.AOI)",
                "tot": "jsonata"
            }
        ],
        "action": "",
        "property": "",
        "from": "",
        "to": "",
        "reg": false,
        "x": 630,
        "y": 500,
        "wires": [
            [
                "7b911bab7a748422",
                "65da5ede557e89e6"
            ]
        ]
    },
    {
        "id": "7b911bab7a748422",
        "type": "debug",
        "z": "8aa436dcdc93598b",
        "name": "Final Result",
        "active": true,
        "tosidebar": true,
        "console": false,
        "tostatus": false,
        "complete": "payload",
        "targetType": "msg",
        "statusVal": "",
        "statusType": "auto",
        "x": 690,
        "y": 580,
        "wires": []
    },
    {
        "id": "65da5ede557e89e6",
        "type": "mqtt out",
        "z": "8aa436dcdc93598b",
```

```
        "name": "Hive Server For Publishing Message ",
        "topic": "pub99932/dev-node3213/telemetry",
        "qos": "2",
        "retain": "",
        "respTopic": "",
        "contentType": "",
        "userProps": "",
        "correl": "",
        "expiry": "",
        "broker": "1c9dc0f62cd53768",
        "x": 830,
        "y": 400,
        "wires": []
    },
    {
        "id": "1c9dc0f62cd53768",
        "type": "mqtt-broker",
        "name": "Hive",
        "broker": "broker.hivemq.com",
        "port": "1883",
        "clientid": "",
        "autoConnect": true,
        "usetls": false,
        "protocolVersion": "4",
        "keepalive": "60",
        "cleansession": true,
        "autoUnsubscribe": true,
        "birthTopic": "",
        "birthQos": "0",
        "birthPayload": "",
        "birthMsg": {},
        "closeTopic": "",
        "closeQos": "0",
        "closePayload": "",
        "closeMsg": {},
        "willTopic": "",
        "willQos": "0",
        "willPayload": "",
        "willMsg": {},
        "userProps": "",
        "sessionExpiry": ""
    }
]
```

Google Cloud Node-Red Flow Code


```
[
    {
        "id": "e07fb1e2ac3479cf",
        "type": "tab",
        "label": "Dissertation_Node_Red_Google_Cloud",
        "disabled": false,
        "info": "",
        "env": []
    },
    {
        "id": "ebe476c6db3057c7",
        "type": "mqtt in",
        "z": "e07fb1e2ac3479cf",
        "name": "Hive Server For Receiving Message ",
        "topic": "pub99932/dev-node3213/telemetry",
        "qos": "2",
        "datatype": "auto-detect",
        "broker": "0f1c470067804c4b",
        "nl": false,
        "rap": true,
        "rh": 0,
        "inputs": 0,
        "x": 220,
        "y": 1180,
        "wires": [
            [
                "9edbda5058a428d0",
                "5c33a6c88de2d0bb",
                "61d30df3dee28cb3",
                "0a5eaacc6e245af2",
                "235b95637b8eba14",
                "2b9317665ed72200",
                "ef6950b7be5054e4",
                "e3baeff913c962e0",
                "4f4b5e2c328be351",
                "7491e55c2c304572",
                "42fbdcd65fb95b15",
                "64d3799dae0caec5",
                "9273b7848d0a4e37",
                "1b7954064aead7ee",
                "5f617d337d36b209"
            ]
        ]
    },
    {
        "id": "4001e1e339a9b330",
        "type": "ui_chart",
        "z": "e07fb1e2ac3479cf",
        "name": "",
        "group": "13bcf5a28cd15de8",
        "order": 2,
        "width": 0,
        "height": 0,
```

```
        "label": "Line Chart of PM2_5",
        "chartType": "line",
        "legend": "false",
        "xformat": "auto",
        "interpolate": "linear",
        "nodata": "",
        "dot": false,
        "ymin": "",
        "ymax": "",
        "removeOlder": "1",
        "removeOlderPoints": "",
        "removeOlderUnit": "3600",
        "cutout": 0,
        "useOneColor": false,
        "useUTC": false,
        "colors": [
            "#fe4406",
            "#aec7e8",
            "#ff7f0e",
            "#2ca02c",
            "#98df8a",
            "#df2626",
            "#ff9896",
            "#9467bd",
            "#c5b0d5"
        ],
        "outputs": 1,
        "useDifferentColor": false,
        "x": 900,
        "y": 340,
        "wires": [
            []
        ]
    },
    {
        "id": "97c9faf403b13435",
        "type": "ui_chart",
        "z": "e07fb1e2ac3479cf",
        "name": "",
        "group": "b86cf1815e4e9ffc",
        "order": 1,
        "width": 0,
        "height": 0,
        "label": "Comparison of AQI With PM2_5",
        "chartType": "line",
        "legend": "true",
        "xformat": "HH:mm:ss",
        "interpolate": "linear",
        "nodata": "",
        "dot": false,
        "ymin": "",
        "ymax": "",
        "removeOlder": 1,
```

```
        "removeOlderPoints": "",
        "removeOlderUnit": "3600",
        "cutout": 0,
        "useOneColor": false,
        "useUTC": false,
        "colors": [
            "#1f77b4",
            "#aec7e8",
            "#ff7f0e",
            "#2ca02c",
            "#98df8a",
            "#d62728",
            "#ff9896",
            "#9467bd",
            "#c5b0d5"
        ],
        "outputs": 1,
        "useDifferentColor": false,
        "x": 1030,
        "y": 1000,
        "wires": [
            []
        ]
    },
    {
        "id": "5c33a6c88de2d0bb",
        "type": "function",
        "z": "e07fb1e2ac3479cf",
        "name": "Data For PM2_5",
        "func": "\nPM2_5 = msg.payload.PM2_5;\nmsg = { topic: 'PM2_5',
        "outputs": 1,
        "timeout": 0,
        "noerr": 0,
        "initialize": "",
        "finalize": "",
        "libs": [],
        "x": 580,
        "y": 340,
        "wires": [
            [
                "4001e1e339a9b330",
                "2a8d38d0d4035ad0"
            ]
        ]
    },
    {
        "id": "2a8d38d0d4035ad0",
        "type": "ui_text",
        "z": "e07fb1e2ac3479cf",
        "group": "13bcf5a28cd15de8",
        "order": 1,
        "width": 0,
        "height": 0,
        "name": "",
```

```
        "label": "Current Value of PM2_5",
        "format": "{{msg.payload}}",
        "layout": "row-spread",
        "x": 970,
        "y": 380,
        "wires": []
    },
    {
        "id": "9edbda5058a428d0",
        "type": "function",
        "z": "e07fb1e2ac3479cf",
        "name": "Data for Comparison of AQI With PM2_5",
        "func": "\nValueToLine1 = msg.payload.AQI;\nValueToLine2 =
{ topic: 'AQI', payload: ValueToLine1 };\nline2 = { topic: 'PM2_5',
        "outputs": 2,
        "timeout": 0,
        "noerr": 0,
        "initialize": "",
        "finalize": "",
        "libs": [],
        "x": 660,
        "y": 1000,
        "wires": [
            [
                "97c9faf403b13435"
            ],
            [
                "97c9faf403b13435"
            ]
        ]
    },
    {
        "id": "2b2304734194c1ef",
        "type": "ui_text",
        "z": "e07fb1e2ac3479cf",
        "group": "30031aab1a31c733",
        "order": 1,
        "width": 0,
        "height": 0,
        "name": "",
        "label": "Current Value of PM10",
        "format": "{{msg.payload}}",
        "layout": "row-spread",
        "x": 960,
        "y": 460,
        "wires": []
    },
    {
        "id": "34ba0e75ee916d3e",
        "type": "ui_chart",
        "z": "e07fb1e2ac3479cf",
        "name": "",
        "group": "30031aab1a31c733",
        "order": 2,
```

```
        "order": 2,
        "width": 0,
        "height": 0,
        "label": "Line Chart of PM10",
        "chartType": "line",
        "legend": "false",
        "xformat": "auto",
        "interpolate": "linear",
        "nodata": "",
        "dot": false,
        "ymin": "",
        "ymax": "",
        "removeOlder": 1,
        "removeOlderPoints": "",
        "removeOlderUnit": "3600",
        "cutout": 0,
        "useOneColor": false,
        "useUTC": false,
        "colors": [
            "#37ff00",
            "#aec7e8",
            "#ff7f0e",
            "#00ff00",
            "#98df8a",
            "#df2626",
            "#ff9896",
            "#9467bd",
            "#c5b0d5"
        ],
        "outputs": 1,
        "useDifferentColor": false,
        "x": 910,
        "y": 420,
        "wires": [
            []
        ]
    },
    {
        "id": "61d30df3dee28cb3",
        "type": "function",
        "z": "e07fb1e2ac3479cf",
        "name": "Data For PM10",
        "func": "\nPM10 = msg.payload.PM10;\nmsg = { topic:
'M10', payload: PM10 };\nreturn msg\n",
        "outputs": 1,
        "timeout": 0,
        "noerr": 0,
        "initialize": "",
        "finalize": "",
        "libs": [],
        "x": 580,
        "y": 420,
        "wires": [
            [
```

```
            [
                "34ba0e75ee916d3e",
                "2b2304734194c1ef"
            ]
        ]
    },
    {
        "id": "35454d15f1ca96c6",
        "type": "ui_text",
        "z": "e07fb1e2ac3479cf",
        "group": "355c181b0888d8ba",
        "order": 2,
        "width": 0,
        "height": 0,
        "name": "",
        "label": "Current Value of O3",
        "format": "{{msg.payload}}",
        "layout": "row-spread",
        "x": 990,
        "y": 620,
        "wires": []
    },
    {
        "id": "d857c0ae9d3f5d04",
        "type": "ui_chart",
        "z": "e07fb1e2ac3479cf",
        "name": "",
        "group": "355c181b0888d8ba",
        "order": 1,
        "width": 0,
        "height": 0,
        "label": "Line Chart of O3",
        "chartType": "line",
        "legend": "false",
        "xformat": "auto",
        "interpolate": "linear",
        "nodata": "",
        "dot": false,
        "ymin": "",
        "ymax": "",
        "removeOlder": 1,
        "removeOlderPoints": "",
        "removeOlderUnit": "3600",
        "cutout": 0,
        "useOneColor": false,
        "useUTC": false,
        "colors": [
            "#0ef1ad",
            "#aec7e8",
            "#ff7f0e",
            "#2ca02c",
            "#98df8a",
            "#df2626",
            "#ff9896",
```

```
            "#9467bd",
            "#c5b0d5"
        ],
        "outputs": 1,
        "useDifferentColor": false,
        "x": 940,
        "y": 580,
        "wires": [
            []
        ]
    },
    {
        "id": "85bc4e69c77e19d3",
        "type": "ui_text",
        "z": "e07fb1e2ac3479cf",
        "group": "2649c32460269116",
        "order": 2,
        "width": 0,
        "height": 0,
        "name": "",
        "label": "Current Value of NO2",
        "format": "{{msg.payload}}",
        "layout": "row-spread",
        "x": 1000,
        "y": 540,
        "wires": []
    },
    {
        "id": "bfdf4b7f894b7229",
        "type": "ui_chart",
        "z": "e07fb1e2ac3479cf",
        "name": "",
        "group": "2649c32460269116",
        "order": 1,
        "width": 0,
        "height": 0,
        "label": "Line Chart of NO2",
        "chartType": "line",
        "legend": "false",
        "xformat": "auto",
        "interpolate": "linear",
        "nodata": "",
        "dot": false,
        "ymin": "",
        "ymax": "",
        "removeOlder": 1,
        "removeOlderPoints": "",
        "removeOlderUnit": "3600",
        "cutout": 0,
        "useOneColor": false,
        "useUTC": false,
        "colors": [
            "#fbff00",
            "#aec7e8",
```

```
            #2ca02c",
            "#98df8a",
            "#df2626",
            "#ff9896",
            "#9467bd",
            "#c5b0d5"
        ],
        "outputs": 1,
        "useDifferentColor": false,
        "x": 930,
        "y": 500,
        "wires": [
            []
        ]
    },
    {
        "id": "0a5eaacc6e245af2",
        "type": "function",
        "z": "e07fb1e2ac3479cf",
        "name": "Data For NO2",
        "func": "\nNO2 = msg.payload.NO2;\nmsg = { topic:
'NO2', payload: NO2 };\nreturn msg\n",
        "outputs": 1,
        "timeout": 0,
        "noerr": 0,
        "initialize": "",
        "finalize": "",
        "libs": [],
        "x": 580,
        "y": 500,
        "wires": [
            [
                "bfdf4b7f894b7229",
                "85bc4e69c77e19d3"
            ]
        ]
    },
    {
        "id": "235b95637b8eba14",
        "type": "function",
        "z": "e07fb1e2ac3479cf",
        "name": "Data For O3",
        "func": "\nO3 = msg.payload.O3;\nmsg = { topic:
 'O3', payload: O3 };\nreturn msg\n",
        "outputs": 1,
        "timeout": 0,
        "noerr": 0,
        "initialize": "",
        "finalize": "",
        "libs": [],
        "x": 570,
        "y": 580,
        "wires": [
            [
```

```
                "d857c0ae9d5f5d04",
                "35454d15f1ca96c6"
            ]
        ]
    },
    {
        "id": "e91cba5da6c5758b",
        "type": "ui_text",
        "z": "e07fb1e2ac3479cf",
        "group": "b1f9da87f11d68ec",
        "order": 2,
        "width": 0,
        "height": 0,
        "name": "",
        "label": "Current Value of SO2",
        "format": "{{msg.payload}}",
        "layout": "row-spread",
        "x": 1000,
        "y": 800,
        "wires": []
    },
    {
        "id": "b2ab4b1f64ac68a4",
        "type": "ui_chart",
        "z": "e07fb1e2ac3479cf",
        "name": "",
        "group": "b1f9da87f11d68ec",
        "order": 1,
        "width": 0,
        "height": 0,
        "label": "Line Chart of SO2",
        "chartType": "line",
        "legend": "false",
        "xformat": "auto",
        "interpolate": "linear",
        "nodata": "",
        "dot": false,
        "ymin": "",
        "ymax": "",
        "removeOlder": 1,
        "removeOlderPoints": "",
        "removeOlderUnit": "3600",
        "cutout": 0,
        "useOneColor": false,
        "useUTC": false,
        "colors": [
            "#006eff",
            "#3f4f64",
            "#d9a578",
            "#2ca02c",
            "#2e55f5",
            "#ff0000",
            "#ff9896",
            "#9467bd",
```

```
        "useDifferentColor": false,
        "x": 950,
        "y": 760,
        "wires": [
            []
        ]
    },
    {
        "id": "cd439213e60fcdbb",
        "type": "ui_text",
        "z": "e07fb1e2ac3479cf",
        "group": "21f86775f7b2ce9a",
        "order": 1,
        "width": 0,
        "height": 0,
        "name": "",
        "label": "Current Value of CO",
        "format": "{{msg.payload}}",
        "layout": "row-spread",
        "x": 1000,
        "y": 720,
        "wires": []
    },
    {
        "id": "30af56c7a11697cd",
        "type": "ui_chart",
        "z": "e07fb1e2ac3479cf",
        "name": "",
        "group": "21f86775f7b2ce9a",
        "order": 2,
        "width": 0,
        "height": 0,
        "label": "Line Chart of CO",
        "chartType": "line",
        "legend": "false",
        "xformat": "auto",
        "interpolate": "linear",
        "nodata": "",
        "dot": false,
        "ymin": "",
        "ymax": "",
        "removeOlder": 1,
        "removeOlderPoints": "",
        "removeOlderUnit": "3600",
        "cutout": 0,
        "useOneColor": false,
        "useUTC": false,
        "colors": [
            "#53d3d5",
            "#aec7e8",
            "#ff7f0e",
            "#2ca02c",
            "#98df8a",
```

```
            "#ff9896",
            "#9467bd",
            "#c5b0d5"
        ],
        "outputs": 1,
        "useDifferentColor": false,
        "x": 930,
        "y": 680,
        "wires": [
            []
        ]
    },
    {
        "id": "2b9317665ed72200",
        "type": "function",
        "z": "e07fb1e2ac3479cf",
        "name": "Data For CO",
        "func": "\nCO = msg.payload.CO;\nmsg = { topic
 'CO', payload: CO };\nreturn msg\n",
        "outputs": 1,
        "timeout": 0,
        "noerr": 0,
        "initialize": "",
        "finalize": "",
        "libs": [],
        "x": 570,
        "y": 680,
        "wires": [
            [
                "30af56c7a11697cd",
                "cd439213e60fcdbb"
            ]
        ]
    },
    {
        "id": "ef6950b7be5054e4",
        "type": "function",
        "z": "e07fb1e2ac3479cf",
        "name": "Data For SO2",
        "func": "\nSO2 = msg.payload.SO2;\nmsg = { topic:
SO2', payload: SO2 };\nreturn msg\n",
        "outputs": 1,
        "timeout": 0,
        "noerr": 0,
        "initialize": "",
        "finalize": "",
        "libs": [],
        "x": 580,
        "y": 760,
        "wires": [
            [
                "b2ab4b1f64ac68a4",
                "e91cba5da6c5758b"
```

```
            ]
        ]
    },
    {
        "id": "e3baeff913c962e0",
        "type": "function",
        "z": "e07fb1e2ac3479cf",
        "name": "Data For AQI",
        "func": "\nAQI = msg.payload.AQI;\nmsg = { topic:
'AQI', payload: AQI };\nreturn msg\n",
        "outputs": 1,
        "timeout": 0,
        "noerr": 0,
        "initialize": "",
        "finalize": "",
        "libs": [],
        "x": 570,
        "y": 840,
        "wires": [
            [
                "c9960c80c420c7f8",
                "5f03471c05b5eedd"
            ]
        ]
    },
    {
        "id": "c9960c80c420c7f8",
        "type": "ui_chart",
        "z": "e07fb1e2ac3479cf",
        "name": "",
        "group": "bf2ced2110105158",
        "order": 1,
        "width": 0,
        "height": 0,
        "label": "Line Chart of AQI",
        "chartType": "line",
        "legend": "false",
        "xformat": "auto",
        "interpolate": "linear",
        "nodata": "",
        "dot": false,
        "ymin": "",
        "ymax": "",
        "removeOlder": 1,
        "removeOlderPoints": "",
        "removeOlderUnit": "3600",
        "cutout": 0,
        "useOneColor": false,
        "useUTC": false,
        "colors": [
            "#560631",
            "#1371ec",
            "#bf2259",
            "#2ca02c",
```

```
            "#98df8a",
            "#df2626",
            "#ff9896",
            "#9467bd",
            "#c5b0d5"
        ],
        "outputs": 1,
        "useDifferentColor": false,
        "x": 950,
        "y": 840,
        "wires": [
            []
        ]
    },
    {
        "id": "5f03471c05b5eedd",
        "type": "ui_text",
        "z": "e07fb1e2ac3479cf",
        "group": "bf2ced2110105158",
        "order": 2,
        "width": 0,
        "height": 0,
        "name": "",
        "label": "Current Value of AQI",
        "format": "{{msg.payload}}",
        "layout": "row-spread",
        "x": 980,
        "y": 880,
        "wires": []
    },
    {
        "id": "a143369d49d67e69",
        "type": "comment",
        "z": "e07fb1e2ac3479cf",
        "name": "Seperating Data For Each Colum",
        "info": "",
        "x": 570,
        "y": 300,
        "wires": []
    },
    {
        "id": "ed88d8255a256ca0",
        "type": "comment",
        "z": "e07fb1e2ac3479cf",
        "name": "visualisation data in line chart and text",
        "info": "",
        "x": 930,
        "y": 300,
        "wires": []
    },
    {
        "id": "4f4b5e2c328be351",
        "type": "debug",
        "z": "e07fb1e2ac3479cf",
```

```
        "name": "debug 1",
        "active": true,
        "tosidebar": true,
        "console": false,
        "tostatus": false,
        "complete": "false",
        "statusVal": "",
        "statusType": "auto",
        "x": 180,
        "y": 1040,
        "wires": []
    },
    {
        "id": "5990c56388fab544",
        "type": "comment",
        "z": "e07fb1e2ac3479cf",
        "name": "Data for Comparsion
of AQI with Other Particles",
        "info": "",
        "x": 620,
        "y": 940,
        "wires": []
    },
    {
        "id": "90c695193bbd04f1",
        "type": "comment",
        "z": "e07fb1e2ac3479cf",
        "name": "Comparsion Line Chart of AQI with Other Particles",
        "info": "",
        "x": 1030,
        "y": 940,
        "wires": []
    },
    {
        "id": "c0ab5d441e70f931",
        "type": "ui_chart",
        "z": "e07fb1e2ac3479cf",
        "name": "",
        "group": "344fa39eeededc7c",
        "order": 1,
        "width": 0,
        "height": 0,
        "label": "Comparison of AQI With PM10",
        "chartType": "line",
        "legend": "true",
        "xformat": "HH:mm:ss",
        "interpolate": "linear",
        "nodata": "",
        "dot": false,
        "ymin": "",
        "ymax": "",
        "removeOlder": 1,
        "removeOlderPoints": "",
        "removeOlderUnit": "3600",
```

```
    "cutout": 0,
    "useOneColor": false,
    "useUTC": false,
    "colors": [
        "#1f77b4",
        "#aec7e8",
        "#ff7f0e",
        "#2ca02c",
        "#98df8a",
        "#d62728",
        "#ff9896",
        "#9467bd",
        "#c5b0d5"
    ],
    "outputs": 1,
    "useDifferentColor": false,
    "x": 1050,
    "y": 1060,
    "wires": [
        []
    ]
,

    "id": "7491e55c2c304572",
    "type": "function",
    "z": "e07fb1e2ac3479cf",
    "name": "Data for Comparison of AQI With PM10",
    "func": "\nValueToLine1 = msg.payload.AQI;\nValueToLine2 =
payload.PM10;\n\nline1 = { topic: 'AQI', payload: ValueToLine1
ine2 = { topic: 'PM10', payload: ValueToLine2 };\n  \nreturn [line1, l
    "outputs": 2,
    "timeout": 0,
    "noerr": 0,
    "initialize": "",
    "finalize": "",
    "libs": [],
    "x": 640,
    "y": 1060,
    "wires": [
        [
            "c0ab5d441e70f931"
        ],
        [
            "c0ab5d441e70f931"
        ]
    ]
,

    "id": "42fbdcd65fb95b15",
    "type": "function",
    "z": "e07fb1e2ac3479cf",
    "name": "Data for Comparison of AQI With NO2",
    "func": "\nValueToLine1 = msg.payload.AQI;\nValueToLine2 = msg.payl
pic: 'AQI', payload: ValueToLine1 };\nline2 = { topic: 'NO2', payload:
```

```
        "outputs": 2,
        "timeout": 0,
        "noerr": 0,
        "initialize": "",
        "finalize": "",
        "libs": [],
        "x": 630,
        "y": 1120,
        "wires": [
            [
                "ff64e1ec0b3116d0"
            ],
            [
                "ff64e1ec0b3116d0"
            ]
        ]
    },
    {
        "id": "64d3799dae0caec5",
        "type": "function",
        "z": "e07fb1e2ac3479cf",
        "name": "Data for Comparison of AQI With O3",
        "func": "\nValueToLine1 = msg.payload.AQI;\nValueToLine2 = msg.payload.O3;\n\nline1 = { t
        "outputs": 2,
        "timeout": 0,
        "noerr": 0,
        "initialize": "",
        "finalize": "",
        "libs": [],
        "x": 610,
        "y": 1180,
        "wires": [
            [
                "238b44693066f830"
            ],
            [
                "238b44693066f830"
            ]
        ]
    },
    {
        "id": "9273b7848d0a4e37",
        "type": "function",
        "z": "e07fb1e2ac3479cf",
        "name": "Data for Comparison of AQI With CO",
        "func": "\nValueToLine1 = msg.payload.AQI;\nValueToLine2 = msg.payload.CO;\n\nline1 = { t
        "outputs": 2,
        "timeout": 0,
        "noerr": 0,
        "initialize": "",
        "finalize": "",
        "libs": [],
        "x": 590,
        "y": 1240,
        "wires": [
            [
                "42a1345300c1c2e3"
            ],
            [
                "42a1345300c1c2e3"
            ]
        ]
    },
    {
        "id": "1b7954064aead7ee",
        "type": "function",
        "z": "e07fb1e2ac3479cf",
        "name": "Data for Comparison of AQI With SO2",
        "func": "\nValueToLine1 = msg.payload.AQI;\nValueToLine2 = msg.payload.SO2;\n\nline1 = {
        "outputs": 2,
        "timeout": 0,
        "noerr": 0,
        "initialize": "",
        "finalize": "",
        "libs": [],
```

```
        "libs": [],
        "x": 570,
        "y": 1300,
        "wires": [
            [
                "c314b9accbcf23ee"
            ],
            [
                "c314b9accbcf23ee"
            ]
        ]
    },
    {
        "id": "5f617d337d36b209",
        "type": "function",
        "z": "e07fb1e2ac3479cf",
        "name": "Data for Comparison of All Particles",
        "func": "\nValueToLine1 = msg.payload.AQI;\nValueToLine2 = msg.payload.PM2_5;\nValueToLine3 = msg.p
PM10', payload: ValueToLine3 };\nline4 = { topic: 'NO2', payload: ValueToLine4 };\nline5 = { topic: 'O3',
        "outputs": 7,
        "timeout": 0,
        "noerr": 0,
        "initialize": "",
        "finalize": "",
        "libs": [],
        "x": 580,
        "y": 1460,
        "wires": [
            [
                "0f9f873f98de7d90"
            ],
            [
                "0f9f873f98de7d90"
            ],
            [
                "0f9f873f98de7d90"
            ],
            [
                "0f9f873f98de7d90"
            ],
            [
                "0f9f873f98de7d90"
            ],
            [
                "0f9f873f98de7d90"
            ],
            [
                "0f9f873f98de7d90"
            ]
        ]
    },
    {
        "id": "0f9f873f98de7d90",
        "type": "ui_chart",
        "z": "e07fb1e2ac3479cf",
        "name": "",
        "group": "9a6c102ad07d5385",
        "order": 1,
        "width": 0,
        "height": 0,
        "label": "Comparison of All Particles In Line Chart",
        "chartType": "line",
        "legend": "true",
        "xformat": "HH:mm:ss",
        "interpolate": "linear",
        "nodata": "",
        "dot": false,
        "ymin": "",
        "ymax": "",
        "removeOlder": 1,
        "removeOlderPoints": "",
        "removeOlderUnit": "3600",
        "cutout": 0,
        "useOneColor": false,
        "useUTC": false,
        "colors": [
            "#1f77b4",
            "#aec7e8",
            "#ff7f0e",
            "#2ca02c",
            "#98df8a",
            "#d62728",
            "#ff9896",
            "#9467bd",
            "#c5b0d5"
        ],
        "outputs": 1,
        "useDifferentColor": false,
        "x": 1020,
        "y": 1460,
        "wires": [
            []
        ]
    },
    {
        "id": "a3684a9b33ad45aa",
        "type": "comment",
        "z": "e07fb1e2ac3479cf",
```

```
        "name": "Data for Comparsion of all  Particles",
        "info": "",
        "x": 600,
        "y": 1360,
        "wires": []
    },
    {
        "id": "bf8895d9e5153c66",
        "type": "comment",
        "z": "e07fb1e2ac3479cf",
        "name": "Comparsion Line Chart of all Particles",
        "info": "",
        "x": 1030,
        "y": 1360,
        "wires": []
    },
    {
        "id": "ff64e1ec0b3116d0",
        "type": "ui_chart",
        "z": "e07fb1e2ac3479cf",
        "name": "",
        "group": "364345a2b278c2e8",
        "order": 1,
        "width": 0,
        "height": 0,
        "label": "Comparison of AQI With NO2",
        "chartType": "line",
        "legend": "true",
        "xformat": "HH:mm:ss",
        "interpolate": "linear",
        "nodata": "",
        "dot": false,
        "ymin": "",
        "ymax": "",
        "removeOlder": 1,
        "removeOlderPoints": "",
        "removeOlderUnit": "3600",
        "cutout": 0,
        "useOneColor": false,
        "useUTC": false,
        "colors": [
            "#1f77b4",
            "#aec7e8",
            "#ff7f0e",
            "#2ca02c",
            "#98df8a",
            "#d62728",
            "#ff9896",
            "#9467bd",
            "#c5b0d5"
        ],
        "outputs": 1,
        "useDifferentColor": false,
        "x": 1060,
        "y": 1120,
        "wires": [
            []
        ]
    },
    {
        "id": "238b44693066f830",
        "type": "ui_chart",
        "z": "e07fb1e2ac3479cf",
        "name": "",
        "group": "0534c83b77178985",
        "order": 1,
        "width": 0,
        "height": 0,
        "label": "Comparison of AQI With O3",
        "chartType": "line",
        "legend": "true",
        "xformat": "HH:mm:ss",
        "interpolate": "linear",
        "nodata": "",
        "dot": false,
        "ymin": "",
        "ymax": "",
        "removeOlder": 1,
        "removeOlderPoints": "",
        "removeOlderUnit": "3600",
        "cutout": 0,
        "useOneColor": false,
        "useUTC": false,
        "colors": [
            "#1f77b4",
            "#aec7e8",
            "#ff7f0e",
            "#2ca02c",
            "#98df8a",
            "#d62728",
            "#ff9896",
            "#9467bd",
            "#c5b0d5"
        ],
        "outputs": 1,
        "useDifferentColor": false,
        "x": 1060,
```

```
        "wires": [
            []
        ]
    },
    {
        "id": "42a1345300c1c2e3",
        "type": "ui_chart",
        "z": "e07fb1e2ac3479cf",
        "name": "",
        "group": "34265e323e9d9b99",
        "order": 1,
        "width": 0,
        "height": 0,
        "label": "Comparison of AQI With CO",
        "chartType": "line",
        "legend": "true",
        "xformat": "HH:mm:ss",
        "interpolate": "linear",
        "nodata": "",
        "dot": false,
        "ymin": "",
        "ymax": "",
        "removeOlder": 1,
        "removeOlderPoints": "",
        "removeOlderUnit": "3600",
        "cutout": 0,
        "useOneColor": false,
        "useUTC": false,
        "colors": [
            "#1f77b4",
            "#aec7e8",
            "#ff7f0e",
            "#2ca02c",
            "#98df8a",
            "#d62728",
            "#ff9896",
            "#9467bd",
            "#c5b0d5"
        ],
        "outputs": 1,
        "useDifferentColor": false,
        "x": 1060,
        "y": 1240,
        "wires": [
            []
        ]
    },
    {
        "id": "c314b9accbcf23ee",
        "type": "ui_chart",
        "z": "e07fb1e2ac3479cf",
        "name": "",
        "group": "f3f5c067d30ab2b1",
        "order": 1,
        "width": 0,
        "height": 0,
        "label": "Comparison of AQI With SO2",
        "chartType": "line",
        "legend": "true",
        "xformat": "HH:mm:ss",
        "interpolate": "linear",
        "nodata": "",
        "dot": false,
        "ymin": "",
        "ymax": "",
        "removeOlder": 1,
        "removeOlderPoints": "",
        "removeOlderUnit": "3600",
        "cutout": 0,
        "useOneColor": false,
        "useUTC": false,
        "colors": [
            "#1f77b4",
            "#aec7e8",
            "#ff7f0e",
            "#2ca02c",
            "#98df8a",
            "#d62728",
            "#ff9896",
            "#9467bd",
            "#c5b0d5"
        ],
        "outputs": 1,
        "useDifferentColor": false,
        "x": 1060,
        "y": 1300,
        "wires": [
            []
        ]
    },
    {
        "id": "ad6d92c983c792d2",
        "type": "ui_spacer",
        "z": "e07fb1e2ac3479cf",
        "name": "spacer",
        "group": "abd9f2aa52ccd0f5",
        "order": 1,
```

```
[
    {
        "id": "0f1c47006780 4c4b",
        "type": "matt-broker",
        "name": "",
        "broker": "broker.hivemq.com",
        "port": "1883",
        "clientid": "",
        "autoConnect": true,
        "usetls": false,
        "protocolVersion": "4",
        "keepalive": "60",
        "cleansession": true,
        "autoUnsubscribe": true,
        "birthTopic": "",
        "birthQos": "0",
        "birthRetain": "false",
        "birthPayload": "",
        "birthMsg": {},
        "closeTopic": "",
        "closeQos": "0",
        "closeRetain": "false",
        "closePayload": "",
        "closeMsg": {},
        "willTopic": "",
        "willQos": "0",
        "willRetain": "false",
        "willPayload": "",
        "willMsg": {},
        "userProps": "",
        "sessionExpiry": ""
    },
    {
        "id": "13bcf5a28cd15de8",
        "type": "ui_group",
        "name": "PM2_5",
        "tab": "0ea5e8f3fa2e9e8a",
        "order": 1,
        "disp": true,
        "width": "6",
        "collapse": false
    },
    {
        "id": "b86cf1815e4e9ffc",
        "type": "ui_group",
        "name": "Comparison of AQI With PM2_5",
        "tab": "1d8ea8e52a1b9b94",
        "order": 2,
        "disp": true,
        "width": "6",
        "collapse": false
    },
    {
        "id": "30031aab1a31c733",
        "type": "ui_group",
        "name": "PM10",
        "tab": "0ea5e8f3fa2e9e8a",
        "order": 2,
        "disp": true,
        "width": "6",
        "collapse": false
    },
    {
        "id": "355c181b0888d8ba",
        "type": "ui_group",
        "name": "O3",
        "tab": "0ea5e8f3fa2e9e8a",
        "order": 4,
        "disp": true,
        "width": "6",
        "collapse": false
    },
    {
        "id": "2649c32460269116",
        "type": "ui_group",
        "name": "NO2",
        "tab": "0ea5e8f3fa2e9e8a",
        "order": 3,
        "disp": true,
        "width": "6",
        "collapse": false
    },
    {
        "id": "b1f9da87f11d68ec",
        "type": "ui_group",
        "name": "SO2",
        "tab": "0ea5e8f3fa2e9e8a",
        "order": 6,
        "disp": true,
        "width": "6",
        "collapse": false
    },
    {
        "id": "21f86775f7b2ce9a",
        "type": "ui_group",
        "name": "CO",
        "tab": "0ea5e8f3fa2e9e8a",
        "order": 5,
        "disp": true
```

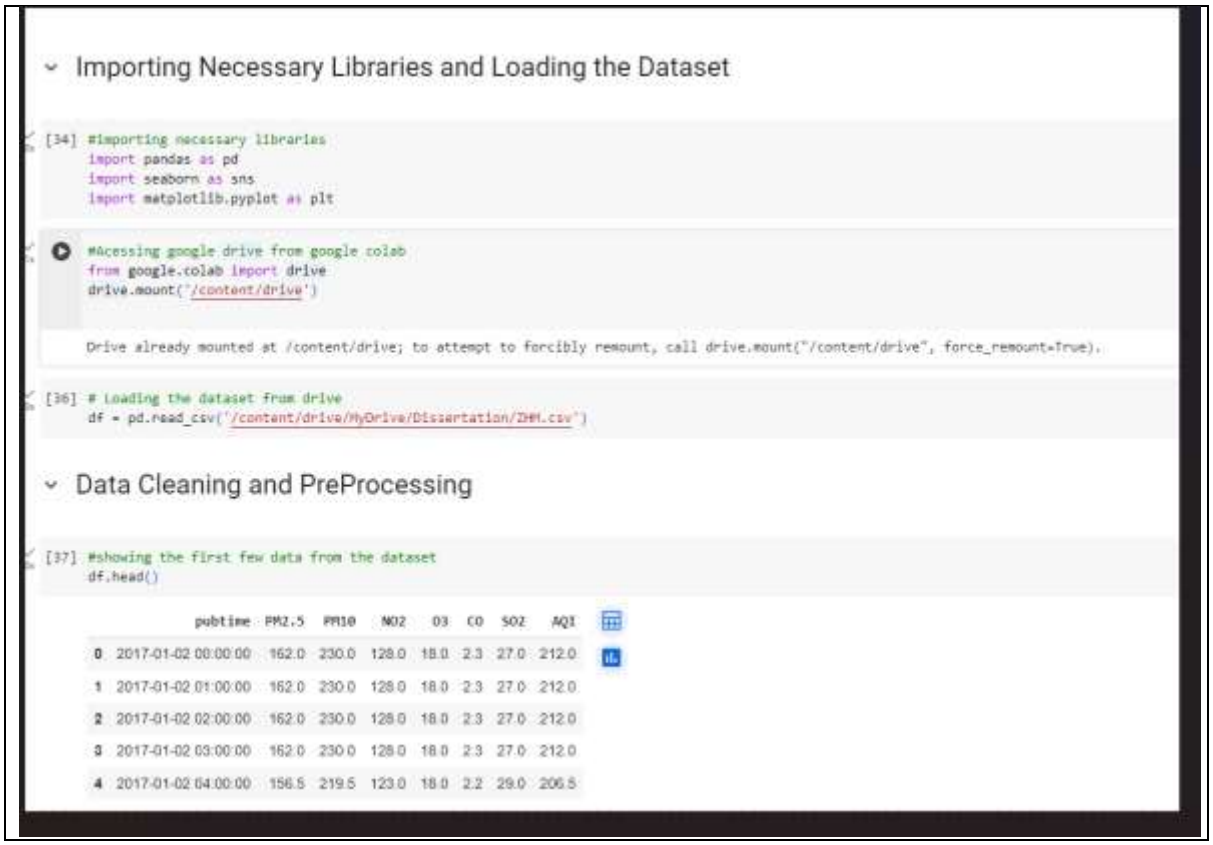
```
        "disp": true,
        "width": "6",
        "collapse": false
    },
    {
        "id": "bf2ced2110105158",
        "type": "ui_group",
        "name": "AQI",
        "tab": "0ea5e8f3fa2e9e8a",
        "order": 7,
        "disp": true,
        "width": "6",
        "collapse": false
    },
    {
        "id": "344fa39eeededc7c",
        "type": "ui_group",
        "name": "Comparison of AQI With PM10",
        "tab": "1d8ea8e52a1b9b94",
        "order": 3,
        "disp": true,
        "width": "6",
        "collapse": false
    },
    {
        "id": "9a6c102ad07d5385",
        "type": "ui_group",
        "name": "Comparison of All Particle",
        "tab": "1d8ea8e52a1b9b94",
        "order": 1,
        "disp": true,
        "width": "18",
        "collapse": false
    },
    {
        "id": "364345a2b278c2e8",
        "type": "ui_group",
        "name": "Comparison of AQI With NO2",
        "tab": "1d8ea8e52a1b9b94",
        "order": 4,
        "disp": true,
        "width": "6",
        "collapse": false
    },
    {
        "id": "0534c83b77178985",
        "type": "ui_group",
        "name": "Comparison of AQI With O3",
        "tab": "1d8ea8e52a1b9b94",
        "order": 5,
        "disp": true,
        "width": "6",
        "collapse": false
    },
    {
        "id": "34265e323e9d9b99",
        "type": "ui_group",
        "name": "Comparison of AQI With CO",
        "tab": "1d8ea8e52a1b9b94",
        "order": 6,
        "disp": true,
        "width": "6",
        "collapse": false
    },
    {
        "id": "f3f5c067d30ab2b1",
        "type": "ui_group",
        "name": "Comparison of AQI With SO2",
        "tab": "1d8ea8e52a1b9b94",
        "order": 7,
        "disp": true,
        "width": "6",
        "collapse": false
    },
    {
        "id": "0ea5e8f3fa2e9e8a",
        "type": "ui_tab",
        "name": "Current Sensor Data",
        "icon": "dashboard",
        "order": 1,
        "disabled": false,
        "hidden": false
    },
    {
```

APPENDIX 2: Setting.js File

```
module.exports = {

    /** The file containing the flows. If not set, defaults to flows_<hostname>.json **/
    flowFile: 'flows.json',
    flowFilePretty: true,
    /** To password protect the Node-RED editor and admin API, the following
     * property can be used. See http://nodered.org/docs/security.html for details.
     */
    adminAuth: {
        type: "credentials",
        users: [{
            username: "admin",
            password: "$2a$08$bmh79LLYXAQieH5Y/DnA7uPvpxgz6i2aafkEyrXy7bF2vnpkeaHgm",
            permissions: "*"
        }]
    },

    /** the tcp port that the Node-RED web server is listening on */
    uiPort: process.env.PORT || 1880,
    /** Configure the logging output */
    logging: {
        /** Only console logging is currently supported */
        console: {

            level: "info",
            /** Whether or not to include metric events in the log output */
            metrics: false,
            /** Whether or not to include audit events in the log output */
            audit: false
        }
    },
```

```
    exportGlobalContextKeys: false,
    externalModules: {
    },
    editorTheme: {
        palette: {
        },
        projects: {
            /** To enable the Projects feature, set this value to true */
            enabled: false,
            workflow: {
                /** Set the default projects workflow mode.
                 *  - manual - you must manually commit changes
                 *  - auto - changes are automatically committed
                 * This can be overridden per-user from the 'Git config'
                 * section of 'User Settings' within the editor
                 */
                mode: "manual"
            }
        },
        codeEditor: {
            /** Select the text editor component used by the editor.
             * Defaults to "ace", but can be set to "ace" or "monaco"
             */
            lib: "ace",
            options: {

                theme: "vs",

            }
        }
    },

    /** Allow the Function node to load additional npm modules directly */
    functionExternalModules: true,

    functionGlobalContext: {
        // os:require('os'),
    },
    /** The maximum length, in characters, of any message sent to the debug sidebar tab */
    debugMaxLength: 1000,
    mqttReconnectTime: 15000,
    /** Retry time in milliseconds for Serial port connections */
    serialReconnectTime: 15000,
}
```

APPENDIX 3: Data Cleaning, Pre-Processing, and Exploratory Data Analysis



Importing Necessary Libraries and Loading the Dataset

```
[34] #importing necessary libraries
     import pandas as pd
     import seaborn as sns
     import matplotlib.pyplot as plt
```

```
     #Acessing google drive from google colab
     from google.colab import drive
     drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
[36] # Loading the dataset from drive
     df = pd.read_csv('/content/drive/MyDrive/Dissertation/ZHM.csv')
```

Data Cleaning and PreProcessing

```
[37] #showing the first few data from the dataset
     df.head()
```

	pubtime	PM2.5	PM10	NO2	O3	CO	SO2	AQI
0	2017-01-02 00:00:00	162.0	230.0	128.0	18.0	2.3	27.0	212.0
1	2017-01-02 01:00:00	162.0	230.0	128.0	18.0	2.3	27.0	212.0
2	2017-01-02 02:00:00	162.0	230.0	128.0	18.0	2.3	27.0	212.0
3	2017-01-02 03:00:00	162.0	230.0	128.0	18.0	2.3	27.0	212.0
4	2017-01-02 04:00:00	156.5	219.5	123.0	18.0	2.2	29.0	206.5

```
[38] #shape of data
     df.shape

     (35040, 8)
```

```
[39] #showing all the columns in dataset
     df.columns

     Index(['pubtime', 'PM2.5', 'PM10', 'NO2', 'O3', 'CO', 'SO2', 'AQI'], dtype='object')
```

```
[40] #showing overall information of whole dataset
     df.info()

     <class 'pandas.core.frame.DataFrame'>
     RangeIndex: 35040 entries, 0 to 35039
     Data columns (total 8 columns):
      #   Column   Non-Null Count  Dtype
     ---  ------   --------------  -----
      0   pubtime  35040 non-null  object
      1   PM2.5    35040 non-null  float64
      2   PM10     35040 non-null  float64
      3   NO2      35040 non-null  float64
      4   O3       35040 non-null  float64
      5   CO       35040 non-null  float64
      6   SO2      35040 non-null  float64
      7   AQI      35040 non-null  float64
     dtypes: float64(7), object(1)
     memory usage: 2.1+ MB
```

```
[41] #showing the descriptive statistics of whole Dataset
     df.describe()
```

	PM2.5	PM10	NO2	O3	CO	SO2	AQI
count	35040.000000	35040.000000	35040.000000	35040.000000	35040.000000	35040.000000	35040.000000
mean	38.710924	69.343559	43.808095	65.447962	0.881120	12.221237	65.993193
std	31.539575	45.944918	26.777590	50.440051	0.403825	13.997214	37.757436
min	1.000000	1.000000	1.000000	1.000000	0.100000	1.000000	5.000000
25%	18.000000	38.000000	24.000000	27.000000	0.600000	6.000000	42.000000
50%	31.000000	58.125000	36.625000	56.000000	0.800000	9.000000	58.000000
75%	50.000000	89.000000	57.000000	91.000000	1.100000	14.000000	79.000000
max	331.000000	527.000000	188.000000	526.000000	6.300000	292.000000	427.000000

```
[42] #checking all the unique values that are present in a dataset
     df.nunique()

     pubtime    35040
     PM2.5       1353
     PM10        2110
     NO2         1088
     O3          1406
     CO           576
     SO2          677
     AQI         1297
     dtype: int64
```

To perform the analysis and different machine learning model, last six month data was extraxted from the dataset.

```
[43] df['pubtime'] = pd.to_datetime(df['pubtime'])

     # start and end dates
     start_date = pd.Timestamp('2020-06-30')
     end_date = pd.Timestamp('2020-12-31')

     # Filtering the DataFrame based on date
     df = df[(df['pubtime'] >= start_date) & (df['pubtime'] <= end_date)]
     df
```

	pubtime	PM2.5	PM10	NO2	O3	CO	SO2	AQI
30600	2020-06-30 00:00:00	11.0	23.0	26.0	70.0	0.7	5.0	23.0
30601	2020-06-30 01:00:00	14.0	23.0	29.0	54.0	0.8	5.0	23.0
30602	2020-06-30 02:00:00	11.0	23.0	31.0	48.0	0.7	5.0	23.0
30603	2020-06-30 03:00:00	12.0	25.0	42.0	26.0	0.7	5.0	25.0
30604	2020-06-30 04:00:00	7.0	29.0	36.0	22.0	0.7	5.0	29.0

[43]

...	...	...	...	...	...	...	...	...	...
35012	2020-12-30 20:00:00	21.0	59.0	35.0	42.0	0.8	6.0	55.0
35013	2020-12-30 21:00:00	22.0	65.0	35.0	40.0	0.8	6.0	58.0
35014	2020-12-30 22:00:00	26.0	64.0	30.0	43.0	0.8	6.0	57.0
35015	2020-12-30 23:00:00	24.0	58.0	30.0	42.0	0.8	6.0	54.0
35016	2020-12-31 00:00:00	23.0	61.0	25.0	46.0	0.8	7.0	56.0

4417 rows × 8 columns

Next steps: View recommended plots

[44]
```
df.shape
```

```
(4417, 8)
```

So the new dataset will have the spahe of 4417 column and 8 rows.

[45]
```
#reindex the order in chronological order
df = df.reindex(index = df.index[::1]).reset_index(drop=True)
df
```

	pubtime	PM2.5	PM10	NO2	O3	CO	SO2	AQI
0	2020-06-30 00:00:00	11.0	23.0	26.0	70.0	0.7	5.0	23.0
1	2020-06-30 01:00:00	14.0	23.0	29.0	54.0	0.8	5.0	23.0
2	2020-06-30 02:00:00	11.0	23.0	31.0	48.0	0.7	5.0	23.0
3	2020-06-30 03:00:00	12.0	25.0	42.0	26.0	0.7	5.0	25.0
4	2020-06-30 04:00:00	7.0	29.0	36.0	22.0	0.7	5.0	29.0
...	...	...	...	...	...	...	...	...
4412	2020-12-30 20:00:00	21.0	59.0	35.0	42.0	0.8	6.0	55.0
4413	2020-12-30 21:00:00	22.0	65.0	35.0	40.0	0.8	6.0	58.0

[45]

4414	2020-12-30 22:00:00	26.0	64.0	30.0	43.0	0.8	6.0	57.0
4415	2020-12-30 23:00:00	24.0	58.0	30.0	42.0	0.8	6.0	54.0
4416	2020-12-31 00:00:00	23.0	61.0	25.0	46.0	0.8	7.0	56.0

4417 rows × 8 columns

Next steps: View recommended plots

[46]
```
#checking all the unique values that are present in a selected range of data
df.nunique()
```

```
pubtime    4417
PM2.5       220
PM10        246
NO2         194
O3          295
CO           87
SO2          67
AQI         280
dtype: int64
```

Checking Null Values

[47]
```
# Check for missing values
print(df.isnull().sum())
```

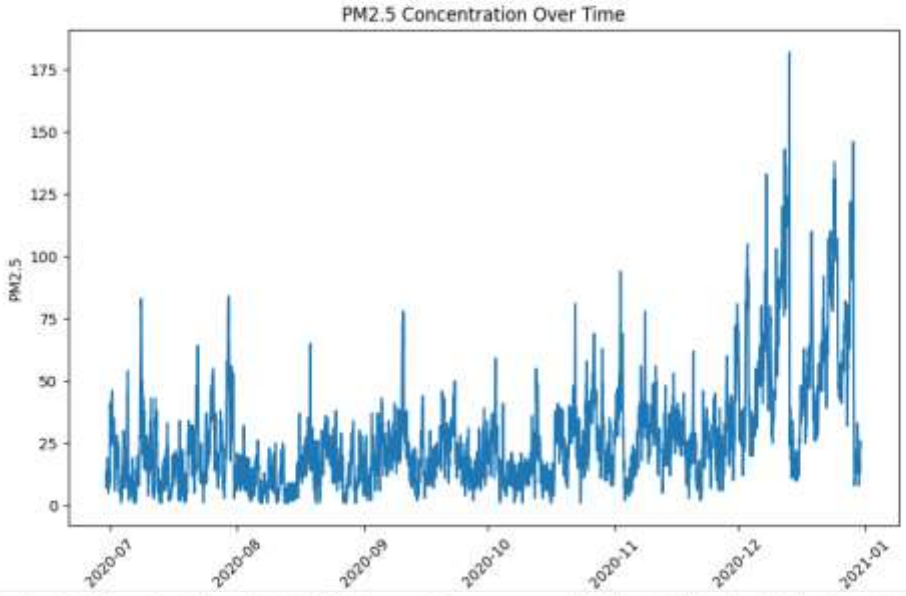
```
pubtime    0
PM2.5      0
PM10       0
NO2        0
O3         0
CO         0
SO2        0
AQI        0
dtype: int64
```

Checking Duplicate Values

```
[48] #Checking duplicated data
     df.loc[df.duplicated()]
```

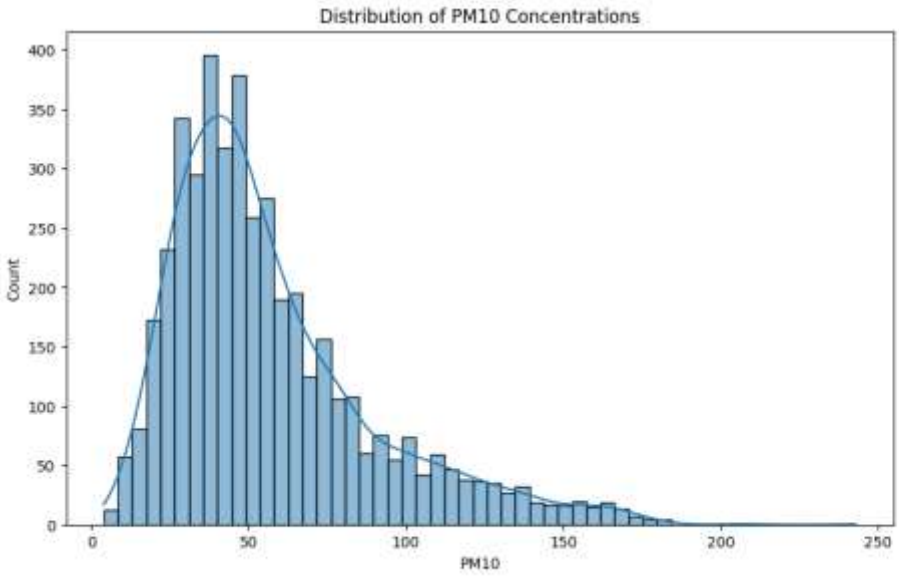
pubtime	PM2.5	PM10	NO2	O3	CO	SO2	AQI

```
[49] # Question 1. How does the PM2.5 concentration vary over time?
     plt.figure(figsize=(10, 6))
     sns.lineplot(data=df, x='pubtime', y='PM2.5')
     plt.xticks(rotation=45)
     plt.title('PM2.5 Concentration Over Time')
     plt.show()
```



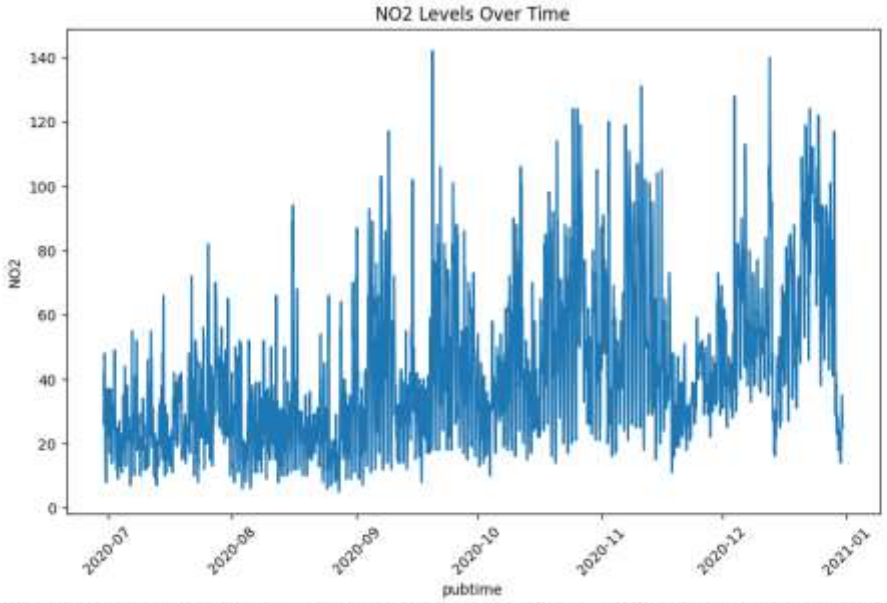
During the mid of December, it shows that the PM2.5 concentrations was recorded highest which is almost 175 and in whole December month it was recorded high as compared to the previous months.

```
[50] # Question 2. What is the distribution of PM10 concentrations?
     plt.figure(figsize=(10, 6))
     sns.histplot(data=df, x='PM10', kde=True)
     plt.title('Distribution of PM10 Concentrations')
     plt.show()
```



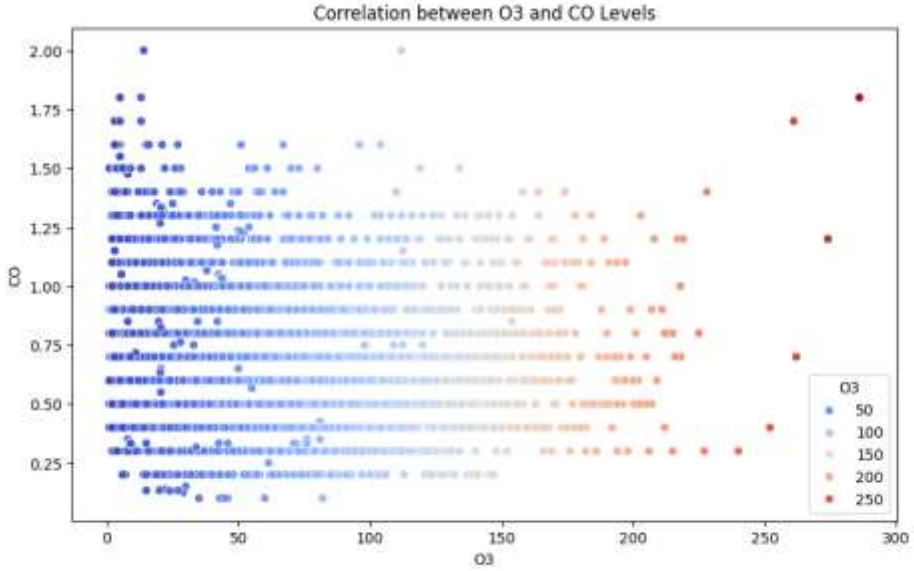
Histogram plot show the PM10 concentrations is skewed to right, where lower concentrations are occurring more frequently than higher concentrations. This suggest that the majority of observation fall in lower concentration range as long tail is extending towards higher values.

```
# Question 3. How does the NO2 levels change over time?
plt.figure(figsize=(10, 6))
sns.lineplot(data=df, x='pubtime', y='NO2')
plt.xticks(rotation=45)
plt.title('NO2 Levels Over Time')
plt.show()
```



Line plot shows that the level of NO2 was fluctuated throughout the different month. Sometime it goes to maximum and sometime it goes to minimum.

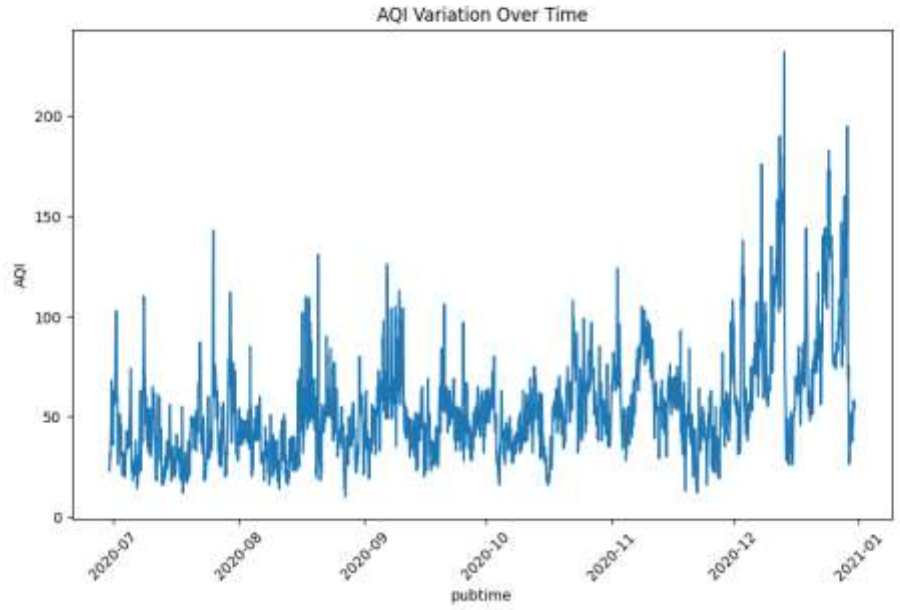
```
[52] # Question 4. Is there any correlation between O3 and CO levels?
     plt.figure(figsize=(10, 6))
     sns.scatterplot(data=df, x='O3', y='CO', hue='O3', palette='coolwarm')
     plt.title('Correlation between O3 and CO Levels')
     plt.show()
```



It show correlation between O3 and CO levels through scatter plot where, each point on plot represent the observed data having X-axis as O3

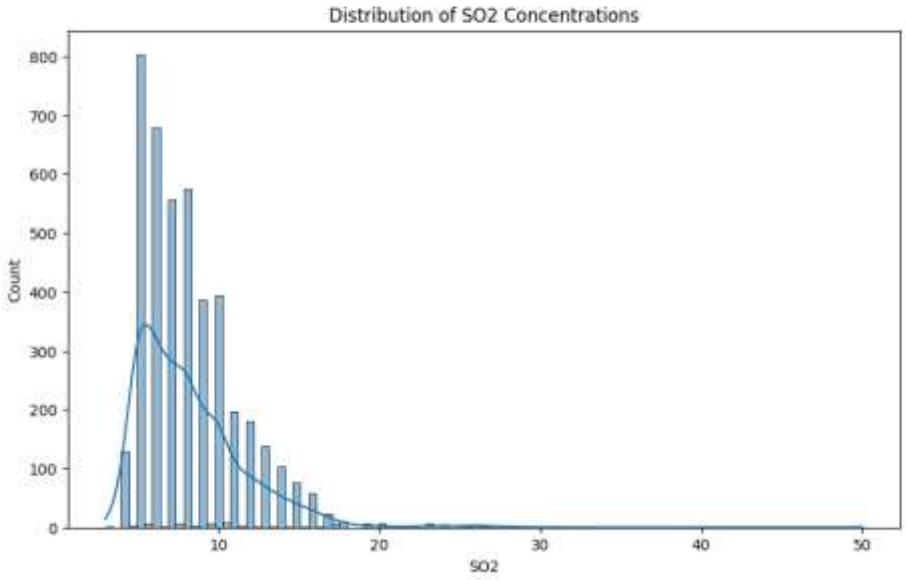
level and Y-axis as CO level. It shows weak negative correlation and due to the point spread around the trends line it can be state that relationship between CO and O3 is not particularly strong.

```
[53] # Question 5. How does  AQI vary across different times?
     plt.figure(figsize=(10, 6))
     sns.lineplot(data=df, x='pubtime', y='AQI')
     plt.xticks(rotation=45)
     plt.title('AQI Variation Over Time')
     plt.show()
```



During the mid of December, it shows that the AQI was recorded highest which is more than 200 and in whole December month it was recorded high as compared to the previous months.

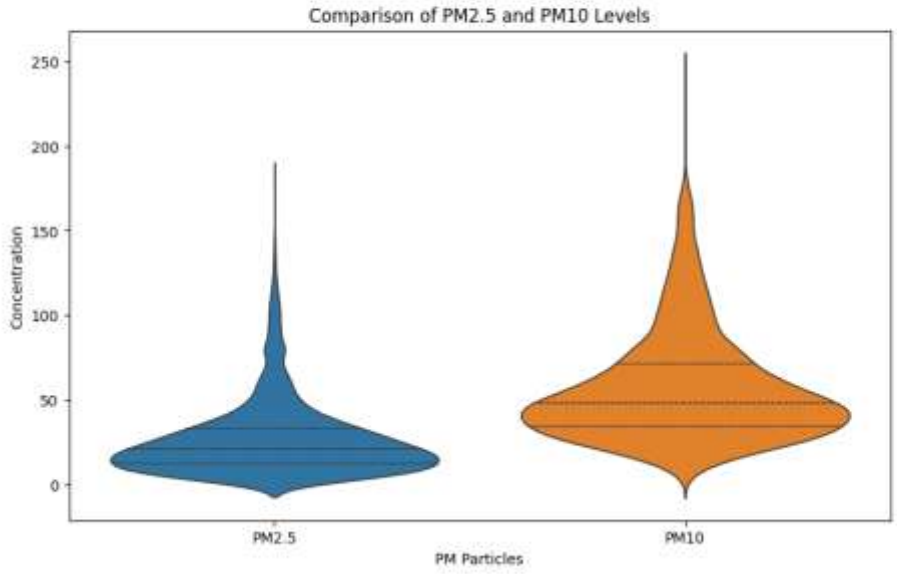
```
[54] # Question 6. What is the distribution of SO2 concentrations?
     plt.figure(figsize=(10, 6))
     sns.histplot(data=df, x='SO2', kde=True)
     plt.title('Distribution of SO2 Concentrations')
     plt.show()
```



Histogram plot show the SO2 concentrations is skewed to right, where lower concentrations are occurring more frequently than higher

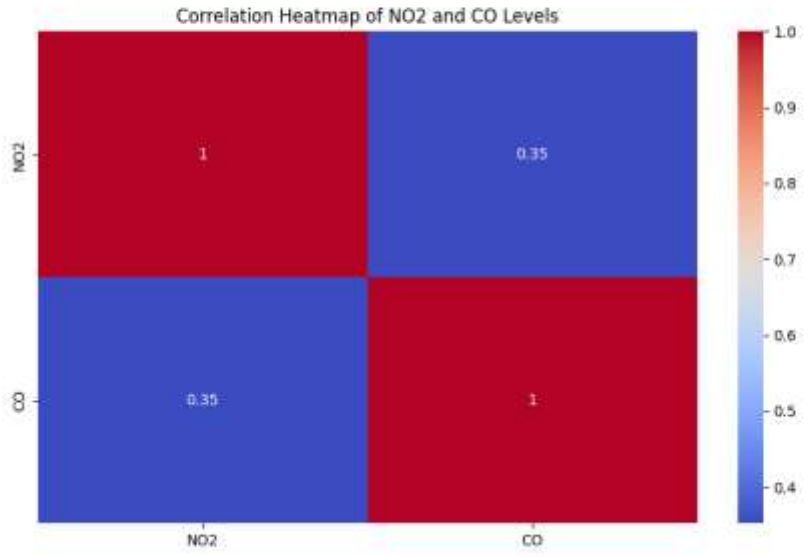
concentrations. This suggest that the majority of observation fall in lower concentration range as long tail is extending towards higher values

```
[55] # Question 7. How do PM2.5 and PM10 levels compare?
     plt.figure(figsize=(10, 6))
     sns.violinplot(data=df[['PM2.5', 'PM10']], inner="quartile")
     plt.title('Comparison of PM2.5 and PM10 Levels')
     plt.xlabel('PM Particles')
     plt.ylabel('Concentration')
     plt.show()
```



Plot show the comparison of PM2.5 and PM10 through violin plot. Each violin represent the observed data with particular particles where width of each violin show the frequency of observation at different concentrations level. Each quartile represents by the horizontal line drawn inside the violin plot.

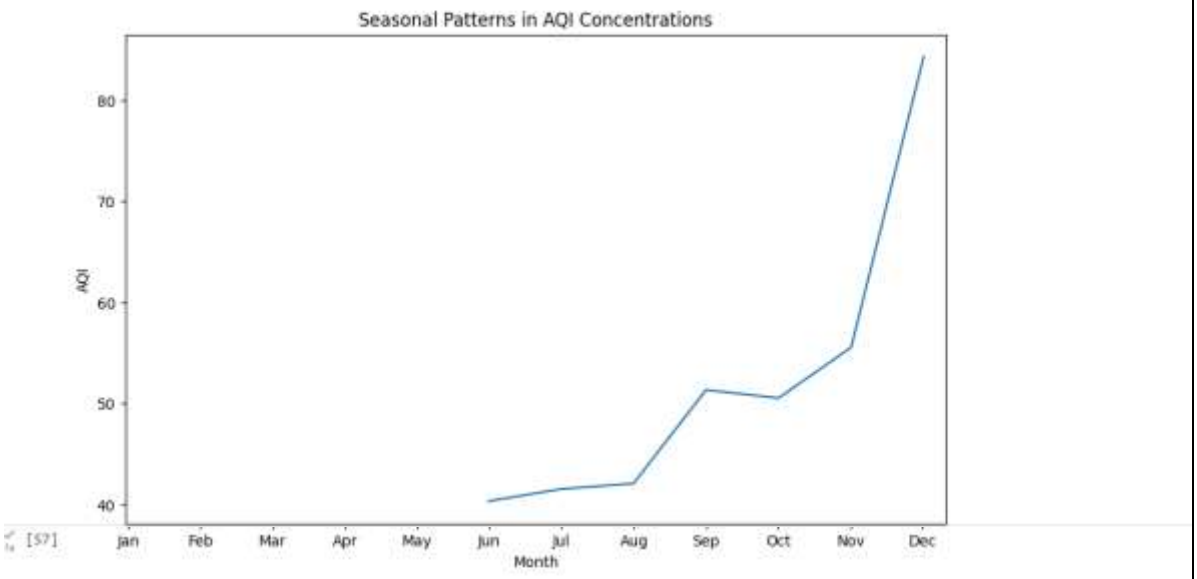
```
[56] #Question 8. How do NO2 and CO levels vary spatially?
     plt.figure(figsize=(10, 6))
     sns.heatmap(data=df[['NO2', 'CO']].corr(), annot=True, cmap='coolwarm')
     plt.title('Correlation Heatmap of NO2 and CO Levels')
     plt.show()
```



Plot show the correlation Heatmap of NO2 and CO levels where correlations coefficient is 0.35 between these two gases which can be state as

moderate positive correlation. This suggest that with higher concentration of CO tend to exhibit higher concentration of NO2 and vice versa.

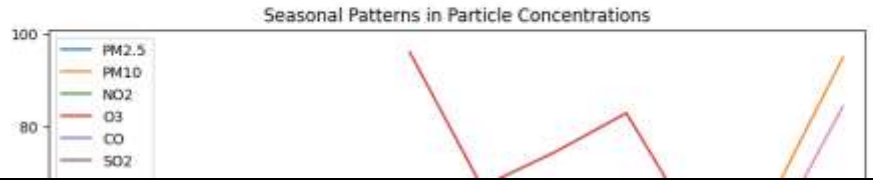
```
[57] # Question 9. IS there any seasonal patterns in AQI concentrations?

     #Extract month temporarily for visualization
     temp_df = df.copy()
     temp_df['Month'] = temp_df['pubtime'].dt.month

     #plotting Sesonal pattern
     plt.figure(figsize=(10, 6))
     sns.lineplot(data=temp_df, x='Month', y='AQI', estimator='mean', err_style=None)
     plt.xticks(range(1, 13), ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec'])
     plt.title('Seasonal Patterns in AQI Concentrations')
     plt.show()
```



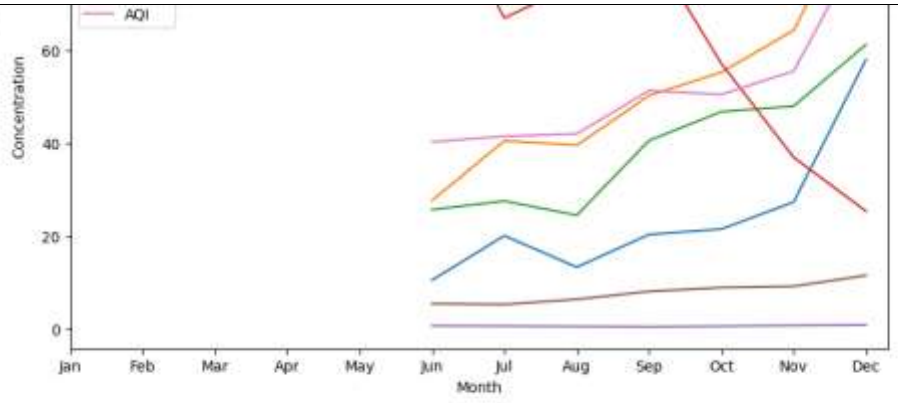
[57]

Plot show the seasonal patterns in AQI concentrations through the line plot. X-axis show the all the month of the year, where Y-axis show the mean of AQI. From the plot it can stated that AQI was slightly increased from June to November and after that it was sky-rocketing and reach the highest level

```
[60] # Question 10. IS there any seasonal patterns in all variable?
     # Extract month temporarily for visualization
     temp_df = df.copy()
     temp_df['Month'] = temp_df['pubtime'].dt.month

     # Plotting seasonal patterns for all particles
     plt.figure(figsize=(10, 6))
     sns.lineplot(data=temp_df, x='Month', y='PM2.5', estimator='mean', err_style=None, label='PM2.5')
     sns.lineplot(data=temp_df, x='Month', y='PM10', estimator='mean', err_style=None, label='PM10')
     sns.lineplot(data=temp_df, x='Month', y='NO2', estimator='mean', err_style=None, label='NO2')
     sns.lineplot(data=temp_df, x='Month', y='O3', estimator='mean', err_style=None, label='O3')
     sns.lineplot(data=temp_df, x='Month', y='CO', estimator='mean', err_style=None, label='CO')
     sns.lineplot(data=temp_df, x='Month', y='SO2', estimator='mean', err_style=None, label='SO2')
     sns.lineplot(data=temp_df, x='Month', y='AQI', estimator='mean', err_style=None, label='AQI')


     plt.xticks(range(1, 13), ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec'])
     plt.title('Seasonal Patterns in Particle Concentrations')
     plt.xlabel('Month')
     plt.ylabel('Concentration')
     plt.legend()
     plt.show()
```



[60]

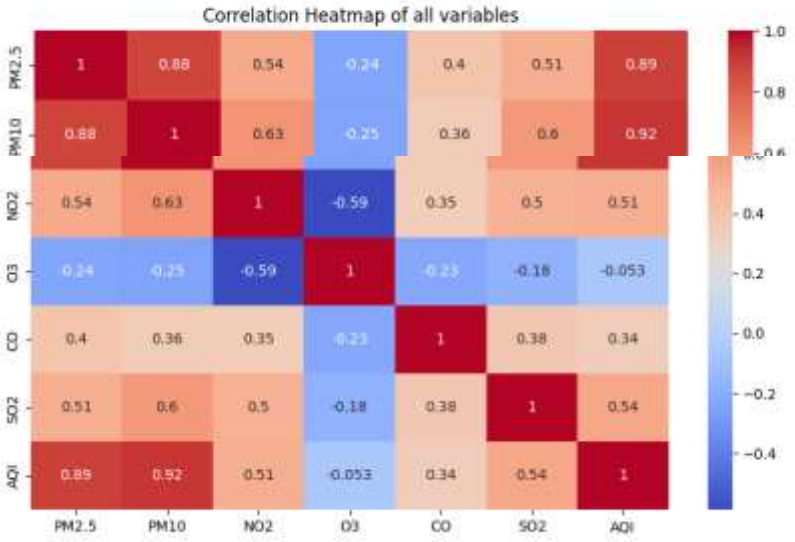


Plot show the seasonal patterns in all seven variables (PM22.5, PM10, NO2, O3, CO, SO2, AQI) with the help of line graph distributed all the month on X-axis and particles concentration on Y-axis. From the graph it can be state that concentration of AQI, PM2.5, PM10, and NO2 was increased in each month towards the ends of year, and O3 concentration was moving downwards in each month towards the ends of year. Whereas the concentration of CO and SO2 was almost same throughout each month.

[61]

```
#Question 10. How do NO2 and CO levels vary spatially?
plt.figure(figsize=(10, 6))
sns.heatmap(data=df[['PM2.5','PM10','NO2','O3', 'CO','SO2','AQI']].corr(), annot=True, cmap='coolwarm')
plt.title('Correlation Heatmap of all variables')
plt.show()
```

[61]



Plot show the correlation Heatmap of all variables levels. Positive correlations are indicated by warmer color red whereas, negative correlations are indicated by cooler color blue. Correlation coefficient closer to 1 indicate the stronger correlation between two variable whereas negative or coefficient close to 0 indicate the weak relationship between two variables.

[66]

```
#File path to save the data
file_path = '/content/drive/MyDrive/Dissertation/clean_data.csv'

# Save DataFrame to CSV file
df.to_csv(file_path, index=False)

print("CSV file saved to Google Drive!")
```

```
CSV file saved to Google Drive!
```

[67]

```
# Loading the clean dataset from drive
df = pd.read_csv('/content/drive/MyDrive/Dissertation/clean_data.csv')
df.head()
```

	oubtime	PM2.5	PM10	NO2	O3	CO	SO2	AQI

0	2020-06-30 00:00:00	11.0	23.0	26.0	70.0	0.7	5.0	23.0
1	2020-06-30 01:00:00	14.0	23.0	29.0	54.0	0.8	5.0	23.0
2	2020-06-30 02:00:00	11.0	23.0	31.0	48.0	0.7	5.0	23.0
3	2020-06-30 03:00:00	12.0	25.0	42.0	26.0	0.7	5.0	25.0
4	2020-06-30 04:00:00	7.0	29.0	36.0	22.0	0.7	5.0	29.0

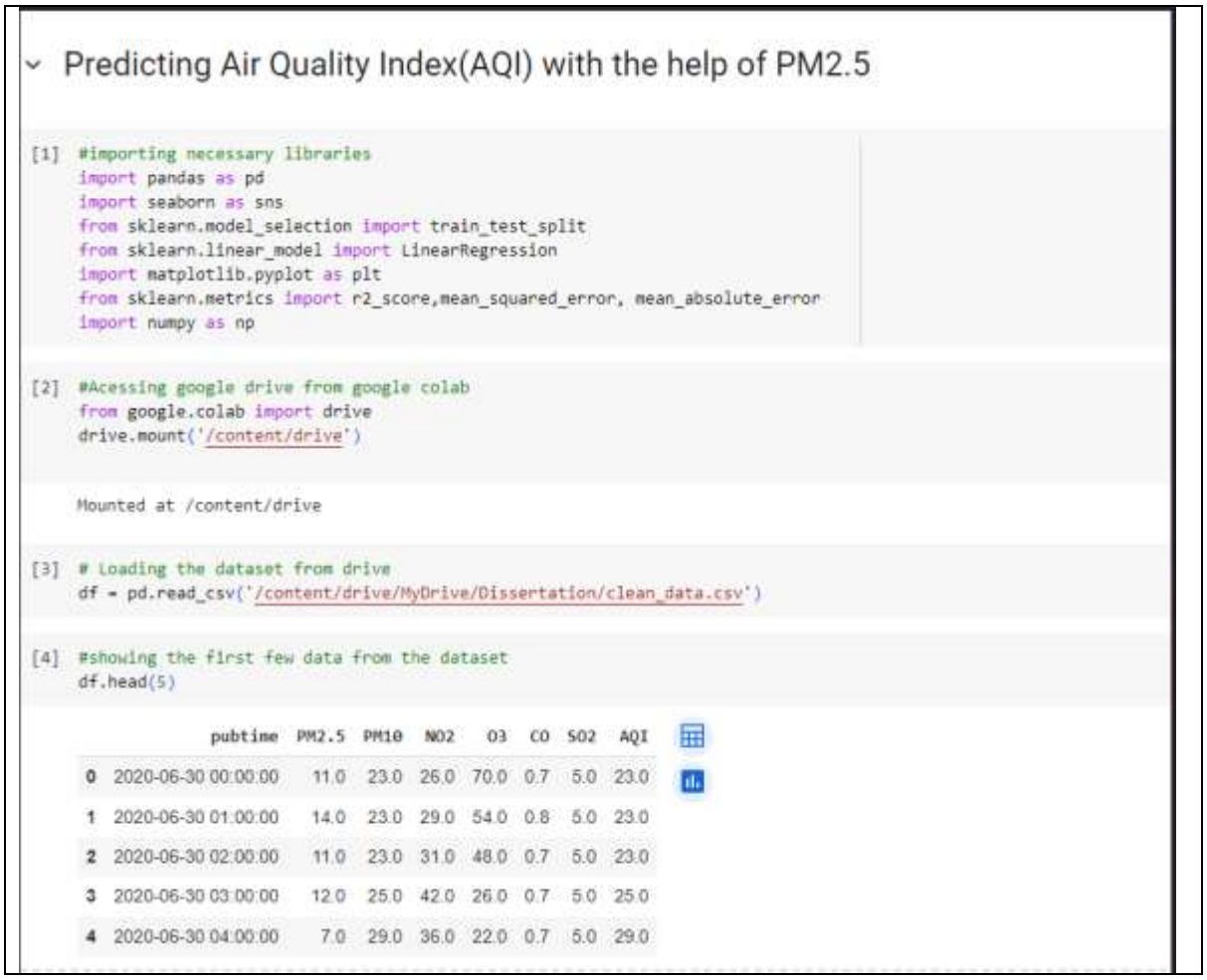
Next steps: View recommended plots

```
[68] df.shape
```

```
(4417, 8)
```

APPENDIX 4: Linear Regression Code

LR Model 1



Predicting Air Quality Index(AQI) with the help of PM2.5

```
[1] #importing necessary libraries
    import pandas as pd
    import seaborn as sns
    from sklearn.model_selection import train_test_split
    from sklearn.linear_model import LinearRegression
    import matplotlib.pyplot as plt
    from sklearn.metrics import r2_score,mean_squared_error, mean_absolute_error
    import numpy as np
```

```
[2] #Acessing google drive from google colab
    from google.colab import drive
    drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
[3] # Loading the dataset from drive
    df = pd.read_csv('/content/drive/MyDrive/Dissertation/clean_data.csv')
```

```
[4] #showing the first few data from the dataset
    df.head(5)
```

	pubtime	PM2.5	PM10	NO2	O3	CO	SO2	AQI
0	2020-06-30 00:00:00	11.0	23.0	26.0	70.0	0.7	5.0	23.0
1	2020-06-30 01:00:00	14.0	23.0	29.0	54.0	0.8	5.0	23.0
2	2020-06-30 02:00:00	11.0	23.0	31.0	48.0	0.7	5.0	23.0
3	2020-06-30 03:00:00	12.0	25.0	42.0	26.0	0.7	5.0	25.0
4	2020-06-30 04:00:00	7.0	29.0	36.0	22.0	0.7	5.0	29.0

```
[5] df.shape
```

```
(4417, 8)
```

Linear Regression Model

```
[6] #dropping all column except feature column PM2.5 and store in x
    x= df.drop(columns = ['pubtime','AQI','PM10','NO2','O3','CO','SO2'])
    x
```

	PM2.5
0	11.0
1	14.0
2	11.0
3	12.0
4	7.0
...	...
4412	21.0
4413	22.0
4414	26.0
4415	24.0
4416	23.0

4417 rows × 1 columns

Next steps: View recommended plots

```
[7] #storing target column in y
    y = df['AQI']
    y
```

```
[7] 0       23.0
    1       23.0
    2       23.0
    3       25.0
    4       29.0
            ...
    4412    55.0
    4413    58.0
    4414    57.0
    4415    54.0
    4416    56.0
    Name: AQI, Length: 4417, dtype: float64
```

```
[8] #splitting test train data by using train_test_split libraries from sklearn
    x_train, x_test, y_train, y_test= train_test_split(x, y, test_size= 0.2,random_state=0)
```

```
[9] #fitting the model
    lr= LinearRegression()
    lr.fit(x_train,y_train)
```

▾ LinearRegression
LinearRegression()

```
[10] #calculating intercept c
     c = lr.intercept_
     c
```

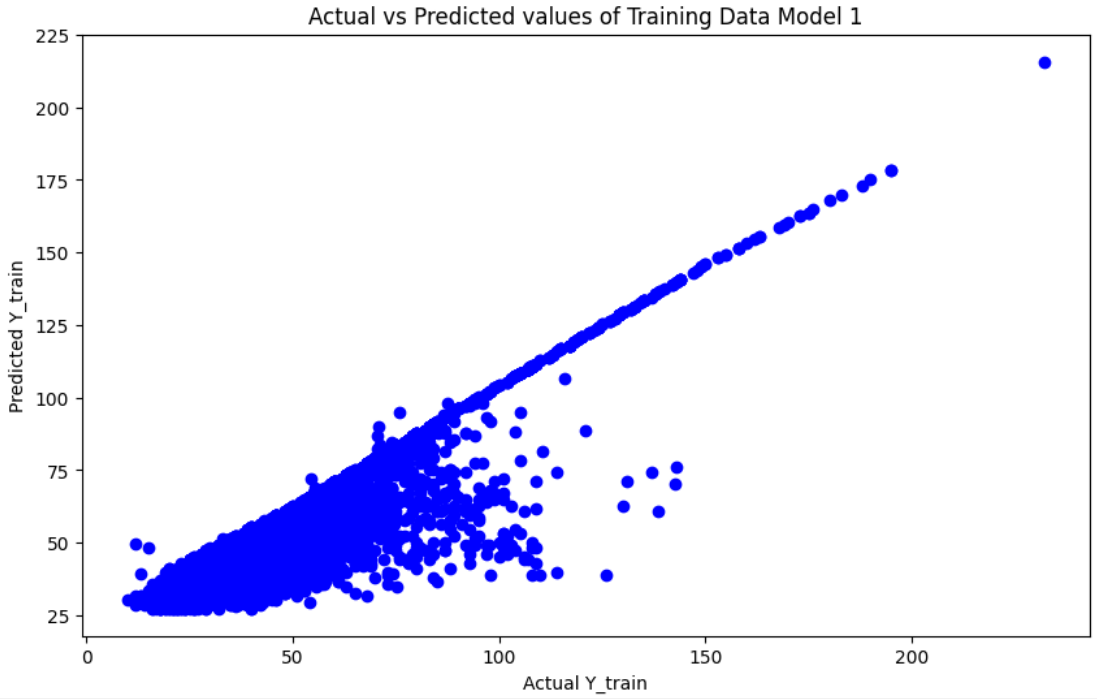
```
     26.340297259337067
```

```
[11] # Calculating coefficient m
     m = lr.coef_
     m
```

```
     array([1.04081298])
```

```
[12] #predicting y training values
     y_pred_train = lr.predict(x_train)
     y_pred_train
```

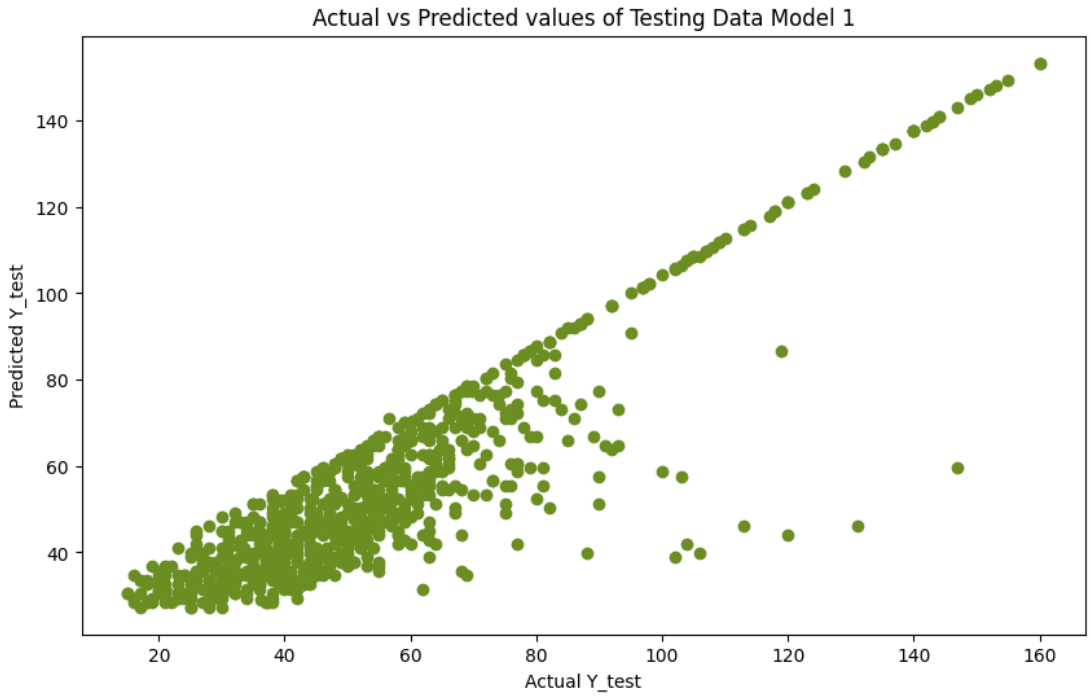
```
[13] # Plotting Training Actual values vs Predicted Values
     plt.figure(figsize=(10, 6))
     # Scatter plot of training data (actual values)
     plt.scatter(y_train, y_pred_train, color='blue')
     # Labels
     plt.xlabel('Actual Y_train')
     plt.ylabel('Predicted Y_train')
     plt.title('Actual vs Predicted values of Training Data Model 1')
     # Display the plot
     plt.show()
```



```
[14] #Prdicitng y testing values
     y_pred_test = lr.predict(x_test)
     y_pred_test
```

```
array([[ 38.83005302,  72.13606839,  61.72793858,  48.19736984,
         40.91167898,  60.6871256 ,  70.05444242,  52.36062176,
         52.36062176,  36.74842706,  36.74842706,  54.44224772,
         36.74842706,  66.93200348,  75.25850733,  45.0749309 ,
         32.0566373 ,  55.4830607 ,  47.15655686,  57.56468666,
         61.72793858,  57.56468666,  31.54436216,  37.78924004,
        108.56452269,  34.6668011 ,  41.95249196,  71.0952554 ,
        118.97265249,  29.4627362 ,  34.6668011 ,  56.52387368,
         51.84021527,  41.95249196,  37.78924004,  73.17688137,
         77.34013329,  44.03411792,  53.40143474,  31.54436216,
         47.15655686,  32.58517514,  72.13606839,  45.0749309 ,
         44.03411792,  67.97281646,  84.62582415,  39.870866  ,
         31.54436216,  44.03411792,  29.4627362 ,  60.6871256 ,
         66.93200348,  31.54436216,  56.52387368,  87.74826309,
         44.03411792,  62.76875156,  71.0952554 , 148.11541593,
         80.46257223,  76.29932031,  30.50354918,  45.0749309 ,
         37.78924004,  53.40143474,  30.50354918, 101.27883183,
         71.0952554 ,  47.15655686,  47.15655686,  35.70761408,
         59.64631262,  88.78907607,  39.870866  ,  29.4627362 ,
        101.27883183,  49.23818282, 114.80940057,  74.21769435,
         31.54436216,  40.91167898,  31.02395567,  80.46257223,
         64.85037752,  61.72793858,  57.56468666,  36.74842706,
         46.11574388,  41.95249196,  75.25850733,  92.95232799,
         65.8911905 ,  35.70761408,  52.23052014,  63.80956454,
         51.31980878,  33.62598812,  35.70761408,  69.01362944,
         85.66663713,  38.83005302,  52.36062176,  38.83005302,
         92.95232799,  65.8911905 ,  47.15655686,  47.15655686,
         35.70761408,  59.64631262,  39.870866  ,  34.6668011 ,
         33.62598812,  66.93200348,  71.0952554 ,  52.36062176,
         38.83005302,  29.4627362 , 137.70728613,  74.21769435,
         62.76875156,  45.0749309 ,  52.36062176,  62.76875156,
         46.11574388,  45.0749309 ,  79.42175925,  97.11557991,
         30.50354918,  46.11574388,  40.91167898, 121.05427845,
         37.78924004,  49.23818282,  39.870866  ,  45.0749309 ,
         45.0749309 ,  49.23818282,  56.52387368,  76.29932031,
        144.99297699,  67.97281646,  70.05444242,  57.56468666,
         38.83005302,  41.95249196,  46.11574388,  58.60549964,
         38.83005302,  40.91167898,  49.23818282,  56.52387368,
         52.36062176,  29.85304107,  62.76875156,  50.2789958 ,
         38.83005302,  48.19736984,  50.2789958 ,  33.62598812,
         56.52387368,  32.58517514, 130.42159527,  33.62598812,
         47.15655686,  35.70761408,  92.95232799,  97.11557991,
```

```
[15] #Plotting  Testing Actual values vs Predicted Vlaues
     plt.figure(figsize=(10, 6))
     # Scatter plot of testing data
     plt.scatter(y_test, y_pred_test, color ='olivedrab')
     # Labels
     plt.xlabel('Actual Y_test')
     plt.ylabel('Predicted Y_test')
     plt.title('Actual vs Predicted values of Testing Data Model 1')
     # Display the plot
     plt.show()
```

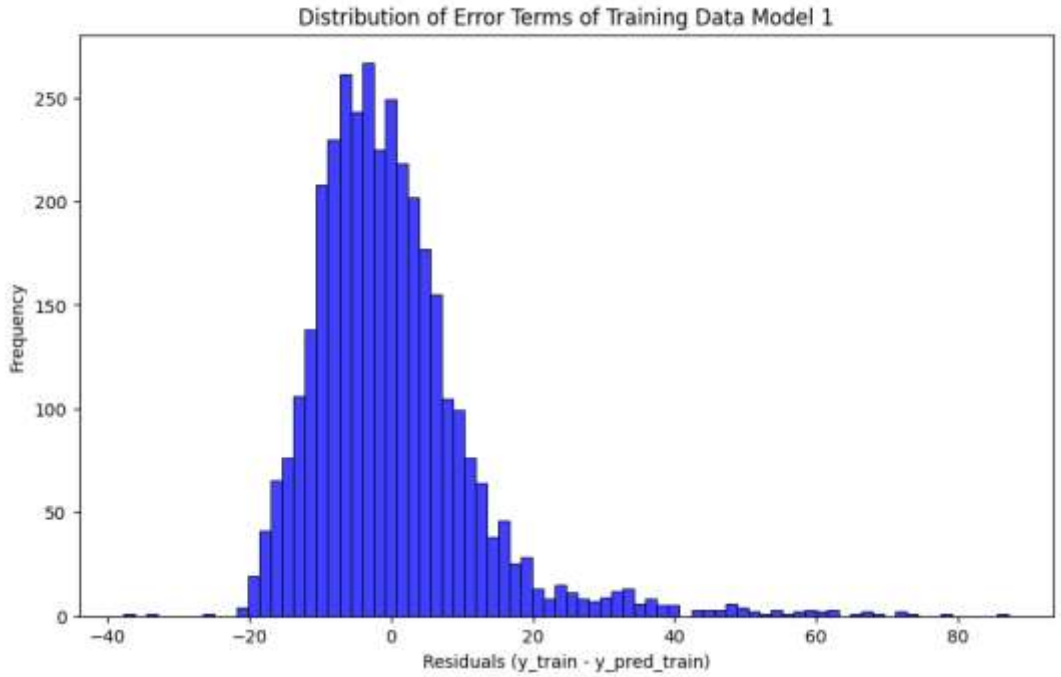



```
[16] #Calculating training data error term for linear regression
     res_train = (y_train - y_pred_train)
     res_train
```

```
4339    72.695558
4275     1.537592
3849    -5.156393
883     -4.115744
134     -5.503549
          ...
1033    -7.156557
3264     9.965882
1653    -2.442248
2607    -9.421923
2732   -11.013629
Name: AQI, Length: 3533, dtype: float64
```

```
[17] # Plotting histogram to show the error terms for trainig data
     plt.figure(figsize=(10, 6))
     # Creating the histogram plot using Seaborn
     sns.histplot(res_train , color = 'blue')

     # Label the plot
     plt.title('Distribution of Error Terms of Training Data Model 1')
     plt.xlabel('Residuals (y_train - y_pred_train)')
     plt.ylabel('Frequency')
     plt.show()
```

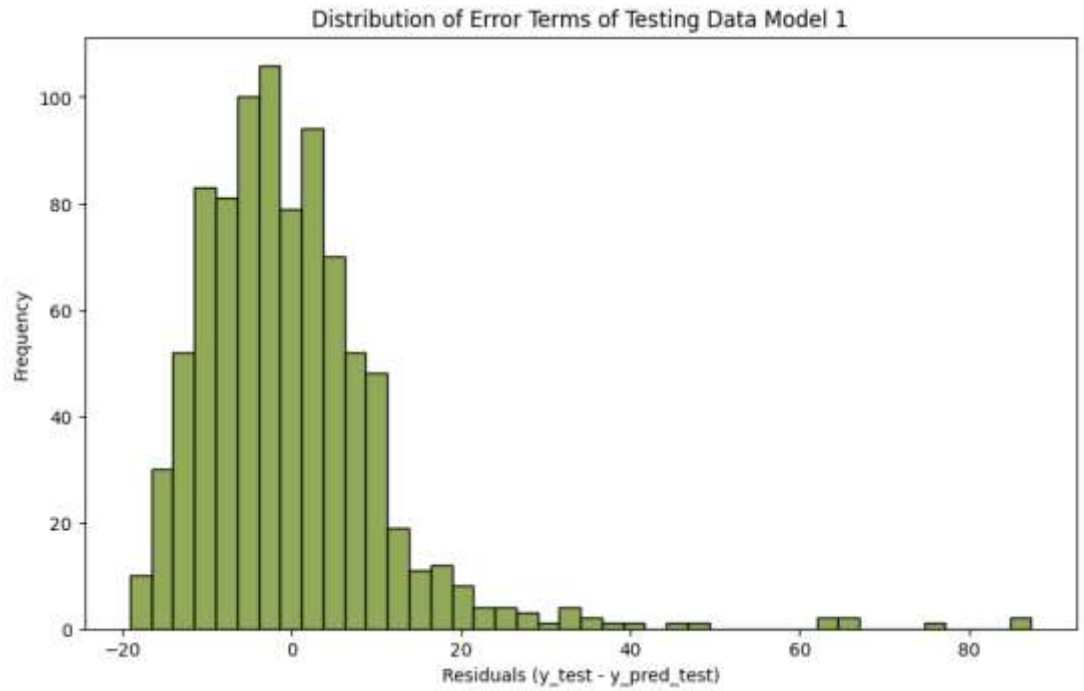
[17]



[18]
```
#Calculating testing data error term for linear regression
res_test = (y_test - y_pred_test)
res_test
```

```
2917    -7.074931
1457    -3.585175
1669    62.047508
2013    -2.483061
4303     2.863932
           ...
2527    -3.095255
2339     1.496451
```

```
[19] # Plotting histogram to show the error terms for testing data
     plt.figure(figsize=(10, 6))
     # Creating the histogram plot using Seaborn
     sns.histplot(res_test , color ='olivedrab')

     # Label the plot
     plt.title('Distribution of Error Terms of Testing Data Model 1')
     plt.xlabel('Residuals (y_test - y_pred_test)')
     plt.ylabel('Frequency')
     plt.show()
```



Evaluation of Linear Regression

Training Data

```
[20] #Calculating Mean Squared Error of Training Data
     mse = mean_squared_error(y_train, y_pred_train)
     print("Mean Squared Error (MSE) of Training Data:", mse)
```

```
Mean Squared Error (MSE) of Training Data: 137.09214220703785
```

```
[21] #calculating Root Mean Squared Error of Training Data
     rmse = np.sqrt(mse)
     print("Root Mean Squared Error (RMSE) of Training Data:", rmse)
```

```
Root Mean Squared Error (RMSE) of Training Data: 11.708635369121282
```

```
[22] #Calculating Mean Absolute Error of Training Data
     mae = mean_absolute_error(y_train, y_pred_train)
     print("Mean Absolute Error (MAE) of Training Data:", mae)
```

```
Mean Absolute Error (MAE) of Training Data: 8.180846600040727
```

```
[23] #Calculating R-Squared of Training Data
     rsquare =  r2_score(y_train, y_pred_train)
     print("R-sqaured of Training Data:", rsquare)
```

```
R-sqaured of Training Data: 0.7959447841318095
```

Testing Data

```
[24] #Calculating Mean Squared Error
     mse = mean_squared_error(y_test, y_pred_test)
     print("Mean Squared Error (MSE) of Testing Data :", mse)
```

```
Mean Squared Error (MSE) of Testing Data : 132.2206699408324
```

```
[25] #calculating Root Mean Squared Error
     rmse = np.sqrt(mse)
     print("Root Mean Squared Error (RMSE) of Testing Data:", rmse)
```

```
Root Mean Squared Error (RMSE) of Testing Data: 11.498724709324613
```

```
[26] #Calculating Mean Absolute Error
     mae = mean_absolute_error(y_test, y_pred_test)
     print("Mean Absolute Error (MAE) of Testing Data:", mae)
```

```
Mean Absolute Error (MAE) of Testing Data: 7.900466019689688
```

```
#Calculating R-Squared  of Testing Data
rsquare =  r2_score(y_test, y_pred_test)
print("R-sqaured of Testing Data:", rsquare)
```

```
R-sqaured of Testing Data: 0.799659153879728
```

LR Model 2

Predicting Air Quality Index(AQI) with the help of PM2.5 and PM10

```
[1] #Importing necessary libraries
    import pandas as pd
    import seaborn as sns
    from sklearn.model_selection import train_test_split
    from sklearn.linear_model import LinearRegression
    import matplotlib.pyplot as plt
    from sklearn.metrics import r2_score,mean_squared_error, mean_absolute_error
    import numpy as np
```

```
[2] #Acessing google drive from google colab
    from google.colab import drive
    drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
[47] # Loading the dataset from drive
     df = pd.read_csv('/content/drive/MyDrive/Dissertation/clean_data.csv')
```

```
[48] #showing the first few data from the dataset
     df.head(5)
```

	pubtime	PM2.5	PM10	NO2	O3	CO	SO2	AQI
0	2020-06-30 00:00:00	11.0	23.0	26.0	70.0	0.7	5.0	23.0
1	2020-06-30 01:00:00	14.0	23.0	29.0	54.0	0.8	5.0	23.0
2	2020-06-30 02:00:00	11.0	23.0	31.0	48.0	0.7	5.0	23.0
3	2020-06-30 03:00:00	12.0	25.0	42.0	26.0	0.7	5.0	25.0
4	2020-06-30 04:00:00	7.0	29.0	36.0	22.0	0.7	5.0	29.0

```
[49] df.shape
```

```
(4417, 8)
```

Linear Regression Model

```
[50] #dropping all column except feature column PM2.5, and PM10 and store in x
     x= df.drop(columns = ['pubtime','AQI','NO2','O3','CO','SO2'])
     x
```

	PM2.5	PM10
0	11.0	23.0
1	14.0	23.0
2	11.0	23.0
3	12.0	25.0
4	7.0	29.0
...	...	...
4412	21.0	59.0
4413	22.0	65.0
4414	26.0	64.0
4415	24.0	58.0
4416	23.0	61.0

4417 rows × 2 columns

Next steps: View recommended plots

```
[51] #storing target column in y
     y = df['AQI']
     y
```

```
0       23.0
1       23.0
2       23.0
3       25.0
4       29.0
        ...
4412    55.0
4413    58.0
4414    57.0
4415    54.0
4416    56.0
Name: AQI, Length: 4417, dtype: float64
```

```
[52] #splitting test train data by using train_test_split libraries from sklearn
     x_train, x_test, y_train, y_test= train_test_split(x, y, test_size= 0.2,random_state=0)
```

```
[53] #fitting the model
     lr= LinearRegression()
     lr.fit(x_train,y_train)
```

```
▾ LinearRegression
LinearRegression()
```

```
[54] #calculating intercept c
     c = lr.intercept_
     c
```

```
15.780111766846716
```

```
[55] # Calculating coefficient m
     m = lr.coef_
     m
```

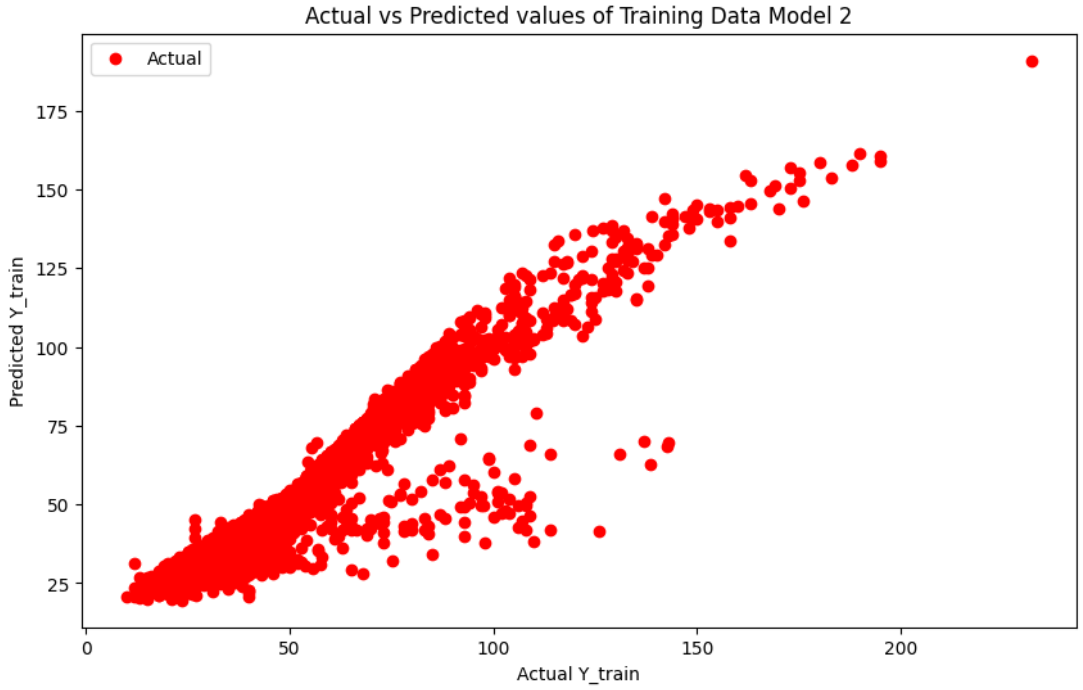
```
array([0.43999371, 0.46367613])
```

```
[56] #predicting y training values
     y_pred_train = lr.predict(x_train)
     y_pred_train
```

```
array([ 68.57188134, 134.40765759,  87.87052966, ...,  52.23477692,
        22.22421276,  63.95880245])
```

```
[57] # Plotting Training Actual values vs Predicted Values
     plt.figure(figsize=(10, 6))

     # Scatter plot of training data (actual values)
     plt.scatter(y_train, y_pred_train, color='red', label='Actual')

     # Labels
     plt.xlabel('Actual Y_train')
     plt.ylabel('Predicted Y_train')
     plt.title('Actual vs Predicted values of Training Data Model 2')
     plt.legend()

     # Display the plot
     plt.show()
```

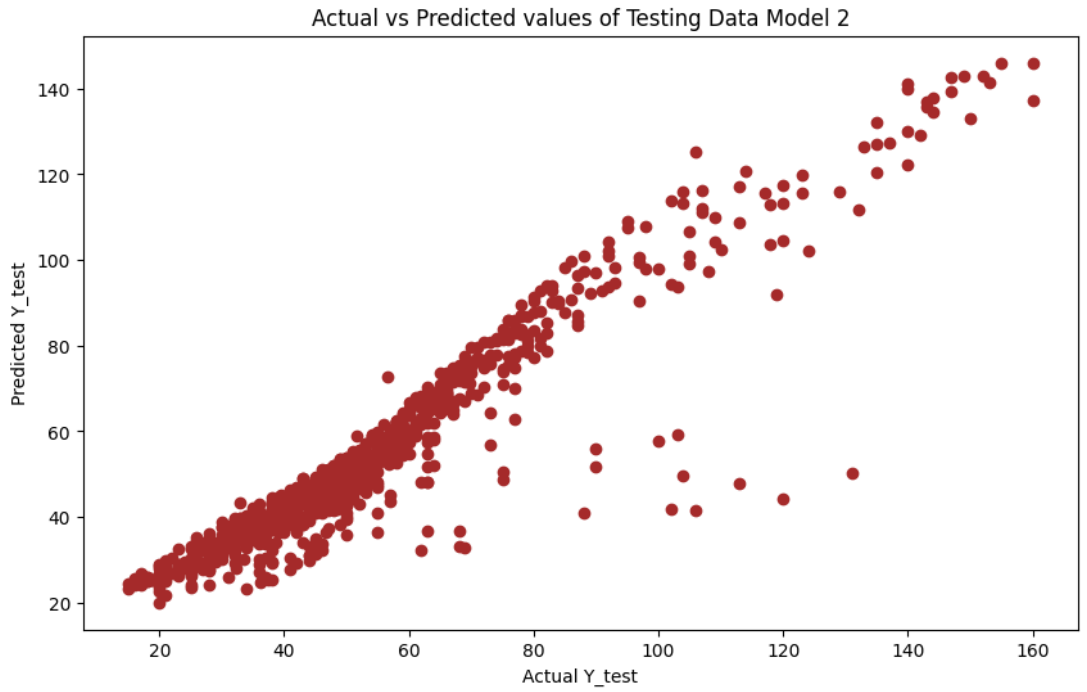
[57]



[58]
```
#Prdicitng y testing values
y_pred_test = lr.predict(x_test)
y_pred_test
```

```
array([[ 42.38913832,  75.94333459,  52.06899999,  44.03070108],
       [ 42.80544961,  78.05854572,  63.00776777,  55.9915508 ],
       [ 55.52787467,  48.00061673,  40.1181225 ,  49.91639626],
       [ 32.69930441,  64.46984342,  73.55390668,  41.78336768],
       [ 25.61545782,  52.67477063,  39.88129832,  48.91799675],
       [ 65.97928391,  45.67226383,  22.61684163,  44.73120139],
       [101.08031326,  35.0650499 ,  58.08307951,  70.86657958],
       [112.89906847,  25.44626325,  29.03726021,  50.33270756],
       [ 44.75024584,  52.54896504,  29.16709782,  72.69127152],
```

```
#Plotting  Testing Actual values vs Predicted Vlaues
plt.figure(figsize=(10, 6))
# Scatter plot of testing data
plt.scatter(y_test, y_pred_test, color ='brown')
# Labels
plt.xlabel('Actual Y_test')
plt.ylabel('Predicted Y_test')
plt.title('Actual vs Predicted values of Testing Data Model 2')
# Display the plot
plt.show()
```

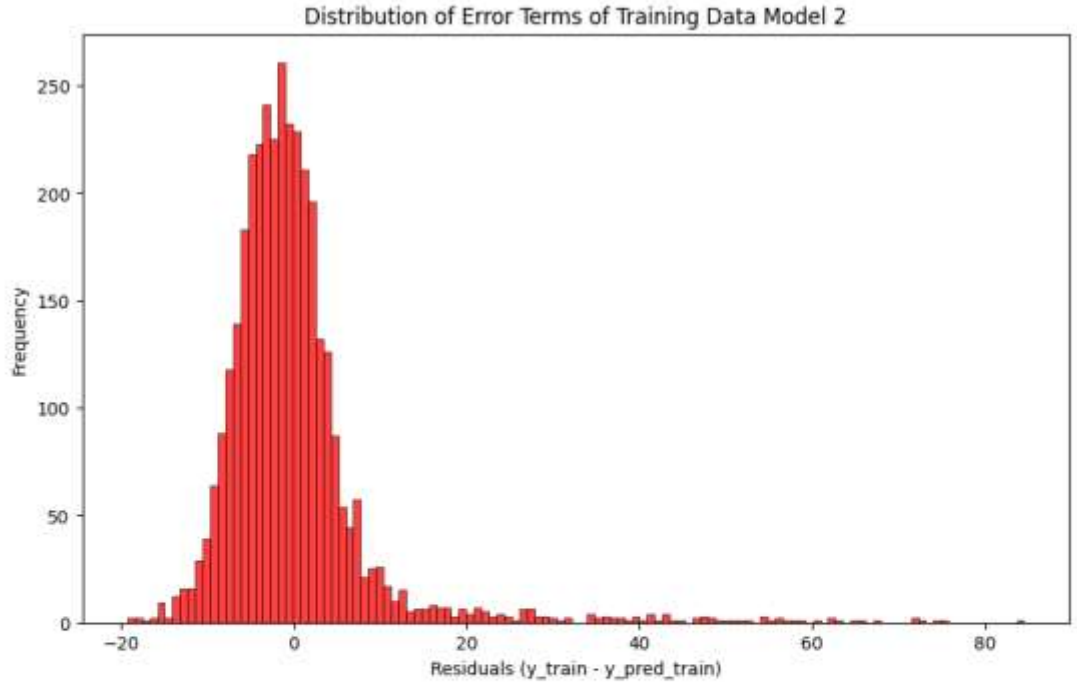



```
[60] #Calculating training data error term for linear regression
     res_train = (y_train - y_pred_train)
     res_train
```

```
4339    74.178119
4275    -1.407658
3849     5.129470
883     -1.614390
134      3.286828
          ...
1033    -3.127031
3264     3.846780
1653    -0.234777
2607    -3.224213
2732    -5.958802
Name: AQI, Length: 3533, dtype: float64
```

```
[61] # Plotting histogram to show the error terms for trainig data
     plt.figure(figsize=(10, 6))
     # Creating the histogram plot using Seaborn
     sns.histplot(res_train , color = 'red')

     # Label the plot
     plt.title('Distribution of Error Terms of Training Data Model 2')
     plt.xlabel('Residuals (y_train - y_pred_train)')
     plt.ylabel('Frequency')
     plt.show()
```

[61]

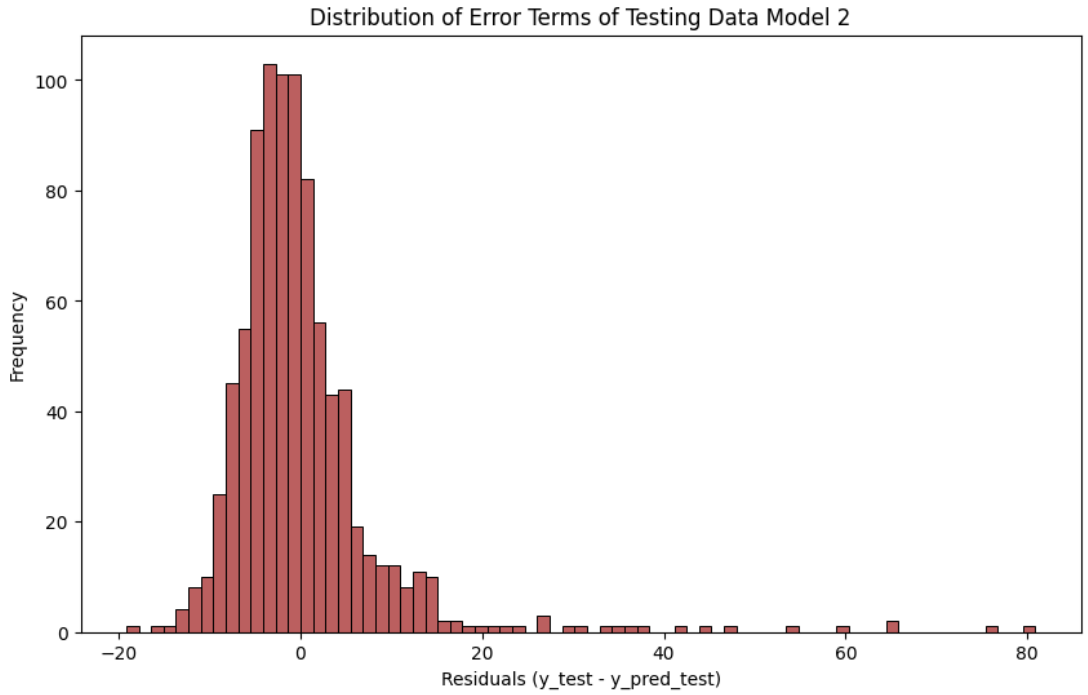


[62]
```
#Calculating testing data error term for linear regression
res_test = (y_test - y_pred_test)
res_test
```

```
2917    -3.319692
1457    -1.939330
1669    54.263091
2013    -1.065799
4303    -6.507448
          ...
2527    -6.575989
2339    -0.377723
```

```
[62] 3354    -1.046224
     3019    -6.112312
     Name: AQI, Length: 884, dtype: float64
```

```
# Plotting histogram to show the error terms for testing data
plt.figure(figsize=(10, 6))
# Creating the histogram plot using Seaborn
sns.histplot(res_test , color ='brown')
# Label the plot
plt.title('Distribution of Error Terms of Testing Data Model 2')
plt.xlabel('Residuals (y_test - y_pred_test)')
plt.ylabel('Frequency')
plt.show()
```



Evaluation of Linear Regression

Training Data

```
[64] #Calculating Mean Squared Error of Training Data
     mse = mean_squared_error(y_train, y_pred_train)
     print("Mean Squared Error (MSE) of Training Data:", mse)
```

```
Mean Squared Error (MSE) of Training Data: 83.78276854729341
```

```
[65] #calculating Root Mean Squared Error of Training Data
     rmse = np.sqrt(mse)
     print("Root Mean Squared Error (RMSE) of Training Data:", rmse)
```

```
Root Mean Squared Error (RMSE) of Training Data: 9.153292770762521
```

```
[66] #Calculating Mean Absolute Error of Training Data
     mae = mean_absolute_error(y_train, y_pred_train)
     print("Mean Absolute Error (MAE) of Training Data:", mae)
```

```
Mean Absolute Error (MAE) of Training Data: 5.30963259190655
```

```
[67] #Calculating R-Squared of Training Data
     rsquare =  r2_score(y_train, y_pred_train)
     print("R-sqaured of Training Data:", rsquare)
```

```
R-sqaured of Training Data: 0.8752932834317113
```

Testing Data

```
[68] #Calculating Mean Squared Error
     mse = mean_squared_error(y_test, y_pred_test)
     print("Mean Squared Error (MSE) of Testing Data :", mse)
```

```
Mean Squared Error (MSE) of Testing Data : 79.53767552774848
```

```
[69] #calculating Root Mean Squared Error
     rmse = np.sqrt(mse)
     print("Root Mean Squared Error (RMSE) of Testing Data:", rmse)
```

```
Root Mean Squared Error (RMSE) of Testing Data: 8.918389738498115
```

```
[70] #Calculating Mean Absolute Error
     mae = mean_absolute_error(y_test, y_pred_test)
     print("Mean Absolute Error (MAE) of Testing Data:", mae)
```

```
Mean Absolute Error (MAE) of Testing Data: 5.210566166369225
```

```
[71] #Calculating R-Squared  of Testing Data
     rsquare =  r2_score(y_test, y_pred_test)
     print("R-sqaured of Testing Data:", rsquare)
```

```
R-sqaured of Testing Data: 0.8794844616896935
```

LR Model 3

Predicting Air Quality Index(AQI) with the help of NO2, O3, CO, and SO2

```
[1]  #importing necessary libraries
     import pandas as pd
     import seaborn as sns
     from sklearn.model_selection import train_test_split
     from sklearn.linear_model import LinearRegression
     import matplotlib.pyplot as plt
     from sklearn.metrics import r2_score,mean_squared_error, mean_absolute_error
     import numpy as np
```

```
[2]  #Acessing google drive from google colab
     from google.colab import drive
     drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
[3]  # Loading the dataset from drive
     df = pd.read_csv('/content/drive/MyDrive/Dissertation/clean_data.csv')
```

```
[4]  #showing the first few data from the dataset
     df.head(5)
```

	pubtime	PM2.5	PM10	NO2	O3	CO	SO2	AQI
0	2020-06-30 00:00:00	11.0	23.0	26.0	70.0	0.7	5.0	23.0
1	2020-06-30 01:00:00	14.0	23.0	29.0	54.0	0.8	5.0	23.0
2	2020-06-30 02:00:00	11.0	23.0	31.0	48.0	0.7	5.0	23.0
3	2020-06-30 03:00:00	12.0	25.0	42.0	26.0	0.7	5.0	25.0
4	2020-06-30 04:00:00	7.0	29.0	36.0	22.0	0.7	5.0	29.0

```
[5] df.shape
```

```
(4417, 8)
```

Linear Regression Model

```
[6] #dropping all column except feature column No2, O3, CO, and SO2 and store in x
    x= df.drop(columns = ['pubtime','AQI','PM2.5','PM10'])
    x
```

	NO2	O3	CO	SO2
0	26.0	70.0	0.7	5.0
1	29.0	54.0	0.8	5.0
2	31.0	48.0	0.7	5.0
3	42.0	26.0	0.7	5.0
4	36.0	22.0	0.7	5.0
...	...	...	...	...
4412	35.0	42.0	0.8	6.0
4413	35.0	40.0	0.8	6.0
4414	30.0	43.0	0.8	6.0
4415	30.0	42.0	0.8	6.0
4416	25.0	46.0	0.8	7.0

4417 rows × 4 columns

Next steps: View recommended plots

```
[7] #storing target column in y
    y = df['AQI']
    y
```

```
[7]  0       23.0
     1       23.0
     2       23.0
     3       25.0
     4       29.0
             ...
     4412    55.0
     4413    58.0
     4414    57.0
     4415    54.0
     4416    56.0
     Name: AQI, Length: 4417, dtype: float64
```

```
[8]  #splitting test train data by using train_test_split libraries from sklearn
     x_train, x_test, y_train, y_test= train_test_split(x, y, test_size= 0.2,random_state=0)
```

```
[9]  #fitting the model
     lr= LinearRegression()
     lr.fit(x_train,y_train)
```

```
▾ LinearRegression
LinearRegression()
```

```
[10] #calculating intercept c
     c = lr.intercept_
     c
```

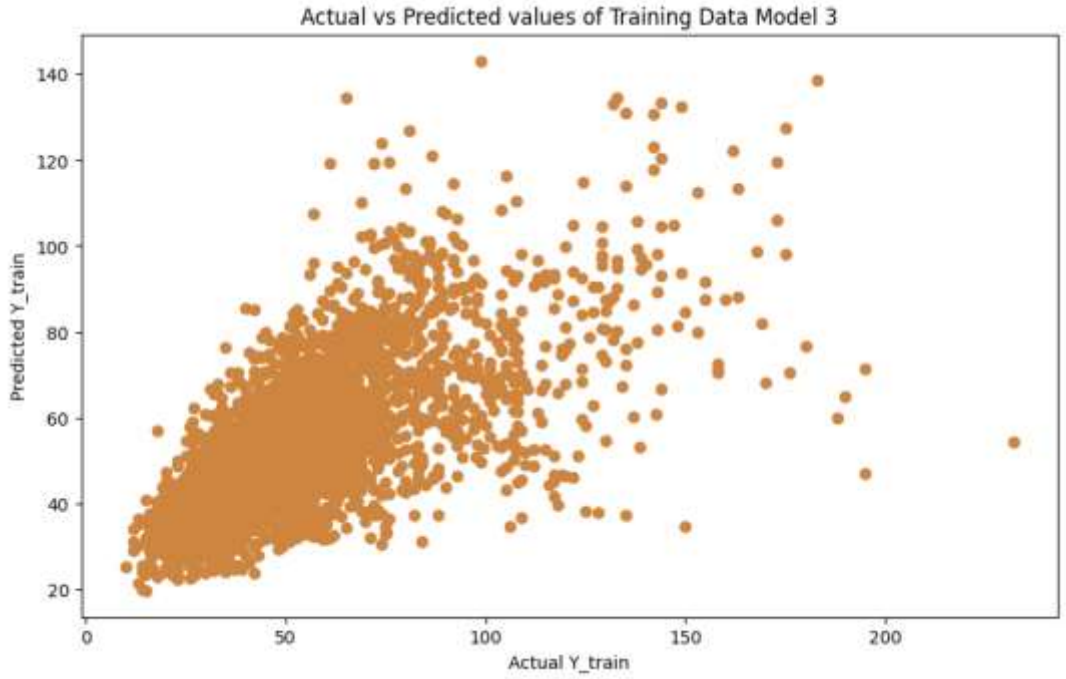
```
-6.101049643142872
```

```
[11] # Calculating coefficient m
     m = lr.coef_
     m
```

```
array([ 0.52839932,  0.17889162, 10.62499604,  2.40649915])
```

```
[12] #predicting y training values
     y_pred_train = lr.predict(x_train)
     y_pred_train
```

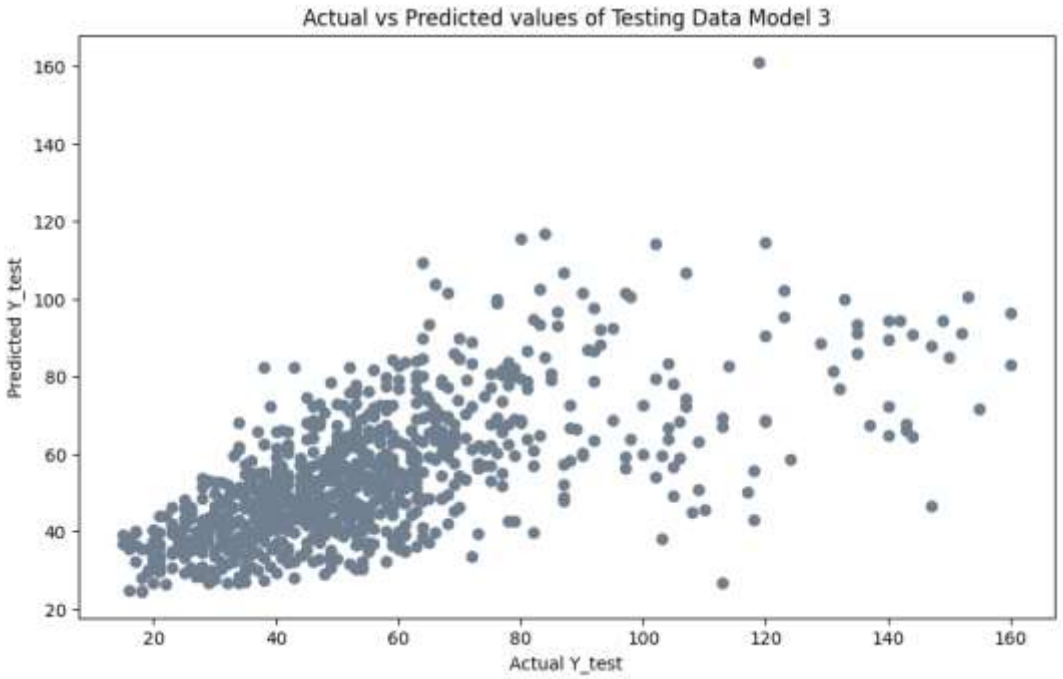
```python
# Plotting Training Actual values vs Predicted Values
plt.figure(figsize=(10, 6))
# Scatter plot of training data (actual values)
plt.scatter(y_train, y_pred_train, color='peru')
# Labels
plt.xlabel('Actual Y_train')
plt.ylabel('Predicted Y_train')
plt.title('Actual vs Predicted values of Training Data Model 3')
# Display the plot
plt.show()
```



```
#Prdicitng y testing values
y_pred_test = lr.predict(x_test)
y_pred_test
```

```
array([ 62.4626914 ,  37.31576964,  66.64779799,  77.75925179,
        53.19200722,  46.30171177,  38.9516803 ,  42.92684317,
        82.75718836,  36.81142572,  27.00548858,  41.50474013,
        38.22728646,  39.37588748,  38.24825968,  63.88988254,
        82.48904162,  51.54492318,  46.15529905,  30.79476313,
        30.89457737,  67.47644352,  54.45054555,  79.90365084,
        48.77698995,  62.63685945,  71.45361231,  41.80071372,
        54.04153909,  34.84931881,  41.5594711 , 102.12572567,
        68.99409864,  86.48233943,  54.29098223,  44.7935684 ,
        54.34943557,  40.97248736,  72.33857068,  46.8892915 ,
        56.79733983,  93.11158369,  63.36275163,  54.04571089,
        33.7441899 ,  48.54489652,  49.11637457,  68.24806405,
        47.26785581,  80.79375839,  40.27802516,  43.16107222,
        33.46341703,  45.67268273,  58.10470556,  68.12929451,
        43.68394585,  55.09984207,  47.774747  ,  34.76656326,
        33.61620314,  57.03781117,  57.64946317,  65.93771297,
        39.26753399,  69.18852467,  46.33791304,  59.4519159 ,
        45.43024967,  30.22296323,  47.55075524,  50.33976055,
        38.79908635,  66.04835003, 114.25933139,  57.58466866,
        67.60185569,  26.6171777 ,  40.96729398,  26.27337097,
        35.14023609,  84.60441488,  51.07438121,  72.64027257,
        36.31925499,  84.97004809,  37.14566116,  60.05792403,
        43.97097147,  51.77280061,  60.8315947 ,  37.17490993,
        53.3536922 ,  74.5317698 ,  61.40314778,  32.08576348,
        64.6442691 ,  72.26663795,  47.30570338,  40.52602811,
        62.42909555,  28.00825598,  68.02654312,  51.14886121,
        78.95516149,  37.51614207,  46.88878392,  46.66794586,
        35.12181613,  35.77687915,  43.28016011,  54.98371616,
        41.72678551,  54.23602132,  44.80751759,  51.26231967,
        78.64637603,  64.91299985,  34.53934137,  63.61279426,
        36.18684611,  43.09796682,  68.78362002,  72.23596237,
        55.13155569,  24.67107012,  63.16678444,  52.20535487,
        35.22750661,  78.94661179,  50.3036357 ,  40.75732294,
        47.00433379,  60.81128901,  33.40420929,  58.07433633,
        58.91656653,  49.57656353,  79.11680226,  41.81524203,
        70.27019418,  38.84299451,  68.30766167,  38.63417125,
        72.81211284,  50.63019158,  48.80119329,  37.15908589,
        37.01201763,  58.80363253,  59.99675458,  32.22332154,
        57.45349598,  66.42876324,  48.53113949,  52.77191718,
        34.94549909,  63.29219661,  57.74319994,  69.88886312,
        35.01941042,  44.41028939,  43.35648762,  53.85799192,
        50.14004366,  90.94833308,  86.37240211,  60.18973692,
```

```
[29] #Plotting  Testing Actual values vs Predicted Vlaues
     plt.figure(figsize=(10, 6))
     # Scatter plot of testing data
     plt.scatter(y_test, y_pred_test, color ='slategrey')
     # Labels
     plt.xlabel('Actual Y_test')
     plt.ylabel('Predicted Y_test')
     plt.title('Actual vs Predicted values of Testing Data Model 3')
     # Display the plot
     plt.show()
```

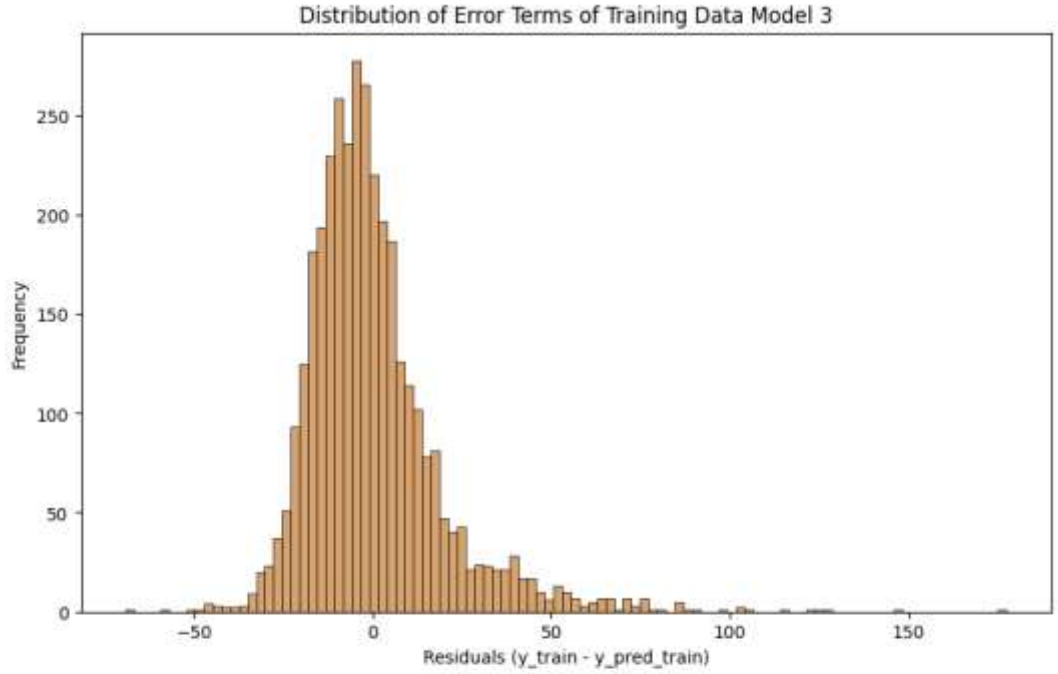



```
[16] #Calculating training data error term for linear regression
     res_train = (y_train - y_pred_train)
     res_train
```

```
4339    81.980863
4275    38.238063
3849    39.494090
883     -3.032717
134     -5.120574
          ...
1033    -5.910028
3264   -16.802927
1653   -26.773784
2607   -14.596526
2732     4.651529
Name: AQI, Length: 3533, dtype: float64
```

```
[17] # Plotting histogram to show the error terms for trainig data
     plt.figure(figsize=(10, 6))
     # Creating the histogram plot using Seaborn
     sns.histplot(res_train , color = 'peru')

     # Label the plot
     plt.title('Distribution of Error Terms of Training Data Model 3')
     plt.xlabel('Residuals (y_train - y_pred_train)')
     plt.ylabel('Frequency')
     plt.show()
```

[17]

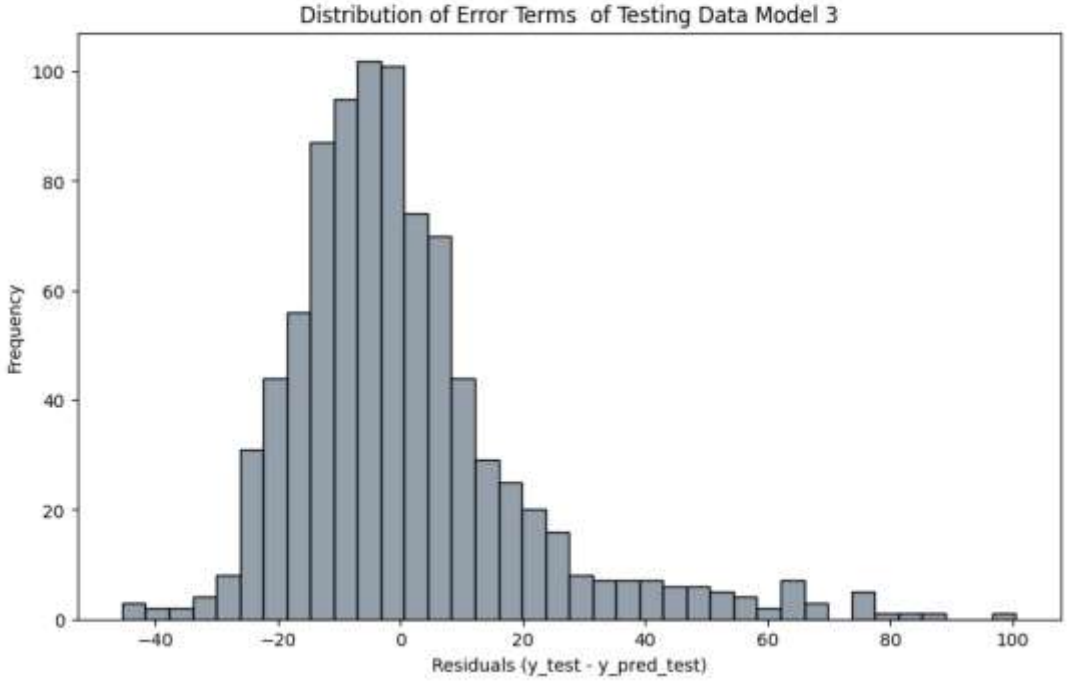


[18]
```
#Calculating testing data error term for linear regression
res_test = (y_test - y_pred_test)
res_test
```

```
2917   -24.462691
1457    -8.315770
1669    37.352202
2013   -24.759252
4303    21.807093
          ...
2527     6.427214
2339   -12.380431
```

```
3019   -33.515620
Name: AQI, Length: 884, dtype: float64
```

```
# Plotting histogram to show the error terms for testing data
plt.figure(figsize=(10, 6))
# Creating the histogram plot using Seaborn
sns.histplot(res_test , color ='slategrey')
# Label the plot
plt.title('Distribution of Error Terms  of Testing Data Model 3')
plt.xlabel('Residuals (y_test - y_pred_test)')
plt.ylabel('Frequency')
plt.show()
```



Evaluation of Linear Regression

Training Data

```
[20] #Calculating Mean Squared Error of Training Data
     mse = mean_squared_error(y_train, y_pred_train)
     print("Mean Squared Error (MSE) of Training Data:", mse)
```

```
Mean Squared Error (MSE) of Training Data: 367.51989118675203
```

```
[21] #calculating Root Mean Squared Error of Training Data
     rmse = np.sqrt(mse)
     print("Root Mean Squared Error (RMSE) of Training Data:", rmse)
```

```
Root Mean Squared Error (RMSE) of Training Data: 19.170808308121806
```

```
[22] #Calculating Mean Absolute Error of Training Data
     mae = mean_absolute_error(y_train, y_pred_train)
     print("Mean Absolute Error (MAE) of Training Data:", mae)
```

```
Mean Absolute Error (MAE) of Training Data: 13.355867871455413
```

```
[23] #Calculating R-Squared of Training Data
     rsquare =  r2_score(y_train, y_pred_train)
     print("R-sqaured of Training Data:", rsquare)
```

```
R-sqaured of Training Data: 0.45296390059534253
```

Testing Data

```
[24] #Calculating Mean Squared Error
     mse = mean_squared_error(y_test, y_pred_test)
     print("Mean Squared Error (MSE) of Testing Data :", mse)
```

```
Mean Squared Error (MSE) of Testing Data : 381.1122419287955
```

```
[25] #calculating Root Mean Squared Error
     rmse = np.sqrt(mse)
     print("Root Mean Squared Error (RMSE) of Testing Data:", rmse)
```

```
Root Mean Squared Error (RMSE) of Testing Data: 19.52209624832322
```

```
[26] #Calculating Mean Absolute Error
     mae = mean_absolute_error(y_test, y_pred_test)
     print("Mean Absolute Error (MAE) of Testing Data:", mae)
```

```
Mean Absolute Error (MAE) of Testing Data: 13.714497260079185
```

```
[27] #Calculating R-Squared  of Testing Data
     rsquare =  r2_score(y_test, y_pred_test)
     print("R-sqaured of Testing Data:", rsquare)
```

```
R-sqaured of Testing Data: 0.4225384801863755
```

LR Model 4

Predicting Air Quality Index(AQI) with the help of PM2.5, PM10, NO2, O3, CO, and SO2

```
[1] #importing necessary libraries
    import pandas as pd
    import seaborn as sns
    from sklearn.model_selection import train_test_split
    from sklearn.linear_model import LinearRegression
    import matplotlib.pyplot as plt
    from sklearn.metrics import r2_score,mean_squared_error, mean_absolute_error
    import numpy as np
```

```
[2] #Acessing google drive from google colab
    from google.colab import drive
    drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
[3] # Loading the dataset from drive
    df = pd.read_csv('/content/drive/MyDrive/Dissertation/clean_data.csv')
```

```
[4] #showing the first few data from the dataset
    df.head(5)
```

	pubtime	PM2.5	PM10	NO2	O3	CO	SO2	AQI
0	2020-06-30 00:00:00	11.0	23.0	26.0	70.0	0.7	5.0	23.0
1	2020-06-30 01:00:00	14.0	23.0	29.0	54.0	0.8	5.0	23.0
2	2020-06-30 02:00:00	11.0	23.0	31.0	48.0	0.7	5.0	23.0
3	2020-06-30 03:00:00	12.0	25.0	42.0	26.0	0.7	5.0	25.0
4	2020-06-30 04:00:00	7.0	29.0	36.0	22.0	0.7	5.0	29.0

```
[5] df.shape
```

```
(4417, 8)
```

Linear Regression Model

```
[6] #dropping all column except feature column PM2.5, PM10, No2, O3, CO and SO2 and store in x
    x= df.drop(columns = ['pubtime','AQI'])
    x
```

	PM2.5	PM10	NO2	O3	CO	SO2
0	11.0	23.0	26.0	70.0	0.7	5.0
1	14.0	23.0	29.0	54.0	0.8	5.0
2	11.0	23.0	31.0	48.0	0.7	5.0
3	12.0	25.0	42.0	26.0	0.7	5.0
4	7.0	29.0	36.0	22.0	0.7	5.0
...	...	...	...	...	...	...
4412	21.0	59.0	35.0	42.0	0.8	6.0
4413	22.0	65.0	35.0	40.0	0.8	6.0
4414	26.0	64.0	30.0	43.0	0.8	6.0
4415	24.0	58.0	30.0	42.0	0.8	6.0
4416	23.0	61.0	25.0	46.0	0.8	7.0

4417 rows × 6 columns

Next steps: View recommended plots

```
[7] #storing target column in y
    y = df['AQI']
    y
```

```
[7] 0       23.0
    1       23.0
    2       23.0
    3       25.0
    4       29.0
            ...
    4412    55.0
    4413    58.0
    4414    57.0
    4415    54.0
    4416    56.0
    Name: AQI, Length: 4417, dtype: float64
```

```
[8] #splitting test train data by using train_test_split libraries from sklearn
    x_train, x_test, y_train, y_test= train_test_split(x, y, test_size= 0.2,random_state=0)
```

```
[9] #fitting the model
    lr= LinearRegression()
    lr.fit(x_train,y_train)
```

```
▾ LinearRegression
LinearRegression()
```

```
[10] #calculating intercept c
     c = lr.intercept_
     c
```

```
5.5155327552498505
```

```
[11] # Calculating coefficient m
     m = lr.coef_
     m
```

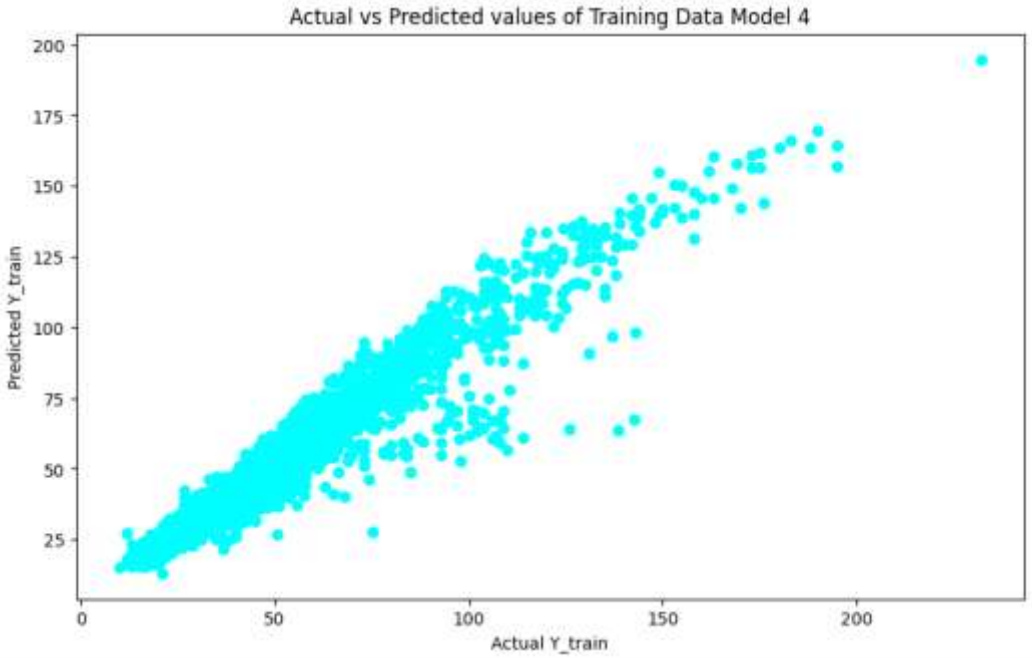
```
array([ 0.47231332,  0.46027295,  0.07279498,  0.12498361,  0.45818509,
       -0.11613176])
```

```
[12] #predicting y training values
     y_pred_train = lr.predict(x_train)
     y_pred_train
```

✓ 0s cc

```
array([ 67.55684847, 133.78640886,  86.17313307, ...,  61.47821071,
        20.83214596,  63.06363449])
```

```
# Plotting Training Actual values vs Predicted Values
plt.figure(figsize=(10, 6))
# Scatter plot of training data (actual values)
plt.scatter(y_train, y_pred_train, color='cyan')
# Labels
plt.xlabel('Actual Y_train')
plt.ylabel('Predicted Y_train')
plt.title('Actual vs Predicted values of Training Data Model 4')
# Display the plot
plt.show()
```



```
[14] #Prdicitng y testing values
     y_pred_test = lr.predict(x_test)
     y_pred_test
```

```
array([[ 37.90626444,  73.10069892,  54.10059445,  39.52320704],
       [ 38.76281681,  77.70598263,  62.20731759,  51.31631212],
       [ 52.87715973,  42.12631535,  35.05752604,  44.91851955],
       [ 28.29768434,  64.63762862,  68.33201464,  36.42987037],
       [ 24.33724041,  57.23138585,  36.69125393,  50.10768155],
       [ 62.76265334,  40.56547932,  16.28424517,  47.04450321],
       [112.81458842,  29.11721298,  58.86938859,  85.33264147],
       [109.10451069,  23.85584646,  24.37162841,  53.76397348],
       [ 39.93882875,  53.91474709,  35.6318966 ,  74.05065917],
       [ 79.46707058,  33.2076812 ,  65.72658307,  34.76191547],
       [ 38.41320134,  18.91600857,  63.62393529,  40.02095517],
       [ 37.98054467,  75.57933726,  87.89978185,  34.41226916],
       [ 24.54065502,  34.02394911,  21.55129955,  71.95658032],
       [ 63.33769933,  33.18995589,  56.88682196,  88.59325326],
       [ 35.76600191,  61.83732931,  68.61377526, 143.54512977],
       [ 72.50029458,  70.09878301,  21.31331063,  56.7129704 ],
       [ 38.4354319 ,  43.15336449,  19.74901151,  98.88958864],
       [ 81.03722574,  55.96337546,  58.80938168,  31.25399736],
       [ 55.04015507,  81.99644039,  36.82343412,  33.02667335],
       [ 87.3347148 ,  56.58716239, 113.17655287,  70.46741868],
       [ 28.28069065,  40.63657507,  18.4710979 ,  81.21648666],
       [ 54.87390991,  61.59820463,  76.27469377,  34.74349512],
       [ 37.44291198,  43.37582889,  86.62436232,  95.6684326 ],
       [ 66.81649002,  38.55249843,  40.25325796,  58.62322032],
       [ 42.67661433,  29.19820041,  32.75150672,  74.12341582],
       [ 82.4894894 ,  39.04227258,  75.44833879,  46.22784361],
       [ 83.05416305,  57.63405586,  38.77618408,  49.26653529],
       [ 29.88090591,  56.00079071,  39.18399195,  41.40501814],
       [ 31.63232554,  61.59417104,  68.51114518,  47.26028878],
       [ 40.43505378,  34.75720011, 137.94245043,  79.03363538],
       [ 62.0094402 ,  35.3073257 ,  55.37789666,  55.5441966 ],
       [ 74.29482089,  44.27620135,  80.78298772, 101.56108894],
       [ 29.45352754,  43.91245467,  35.57756693, 116.94096457],
       [ 38.91079623,  61.95963901,  47.30386605,  40.37555523],
       [ 36.55243481,  53.07272693,  64.05436948,  82.77590258],
       [144.82914679,  74.05304418,  54.99457029,  62.91886694],
       [ 38.96827789,  45.34007928,  48.01484209,  48.56704616],
       [ 36.18134111,  34.48032585,  48.34255286,  61.04532494],
       [ 52.4507506 ,  21.27758593,  68.56327072,  53.79852748],
       [ 32.34825505,  32.53652429,  51.50070359,  26.89384962],
       [ 57.67878331,  23.1860883 , 109.31919977,  35.27012335],
       [ 47.81450997,  40.09420704,  91.97366268, 100.46681764],
```

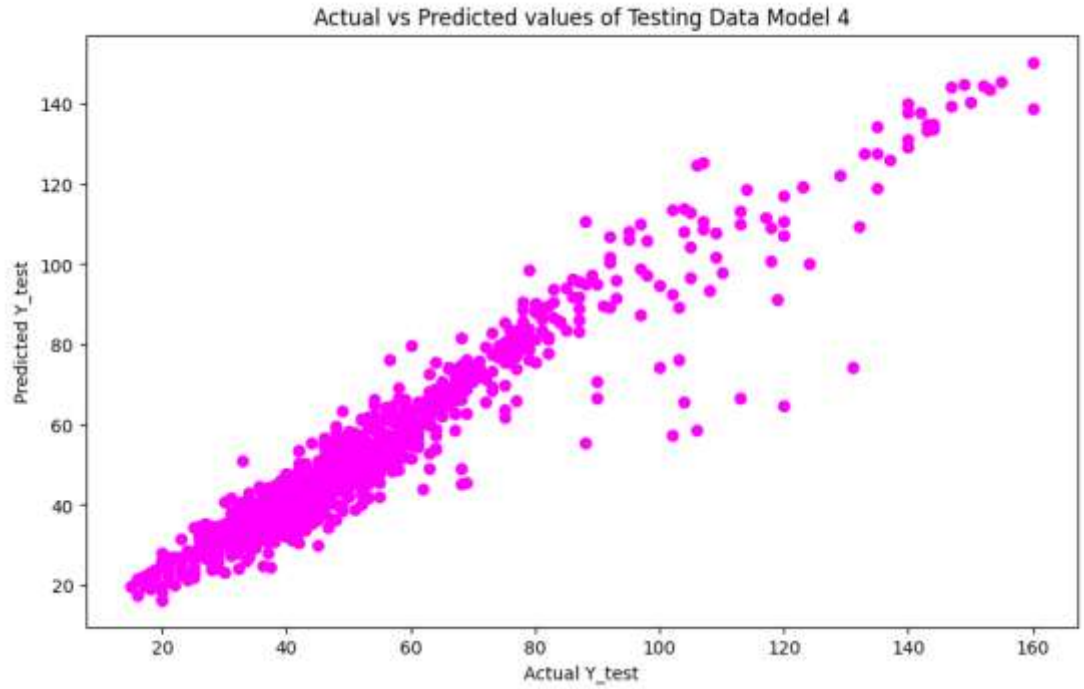
[14]

```
 29.88090591,  56.00079071,  39.18399195,  41.40501814,
 31.63232554,  61.59417104,  68.51114518,  47.26028878,
 40.43505378,  34.75720011, 137.94245043,  79.03363538,
 62.0094402 ,  35.3073257 ,  55.37789666,  55.5441966 ,
 74.29482089,  44.27620135,  80.78298772, 101.56108894,
 29.45352754,  43.91245467,  35.57756693, 116.94096457,
 38.91079623,  61.95963901,  47.30386605,  40.37555523,
 36.55243481,  53.07272693,  64.05436948,  82.77590258,
144.82914679,  74.05304418,  54.99457029,  62.91886694,
 38.96827789,  45.34007928,  48.01484209,  48.56704616,
 36.18134111,  34.48032585,  48.34255286,  61.04532494,
 52.4507506 ,  21.27758593,  68.56327072,  53.79852748,
 32.34825505,  32.53652429,  51.50070359,  26.89384962,
 57.67878331,  23.1860883 , 109.31919977,  35.27012335,
 47.81450997,  40.09420704,  91.97366268, 100.46681764,
 97.13156327,  50.71308742,  49.91737571,  37.9292411 ,
 29.60754046,  42.05055557,  57.59894397,  45.99224841,
 41.27434604,  53.34039397,  34.16023156,  96.55067703,
 62.79549186,  45.96223987,  30.80885434,  37.36652734,
 23.23440378,  37.76184011,  68.22405563,  35.05826124,
 55.30549604,  33.14740196,  41.53398346,  60.79349415,
119.14574509,  63.43015076,  91.58530473,  57.9157267 ,
 53.95569283,  69.85151273,  28.22024033,  42.3211966 ,
 44.49335185,  35.15157202,  32.76466877,  36.57935648,
 43.0377386 ,  50.47326865,  71.17495576,  30.98105855,
 56.1193578 ,  53.34574028,  57.86622213,  55.15871193,
 70.71017668,  50.77025676,  44.0764894 ,  45.19377324,
 24.97742007,  55.17554737,  64.68566894,  83.79371095,
 31.42011806,  36.28079217,  68.90216744,  48.13822699,
 48.24279242,  56.81957811,  49.4759478 ,  70.91741003,
 32.027498  ,  50.82163979,  60.39219492,  75.04735122])
```

[15]

```
#Plotting  Testing Actual values vs Predicted Vlaues
plt.figure(figsize=(10, 6))
# Scatter plot of testing data
plt.scatter(y_test, y_pred_test, color ='magenta')
# Labels
plt.xlabel('Actual Y_test')
plt.ylabel('Predicted Y_test')
plt.title('Actual vs Predicted values of Testing Data Model 4')
# Display the plot
plt.show()
```

[15]

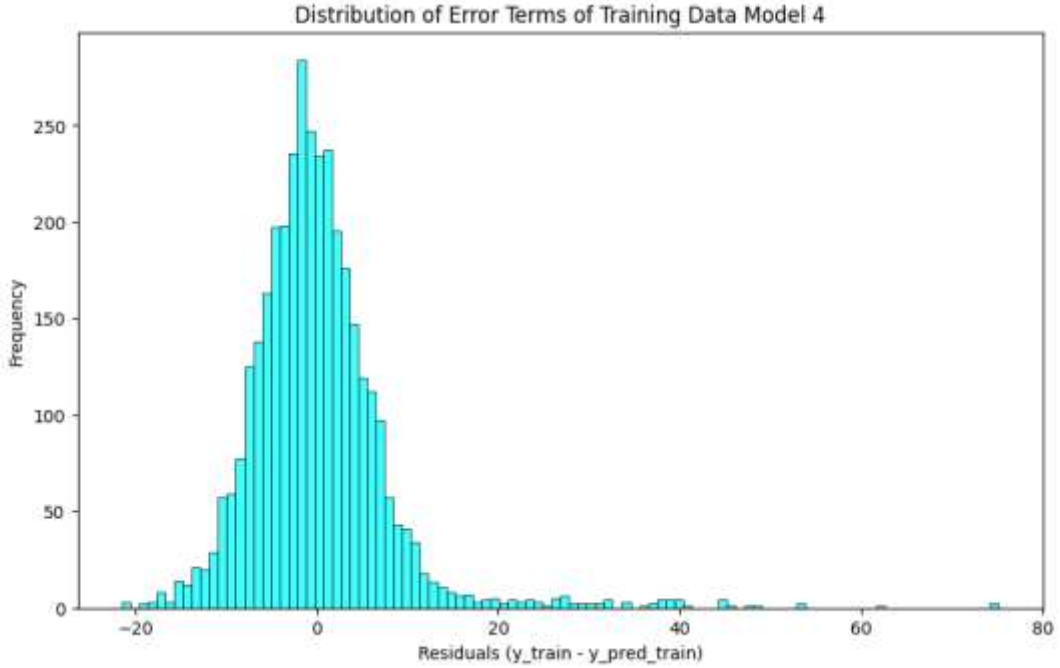


[16]
```
#Calculating training data error term for linear regression
res_train = (y_train - y_pred_train)
res_train
```

```
4339    75.193152
4275    -0.786409
3849     6.826867
883     -6.658975
134      3.075921
           ...
1033    -2.110831
3264     6.847799
1653    -9.478211
```

```
2732    -5.063634
Name: AQI, Length: 3533, dtype: float64
```

```
# Plotting histogram to show the error terms for trainig data
plt.figure(figsize=(10, 6))
# Creating the histogram plot using Seaborn
sns.histplot(res_train , color = 'cyan')
# Label the plot
plt.title('Distribution of Error Terms of Training Data Model 4')
plt.xlabel('Residuals (y_train - y_pred_train)')
plt.ylabel('Frequency')
plt.show()
```

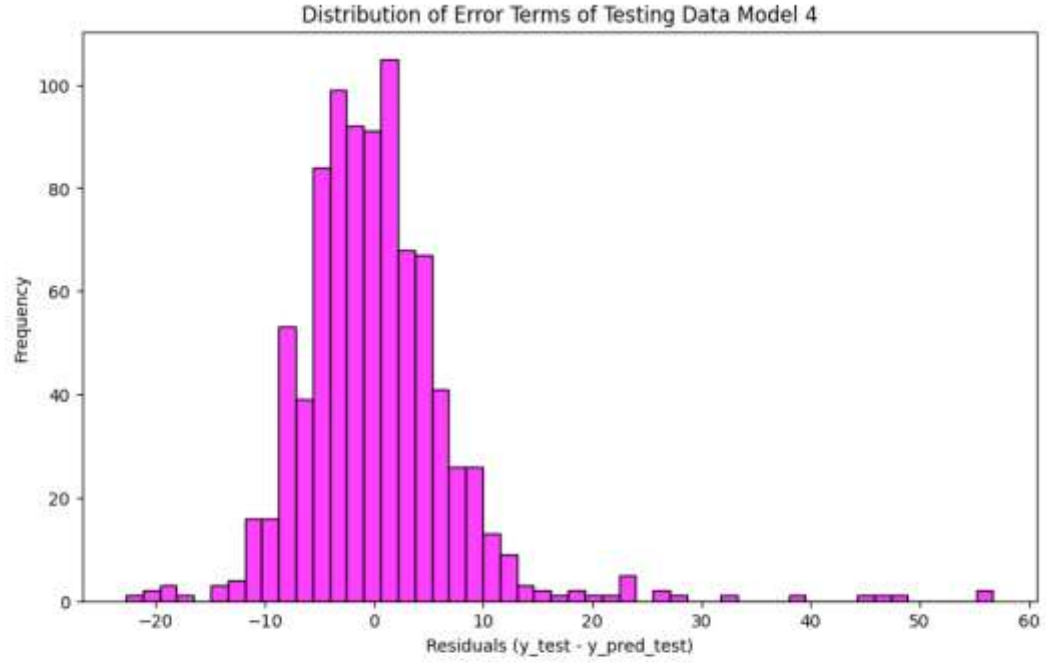


[18]
```
#Calculating testing data error term for linear regression
res_test = (y_test - y_pred_test)
res_test
```

```
2917    -4.958057
1457    -3.901265
1669    38.430742
2013    -0.535255
4303    -2.512761
          ...
2527    -2.917410
2339    -0.027498
3276     2.178360
3354    -1.392195
3019    -7.047351
Name: AQI, Length: 884, dtype: float64
```

[19]
```
# Plotting histogram to show the error terms for testing data
plt.figure(figsize=(10, 6))
# Creating the histogram plot using Seaborn
sns.histplot(res_test , color ='magenta')

# Label the plot
plt.title('Distribution of Error Terms of Testing Data Model 4')
plt.xlabel('Residuals (y_test - y_pred_test)')
plt.ylabel('Frequency')
plt.show()
```

[19]



Evaluation of Linear Regression

Training Data

```
#Calculating Mean Squared Error of Training Data
mse = mean_squared_error(y_train, y_pred_train)
print("Mean Squared Error (MSE) of Training Data:", mse)
```

```
Mean Squared Error (MSE) of Training Data: 58.33351774880525
```

```
[21] #calculating Root Mean Squared Error of Training Data
     rmse = np.sqrt(mse)
     print("Root Mean Squared Error (RMSE) of Training Data:", rmse)
```

```
Root Mean Squared Error (RMSE) of Training Data: 7.637638231076754
```

```
[22] #Calculating Mean Absolute Error of Training Data
     mae = mean_absolute_error(y_train, y_pred_train)
     print("Mean Absolute Error (MAE) of Training Data:", mae)
```

```
Mean Absolute Error (MAE) of Training Data: 5.095290973689186
```

```
[23] #Calculating R-Squared of Training Data
     rsquare =  r2_score(y_train, y_pred_train)
     print("R-sqaured of Training Data:", rsquare)
```

```
R-sqaured of Training Data: 0.9131732981558712
```

Testing Data

```
[24] #Calculating Mean Squared Error
     mse = mean_squared_error(y_test, y_pred_test)
     print("Mean Squared Error (MSE) of Testing Data :", mse)
```

```
Mean Squared Error (MSE) of Testing Data : 55.27674743256018
```

```
[25] #calculating Root Mean Squared Error
     rmse = np.sqrt(mse)
     print("Root Mean Squared Error (RMSE) of Testing Data:", rmse)
```

```
Root Mean Squared Error (RMSE) of Testing Data: 7.434833382972356
```

```
[26] #Calculating Mean Absolute Error
     mae = mean_absolute_error(y_test, y_pred_test)
     print("Mean Absolute Error (MAE) of Testing Data:", mae)
```

```
Mean Absolute Error (MAE) of Testing Data: 5.014977049917288
```

```
[27] #Calculating R-Squared  of Testing Data
     rsquare =  r2_score(y_test, y_pred_test)
     print("R-sqaured of Testing Data:", rsquare)
```

```
R-sqaured of Testing Data: 0.9162446359077496
```

APPENDIX 5: LSTM Code

LSTM Model 1

Predicting Air Quality Index(AQI) With single Itterative Column

```
#importing necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential, load_model
from keras.layers import LSTM, Dense
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
```

```
[92] #acessing google drive from google colab
     from google.colab import drive
     drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount
```

```
[94] # Load the dataset
     df = pd.read_csv('/content/drive/MyDrive/Dissertation/clean_data.csv')
     df
```

	pubtime	PM2.5	PM10	NO2	O3	CO	SO2	AQI
0	2020-06-30 00:00:00	11.0	23.0	26.0	70.0	0.7	5.0	23.0
1	2020-06-30 01:00:00	14.0	23.0	29.0	54.0	0.8	5.0	23.0
2	2020-06-30 02:00:00	11.0	23.0	31.0	48.0	0.7	5.0	23.0
3	2020-06-30 03:00:00	12.0	25.0	42.0	26.0	0.7	5.0	25.0
4	2020-06-30 04:00:00	7.0	29.0	36.0	22.0	0.7	5.0	29.0

[94]

4412	2020-12-30 20:00:00	21.0	59.0	35.0	42.0	0.8	6.0	55.0
4413	2020-12-30 21:00:00	22.0	65.0	35.0	40.0	0.8	6.0	58.0
4414	2020-12-30 22:00:00	26.0	64.0	30.0	43.0	0.8	6.0	57.0
4415	2020-12-30 23:00:00	24.0	58.0	30.0	42.0	0.8	6.0	54.0
4416	2020-12-31 00:00:00	23.0	61.0	25.0	46.0	0.8	7.0	56.0

4417 rows × 8 columns

Next steps: View recommended plots

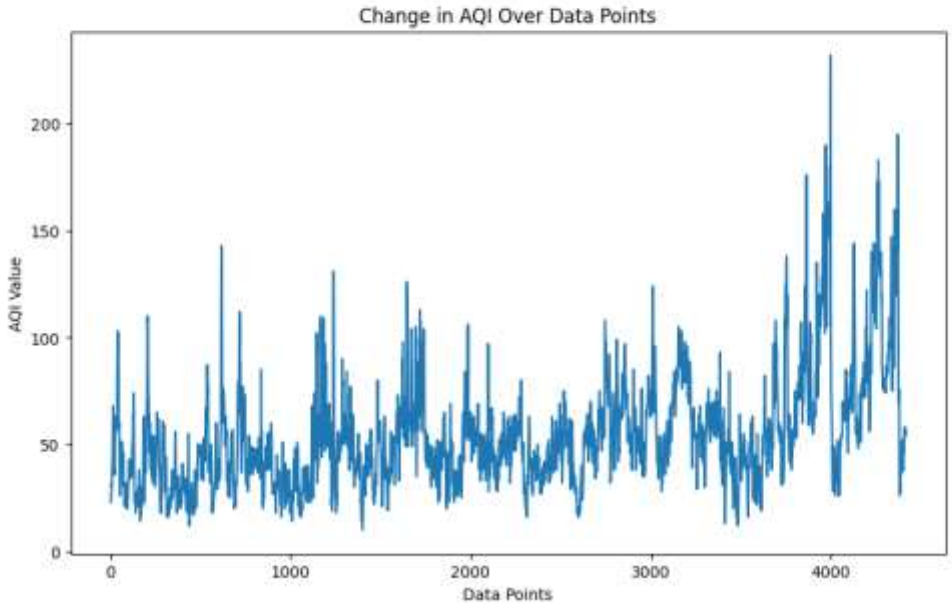
LSTM Model Creation and Training

[95]
```
#Selecting only one column to predict AQI from AQI previous data
df= df[['AQI']]
df.head()
```

	AQI
0	23.0
1	23.0
2	23.0
3	25.0
4	29.0

Next steps: View recommended plots

```
# Plotting how AQI values are changing
plt.figure(figsize=(10, 6))
plt.plot(df['AQI'])

# Adding labels and title
plt.xlabel('Data Points')
plt.ylabel('AQI Value')
plt.title('Change in AQI Over Data Points')

plt.show()
```



```
[97] #spliting dataset for training and testing phase
     train_size = int(len(df) * 0.8)
     train_data = df.iloc[:train_size]
     test_data = df.iloc[train_size:]
```

```
[98] #scaling dataset to 0 and 1 by using MinMaxScaler
     scaler = MinMaxScaler(feature_range=(0, 1))
     train_scaled = scaler.fit_transform(train_data)
     test_scaled = scaler.transform(test_data)
```

```
[99] #Creating input-output pairs by splitting dataset into a sequence of input_size
     def create_sequences(data, input_size):
         X, y = [], []
     #
         for i in range(len(data) - input_size):
             X.append(data[i:i + input_size])
             y.append(data[i + input_size])
         return np.array(X), np.array(y)

     input_size = 24

     X_train, y_train = create_sequences(train_scaled, input_size)
     X_test, y_test = create_sequences(test_scaled, input_size)
```

```
[100] #priniting the shape of x_train and X_test data
      print(X_train.shape)
      print(X_test.shape)
```

```
(3509, 24, 1)
(860, 24, 1)
```

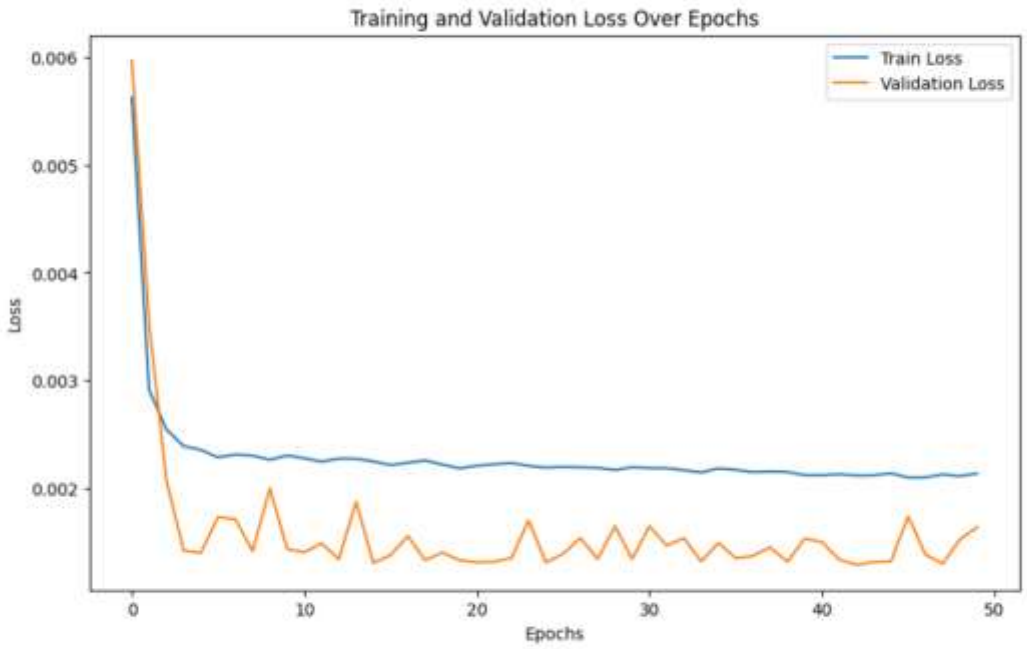
```
[101] #creating a LSTM model
      model = Sequential()
      model.add(LSTM(100, activation='relu',input_shape=(X_train.shape[1], X_train.shape[2])))
      model.add(Dense(1))
      model.compile(optimizer='adam', loss='mse')
```

```
[102] #fitting the model
      history = model.fit(X_train, y_train, epochs=50, batch_size=1,validation_split=0.2)
```

```
Epoch 1/50
2807/2807 [==============================] - 47s 16ms/step - loss: 0.0056 - val_loss: 0.0060
Epoch 2/50
2807/2807 [==============================] - 54s 19ms/step - loss: 0.0029 - val_loss: 0.0035
Epoch 3/50
2807/2807 [==============================] - 43s 15ms/step - loss: 0.0025 - val_loss: 0.0021
Epoch 4/50
2807/2807 [==============================] - 48s 17ms/step - loss: 0.0024 - val_loss: 0.0014
Epoch 5/50
2807/2807 [==============================] - 44s 16ms/step - loss: 0.0024 - val_loss: 0.0014
Epoch 6/50
2807/2807 [==============================] - 48s 17ms/step - loss: 0.0023 - val_loss: 0.0017
Epoch 7/50
2807/2807 [==============================] - 44s 16ms/step - loss: 0.0023 - val_loss: 0.0017
Epoch 8/50
2807/2807 [==============================] - 47s 17ms/step - loss: 0.0023 - val_loss: 0.0014
Epoch 9/50
2807/2807 [==============================] - 43s 15ms/step - loss: 0.0023 - val_loss: 0.0020
Epoch 10/50
2807/2807 [==============================] - 44s 16ms/step - loss: 0.0023 - val_loss: 0.0014
Epoch 11/50
2807/2807 [==============================] - 48s 17ms/step - loss: 0.0023 - val_loss: 0.0014
Epoch 12/50
2807/2807 [==============================] - 44s 16ms/step - loss: 0.0022 - val_loss: 0.0015
Epoch 13/50
2807/2807 [==============================] - 48s 17ms/step - loss: 0.0023 - val_loss: 0.0013
Epoch 14/50
2807/2807 [==============================] - 46s 16ms/step - loss: 0.0023 - val_loss: 0.0019
Epoch 15/50
```

```
[102] 2807/2807 [==============================] - 46s 16ms/step - loss: 0.0022 - val_loss: 0.0013
      Epoch 16/50
      2807/2807 [==============================] - 46s 16ms/step - loss: 0.0022 - val_loss: 0.0014
      Epoch 17/50
      2807/2807 [==============================] - 43s 15ms/step - loss: 0.0022 - val_loss: 0.0016
      Epoch 18/50
      2807/2807 [==============================] - 44s 16ms/step - loss: 0.0023 - val_loss: 0.0013
      Epoch 19/50
      2807/2807 [==============================] - 46s 16ms/step - loss: 0.0022 - val_loss: 0.0014
      Epoch 20/50
      2807/2807 [==============================] - 44s 16ms/step - loss: 0.0022 - val_loss: 0.0013
      Epoch 21/50
      2807/2807 [==============================] - 45s 16ms/step - loss: 0.0022 - val_loss: 0.0013
      Epoch 22/50
      2807/2807 [==============================] - 44s 16ms/step - loss: 0.0022 - val_loss: 0.0013
      Epoch 23/50
      2807/2807 [==============================] - 46s 16ms/step - loss: 0.0022 - val_loss: 0.0014
      Epoch 24/50
      2807/2807 [==============================] - 45s 16ms/step - loss: 0.0022 - val_loss: 0.0017
      Epoch 25/50
      2807/2807 [==============================] - 43s 15ms/step - loss: 0.0022 - val_loss: 0.0013
      Epoch 26/50
      2807/2807 [==============================] - 52s 19ms/step - loss: 0.0022 - val_loss: 0.0014
      Epoch 27/50
      2807/2807 [==============================] - 46s 16ms/step - loss: 0.0022 - val_loss: 0.0015
      Epoch 28/50
      2807/2807 [==============================] - 45s 16ms/step - loss: 0.0022 - val_loss: 0.0013
      Epoch 29/50
      2807/2807 [==============================] - 43s 15ms/step - loss: 0.0022 - val_loss: 0.0016
      Epoch 30/50
      2807/2807 [==============================] - 46s 16ms/step - loss: 0.0022 - val_loss: 0.0013
      Epoch 31/50
      2807/2807 [==============================] - 44s 16ms/step - loss: 0.0022 - val_loss: 0.0016
      Epoch 32/50
      2807/2807 [==============================] - 44s 16ms/step - loss: 0.0022 - val_loss: 0.0015
      Epoch 33/50
      2807/2807 [==============================] - 44s 16ms/step - loss: 0.0022 - val_loss: 0.0015
      Epoch 34/50
      2807/2807 [==============================] - 47s 17ms/step - loss: 0.0021 - val_loss: 0.0013
      Epoch 35/50
      2807/2807 [==============================] - 44s 16ms/step - loss: 0.0022 - val_loss: 0.0015
[102] 2807/2807 [==============================] - 44s 16ms/step - loss: 0.0022 - val_loss: 0.0015
      Epoch 36/50
      2807/2807 [==============================] - 46s 16ms/step - loss: 0.0022 - val_loss: 0.0013
      Epoch 37/50
      2807/2807 [==============================] - 44s 16ms/step - loss: 0.0021 - val_loss: 0.0014
      Epoch 38/50
      2807/2807 [==============================] - 46s 16ms/step - loss: 0.0022 - val_loss: 0.0015
      Epoch 39/50
      2807/2807 [==============================] - 44s 16ms/step - loss: 0.0022 - val_loss: 0.0013
      Epoch 40/50
      2807/2807 [==============================] - 45s 16ms/step - loss: 0.0021 - val_loss: 0.0015
      Epoch 41/50
      2807/2807 [==============================] - 51s 18ms/step - loss: 0.0021 - val_loss: 0.0015
      Epoch 42/50
      2807/2807 [==============================] - 44s 16ms/step - loss: 0.0021 - val_loss: 0.0013
      Epoch 43/50
      2807/2807 [==============================] - 46s 16ms/step - loss: 0.0021 - val_loss: 0.0013
      Epoch 44/50
      2807/2807 [==============================] - 44s 16ms/step - loss: 0.0021 - val_loss: 0.0013
      Epoch 45/50
      2807/2807 [==============================] - 47s 17ms/step - loss: 0.0021 - val_loss: 0.0013
      Epoch 46/50
      2807/2807 [==============================] - 48s 17ms/step - loss: 0.0021 - val_loss: 0.0017
      Epoch 47/50
      2807/2807 [==============================] - 47s 17ms/step - loss: 0.0021 - val_loss: 0.0014
      Epoch 48/50
      2807/2807 [==============================] - 47s 17ms/step - loss: 0.0021 - val_loss: 0.0013
      Epoch 49/50
      2807/2807 [==============================] - 47s 17ms/step - loss: 0.0021 - val_loss: 0.0015
      Epoch 50/50
      2807/2807 [==============================] - 45s 16ms/step - loss: 0.0021 - val_loss: 0.0016

[106] # Plotting the Loss Values of a Fitted Model
      plt.figure(figsize=(10, 6))
      plt.plot(history.history['loss'], label='Train Loss')
      plt.plot(history.history['val_loss'], label='Validation Loss')
      plt.legend()
      plt.xlabel('Epochs')
      plt.ylabel('Loss')
```

```
[106] plt.title('Training and Validation Loss Over Epochs')
      plt.show()
```




```
[107] #predicting train and test data from x_train and x_test
      train_predicted = model.predict(X_train.reshape(X_train.shape))
      test_predicted = model.predict(X_test.reshape(X_test.shape))
```

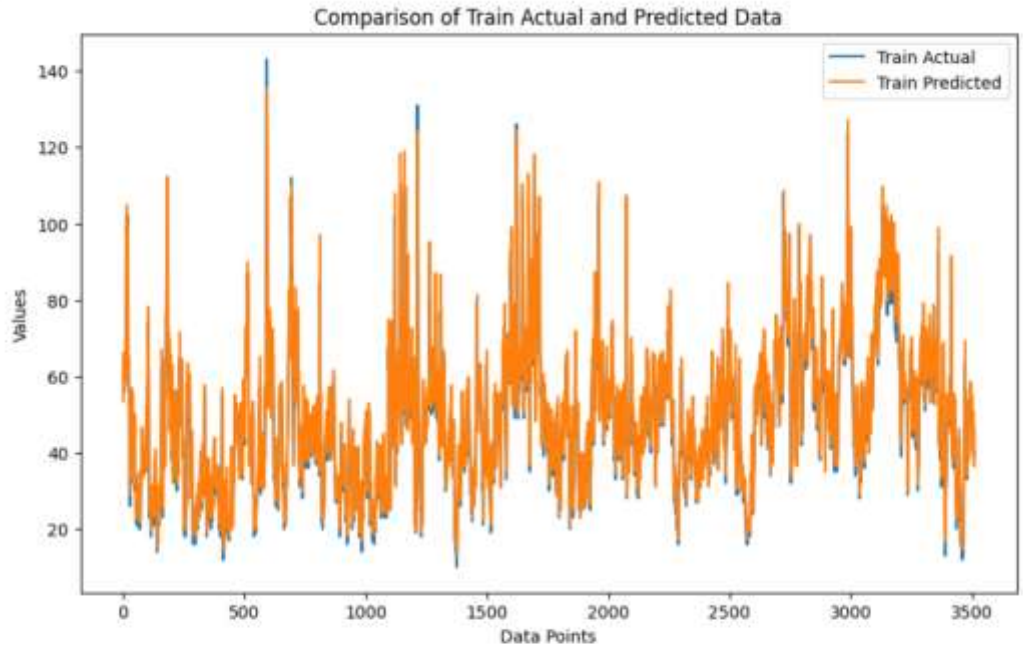
```
[107] #Scale back to orginal scale for both train and test predicted data
      train_predicted = scaler.inverse_transform(train_predicted)
      test_predicted = scaler.inverse_transform(test_predicted)

      #Scale back to orginal scale for both train and test actual data
      train_actual = scaler.inverse_transform(y_train)
      test_actual = scaler.inverse_transform(y_test)
```

```
110/110 [==============================] - 1s 7ms/step
27/27 [==============================] - 0s 8ms/step
```

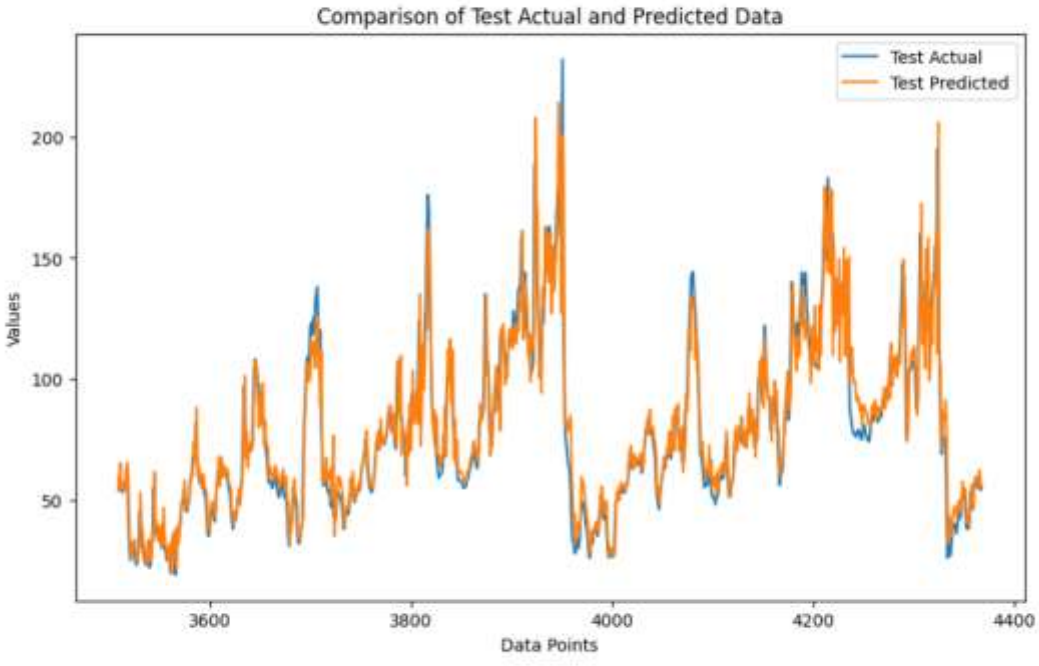
```
      #creating numpy array data for plotting actual and predicted data
      x1 = np.arange(0, len(train_actual))
      x2 = np.arange(len(train_actual), len(train_actual)+len(test_actual))
```

```
[109] # Plotting Train Actual and Predicted Data
      plt.figure(figsize=(10, 6))
      plt.plot(x1, train_actual)
      plt.plot(x1, train_predicted)
      plt.legend(['Train Actual', 'Train Predicted'])
      plt.xlabel('Data Points')
      plt.ylabel('Values')
      plt.title('Comparison of Train Actual and Predicted Data')
      plt.show()
```

[109]

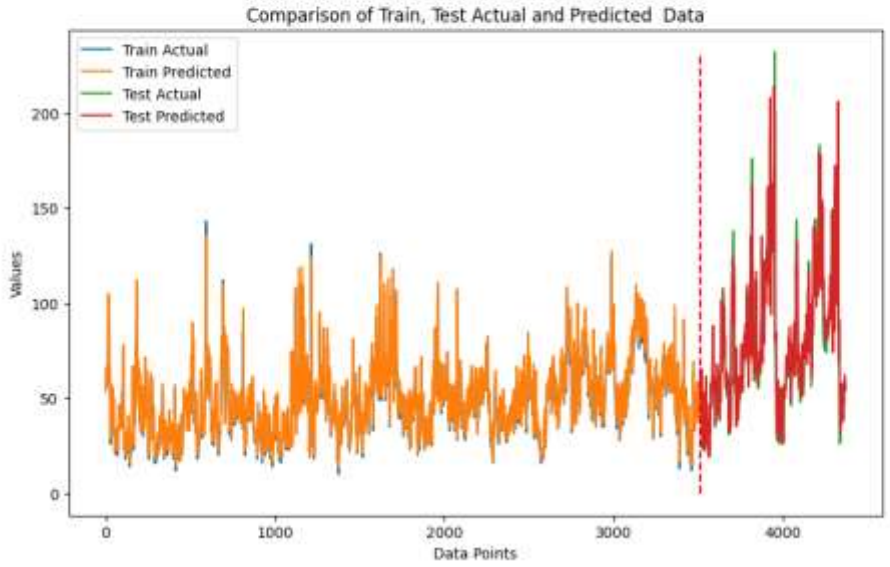



```
[110] # Plotting Test Actual and Predicted Data
      plt.figure(figsize=(10, 6))
      plt.plot(x2, test_actual)
      plt.plot(x2, test_predicted)
      plt.legend(['Test Actual', 'Test Predicted'])
      plt.xlabel('Data Points')
[110] plt.ylabel('Values')
      plt.title('Comparison of Test Actual and Predicted Data')
      plt.show()
```




```
[111] # Plotting Both Train and Test Actual Data
      plt.figure(figsize=(10, 6))
```

```
plt.plot(x1, train_actual)
plt.plot(x1, train_predicted)
plt.plot(x2, test_actual)
plt.plot(x2, test_predicted)
plt.legend(['Train Actual', 'Train Predicted', 'Test Actual', 'Test Predicted'])
plt.xlabel('Data Points')
plt.ylabel('Values')
plt.title('Comparison of Train, Test Actual and Predicted  Data')
plt.vlines(x=len(train_actual), color='r', linestyles='dashed', ymin=0, ymax=max(test_actual))
plt.show()
```




```
#Evulate the model performance

#Mean Squared Error For Testing and Training data
mse_train = mean_squared_error(train_actual, train_predicted)
print("Train Mean Squared Error :", mse_train)
mse_test = mean_squared_error(test_actual, test_predicted)
print("Test Mean Squared Error:", mse_test)

# Root Mean Squared Error For Testing and Training data
rmse_train = np.sqrt(mse_train)
print("Train Root Mean Squared Error:", rmse_train)
rmse_test = np.sqrt(mse_test)
print("Test Root Mean Squared Error:", rmse_test)

#Mean Absolute Error For Testing and Training data
mae_train = mean_absolute_error(train_actual, train_predicted)
print("Train Mean Absolute Error:", mae_train)
mae_test = mean_absolute_error(test_actual, test_predicted)
print("Test Mean Absolute Error:", mae_test)

#R-Sqaured For Testing and Training data
r2_train = r2_score(train_actual, train_predicted)
print("Train R-squared:", r2_train)
r2_test = r2_score(test_actual, test_predicted)
print("Test R-squared:", r2_test)
```

```
Train Mean Squared Error : 33.630915721336855
Test Mean Squared Error: 171.988817144974
Train Root Mean Squared Error: 5.799216819652189
Test Root Mean Squared Error: 13.114450699323019
Train Mean Absolute Error: 3.903921513518026
Test Mean Absolute Error: 8.162672923332037
Train R-squared: 0.9023962686614279
Test R-squared: 0.8652186099787609
```

LSTM Model 2

Predicting Air Quality Index(AQI) With single multivariate Column PM2.5, PM10

```
[1] #importing necessary libraries
    import pandas as pd
    import numpy as np
    import matplotlib.pyplot as plt
    from sklearn.preprocessing import MinMaxScaler
    from keras.models import Sequential, load_model
    from keras.layers import LSTM, Dense
    from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
```

```
[2] #acessing google drive from google colab
    from google.colab import drive
    drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
[3] # Load the dataset
    df = pd.read_csv('/content/drive/MyDrive/Dissertation/clean_data.csv')
    df
```

	pubtime	PM2.5	PM10	NO2	O3	CO	SO2	AQI
0	2020-06-30 00:00:00	11.0	23.0	26.0	70.0	0.7	5.0	23.0
1	2020-06-30 01:00:00	14.0	23.0	29.0	54.0	0.8	5.0	23.0
2	2020-06-30 02:00:00	11.0	23.0	31.0	48.0	0.7	5.0	23.0
3	2020-06-30 03:00:00	12.0	25.0	42.0	26.0	0.7	5.0	25.0
4	2020-06-30 04:00:00	7.0	29.0	36.0	22.0	0.7	5.0	29.0

[3]

4412	2020-12-30 20:00:00	21.0	59.0	35.0	42.0	0.8	6.0	55.0
4413	2020-12-30 21:00:00	22.0	65.0	35.0	40.0	0.8	6.0	58.0
4414	2020-12-30 22:00:00	26.0	64.0	30.0	43.0	0.8	6.0	57.0
4415	2020-12-30 23:00:00	24.0	58.0	30.0	42.0	0.8	6.0	54.0
4416	2020-12-31 00:00:00	23.0	61.0	25.0	46.0	0.8	7.0	56.0

4417 rows × 8 columns

Next steps: View recommended plots

LSTM Model Creation and Training

[4]
```
#Selecting only one column to predict AQI from AQI previous data
df= df[['PM2.5','PM10','AQI']]
df.head()
```

	PM2.5	PM10	AQI
0	11.0	23.0	23.0
1	14.0	23.0	23.0
2	11.0	23.0	23.0
3	12.0	25.0	25.0
4	7.0	29.0	29.0

Next steps: View recommended plots

[5]
```
#spliting dataset for training and testing phase
train_size = int(len(df) * 0.8)
train_data = df.iloc[:train_size]
test_data = df.iloc[train_size:]
```

[6]
```
#scaling dataset to 0 and 1 by using MinMaxScaler
scaler = MinMaxScaler(feature_range=(0, 1))
train_scaled = scaler.fit_transform(train_data)
test_scaled = scaler.transform(test_data)
```

[7]
```
#Creating input-output pairs by splitting dataset into a sequence of input_size
def create_sequences(data, input_size):
    X, y = [], []
    for i in range(len(data) - input_size):
        X.append(data[i:i + input_size, :])
        y.append(data[i + input_size, -1])
    return np.array(X), np.array(y)

input_size = 24

X_train, y_train = create_sequences(train_scaled, input_size)
X_test, y_test = create_sequences(test_scaled, input_size)
```

[8]
```
#priniting the shape of x_train and X_test data
print(X_train.shape)
print(X_test.shape)
```

```
(3509, 24, 3)
(860, 24, 3)
```

[9]
```
#creating a LSTM model
model = Sequential()
```

```
[9]  model.add(LSTM(100, activation='relu',input_shape=(X_train.shape[1], X_train.shape[2])))
     model.add(Dense(1))
     model.compile(optimizer='adam', loss='mse')
```

```
[10] #fitting the model
     history = model.fit(X_train, y_train, epochs=50, batch_size=1,validation_split=0.2)
```

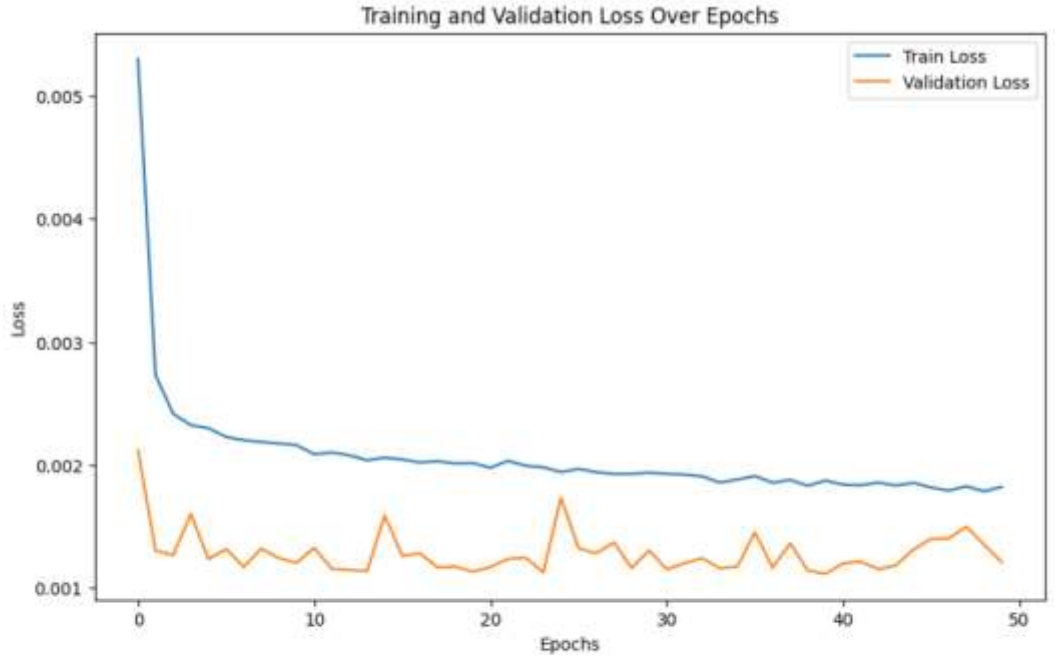
```
Epoch 1/50
2807/2807 [==============================] - 56s 20ms/step - loss: 0.0053 - val_loss: 0.0021
Epoch 2/50
2807/2807 [==============================] - 45s 16ms/step - loss: 0.0027 - val_loss: 0.0013
Epoch 3/50
2807/2807 [==============================] - 48s 17ms/step - loss: 0.0024 - val_loss: 0.0013
Epoch 4/50
2807/2807 [==============================] - 47s 17ms/step - loss: 0.0023 - val_loss: 0.0016
Epoch 5/50
2807/2807 [==============================] - 44s 16ms/step - loss: 0.0023 - val_loss: 0.0012
Epoch 6/50
2807/2807 [==============================] - 46s 17ms/step - loss: 0.0022 - val_loss: 0.0013
Epoch 7/50
2807/2807 [==============================] - 46s 16ms/step - loss: 0.0022 - val_loss: 0.0012
Epoch 8/50
2807/2807 [==============================] - 48s 17ms/step - loss: 0.0022 - val_loss: 0.0013
Epoch 9/50
2807/2807 [==============================] - 45s 16ms/step - loss: 0.0022 - val_loss: 0.0012
Epoch 10/50
2807/2807 [==============================] - 46s 17ms/step - loss: 0.0022 - val_loss: 0.0012
Epoch 11/50
2807/2807 [==============================] - 46s 16ms/step - loss: 0.0021 - val_loss: 0.0013
Epoch 12/50
2807/2807 [==============================] - 46s 16ms/step - loss: 0.0021 - val_loss: 0.0012
Epoch 13/50
2807/2807 [==============================] - 47s 17ms/step - loss: 0.0021 - val_loss: 0.0011
Epoch 14/50
2807/2807 [==============================] - 45s 16ms/step - loss: 0.0020 - val_loss: 0.0011
Epoch 15/50
2807/2807 [==============================] - 46s 16ms/step - loss: 0.0021 - val_loss: 0.0016
Epoch 16/50
2807/2807 [==============================] - 46s 16ms/step - loss: 0.0020 - val_loss: 0.0013
```

```
[10] Epoch 17/50
2807/2807 [==============================] - 49s 17ms/step - loss: 0.0020 - val_loss: 0.0013
Epoch 18/50
2807/2807 [==============================] - 47s 17ms/step - loss: 0.0020 - val_loss: 0.0012
Epoch 19/50
2807/2807 [==============================] - 46s 16ms/step - loss: 0.0020 - val_loss: 0.0012
Epoch 20/50
2807/2807 [==============================] - 48s 17ms/step - loss: 0.0020 - val_loss: 0.0011
Epoch 21/50
2807/2807 [==============================] - 49s 18ms/step - loss: 0.0020 - val_loss: 0.0012
Epoch 22/50
2807/2807 [==============================] - 46s 16ms/step - loss: 0.0020 - val_loss: 0.0012
Epoch 23/50
2807/2807 [==============================] - 45s 16ms/step - loss: 0.0020 - val_loss: 0.0012
Epoch 24/50
2807/2807 [==============================] - 46s 16ms/step - loss: 0.0020 - val_loss: 0.0011
Epoch 25/50
2807/2807 [==============================] - 50s 18ms/step - loss: 0.0019 - val_loss: 0.0017
Epoch 26/50
2807/2807 [==============================] - 50s 18ms/step - loss: 0.0020 - val_loss: 0.0013
Epoch 27/50
2807/2807 [==============================] - 45s 16ms/step - loss: 0.0019 - val_loss: 0.0013
Epoch 28/50
2807/2807 [==============================] - 46s 16ms/step - loss: 0.0019 - val_loss: 0.0014
Epoch 29/50
2807/2807 [==============================] - 46s 16ms/step - loss: 0.0019 - val_loss: 0.0012
Epoch 30/50
2807/2807 [==============================] - 54s 19ms/step - loss: 0.0019 - val_loss: 0.0013
Epoch 31/50
2807/2807 [==============================] - 46s 16ms/step - loss: 0.0019 - val_loss: 0.0012
Epoch 32/50
2807/2807 [==============================] - 45s 16ms/step - loss: 0.0019 - val_loss: 0.0012
Epoch 33/50
2807/2807 [==============================] - 48s 17ms/step - loss: 0.0019 - val_loss: 0.0012
Epoch 34/50
2807/2807 [==============================] - 53s 19ms/step - loss: 0.0019 - val_loss: 0.0012
Epoch 35/50
2807/2807 [==============================] - 49s 17ms/step - loss: 0.0019 - val_loss: 0.0012
Epoch 36/50
2807/2807 [==============================] - 48s 17ms/step - loss: 0.0019 - val_loss: 0.0015
```

[10]
```
Epoch 37/50
2807/2807 [==============================] - 53s 19ms/step - loss: 0.0019 - val_loss: 0.0012
Epoch 38/50
2807/2807 [==============================] - 54s 19ms/step - loss: 0.0019 - val_loss: 0.0014
Epoch 39/50
2807/2807 [==============================] - 47s 17ms/step - loss: 0.0018 - val_loss: 0.0011
Epoch 40/50
2807/2807 [==============================] - 58s 21ms/step - loss: 0.0019 - val_loss: 0.0011
Epoch 41/50
2807/2807 [==============================] - 47s 17ms/step - loss: 0.0018 - val_loss: 0.0012
Epoch 42/50
2807/2807 [==============================] - 52s 19ms/step - loss: 0.0018 - val_loss: 0.0012
Epoch 43/50
2807/2807 [==============================] - 48s 17ms/step - loss: 0.0019 - val_loss: 0.0012
Epoch 44/50
2807/2807 [==============================] - 45s 16ms/step - loss: 0.0018 - val_loss: 0.0012
Epoch 45/50
2807/2807 [==============================] - 45s 16ms/step - loss: 0.0019 - val_loss: 0.0013
Epoch 46/50
2807/2807 [==============================] - 46s 16ms/step - loss: 0.0018 - val_loss: 0.0014
Epoch 47/50
2807/2807 [==============================] - 47s 17ms/step - loss: 0.0018 - val_loss: 0.0014
Epoch 48/50
2807/2807 [==============================] - 46s 17ms/step - loss: 0.0018 - val_loss: 0.0015
Epoch 49/50
2807/2807 [==============================] - 44s 16ms/step - loss: 0.0018 - val_loss: 0.0014
Epoch 50/50
2807/2807 [==============================] - 45s 16ms/step - loss: 0.0018 - val_loss: 0.0012
```

[11]
```
# Plotting the Loss Values of a Fitted Model
plt.figure(figsize=(10, 6))
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.legend()
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Training and Validation Loss Over Epochs')
plt.show()
```

[11]



[12]
```
#predicting train and test data from x_train and x_test
train_predicted = model.predict(X_train.reshape(X_train.shape))
test_predicted = model.predict(X_test.reshape(X_test.shape))
```

```
[12] 110/110 [==============================] - 1s 10ms/step
     27/27 [==============================] - 0s 9ms/step
```

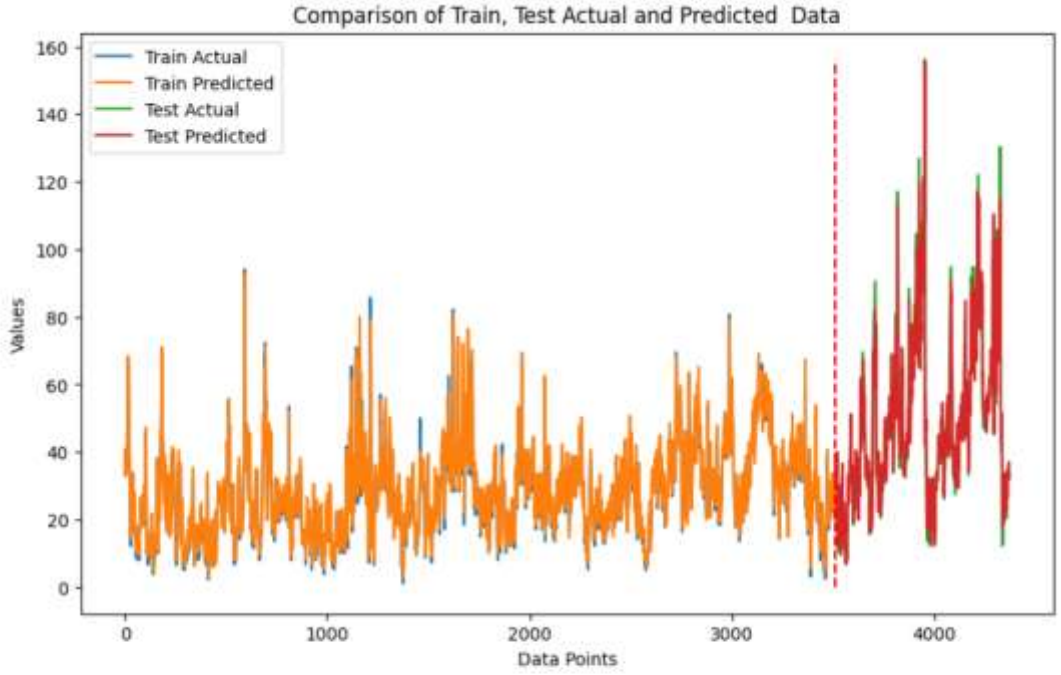
```
[13] #Repeating three column of same data to inverse it into same shape
     train_prediction_copies = np.repeat(train_predicted, 3, axis = -1  )
     test_prediction_copies = np.repeat(test_predicted, 3, axis = -1  )
     train_actual = np.repeat(y_train, 3, axis = -1  )
     test_actual = np.repeat(y_test, 3, axis = -1  )
```

```
[14] #Scale back to orginal scale for both train and test predicted data by selecting first colum from repeated cloum
     train_predicted=scaler.inverse_transform(np.reshape(train_prediction_copies,(len(train_predicted),3)))[:,0]
     test_predicted=scaler.inverse_transform(np.reshape(test_prediction_copies,(len(test_predicted),3)))[:,0]

     #Scale back to orginal scale for both train and test actual databy selecting first colum from repeated cloum
     train_actual=scaler.inverse_transform(np.reshape(train_actual,(len(X_train),3)))[:,0]
     test_actual=scaler.inverse_transform(np.reshape(test_actual,(len(y_test),3)))[:,0]
```

```
[15] #creating numpy array data for plotting actual and predicted data
     x1 = np.arange(0, len(train_actual))
     x2 = np.arange(len(train_actual), len(train_actual)+len(test_actual))
```

```
[16] # Plotting Both Train and Test Actual Data
     plt.figure(figsize=(10, 6))
     plt.plot(x1, train_actual)
     plt.plot(x1, train_predicted)
     plt.plot(x2, test_actual)
     plt.plot(x2, test_predicted)
     plt.legend(['Train Actual', 'Train Predicted', 'Test Actual', 'Test Predicted'])
     plt.xlabel('Data Points')
     plt.ylabel('Values')
     plt.title('Comparison of Train, Test Actual and Predicted  Data')
     plt.vlines(x=len(train_actual), color='r', linestyles='dashed', ymin=0, ymax=max(test_actual))
     plt.show()
```

[16]




```
[17] #Evulate the model performance

     #Mean Squared Error For Testing and Training data
     mse_train = mean_squared_error(train_actual, train_predicted)
     print("Train Mean Squared Error :", mse_train)
     mse_test = mean_squared_error(test_actual, test_predicted)
```

```
[17] print("Test Mean Squared Error:", mse_test)

     # Root Mean Squared Error For Testing and Training data
     rmse_train = np.sqrt(mse_train)
     print("Train Root Mean Squared Error:", rmse_train)
     rmse_test = np.sqrt(mse_test)
     print("Test Root Mean Squared Error:", rmse_test)

     #Mean Absolute Error For Testing and Training data
     mae_train = mean_absolute_error(train_actual, train_predicted)
     print("Train Mean Absolute Error:", mae_train)
     mae_test = mean_absolute_error(test_actual, test_predicted)
     print("Test Mean Absolute Error:", mae_test)

     #R-Sqaured For Testing and Training data
     r2_train = r2_score(train_actual, train_predicted)
     print("Train R-squared:", r2_train)
     r2_test = r2_score(test_actual, test_predicted)
     print("Test R-squared:", r2_test)

     Train Mean Squared Error : 14.248789615968299
     Test Mean Squared Error: 62.09250484686159
     Train Root Mean Squared Error: 3.774756894949435
     Test Root Mean Squared Error: 7.8798797482488006
     Train Mean Absolute Error: 2.62633760204142
     Test Mean Absolute Error: 4.93700171425368
     Train R-squared: 0.9154247895532461
     Test R-squared: 0.9004809352319894
```

LSTM Model 3

Predicting Air Quality Index(AQI) With single multivariate Column NO2, O3, CO, SO2

```
#importing necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential, load_model
from keras.layers import LSTM, Dense
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
```

```
[2] #acessing google drive from google colab
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
[3] # Load the dataset
df = pd.read_csv('/content/drive/MyDrive/Dissertation/clean_data.csv')
df
```

	pubtime	PM2.5	PM10	NO2	O3	CO	SO2	AQI
0	2020-06-30 00:00:00	11.0	23.0	26.0	70.0	0.7	5.0	23.0
1	2020-06-30 01:00:00	14.0	23.0	29.0	54.0	0.8	5.0	23.0
2	2020-06-30 02:00:00	11.0	23.0	31.0	48.0	0.7	5.0	23.0
3	2020-06-30 03:00:00	12.0	25.0	42.0	26.0	0.7	5.0	25.0
4	2020-06-30 04:00:00	7.0	29.0	36.0	22.0	0.7	5.0	29.0

[3]

4412	2020-12-30 20:00:00	21.0	59.0	35.0	42.0	0.8	6.0	55.0
4413	2020-12-30 21:00:00	22.0	65.0	35.0	40.0	0.8	6.0	58.0
4414	2020-12-30 22:00:00	26.0	64.0	30.0	43.0	0.8	6.0	57.0
4415	2020-12-30 23:00:00	24.0	58.0	30.0	42.0	0.8	6.0	54.0
4416	2020-12-31 00:00:00	23.0	61.0	25.0	46.0	0.8	7.0	56.0

4417 rows × 8 columns

Next steps: View recommended plots

LSTM Model Creation and Training

[4]
```
#Selecting only one column to predict AQI from AQI previous data
df= df[['NO2','O3','CO','SO2','AQI']]
df.head()
```

	NO2	O3	CO	SO2	AQI
0	26.0	70.0	0.7	5.0	23.0
1	29.0	54.0	0.8	5.0	23.0
2	31.0	48.0	0.7	5.0	23.0
3	42.0	26.0	0.7	5.0	25.0
4	36.0	22.0	0.7	5.0	29.0

Next steps: View recommended plots

[5]
```
#spliting dataset for training and testing phase
train_size = int(len(df) * 0.8)
train_data = df.iloc[:train_size]
test_data = df.iloc[train_size:]
```

[6]
```
#scaling dataset to 0 and 1 by using MinMaxScaler
scaler = MinMaxScaler(feature_range=(0, 1))
train_scaled = scaler.fit_transform(train_data)
test_scaled = scaler.transform(test_data)
```

[7]
```
#Creating input-output pairs by splitting dataset into a sequence of input_size
def create_sequences(data, input_size):
    X, y = [], []
    for i in range(len(data) - input_size):
        X.append(data[i:i + input_size, :])
        y.append(data[i + input_size, -1])
    return np.array(X), np.array(y)

input_size = 24

X_train, y_train = create_sequences(train_scaled, input_size)
X_test, y_test = create_sequences(test_scaled, input_size)
```

[8]
```
#priniting the shape of x_train and X_test data
print(X_train.shape)
print(X_test.shape)
```

```
(3509, 24, 5)
(860, 24, 5)
```

[9]
```
#creating a LSTM model
model = Sequential()
```

```
[9]  model.add(LSTM(100, activation='relu',input_shape=(X_train.shape[1], X_train.shape[2])))
     model.add(Dense(1))
     model.compile(optimizer='adam', loss='mse')
```

```
[10] #fitting the model
     history = model.fit(X_train, y_train, epochs=50, batch_size=1,validation_split=0.2)
```

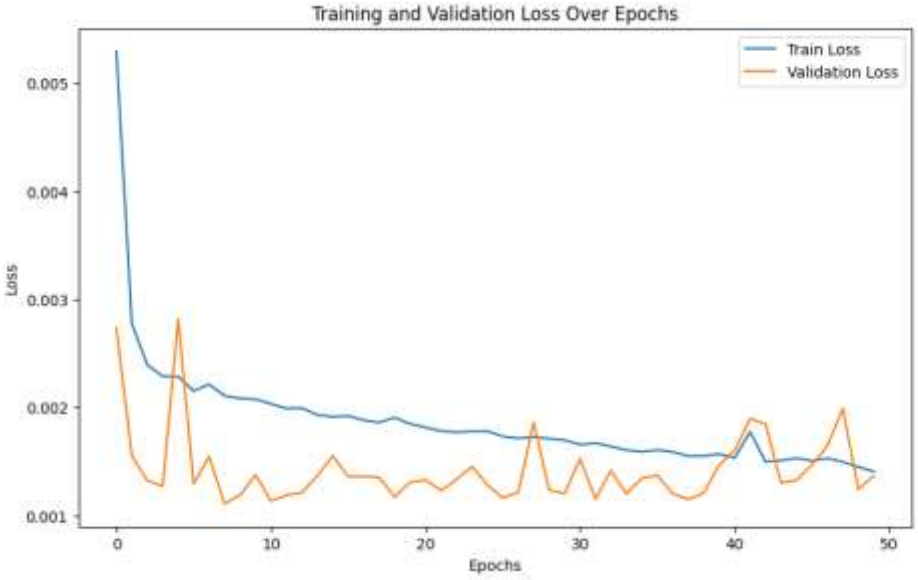
```
Epoch 1/50
2807/2807 [==============================] - 46s 16ms/step - loss: 0.0053 - val_loss: 0.0027
Epoch 2/50
2807/2807 [==============================] - 42s 15ms/step - loss: 0.0028 - val_loss: 0.0016
Epoch 3/50
2807/2807 [==============================] - 42s 15ms/step - loss: 0.0024 - val_loss: 0.0013
Epoch 4/50
2807/2807 [==============================] - 43s 15ms/step - loss: 0.0023 - val_loss: 0.0013
Epoch 5/50
2807/2807 [==============================] - 42s 15ms/step - loss: 0.0023 - val_loss: 0.0028
Epoch 6/50
2807/2807 [==============================] - 46s 16ms/step - loss: 0.0022 - val_loss: 0.0013
Epoch 7/50
2807/2807 [==============================] - 42s 15ms/step - loss: 0.0022 - val_loss: 0.0015
Epoch 8/50
2807/2807 [==============================] - 41s 14ms/step - loss: 0.0021 - val_loss: 0.0011
Epoch 9/50
2807/2807 [==============================] - 46s 16ms/step - loss: 0.0021 - val_loss: 0.0012
Epoch 10/50
2807/2807 [==============================] - 41s 15ms/step - loss: 0.0021 - val_loss: 0.0014
Epoch 11/50
2807/2807 [==============================] - 45s 16ms/step - loss: 0.0020 - val_loss: 0.0011
Epoch 12/50
2807/2807 [==============================] - 42s 15ms/step - loss: 0.0020 - val_loss: 0.0012
Epoch 13/50
2807/2807 [==============================] - 41s 15ms/step - loss: 0.0020 - val_loss: 0.0012
Epoch 14/50
2807/2807 [==============================] - 41s 15ms/step - loss: 0.0019 - val_loss: 0.0014
Epoch 15/50
2807/2807 [==============================] - 43s 15ms/step - loss: 0.0019 - val_loss: 0.0016
Epoch 16/50
2807/2807 [==============================] - 43s 15ms/step - loss: 0.0019 - val_loss: 0.0014
Epoch 17/50
```

```
[10] 2807/2807 [==============================] - 41s 15ms/step - loss: 0.0019 - val_loss: 0.0014
Epoch 18/50
2807/2807 [==============================] - 42s 15ms/step - loss: 0.0019 - val_loss: 0.0014
Epoch 19/50
2807/2807 [==============================] - 43s 15ms/step - loss: 0.0019 - val_loss: 0.0012
Epoch 20/50
2807/2807 [==============================] - 43s 15ms/step - loss: 0.0018 - val_loss: 0.0013
Epoch 21/50
2807/2807 [==============================] - 47s 17ms/step - loss: 0.0018 - val_loss: 0.0013
Epoch 22/50
2807/2807 [==============================] - 43s 15ms/step - loss: 0.0018 - val_loss: 0.0012
Epoch 23/50
2807/2807 [==============================] - 43s 15ms/step - loss: 0.0018 - val_loss: 0.0013
Epoch 24/50
2807/2807 [==============================] - 42s 15ms/step - loss: 0.0018 - val_loss: 0.0015
Epoch 25/50
2807/2807 [==============================] - 43s 15ms/step - loss: 0.0018 - val_loss: 0.0013
Epoch 26/50
2807/2807 [==============================] - 44s 16ms/step - loss: 0.0017 - val_loss: 0.0012
Epoch 27/50
2807/2807 [==============================] - 42s 15ms/step - loss: 0.0017 - val_loss: 0.0012
Epoch 28/50
2807/2807 [==============================] - 40s 14ms/step - loss: 0.0017 - val_loss: 0.0019
Epoch 29/50
2807/2807 [==============================] - 40s 14ms/step - loss: 0.0017 - val_loss: 0.0012
Epoch 30/50
2807/2807 [==============================] - 40s 14ms/step - loss: 0.0017 - val_loss: 0.0012
Epoch 31/50
2807/2807 [==============================] - 42s 15ms/step - loss: 0.0017 - val_loss: 0.0015
Epoch 32/50
2807/2807 [==============================] - 39s 14ms/step - loss: 0.0017 - val_loss: 0.0012
Epoch 33/50
2807/2807 [==============================] - 40s 14ms/step - loss: 0.0016 - val_loss: 0.0014
Epoch 34/50
2807/2807 [==============================] - 40s 14ms/step - loss: 0.0016 - val_loss: 0.0012
Epoch 35/50
2807/2807 [==============================] - 40s 14ms/step - loss: 0.0016 - val_loss: 0.0013
Epoch 36/50
2807/2807 [==============================] - 43s 15ms/step - loss: 0.0016 - val_loss: 0.0014
Epoch 37/50
```

```
[10] 2807/2807 [==============================] - 40s 14ms/step - loss: 0.0016 - val_loss: 0.0012
     Epoch 38/50
     2807/2807 [==============================] - 41s 15ms/step - loss: 0.0016 - val_loss: 0.0012
     Epoch 39/50
     2807/2807 [==============================] - 39s 14ms/step - loss: 0.0016 - val_loss: 0.0012
     Epoch 40/50
     2807/2807 [==============================] - 41s 15ms/step - loss: 0.0016 - val_loss: 0.0015
     Epoch 41/50
     2807/2807 [==============================] - 42s 15ms/step - loss: 0.0015 - val_loss: 0.0016
     Epoch 42/50
     2807/2807 [==============================] - 44s 16ms/step - loss: 0.0018 - val_loss: 0.0019
     Epoch 43/50
     2807/2807 [==============================] - 43s 15ms/step - loss: 0.0015 - val_loss: 0.0018
     Epoch 44/50
     2807/2807 [==============================] - 42s 15ms/step - loss: 0.0015 - val_loss: 0.0013
     Epoch 45/50
     2807/2807 [==============================] - 41s 15ms/step - loss: 0.0015 - val_loss: 0.0013
     Epoch 46/50
     2807/2807 [==============================] - 40s 14ms/step - loss: 0.0015 - val_loss: 0.0015
     Epoch 47/50
     2807/2807 [==============================] - 41s 15ms/step - loss: 0.0015 - val_loss: 0.0016
     Epoch 48/50
     2807/2807 [==============================] - 39s 14ms/step - loss: 0.0015 - val_loss: 0.0020
     Epoch 49/50
     2807/2807 [==============================] - 41s 15ms/step - loss: 0.0015 - val_loss: 0.0012
     Epoch 50/50
     2807/2807 [==============================] - 42s 15ms/step - loss: 0.0014 - val_loss: 0.0014
```

```
[11] # Plotting the Loss Values of a Fitted Model
     plt.figure(figsize=(10, 6))
     plt.plot(history.history['loss'], label='Train Loss')
     plt.plot(history.history['val_loss'], label='Validation Loss')
     plt.legend()
     plt.xlabel('Epochs')
     plt.ylabel('Loss')
     plt.title('Training and Validation Loss Over Epochs')
     plt.show()
```

[11]




```
[12] #predicting train and test data from x_train and x_test
     train_predicted = model.predict(X_train.reshape(X_train.shape))
     test_predicted = model.predict(X_test.reshape(X_test.shape))
```

```
110/110 [==============================] - 2s 9ms/step
27/27 [==============================] - 0s 7ms/step
```

```
[12]
110/110 [==============================] - 2s 9ms/step
27/27 [==============================] - 0s 7ms/step
```

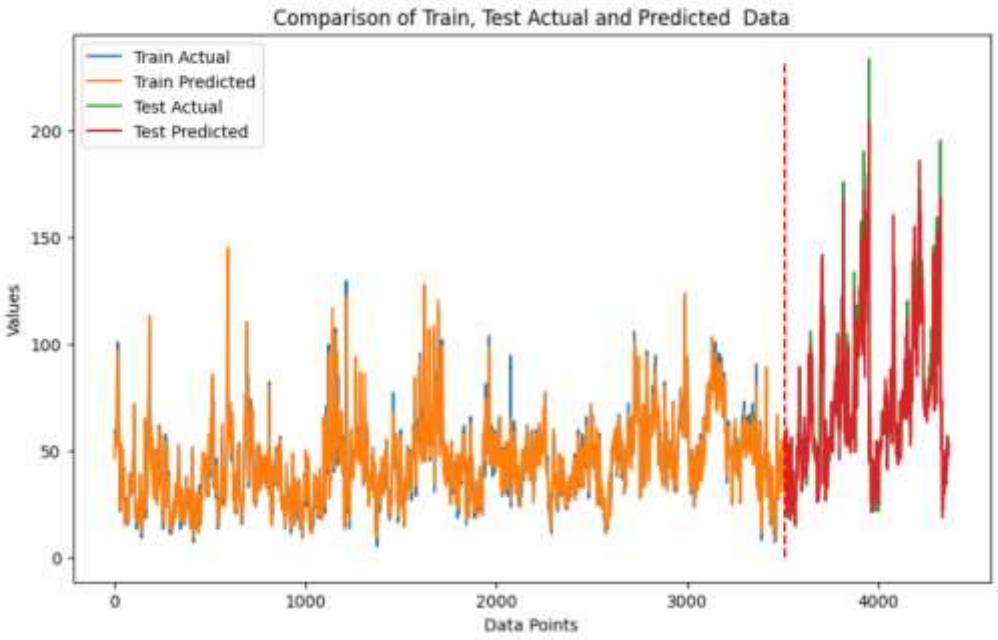
```
[13] #Repeating three column of same data to inverse it into same shape
train_prediction_copies = np.repeat(train_predicted, 5, axis = -1  )
test_prediction_copies = np.repeat(test_predicted, 5, axis = -1  )
train_actual = np.repeat(y_train, 5, axis = -1  )
test_actual = np.repeat(y_test, 5, axis = -1  )
```

```
#Scale back to orginal scale for both train and test predicted data by selecting first colum from repeated cloum
train_predicted=scaler.inverse_transform(np.reshape(train_prediction_copies,(len(train_predicted),5)))[:,0]
test_predicted=scaler.inverse_transform(np.reshape(test_prediction_copies,(len(test_predicted),5)))[:,0]

#Scale back to orginal scale for both train and test actual databy selecting first colum from repeated cloum
train_actual=scaler.inverse_transform(np.reshape(train_actual,(len(X_train),5)))[:,0]
test_actual=scaler.inverse_transform(np.reshape(test_actual,(len(y_test),5)))[:,0]
```

```
[15] #creating numpy array data for plotting actual and predicted data
x1 = np.arange(0, len(train_actual))
x2 = np.arange(len(train_actual), len(train_actual)+len(test_actual))
```

```
[16] # Plotting Both Train and Test Actual Data
plt.figure(figsize=(10, 6))
plt.plot(x1, train_actual)
plt.plot(x1, train_predicted)
plt.plot(x2, test_actual)
plt.plot(x2, test_predicted)
plt.legend(['Train Actual', 'Train Predicted', 'Test Actual', 'Test Predicted'])
plt.xlabel('Data Points')
plt.ylabel('Values')
plt.title('Comparison of Train, Test Actual and Predicted  Data')
plt.vlines(x=len(train_actual), color='r', linestyles='dashed', ymin=0, ymax=max(test_actual))
plt.show()
```

```
[16]
```




```
[17] #Evulate the model performance

#Mean Squared Error For Testing and Training data
mse_train = mean_squared_error(train_actual, train_predicted)
print("Train Mean Squared Error :", mse_train)
mse_test = mean_squared_error(test_actual, test_predicted)
```

```
[17] print("Test Mean Squared Error:", mse_test)

     # Root Mean Squared Error For Testing and Training data
     rmse_train = np.sqrt(mse_train)
     print("Train Root Mean Squared Error:", rmse_train)
     rmse_test = np.sqrt(mse_test)
     print("Test Root Mean Squared Error:", rmse_test)

     #Mean Absolute Error For Testing and Training data
     mae_train = mean_absolute_error(train_actual, train_predicted)
     print("Train Mean Absolute Error:", mae_train)
     mae_test = mean_absolute_error(test_actual, test_predicted)
     print("Test Mean Absolute Error:", mae_test)

     #R-Squared For Testing and Training data
     r2_train = r2_score(train_actual, train_predicted)
     print("Train R-squared:", r2_train)
     r2_test = r2_score(test_actual, test_predicted)
     print("Test R-squared:", r2_test)

     Train Mean Squared Error : 25.051413737837454
     Test Mean Squared Error: 150.88986985087803
     Train Root Mean Squared Error: 5.005138733125931
     Test Root Mean Squared Error: 12.283723777864676
     Train Mean Absolute Error: 3.5121789581446965
     Test Mean Absolute Error: 7.942705302856291
     Train R-squared: 0.9314792488217163
     Test R-squared: 0.8885572159797983
```

LSTM Model 4

Predicting Air Quality Index(AQI) With single multivariate Column PM2.5, PM10,NO2 O3, CO, SO2

```
[1] #Importing necessary libraries
    import pandas as pd
    import numpy as np
    import matplotlib.pyplot as plt
    from sklearn.preprocessing import MinMaxScaler
    from keras.models import Sequential, load_model
    from keras.layers import LSTM, Dense
    from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
```

```
[2] #acessing google drive from google colab
    from google.colab import drive
    drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
[3] # Load the dataset
    df = pd.read_csv('/content/drive/MyDrive/Dissertation/clean_data.csv')
    df
```

	pubtime	PM2.5	PM10	NO2	O3	CO	SO2	AQI
0	2020-06-30 00:00:00	11.0	23.0	26.0	70.0	0.7	5.0	23.0
1	2020-06-30 01:00:00	14.0	23.0	29.0	54.0	0.8	5.0	23.0
2	2020-06-30 02:00:00	11.0	23.0	31.0	48.0	0.7	5.0	23.0
3	2020-06-30 03:00:00	12.0	25.0	42.0	26.0	0.7	5.0	25.0

[3]

4	2020-06-30 04:00:00	7.0	29.0	36.0	22.0	0.7	5.0	29.0
...	...	...	...	...	...	...	...	...	...
4412	2020-12-30 20:00:00	21.0	59.0	35.0	42.0	0.8	6.0	55.0
4413	2020-12-30 21:00:00	22.0	65.0	35.0	40.0	0.8	6.0	58.0
4414	2020-12-30 22:00:00	26.0	64.0	30.0	43.0	0.8	6.0	57.0
4415	2020-12-30 23:00:00	24.0	58.0	30.0	42.0	0.8	6.0	54.0
4416	2020-12-31 00:00:00	23.0	61.0	25.0	46.0	0.8	7.0	56.0

4417 rows × 8 columns

Next steps: View recommended plots

LSTM Model Creation and Training

[4]
```
#Selecting only one column to predict AQI from AQI previous data
df= df[['PM2.5','PM10','NO2','O3','CO','SO2','AQI']]
df.head()
```

	PM2.5	PM10	NO2	O3	CO	SO2	AQI
0	11.0	23.0	26.0	70.0	0.7	5.0	23.0
1	14.0	23.0	29.0	54.0	0.8	5.0	23.0
2	11.0	23.0	31.0	48.0	0.7	5.0	23.0
3	12.0	25.0	42.0	26.0	0.7	5.0	25.0
4	7.0	29.0	36.0	22.0	0.7	5.0	29.0

[5]
```
#spliting dataset for training and testing phase
train_size = int(len(df) * 0.8)
train_data = df.iloc[:train_size]
test_data = df.iloc[train_size:]
```

[6]
```
#scaling dataset to 0 and 1 by using MinMaxScaler
scaler = MinMaxScaler(feature_range=(0, 1))
train_scaled = scaler.fit_transform(train_data)
test_scaled = scaler.transform(test_data)
```

[7]
```
#Creating input-output pairs by splitting dataset into a sequence of input_size
def create_sequences(data, input_size):
    X, y = [], []
    for i in range(len(data) - input_size):
        X.append(data[i:i + input_size, :])
        y.append(data[i + input_size, -1])
    return np.array(X), np.array(y)

input_size = 24

X_train, y_train = create_sequences(train_scaled, input_size)
X_test, y_test = create_sequences(test_scaled, input_size)
```

[8]
```
#priniting the shape of x_train and X_test data
print(X_train.shape)
print(X_test.shape)
```

```
(3509, 24, 7)
(860, 24, 7)
```

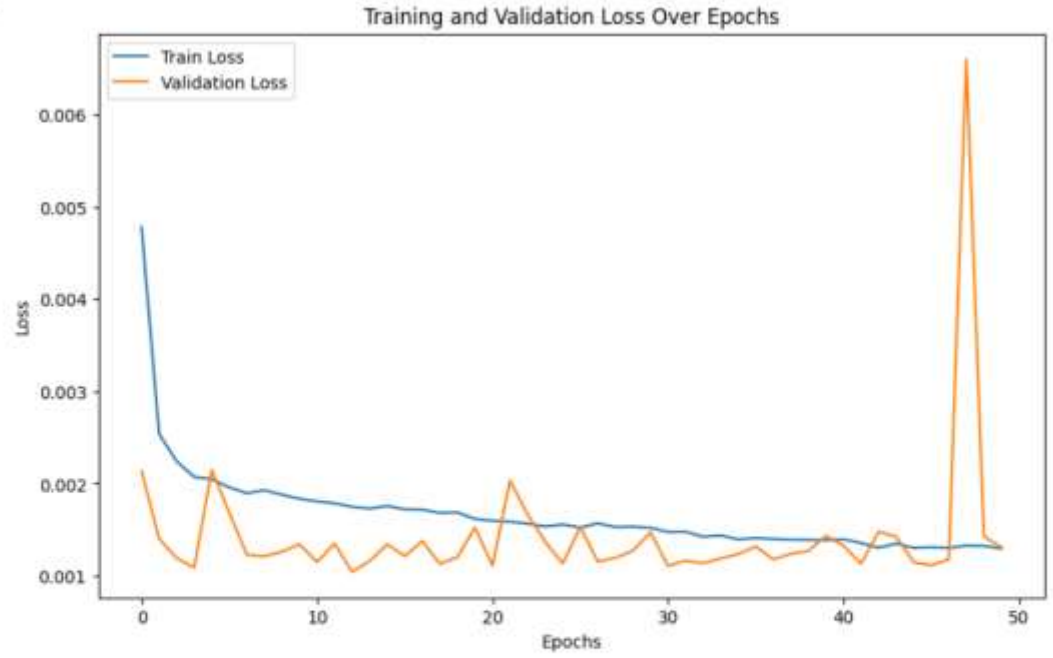
[9]
```
#creating a LSTM model
model = Sequential()
```

```
[9] model.add(LSTM(100, activation='relu',input_shape=(X_train.shape[1], X_train.shape[2])))
    model.add(Dense(1))
    model.compile(optimizer='adam', loss='mse')

[10] #fitting the model
     history = model.fit(X_train, y_train, epochs=50, batch_size=1,validation_split=0.2)

Epoch 1/50
2807/2807 [==============================] - 53s 18ms/step - loss: 0.0048 - val_loss: 0.0021
Epoch 2/50
2807/2807 [==============================] - 39s 14ms/step - loss: 0.0025 - val_loss: 0.0014
Epoch 3/50
2807/2807 [==============================] - 40s 14ms/step - loss: 0.0022 - val_loss: 0.0012
Epoch 4/50
2807/2807 [==============================] - 42s 15ms/step - loss: 0.0021 - val_loss: 0.0011
Epoch 5/50
2807/2807 [==============================] - 40s 14ms/step - loss: 0.0020 - val_loss: 0.0021
Epoch 6/50
2807/2807 [==============================] - 39s 14ms/step - loss: 0.0020 - val_loss: 0.0017
Epoch 7/50
2807/2807 [==============================] - 41s 15ms/step - loss: 0.0019 - val_loss: 0.0012
Epoch 8/50
2807/2807 [==============================] - 39s 14ms/step - loss: 0.0019 - val_loss: 0.0012
Epoch 9/50
2807/2807 [==============================] - 39s 14ms/step - loss: 0.0019 - val_loss: 0.0013
Epoch 10/50
2807/2807 [==============================] - 41s 15ms/step - loss: 0.0018 - val_loss: 0.0013
Epoch 11/50
2807/2807 [==============================] - 37s 13ms/step - loss: 0.0018 - val_loss: 0.0012
Epoch 12/50
2807/2807 [==============================] - 41s 15ms/step - loss: 0.0018 - val_loss: 0.0013
Epoch 13/50
2807/2807 [==============================] - 39s 14ms/step - loss: 0.0017 - val_loss: 0.0010
Epoch 14/50
2807/2807 [==============================] - 38s 14ms/step - loss: 0.0017 - val_loss: 0.0012
Epoch 15/50
2807/2807 [==============================] - 42s 15ms/step - loss: 0.0018 - val_loss: 0.0013
Epoch 16/50
2807/2807 [==============================] - 37s 13ms/step - loss: 0.0017 - val_loss: 0.0012

[10] Epoch 17/50
2807/2807 [==============================] - 39s 14ms/step - loss: 0.0017 - val_loss: 0.0014
Epoch 18/50
2807/2807 [==============================] - 39s 14ms/step - loss: 0.0017 - val_loss: 0.0011
Epoch 19/50
2807/2807 [==============================] - 38s 14ms/step - loss: 0.0017 - val_loss: 0.0012
Epoch 20/50
2807/2807 [==============================] - 40s 14ms/step - loss: 0.0016 - val_loss: 0.0015
Epoch 21/50
2807/2807 [==============================] - 41s 15ms/step - loss: 0.0016 - val_loss: 0.0011
Epoch 22/50
2807/2807 [==============================] - 39s 14ms/step - loss: 0.0016 - val_loss: 0.0020
Epoch 23/50
2807/2807 [==============================] - 39s 14ms/step - loss: 0.0016 - val_loss: 0.0017
Epoch 24/50
2807/2807 [==============================] - 41s 15ms/step - loss: 0.0015 - val_loss: 0.0014
Epoch 25/50
2807/2807 [==============================] - 37s 13ms/step - loss: 0.0016 - val_loss: 0.0011
Epoch 26/50
2807/2807 [==============================] - 40s 14ms/step - loss: 0.0015 - val_loss: 0.0015
Epoch 27/50
2807/2807 [==============================] - 41s 15ms/step - loss: 0.0016 - val_loss: 0.0012
Epoch 28/50
2807/2807 [==============================] - 39s 14ms/step - loss: 0.0015 - val_loss: 0.0012
Epoch 29/50
2807/2807 [==============================] - 40s 14ms/step - loss: 0.0015 - val_loss: 0.0013
Epoch 30/50
2807/2807 [==============================] - 37s 13ms/step - loss: 0.0015 - val_loss: 0.0015
Epoch 31/50
2807/2807 [==============================] - 43s 15ms/step - loss: 0.0015 - val_loss: 0.0011
Epoch 32/50
2807/2807 [==============================] - 41s 15ms/step - loss: 0.0015 - val_loss: 0.0012
Epoch 33/50
2807/2807 [==============================] - 39s 14ms/step - loss: 0.0014 - val_loss: 0.0011
Epoch 34/50
2807/2807 [==============================] - 38s 14ms/step - loss: 0.0014 - val_loss: 0.0012
Epoch 35/50
2807/2807 [==============================] - 39s 14ms/step - loss: 0.0014 - val_loss: 0.0012
Epoch 36/50
2807/2807 [==============================] - 38s 13ms/step - loss: 0.0014 - val_loss: 0.0013
Epoch 37/50
```

```
[10] 2807/2807 [==============================] - 41s 15ms/step - loss: 0.0014 - val_loss: 0.0012
     Epoch 38/50
     2807/2807 [==============================] - 41s 15ms/step - loss: 0.0014 - val_loss: 0.0012
     Epoch 39/50
     2807/2807 [==============================] - 39s 14ms/step - loss: 0.0014 - val_loss: 0.0013
     Epoch 40/50
     2807/2807 [==============================] - 40s 14ms/step - loss: 0.0014 - val_loss: 0.0014
     Epoch 41/50
     2807/2807 [==============================] - 38s 13ms/step - loss: 0.0014 - val_loss: 0.0013
     Epoch 42/50
     2807/2807 [==============================] - 44s 16ms/step - loss: 0.0014 - val_loss: 0.0011
     Epoch 43/50
     2807/2807 [==============================] - 44s 16ms/step - loss: 0.0013 - val_loss: 0.0015
     Epoch 44/50
     2807/2807 [==============================] - 39s 14ms/step - loss: 0.0014 - val_loss: 0.0014
     Epoch 45/50
     2807/2807 [==============================] - 39s 14ms/step - loss: 0.0013 - val_loss: 0.0011
     Epoch 46/50
     2807/2807 [==============================] - 39s 14ms/step - loss: 0.0013 - val_loss: 0.0011
     Epoch 47/50
     2807/2807 [==============================] - 41s 14ms/step - loss: 0.0013 - val_loss: 0.0012
     Epoch 48/50
     2807/2807 [==============================] - 39s 14ms/step - loss: 0.0013 - val_loss: 0.0066
     Epoch 49/50
     2807/2807 [==============================] - 37s 13ms/step - loss: 0.0013 - val_loss: 0.0014
     Epoch 50/50
     2807/2807 [==============================] - 39s 14ms/step - loss: 0.0013 - val_loss: 0.0013
```

```
[11] # Plotting the Loss Values of a Fitted Model
     plt.figure(figsize=(10, 6))
     plt.plot(history.history['loss'], label='Train Loss')
     plt.plot(history.history['val_loss'], label='Validation Loss')
     plt.legend()
     plt.xlabel('Epochs')
     plt.ylabel('Loss')
     plt.title('Training and Validation Loss Over Epochs')
     plt.show()
```

```
[11]
```




```
[12] #predicting train and test data from x_train and x_test
     train_predicted = model.predict(X_train.reshape(X_train.shape))
     test_predicted = model.predict(X_test.reshape(X_test.shape))
```

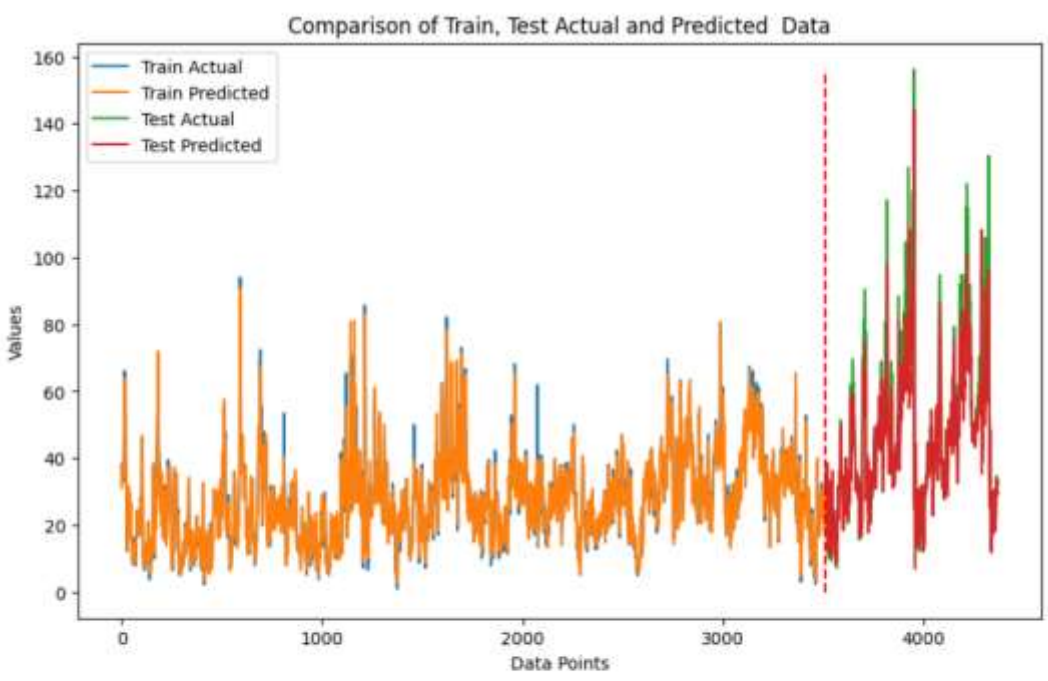
```
     110/110 [==============================] - 1s 8ms/step
     27/27 [==============================] - 0s 7ms/step
```

```
[13] #Repeating three column of same data to inverse it into same shape
     train_prediction_copies = np.repeat(train_predicted, 7, axis = -1  )
     test_prediction_copies = np.repeat(test_predicted, 7, axis = -1  )
     train_actual = np.repeat(y_train, 7, axis = -1  )
     test_actual = np.repeat(y_test, 7, axis = -1  )
```

```
[14] #Scale back to orginal scale for both train and test predicted data by selecting first colum from repeated cloum
     train_predicted=scaler.inverse_transform(np.reshape(train_prediction_copies,(len(train_predicted),7)))[:,0]
     test_predicted=scaler.inverse_transform(np.reshape(test_prediction_copies,(len(test_predicted),7)))[:,0]

     #Scale back to orginal scale for both train and test actual databy selecting first colum from repeated cloum
     train_actual=scaler.inverse_transform(np.reshape(train_actual,(len(X_train),7)))[:,0]
     test_actual=scaler.inverse_transform(np.reshape(test_actual,(len(y_test),7)))[:,0]
```

```
#creating numpy array data for plotting actual and predicted data
x1 = np.arange(0, len(train_actual))
x2 = np.arange(len(train_actual), len(train_actual)+len(test_actual))
```

```
# Plotting Both Train and Test Actual Data
plt.figure(figsize=(10, 6))
plt.plot(x1, train_actual)
plt.plot(x1, train_predicted)
plt.plot(x2, test_actual)
plt.plot(x2, test_predicted)
plt.legend(['Train Actual', 'Train Predicted', 'Test Actual', 'Test Predicted'])
plt.xlabel('Data Points')
plt.ylabel('Values')
plt.title('Comparison of Train, Test Actual and Predicted  Data')
plt.vlines(x=len(train_actual), color='r', linestyles='dashed', ymin=0, ymax=max(test_actual))
plt.show()
```




```
[17] #Evulate the model performance

     #Mean Squared Error For Testing and Training data
     mse_train = mean_squared_error(train_actual, train_predicted)
     print("Train Mean Squared Error :", mse_train)
     mse_test = mean_squared_error(test_actual, test_predicted)
```

```
[17] print("Test Mean Squared Error:", mse_test)

     # Root Mean Squared Error For Testing and Training data
     rmse_train = np.sqrt(mse_train)
     print("Train Root Mean Squared Error:", rmse_train)
     rmse_test = np.sqrt(mse_test)
     print("Test Root Mean Squared Error:", rmse_test)

     #Mean Absolute Error For Testing and Training data
     mae_train = mean_absolute_error(train_actual, train_predicted)
     print("Train Mean Absolute Error:", mae_train)
     mae_test = mean_absolute_error(test_actual, test_predicted)
     print("Test Mean Absolute Error:", mae_test)

     #R-Sqaured For Testing and Training data
     r2_train = r2_score(train_actual, train_predicted)
     print("Train R-squared:", r2_train)
     r2_test = r2_score(test_actual, test_predicted)
     print("Test R-squared:", r2_test)

     Train Mean Squared Error : 9.994013983529502
     Test Mean Squared Error: 113.46995201264183
     Train Root Mean Squared Error: 3.161331046178097
     Test Root Mean Squared Error: 10.652227561061668
     Train Mean Absolute Error: 2.2178106592340368
     Test Mean Absolute Error: 6.516117070379485
     Train R-squared: 0.9406794641056696
     Test R-squared: 0.8181354813850784
```