STAT 654

PREDICTING AIR POLLUTANT CONCENTRATION WITH MULTISENSOR DATA

GROUP 6

Priyadharshini Ramesh Kumar - 534002478

Hasitha Varada - 734004713

Tejashri Kelhe - 135001931

CONTRIBUTION

All three team members—Priyadarshini Ramesh Kumar, Tejashri Kelhe, and Hasitha Varada—contributed equally to every facet of this project, demonstrating a fully collaborative effort. We split up our shared effort equally between writing the report, creating the presentation, and developing the code. Every team member contributed their special knowledge and skills to the project, which helped at every level—from the early phases of code development to the finished report draft. Their knowledge and passion have increased with each stage of this project, thanks to equal work. The team's dedication to preserving a harmonious and cooperative working dynamic is demonstrated by the equitable division of labor and the placement of shared responsibility.

ABSTRACT

Pollution is a great problem for public health and environment protection. In this project, we aim to study the Air Quality dataset, where we try to find how pollution varies during the day, when the peaks happen, and how pollutant concentrations are related to each other in order to predict these pollutant concentrations. The data is collected from a gas multi-sensor device that is deployed in a heavily polluted area of an Italian city. The dataset is collected from March 10th 2004 to February 10th 2005 (one year). We are given the hourly averaged responses of 12 arrays of gas multi-sensor exposed to different gaseous mixtures and an hourly averaged reference value which represents the ground truth. The data contains various pieces of information like, date and time attributes, concentrations of various pollutants (CO, NMHC, C6H6, NOx, NO2, and the O3), and meteorological attributes, like, temperature, relative humidity and absolute humidity. The target is to estimate the gas concentration for a particular pollutant using various Machine Learning models.

1. INTRODUCTION

1.1 PROBLEM STATEMENT

One of the biggest risks to human health, urban sustainability, and environmental quality is air pollution. Pollutants like CO have continued to climb to dangerous levels despite stringent restrictions and improvements in emission management, especially in metropolitan areas. It is challenging to forecast and understand air quality due to its complexity and wide range of influencing elements. Predictive analytics is therefore essential for understanding and forecasting pollutant levels, which is essential for putting preventive measures meant to lessen air pollution into action.

Air quality is the issue that we are attempting to resolve because it is so vital to public health. We have access to a dataset of hourly measurements collected over the course of a full year in a contaminated Italian city from multiple metal oxide chemical sensors. Our work's primary goal is to accurately estimate a specific pollutant's concentration using machine learning algorithms and data from a chemical sensor network. Our focus in pollution prediction is on carbon monoxide.

1.2 OBJECTIVE

Our research has two main objectives, both of which are crucial to advancing our understanding of and ability to manage air quality in urban settings. First, to comprehend how different types of air pollution interact with one another. Such an investigation would identify the sources of emissions and the fluctuations to which they are susceptible in response to changes in diverse environmental variables. These kinds of discoveries are essential for both public health and urban development since they direct practical approaches to pollution control.

Our second goal is to create advanced predictive models that can identify intricate patterns in the data, demonstrating the integration of machine learning and statistical research. By applying machine learning techniques, we should be able to create a reliable forecast model of the amounts of pollutants. We hope to see a predictive system using these models that can forecast rising pollution levels and alert the public to impending poor air quality so they can take the appropriate precautions.

The ultimate objective is to construct a predictive machine learning model capable of estimating pollutant concentration levels in the environment based on multisensor data, thus contributing to more effective air pollution monitoring and potentially enabling preemptive measures to safeguard public health.

2. LITERATURE SURVEY

Author Name/ Journal/ Year	**Title of the paper**	**Approach**	**Pros/Cons**
World Health Organization, 2022	Ambient (Outdoor) Air Pollution [1]	The fact sheet synthesizes data from global health studies and provides an overview of the health impacts associated with exposure to fine particulate matter, carbon monoxide (CO), ozone (O3), nitrogen dioxide (NO2), and sulfur dioxide (SO2).	Offers global and up-to-date insights into the health burden from air pollution. Lacks detailed technical data and analysis that might be found in full research papers.
Ishita Chanana, Aparajita Sharma, Pradeep Kumar, Lokender Kumar, Sourabh Kulshreshtha, Sanjay Kumar/ Fire / 2023	Combustion and Stubble Burning: A Major Concern for the Environment and Human Health [2]	It talks about the issues related to uncontrolled combustion activities, the origins and distribution of pollutants, and methods for mitigating their effects on the environment and public health.	Discusses the broad range of pollutants from such activities, including greenhouse gases and particulate matter. The complexity of some discussed mitigation strategies may not be easily understood by laypersons without sufficient background knowledge.

Vicki MacMurdo (Anoka-Ramsey Community College) / LibreTexts/ 2021	16.4:Air Pollution [3]	The document provides a comprehensive educational overview of air pollution, detailing types of pollutants, their sources, and effects on health and the environment.	It discusses both natural and anthropogenic sources of air pollution, with an emphasis on understanding human-caused pollution which is actionable and can be mitigated. The paper may not address specific case studies or provide localized insights into air pollution issues.
Our Nation's Air - Trends Through / U.S/ Environmental Protection Agency (EPA) / 2020	Our Nation's Air: Trends Through 2020 [4]	It shows a significant reduction in the emissions of key pollutants by 78%, alongside a growing economy. The report notes declines in average concentrations of harmful air pollutants across the nation from 1990 to 2020, including reductions in carbon monoxide, lead, nitrogen dioxide, ozone, and particulate matter	Significant reduction in key air pollutants since 1970, with a 78% decrease in the emissions of criteria pollutants. Air quality is still a concern, especially in areas affected by wildfires and other natural events that can worsen pollution levels.

3. DATA PREPROCESSING AND EDA

3.1 INTRODUCTION

This portion of our task was preprocessing, which we had to complete prior to data analysis. Preprocessing mostly involved resolving anomalies and missing values, as well as sorting out discrepancies. This stage serves as the basis for exploratory data analysis (EDA), which is how we start identifying the fundamentals of the data and get preliminary insights while preparing the dataset for a more sophisticated approach to machine learning techniques.

3.2 ABOUT THE DATASET

The Air Quality dataset contains 9,357 records and 15 columns, offering a snapshot of urban air quality over a calendar year. It is available in the UCI repository [5]. The sensors are designed to collect data on several air pollutants, including nitrogen dioxide (NO2), total nitrogen oxides (NOx), benzene (C6H6), non-methane hydrocarbons (NMHCs), ozone (O3), and carbon monoxide (CO).As demonstrated in Figure 3.2.1, our dataset includes features from a diverse range of environmental parameters—from humidity and temperature through concentration levels of various pollutants to associated sensor readings for target gases. This dataset contains the values of date and time, hourly averaged concentrations of various pollutants (ground truth values), and hourly averaged sensor readings that are targeted to detect specific gases. Values for absolute, relative, and temperature humidity are also available. Therefore, the 15 columns are: 'Date', 'Time', 'CO(GT)', 'NMHC(GT)', 'C6H6(GT)', 'NOx(GT)', 'NO2(GT)', 'PT08.S3(NOx)', 'PT08.S4(NO2)', 'PT08.S5(O3)', 'PT08.S1(CO)', 'PT08.S2(NMHC)', 'T', 'RH', and 'AH'.

Where, CO(GT)', 'NMHC(GT)', 'C6H6(GT)', 'NOx(GT)', 'NO2(GT)' are the ground truth values of the pollutants, 'PT08.S3(NOx)', 'PT08.S4(NO2)', 'PT08.S5(O3)', 'PT08.S1(CO)', 'PT08.S2(NMHC)' are their respective sensor readings and 'T', 'RH', and 'AH' represent temperature, relative humidity, and absolute humidity, respectively.

	Date	Time	CO(GT)	PT08.S1(CO)	NMHC(GT)	C6H6(GT)	PT08.S2(NMHC)	NOx(GT)	PT08.S3(NOx)	NO2(GT)	PT08.S4(NO2)	PT08.S5(O3)	T	RH	AH
0	2004-03-10	18:00:00	2.6	1360.00	150	11.881723	1045.50	166.0	1056.25	113.0	1692.00	1267.50	13.600	48.875001	0.757754
1	2004-03-10	19:00:00	2.0	1292.25	112	9.397165	954.75	103.0	1173.75	92.0	1558.75	972.25	13.300	47.700000	0.725487
2	2004-03-10	20:00:00	2.2	1402.00	88	8.997817	939.25	131.0	1140.00	114.0	1554.50	1074.00	11.900	53.975000	0.750239
3	2004-03-10	21:00:00	2.2	1375.50	80	9.228796	948.25	172.0	1092.00	122.0	1583.75	1203.25	11.000	60.000000	0.786713
4	2004-03-10	22:00:00	1.6	1272.25	51	6.518224	835.50	131.0	1205.00	116.0	1490.00	1110.00	11.150	59.575001	0.788794
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
9352	2005-04-04	10:00:00	3.1	1314.25	-200	13.529605	1101.25	471.7	538.50	189.8	1374.25	1728.50	21.850	29.250000	0.756824
9353	2005-04-04	11:00:00	2.4	1162.50	-200	11.355157	1027.00	353.3	603.75	179.2	1263.50	1269.00	24.325	23.725000	0.711864
9354	2005-04-04	12:00:00	2.4	1142.00	-200	12.374538	1062.50	293.0	603.25	174.7	1240.75	1092.00	26.900	18.350000	0.640649
9355	2005-04-04	13:00:00	2.1	1002.50	-200	9.547187	960.50	234.5	701.50	155.7	1041.00	769.75	28.325	13.550000	0.513866
9356	2005-04-04	14:00:00	2.2	1070.75	-200	11.932060	1047.25	265.2	654.00	167.7	1128.50	816.00	28.500	13.125000	0.502804

9357 rows × 15 columns

Fig 3.2.1 Air Quality Dataset

3.3 PREPROCESSING

Missing values analysis and data cleaning: The missing values analysis is critical as it points the required preprocessing actions, such as deleting, imputing, or otherwise filling in the gaps in the data to ensure high-quality modeling and analysis. In this dataset, a measurement reading of "-200" means that the sensor did not capture data; thus, entries with such values are considered to be missing. In this way, we excluded NMHC(GT) and other columns with a high percentage of missing data. The percentage of missing values for each column is presented in Figure 3.3.1. This task provides integrity to the study as it enabled us to work on more precise and full data.

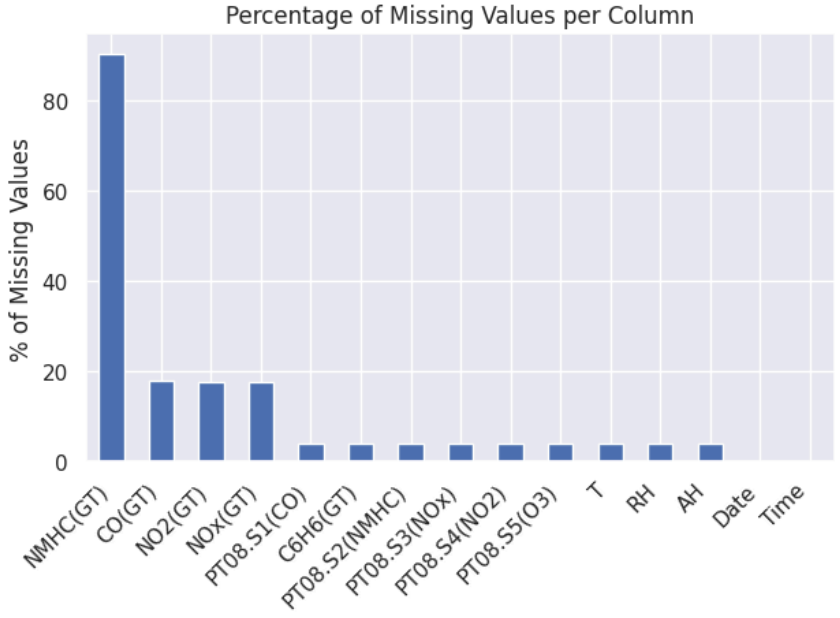

Fig 3.3.1 Percentage of Missing Values per column

The bar chart above gives an idea of the distribution of missing values in every column of the Air Quality dataset. The column 'NMHC(GT)' was dropped from the dataset due to having a remarkably high number of missing values, above 90%. The number of such missing data is a very vast proportion and therefore excluded from the dataset for the sake of the quality of the study. Other columns, including 'CO(GT)', 'NO2(GT)', and 'NOx(GT)', also have significant numbers of missing data, where each has about 18% of their values missing. The other columns, 'PT08.S1(CO)', 'C6H6(GT)', 'PT08.S2(NMHC)', 'PT08.S3(NOx)', 'PT08.S4(NO2)', 'PT08.S5(O3)', and 'T', which represent temperature, relative humidity, and absolute humidity, all have a uniform and much lower percentage of missing values, which account for approximately 3.91% of the values in these columns.

The Pandas module in Python, we used the 'df.dropna()' function to drop all rows with missing entries in the DataFrame 'df'. It did this so that only rows full of data would remain in the dataset. We did that because rows full of missing entries were few compared to the columns other than 'NMHC(GT)'. Our analysis ended with the observation that the removal of these rows would not greatly diminish the size of the dataset. As part of data cleaning, the cleaned dataset became 6,941 rows and 14 columns, and that is the size of the remaining dataset after cleaning it.

Time parsing: Time parsing refers to the process by which the date and time information is converted into structured form that is compatible with detailed data analysis. This process is essential as we attempt to delve into the timing patterns observed within the dataset. This allows us to analyze fluctuations in pollutant concentrations on a month-to-month, week-to-week, and hourly basis. In this way, we enhance our capability to analyze trends and detect specific periods where the concentrations of pollutants are on the rise or decline. We make such an analysis not because it is merely observation but because it forms a foundation on which to base the understanding of periodic changes in air quality and the identification of triggers or correlation within our dataset. Of course, our goal is not to create a time series model but to use these concepts to inform the descriptive and inferential statistics that ultimately could inform further analysis and decision-making processes.

After data transformation, the dataset configuration was done with 6,941 data entries for 17 columns. That showed the expanded framework with newly added time-related variables. The extended database will include an array of original and derived columns: 'Date'—the original date stamp of data capture, 'Month', 'Weekday', and 'Hour'—the date and time segmented into more granular units, and 'DateTime'—a combination of date and time for temporal reference. It is one of the secrets to making a more in-depth analysis of air quality during various periods. One gains insight into the fluctuation of air pollutant levels with the passage of time.

Duplicate Records: To ensure the accuracy of our dataset, we searched for any duplication records. We did this by running the `df.duplicated().sum()` function in the pandas module of Python. This function finds every row for identical copies in the dataset. When we ran this function, the output was zero. In other words, there was no duplication in the data. No duplication assures us that every record in our dataset represents a distinct set of observations-a crucial attribute for ensuring the accuracy of our analysis and any inferences we may later make based on this data.

3.4 OUTLIER DETECTION

Identification: To visualize the data's distribution and potentially spot outliers, a box plot was made for each pollutant and sensor response variable. In the boxplots shown in Figure 3.4.1 for 'CO(GT)', 'PT08.S1(CO)', and 'C6H6(GT)', any data point lying outside the extended lines, or "whiskers," is considered an outlier. These whiskers go out 1.5 times the distance between the 25th and 75th percentiles, known as the interquartile range (IQR).

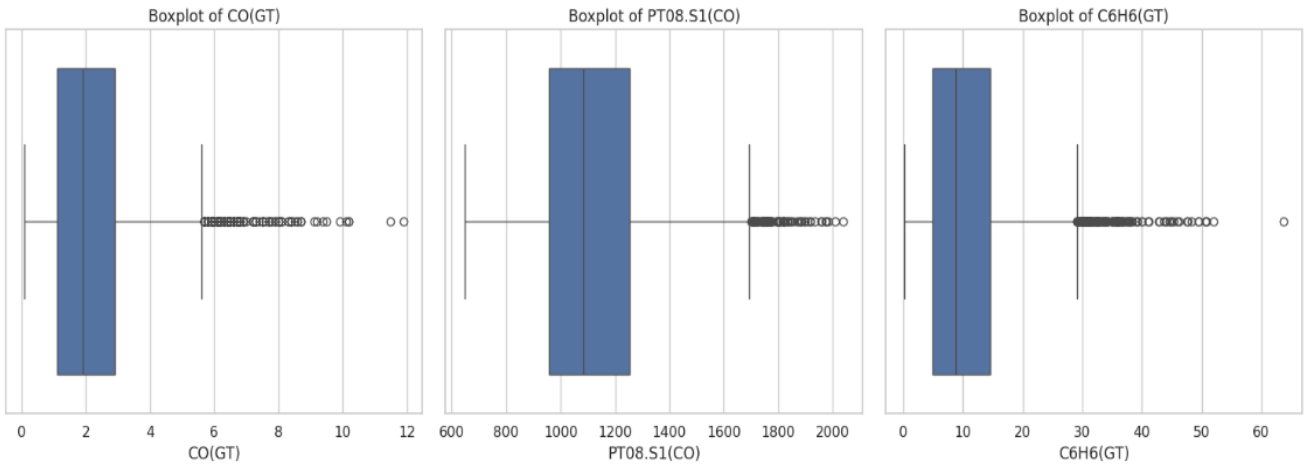

Fig 3.4.1 Boxplots of CO(GT), PT08.S1(CO), C6H6(GT)

Treatment: To handle outliers, we calculated lower and upper boundaries using the Interquartile Range (IQR) method. This helped us figure out which values were extreme enough to be considered outliers. In Figure 3.4.2, we showcase the distributions for each column after outliers have been removed. Our approach was conservative—we only removed the most extreme deviations by setting our threshold at 1.4 times the IQR, slightly less than the commonly used 1.5, to keep as much data as possible. By removing 846 outliers in this way, our goal was to retain data that best represents the usual environmental conditions without the distortion of extreme values. The dataset is now of the dimension 6,095 rows and 17 columns.

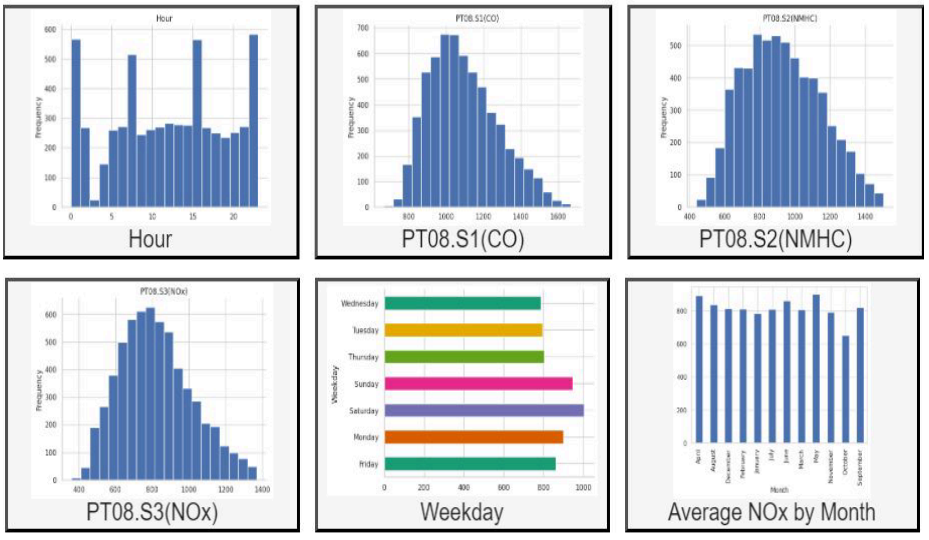

Fig 3.4.2 Distributions of every column

Exclusion of Reference Analyzer Data: We focused our analysis on the sensor response rather than the actual concentrations of pollutants. Consequently, we decided to omit the columns 'CO (GT)', 'NOx (GT)', 'C6H6 (GT)', and 'NO2 (GT)' from our dataset. These columns, which represent the

ground truth measurements from reference analyzers, were less relevant to our study's objectives. By removing these columns, we streamlined the dataset to align with our research focus on sensor behavior and data**.** The dataset is now of the dimension 6,095 rows and 13 columns.

3.5 EXPLORATORY DATA ANALYSIS (EDA)

For our EDA, we have split the dataset into subsets based on month, week, and hour. We created a separate list of DataFrames for each month, day, and hour from a bigger DataFrame named df_new. These lists, month_df_list, day_df_list, and hour_df_list, are then populated with subsets of df_new that correspond to each category. After filtering, our code generates bar plots to visualize average monthly, weekly, and hourly concentrations of carbon monoxide detected by the sensor PT08.S1(CO). We primarily focus on 'CO' as we are interested in predicting it.

Monthly Analysis: We investigate a variation in CO levels on a monthly basis as recorded by the PT08.S1(CO) sensor by making a bar plot. Figure 3.5.1 represents a bar chart that shows a relatively stable pattern of average CO concentration across the year with small variations. The months August and October show a marginally lower and higher average readings respectively, which may indicate seasonal influences or other factors affecting CO levels.

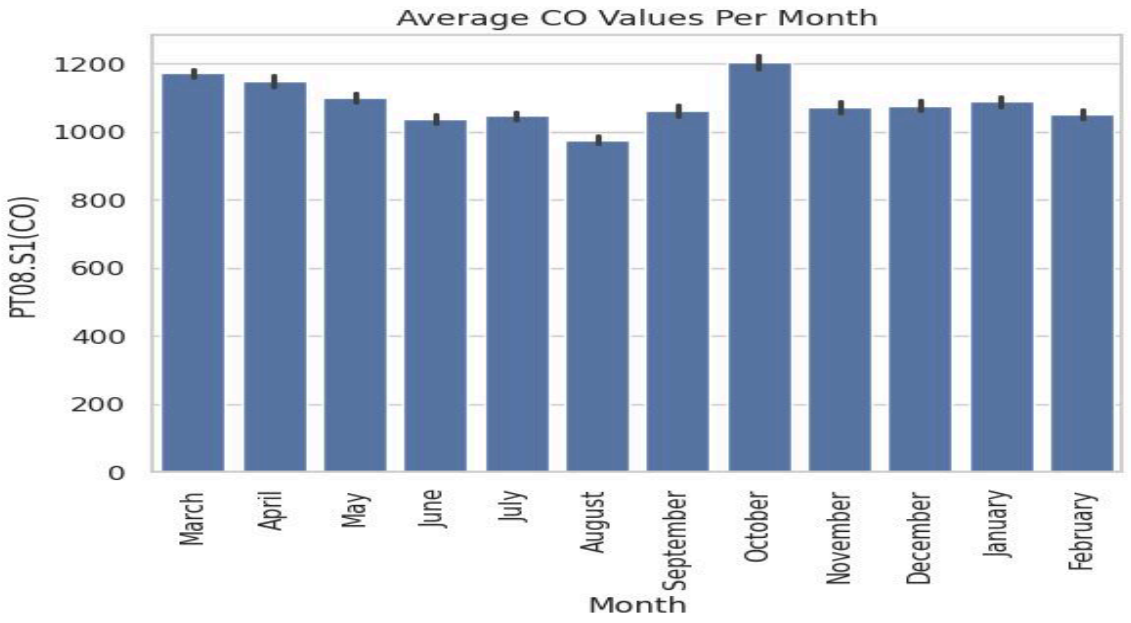

Fig 3.5.1 Average CO values per month

Day of the week CO emission analysis: Figure 3.5.2 reflects average carbon monoxide readings measured with the help of the 'PT08.S1(CO)' sensor for each day of the week. The readings are not very much fluctuated .i.e., the difference between days is very small, and hence, the amount of CO emitted appears to be similar on a daily basis. This might indicate that such additive factors contributing to CO, such as traffic patterns and industrial activity, remain constant throughout the week

of the region being monitored. It is possible that such readings are influenced by periodic human activities, wherein the difference between weekdays and weekends is not very large in terms of activities that enhance the level of CO. Such information might prove very valuable in terms of urban planning and environmental legislation in terms of controlling air pollution.

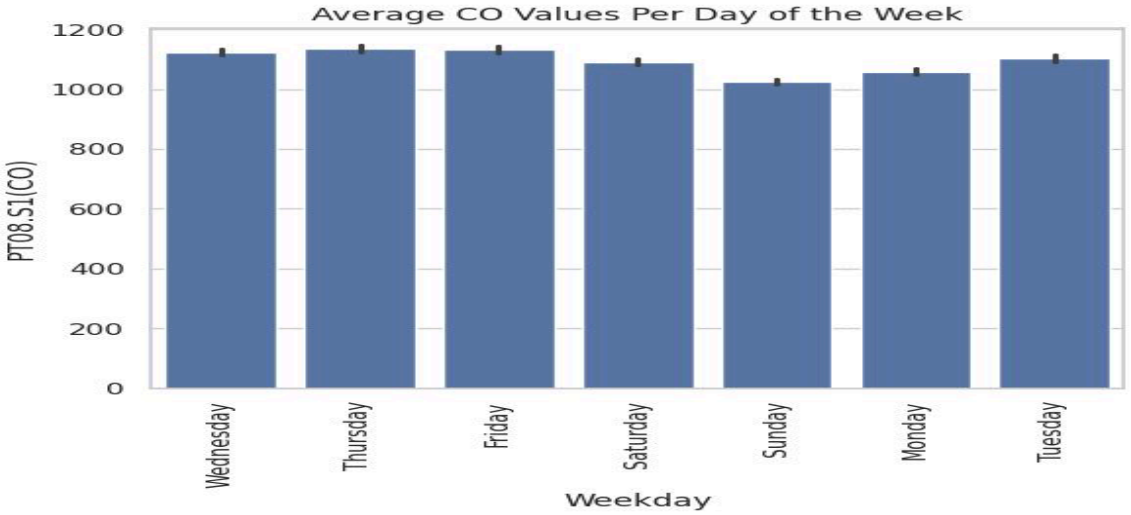

Fig 3.5.2 Average CO values per day of the week

Hourly Analysis: In figure 3.5.3, the bar chart depicts the average carbon monoxide (CO) levels as detected by the 'PT08.S1(CO)' sensor across different hours of the day. The pattern shows a cyclical nature of CO concentrations, with lower levels during the early hours of the morning and higher levels during the day, which could correlate with typical daily traffic and industrial activity patterns. Peak values appear to occur in the evening hours, which might reflect increased vehicular congestion during rush hour or specific evening activities that contribute to higher CO emissions. The information provided by this hourly distribution is valuable for identifying critical times when interventions for reducing CO exposure might be most necessary.

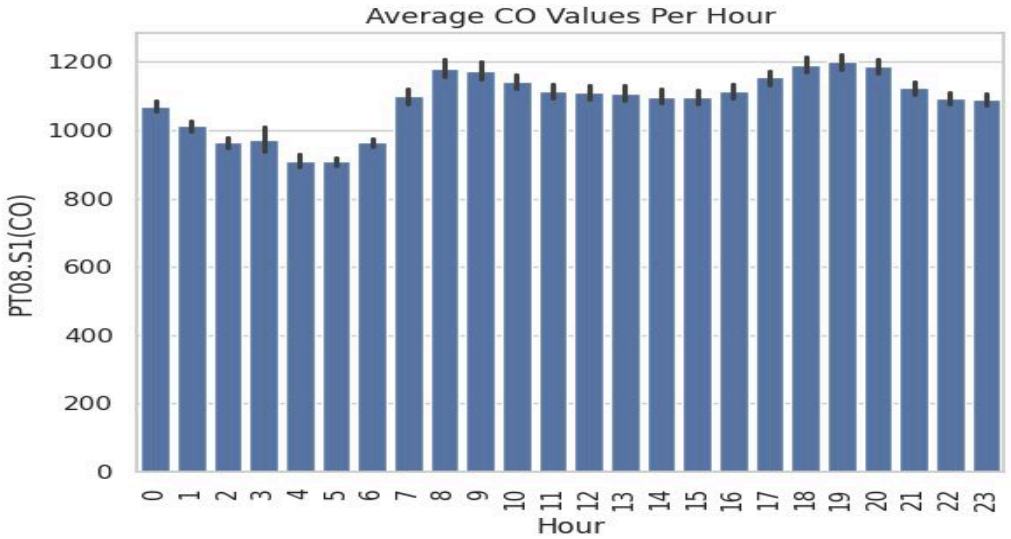

Fig 3.5.3 Average CO values per hour

Feature Selection: For the subsequent phases of our study, we have chosen to focus exclusively on features that represent sensor-derived pollutant concentrations and key environmental parameters. Specifically, we will consider the following variables: 'PT08.S1(CO)' for carbon monoxide, 'PT08.S2(NMHC)' for non-methane hydrocarbons, 'PT08.S3(NOx)' for nitrogen oxides, 'PT08.S4(NO2)' for nitrogen dioxide, and 'PT08.S5(O3)' for ozone. Additionally, environmental features such as temperature ('T'), relative humidity ('RH'), and absolute humidity ('AH') will be included in our analysis. This decision is rooted in our domain interest and the insights gained from our exploratory data analysis (EDA), which indicated that these variables are integral to understanding the factors influencing air quality and pollutant behavior. By narrowing our feature set, we aim to refine our model's predictive capability and ensure that our analysis remains aligned with the specific objectives of our research.

Correlation Analysis: In figure 3.5.4, The correlation matrix heatmap, as depicted in the visualization, provides a quantitative analysis of the interdependencies among various pollutant measurements and environmental parameters. A distinct positive correlation is observed between the sensor readings for carbon monoxide ('PT08.S1(CO)') and non-methane hydrocarbons ('PT08.S2(NMHC)'), indicating a tendency for these pollutants to increase in unison. Inversely, nitrogen oxides as detected by 'PT08.S3(NOx)' demonstrate a pronounced negative correlation with both 'PT08.S1(CO)' and 'PT08.S2(NMHC)', highlighting a contrasting relationship where an increase in nitrogen oxides is associated with a decrease in CO and NMHC levels. Additionally, ozone levels recorded by 'PT08.S5(O3)' are positively correlated with 'PT08.S1(CO)' and 'PT08.S2(NMHC)', but negatively with 'PT08.S3(NOx)', suggesting that factors elevating ozone concentrations may conversely suppress nitrogen oxides. Furthermore, there is a significant positive correlation between temperature and nitrogen dioxide ('PT08.S4(NO2)'), implying that NO2 levels could escalate with rising temperatures. This comprehensive correlation analysis is instrumental in elucidating the complex interactions between air pollutants and environmental conditions, offering crucial insights for air quality management and policy formulation.

Correlations with Environmental Factors: In figure 3.5.5, while reviewing the sensor data for carbon monoxide ('CO'), we observed notable trends and patterns. The 'CO' levels detected by the 'PT08.S1(CO)' sensor exhibit potential correlations with environmental variables. The scatterplot data suggests a relationship between 'CO' levels and temperature, an insight that aligns with established knowledge that higher temperatures can increase the release of 'CO' from certain sources. Additionally, while reviewing the data distributions, we found the 'CO' levels to be normally distributed on most days, suggesting a typical and expected urban emission pattern. However, the presence of outliers in the data indicates sporadic spikes in 'CO' levels, which warrant further investigation to identify possible extraordinary emission events or anomalies in sensor performance. These insights are invaluable for forming a targeted approach to air quality management and pollution mitigation strategies.

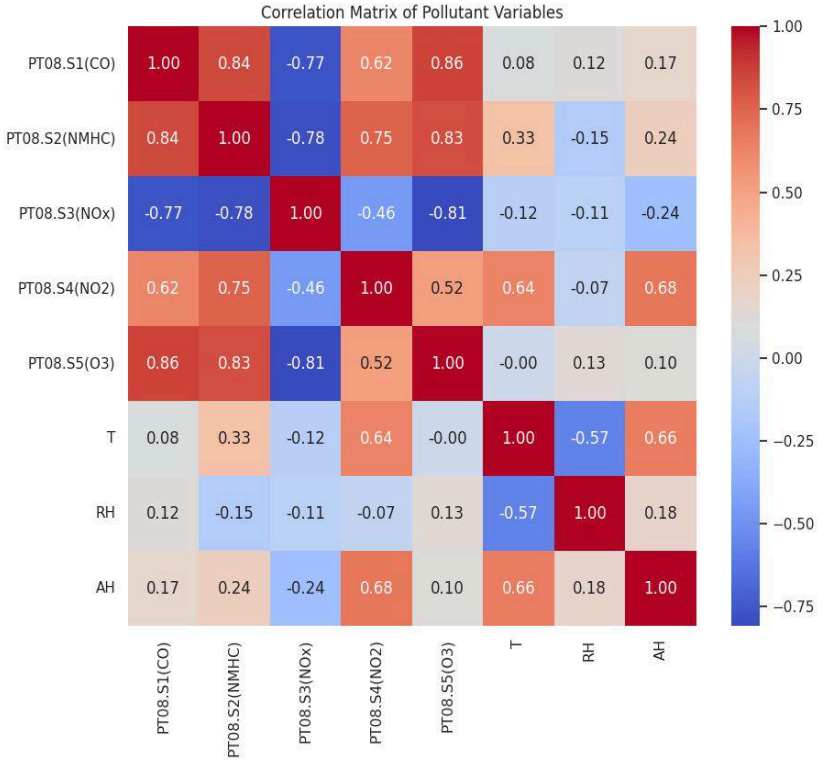

Fig 3.5.4 Correlation Matrix

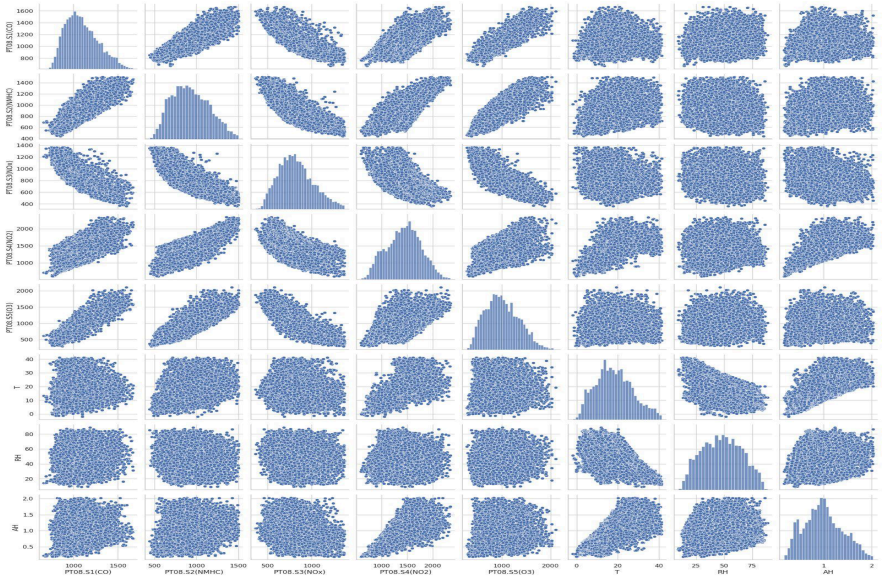

Fig 3.5.5 Pairplot of all variables

4.METHODOLOGY

4.1 INTRODUCTION

In this part we are predicting CO concentrations, our dependent variable is 'PT08.S1(CO)' using the other pollutants in the dataset, i.e. 'PT08.S2(NMHC)', 'PT08.S3(NOx)', 'PT08.S4(NO2)', 'PT08.S5(O3)' which are the independent variables. To build our machine learning models, we use a standard process where we train them on most of the data and then test them on a smaller part to see how well they work. We use 80% of the data for training and set aside 20% to check the models' performance. Further we have scaled the data using sklearn.preprocessing.StandardScaler() [12]. StandardScaler is utilized to bring all data points to a uniform scale by normalizing the input data. Now, the dataset consists of 6095 rows and 5 columns. This dataset is then used for training and validation for all the models.

4.2 LINEAR REGRESSION

One of the machine learning models used for CO prediction in this work is Multiple Linear Regression (MLR) using scikit-learn's linear_model.LinearRegression [6]. MLR is a statistical method to make a linear relationship between a dependent variable and various independent variables. The dependent variable is the one you want to predict, and the independent variables are those that predict the behavior of the dependent variable. In our study, the dependent variable is CO concentration, and all other pollutant concentrations are independent variables. The fundamental equation of MLR can be formulated as:

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + ... + \beta n Xn + \varepsilon$$

The projected CO concentration (Y) is expressed as a function of an intercept (β_0), independent variable coefficients (β_1 to βn), and a random error term (ε) in the linear regression model. The model finds these values for the coefficients β that minimize the error term ε across all the data points. Basically, this provides a best-fit line through this multidimensional space of independent variables. This allows us to predict the CO concentration for new, unseen data based on the values of the other pollutants and the meteorological measurements.

4.3 DECISION TREE REGRESSOR

We have also implemented Decision Tree Regression using sklearn.tree.DecisionTreeRegressor [7]. This method creates a tree-like model where every internal node is split over some feature and some particular threshold, and the procedure iteratively partitions the data over these splits till achieving leaf nodes that represent the predicted CO concentration values. The Decision tree regression model benefits from nonlinear relationships between the variables. It can also give insight into the decision-making process because of the structure of the tree. But, it can tend to be prone to overfitting when not tuned appropriately.

$$\hat{y} = \sum(i=1 \text{ to } N)\ ci * I(x \in Ri)$$

The constant values (ci) associated with each region (Ri) specified by the characteristics (x) are added up to predict the CO concentration ($\hat{y}$), and the indicator function $I(x \in Ri)$ determines whether a given data point belongs in that region.

4.4 ENSEMBLE LEARNING

Ensemble learning is a technique in machine learning[13], where the output of multiple models is pooled to give a better overall prediction. When these outputs are pooled together, we have a more robust, and more exact prediction. Ensemble methods train multiple models on the same data and mix the results. This results in improved accuracy, where it can handle complex relationships, and avoids being over-dependent on any one model that's flawed. Two of the most common ensemble methods .i.e. Random Forest (a collection of individual decision trees) and Gradient Boosting, which builds models sequentially to improve on one another, were also implemented as a part of our experimentation.

4.4.1 Random Forest

We use a technique called the Random Forest model to predict pollutant concentration. Random Forest is like a team of decision trees working together to make better predictions to avoid mistakes that a single tree might make and thus cooperate to enhance prediction accuracy. We implement it using sklearn.ensemble.RandomForestRegressor [8]. Random Forest lessens the possibility of overfitting that is typical with a single decision tree by averaging the output of several trees. This model is a powerful tool for predicting pollutant concentration as it can explain a substantial percentage of the variance in the CO concentration data, as evidenced by its R-squared value of 0.87.

4.4.2 Gradient Boosting

Gradient Boosting is an ensemble method that is a very robust method, we've implemented it using sklearn.ensemble.GradientBoostingRegressor [9]. Gradient boosting constructs trees in a step-by-step fashion, wherein it tries to fix the mistakes of the prior trees. The RMSE and R-squared values of this technique demonstrated promise in our investigation, indicating that it has successfully captured the underlying patterns of the data. Its results are promising even though its accuracy didn't exceed the Random Forest accuracy.

4.5 SUPPORT VECTOR REGRESSOR

Support Vector Machines, which are typically employed for classification tasks, provide the foundation for the Support Vector Regressor (SVR), using sklearn.svm.SVR [10]. This idea is extended to continuous outputs by SVR, which seek to match the optimal line within an epsilon threshold error margin. Because it makes use of kernel functions, which convert the input data into a higher-dimensional space where defining a linear separator is simpler, this method excels at managing non-linear relationships. SVR has the ability to capture the complexities of the relationship that can exist between highly non-linear and complicated inputs (such as different contaminants and environmental conditions) and outputs (such as sensor readings) in the context of environmental data. This is accomplished by concentrating on the data points that are closest to the decision boundary—the most important data points, or support vectors. This makes SVR less susceptible to data noise or outliers, which are common in real-world sensor data.

$$\text{Minimize: } 1/2 \, ||w||^2 + C * \Sigma(\max(0, 1 - y_i * (w^T * x_i + b)))$$

||w||^2 (weight vector norm squared) controls model complexity + C * Σ(max(0, 1 - yi * (w^T * xi + b))) (hinge loss sum) for SVR.

4.6 MLP

Multi-layer Perceptron (MLP) is an artificial neural network with multiple layers of nodes that utilize feedforward architecture for learning complex data relationships. It consists of layers of nodes, including an input layer for features, hidden layers for pattern learning, and an output layer for predictions. Each node applies an activation function to introduce nonlinearity. The network adjusts weights and biases through feedforward propagation and backpropagation to minimize prediction errors. MLPs are versatile and can be used for regression, classification, and unsupervised tasks. In this case, we use it to predict air pollutant concentrations i.e. a regression task. The mathematical description of a Multi-layer Perceptron (MLP) involves a series of transformations applied to input data. Here's a breakdown of the equations governing the feedforward propagation process:

Input Layer: The input layer directly passes input features (x_1, x_2, ..., x_n) to the first hidden layer without any transformation.

Hidden Layers: Each neuron (node) in a hidden layer computes a weighted sum of its inputs followed by an activation function. For neuron (j) in hidden layer (i), the weighted sum is:

$$z_{ij} = \sum_{k=1}^{n} w_{ijk} \cdot h_{i-1,k} + b_{ij}$$

where *wijk* is the weight connecting neuron (k) in the previous layer to neuron (j), $hi-1,k$ is the activation of neuron (k) in the previous layer, and *bij* is the bias term. The activation is computed as: $hij=f(zij)$, where $f(\cdot)$ is the activation function.

Output Layer: Similar to hidden layers, the output layer computes the weighted sum of inputs and applies an activation function. For output neuron (k), the weighted sum is:

$$z_k = \sum_{j=1}^{m} w_{jk} \cdot h_{L,j} + b_k$$

Where *wijk* is the weight connecting neuron (k) in the previous layer to neuron (j), $hi-1,k$ is the activation of neuron (k) in the previous layer, and *bij* is the bias term. The activation is computed as: $hij=f(zij)$, where $f(\cdot)$ is the activation function.

In summary, the MLP equation involves computing weighted sums of inputs and applying activation functions at each neuron in hidden layers and the output layer. The output is obtained by passing input data through multiple hidden layers and generating predictions or outputs at the output layer. In our code we have implemented MLP using ‘sklearn.neural_network.MLPRegressor’ [14] which is a class from the neural_network module of the scikit-learn library.

4.6.1 Optimization

To improve the output of our MLP (Multi-layer Perceptron) model, we performed Hyperparameter Tuning, in which we experimented with different values for hyperparameters such as the number of hidden layers, the number of neurons in each layer, activation functions, learning rate, and batch size. Hyperparameter tuning helps us find the optimal configuration for our model.

We've utilized `GridSearchCV` to systematically explore the hyperparameter space and determine the optimal configuration for the MLP model. Short for "Grid Search Cross-Validation," it's a powerful tool within the scikit-learn library, automating hyperparameter tuning. This technique

exhaustively searches through a specified grid of hyperparameter values, cross-validating each combination to identify the best set of hyperparameters for the model and dataset. By streamlining the hyperparameter tuning process, `GridSearchCV` saves time and effort, ensuring fine-tuning for optimal performance. It eliminates manual trial and error, providing a specific implementation of grid search that combines with cross-validation for efficient parameter optimization.

In our code, we implemented the MLP model using `GridSearchCV`. Initially, we split the data into training and test sets and standardized the features using `StandardScaler` from `sklearn.preprocessing`. Subsequently, we defined a grid of hyperparameters for the MLP model, encompassing hidden layer sizes, activation functions, and solvers. Leveraging `GridSearchCV`, we conducted a cross-validated grid search to identify the best combination of hyperparameters for the model. Upon obtaining the optimal model, we trained it on the training data and evaluated its performance on the test set. Following this, we calculated and displayed metrics such as R-squared score and Root Mean Squared Error (RMSE), alongside a scatter plot illustrating the comparison between actual and predicted values.

Through the fine-tuning process facilitated by `GridSearchCV`, we determined the optimal parameters for the MLP model: **{'activation': 'relu', 'hidden_layer_sizes': (128, 64, 32), 'solver': 'lbfgs'}**. The `hidden_layer_sizes` parameter delineates the architecture of the neural network, comprising three hidden layers with 128, 64, and 32 neurons, respectively, thereby influencing the model's capacity to capture intricate data patterns. The activation function 'relu' specifies the Rectified Linear Unit activation function, augmenting the model's capability to learn complex relationships. For optimization, the 'lbfgs' solver was selected, harnessing the Limited-memory Broyden–Fletcher–Goldfarb–Shanno algorithm, known for its computational efficiency and suitability for shallow networks. This parameter configuration enhances the MLPRegressor model's predictive performance on the given task, reflecting meticulous consideration of architectural and training settings to attain optimal results.

5. RESULTS AND DISCUSSION

5.1 RESULTS

The estimation of the air pollutant concentrations is very essential for the prediction of environmental impacts and the implementation of effective mitigation strategies. We conducted the whole analysis with different regression models in order to predict the concentration of air pollutants based on other pollutant concentrations within the dataset. We tried out the model performance with the Root Mean Squared Error (RMSE) and R-squared (R2) index for every model applied to gain an understanding of the predictive capabilities of each model.

Table 5.1 summarizes performance for various regression models in predicting concentrations of air pollutants. From these methods, the MLP recorded the lowest RMSE of 64.901 and also had the highest R2 score of 0.876, proving the best performance of the model. Random Forest also makes the strongest performance as it recorded an RMSE of 65.784 and an R2 score of 0.872. However, these results confirm that machine learning methods can give predictive ability for the concentrations of air pollutants with great accuracy for the use in decision-making for environmental management.

Table 5.1 Comparison of Models

Model	RMSE	R2
MLP	**64.901533**	**0.875659**
Random Forest	**65.790358**	**0.872230**
Gradient Boosting	72.529068	0.844716
Linear Regression	83.855845	0.792427
Decision Tree Regression	87.089389	0.776110
SVR	89.985874	0.760970

5.2 VISUALIZATION

In the following subsection, we show visual representations of the performance of our two best models: the tuned MLP model and the Random Forest model. These scatter plots present the actual and predicted values. We get insights into the accuracy of prediction for both models in relation to the true values of the target variable. From the plots, we will get a view of how well the models predict actual observations. The best thing is that the points on the plot should resemble a trend of a diagonal line, for strong correspondence between the predictions and the actual observations. This is a visualization that will help us understand the success of the models in predicting the target variable, and thereby inform decisions on their suitability for air quality prediction tasks.

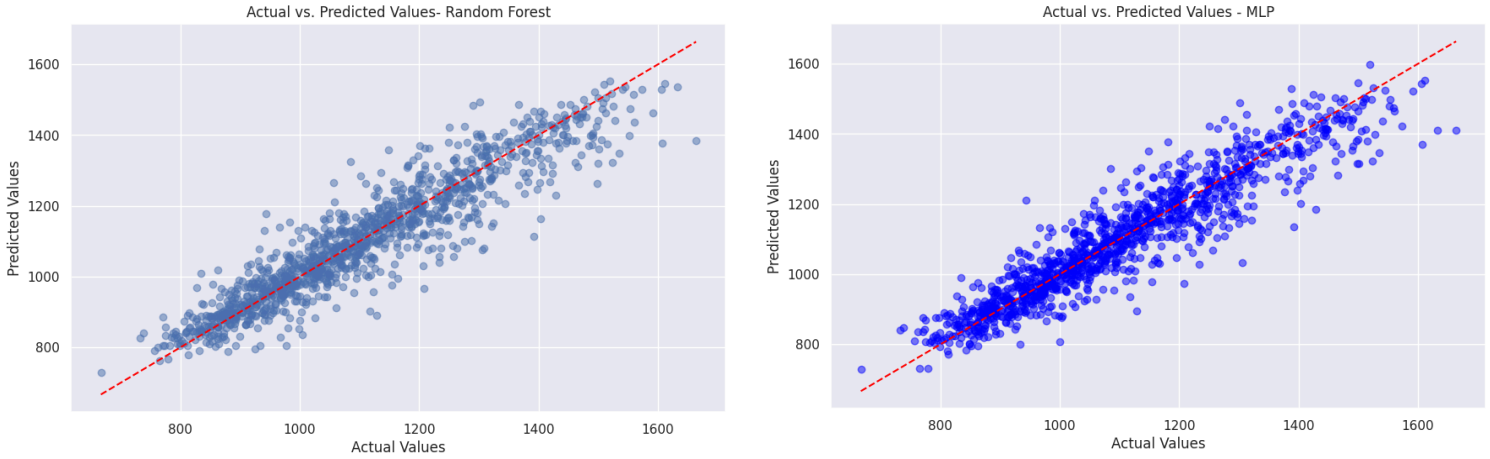

Figure 5.2.2 MLP: Actual VS Predicted Values

Especially in figure 5.2.1, we see that the Random Forest model has good predictive power with many of the data points appearing near to the red dashed line, which is a good sign. However, there is a slight decline in accuracy for higher actual values, especially those greater than 1400, as evidenced by the dispersion of points from the red dashed line. The plot gives a very strong

aggregation of points near the middle value, which means that actual values tend to be in the interval. So the model will put more predictive energy in that interval. The fact that no indication of overfitting is present with overly tight clustering of points along the predictive line means that the model can generalize. However, the deviations in high-value predictions mean that there is room for refinement—through hyperparameter tuning, for instance.

Figure 5.2.2: The Multilayer Perceptron MLP model is remarkably accurate. Indeed, from the close clustering of data points around the predictive line, it is possible to see that the MLP model's predicted outcomes are a remarkably good match with the actual values in almost the entire spectrum of data. The magnitude of error, deducible from the position of the data points relative to the red dashed line, is presumably almost negligible—meaning there is no apparent systemic bias in the MLP model's predictions. As with the Random Forest model, there are more densely packed points in the central part of the plot, which may be taken as an indication of a more significant number of real observations in that range, giving the MLP model a much richer dataset on which to train and perhaps make its predictions more accurate.

5.3 FEATURE IMPORTANCE

However, the MLP model, with several hidden layers, may be a more intricate and harder-to-understand model than a Random Forest model, which—despite being an ensemble—offers interpretability through features like feature importance.

They are inherently complex owing to their architecture. The models can have one or more hidden layers between the input and output layers. For every neuron in the hidden layers, multiple sets of weights and biases are used to process input data. These weights and biases bring about a high degree of non-linearity for how the neurons decide something, which makes it very difficult to actually trace back the contribution of each input feature in what it contributes to the output. This property of MLPs makes the job of deriving feature importance considerably difficult, for MLPs do not immediately offer any direct account of the feature importances.

Random Forests are a more interpretable alternative, though. These models are an ensemble of decision trees; each tree makes a contribution to the final prediction. Decision trees in a Random Forest partition the data into the target categories or predicted values for regression tasks using a series of decisions based on the values of input features. Examination of feature importance is an inherent part of the decision-making process in these trees; the most effective splitters of the data are by the features, so they are deemed more important. This information can be quantified and aggregated across all the trees in a Random Forest to produce a global estimate of feature importance. This measure is useful in interpreting the influence of the features on the predictions and has the added advantage of conveying the decision logic of the model to stakeholders.

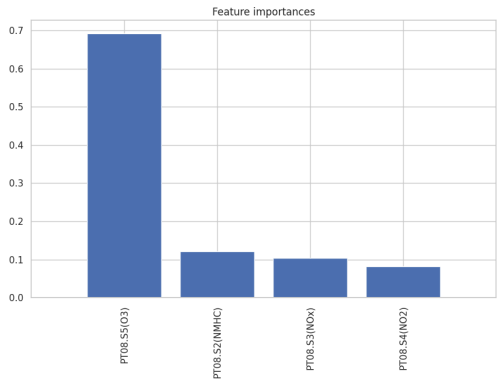

Figure 5.3.1 Random Forest: Feature Importance

The above bar chart illustrates the feature importances derived from a Random Forest model. Feature importance is a technique which is used to understand the contribution of each input variable to the predictive power of a model. The above chart shows four features, with their respective importances scored between 0 and 1:

1. PT08.S5(O3) has the highest importance score of approximately 0.69, making it the most influential feature in the model's predictions.
2. PT08.S2(NMHC) is the second most important feature but significantly less so, with a score of around 0.12.
3. PT08.S3(NOx) follows closely with a score of about 0.1.
4. PT08.S4(NO2) has the lowest importance score of the four, around 0.08.

From the above, we can see which features are influencing the model's predictions the most, and by what margin, which is valuable for understanding the model.

6. CONCLUSION AND FUTURE WORK

6.1 CONCLUSION

After evaluating the performance of our Random Forest and Multi-Layer Perceptron (MLP) models, we find that the Random Forest model outperforms the MLP, considering several key factors. Firstly, Random Forests offer greater interpretability through feature importance scores compared to the complexity of MLPs, particularly with multiple layers. Secondly, Random Forests demonstrate superior computational efficiency, especially on large datasets, whereas MLPs may require more computational resources, particularly with multiple layers. Random Forests, especially with no tuning, can be quick to train and make predictions, whereas MLP models with multiple layers may take longer, especially during training. Additionally, Random Forests handle mixed data types well without extensive preprocessing, whereas MLPs may necessitate more preprocessing steps, especially for high-dimensional data. Furthermore, Random Forests exhibit better generalization and robustness to outliers and noise due to their ensemble nature, in contrast to the potential overfitting and sensitivity to outliers in MLPs, particularly in deeper networks. Based on these considerations, we conclude that the Random Forest model is better suited for predicting air pollutant concentrations with multisensor data.

6.2 FUTURE WORK

In our future work, we want to perform these predictions for all the other pollutants as well. We also plan to explore the integration of external data sources, such as satellite imagery, traffic patterns, industrial emissions data, and socio-economic indicators, to enhance the predictive models. This multi-source data fusion approach promises a more comprehensive understanding of the factors influencing air quality. Additionally, we aim to develop real-time monitoring and prediction systems based on our best-performing model, the Random Forest. These systems will deliver timely insights and alerts to relevant stakeholders, potentially by deploying the model in a cloud-based or edge computing environment for efficient processing of streaming data. Furthermore, we intend to refine the hyperparameters of the Random Forest model to potentially improve performance further. This will involve experimenting with different parameter settings, such as tree depth, number of estimators, or feature subsampling, using techniques like RandomizedSearchCV. Through these efforts, we aim to enhance the accuracy, interpretability, and applicability of our air quality prediction system, ultimately contributing to better environmental management and public health outcomes.

REFERENCES

[1] World Health Organization. (2022, December 19). Ambient (outdoor) air quality and health. Who.int; World Health Organization: WHO. https://www.who.int/news-room/fact-sheets/detail/ambient-(outdoor)-air-quality-and-health

[2] Chanana, Ishita, et al. "Combustion and Stubble Burning: A Major Concern for the Environment and Human Health." Fire, vol. 6, no. 2, 20 Feb. 2023, p. 79, https://doi.org/10.3390/fire6020079.

[3] 16.4: Air Pollution. (2023, April 17). Chemistry LibreTexts. https://chem.libretexts.org/Courses/Anoka-Ramsey_Community_College/Introduction_to_Chemistry/16%3A_Environmental_Chemistry/16.04%3A_Air_Pollution

[4] Radiation, U. S. E. O. of A. and. (n.d.). Air Quality Trends Show Clean Air Progress. Gispub.epa.gov. https://gispub.epa.gov/air/trendsreport/2021/#growth

[5] UCI Machine Learning Repository. (n.d.). Archive.ics.uci.edu. https://archive.ics.uci.edu/dataset/360/air+quality

[6] scikit-learn developers. (2019). sklearn.linear_model.LinearRegression — scikit-learn 0.22 documentation. Scikit-Learn.org. https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html

[7] sklearn.tree.DecisionTreeRegressor — scikit-learn 0.23.2 documentation. (n.d.). Scikit-Learn.org. https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeRegressor.html

[8] scikit-learn. (2018). 3.2.4.3.2. sklearn.ensemble.RandomForestRegressor — scikit-learn 0.20.3 documentation. Scikit-Learn.org. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html

[9] 3.2.4.3.6. sklearn.ensemble.GradientBoostingRegressor — scikit-learn 0.21.2 documentation. (2009). Scikit-Learn.org. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingRegressor.html

[10] sklearn.svm.SVR — scikit-learn 0.23.1 documentation. (n.d.). Scikit-Learn.org. https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVR.html

[11] Scikit-Learn. (2019). sklearn.preprocessing.StandardScaler — scikit-learn 0.21.2 documentation. Scikit-Learn.org. https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html

[12] California Air Resources Board. (2022). Carbon Monoxide & Health | California Air Resources Board. Ww2.Arb.ca.gov. https://ww2.arb.ca.gov/resources/carbon-monoxide-and-health

[13] 1.11. Ensemble methods — scikit-learn 0.22.1 documentation. (2012). Scikit-Learn.org. https://scikit-learn.org/stable/modules/ensemble.html

[14] sklearn.neural_network.MLPRegressor — scikit-learn 0.21.3 documentation. (2010)

CODE:

```
import pandas as pd
import numpy as np
```

```
import random
import datetime
import seaborn as sns
import matplotlib.pyplot as plt
from scipy.stats import pearsonr
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
from sklearn.linear_model import LinearRegression, Lasso, Ridge, ElasticNet
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.svm import SVR
from sklearn.metrics import r2_score, mean_squared_error
from keras.models import Sequential
from keras.layers import LSTM, Dense, Dropout
from keras.optimizers import Adam
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
from sklearn.preprocessing import StandardScaler
from sklearn.neural_network import MLPRegressor
from math import sqrt

import warnings
warnings.filterwarnings("ignore")
%matplotlib inline
sns.set(color_codes=True)
df = pd.read_excel("/content/AirQualityUCI.xlsx")
df
df.replace(to_replace = -200, value = np.nan, inplace = True)
df.info()
# % of null values present in each colunmn
```

```
percent_NaN = []
columns = df.columns
for col in columns:
  pNaN =  (df[col].isna().sum()/df.shape[0]) * 100 #sum NaN instances in each column. Divide by total rows
  percent_NaN.append(pNaN)
nan_percent_df = pd.DataFrame(percent_NaN,
                              index=columns,
columns=['%_NaN_in_Column']).sort_values('%_NaN_in_Column',ascending = False)
nan_percent_df
percent_NaN = (df.isnull().sum() / df.shape[0]) * 100
nan_percent_df = pd.DataFrame(percent_NaN, columns=['%_NaN_in_Column']).sort_values(by='%_NaN_in_Column', ascending=False)

# Plotting the bar chart
plt.figure(figsize=(10, 5))  # Adjust the size of the figure as needed
nan_percent_df.plot(kind='bar', legend=False)
plt.title('Percentage of Missing Values per Column')
plt.ylabel('% of Missing Values')
plt.xticks(rotation=45, ha='right')  # Rotate the x labels to show them better
plt.tight_layout()  # Adjust the padding of the figure

# Display the plot
plt.show()
df.drop('NMHC(GT)', axis=1, inplace=True, errors = 'ignore')
df = df.dropna()
df.head()
df.shape
df['Date'] = df['Date'].astype(str)
df['Time'] = df['Time'].astype(str)
```

```
df['DateTime'] =  (df.Date) + ' ' + (df.Time)

df.DateTime = df.DateTime.apply(lambda x: datetime.datetime.strptime(x, '%Y-%m-%d %H:%M:%S'))

df.head()

df['Weekday'] = df['DateTime'].dt.day_name()

df['Month']   = df['DateTime'].dt.month_name()

df['Hour']    = df['DateTime'].dt.hour

df['Date']    = pd.to_datetime(df['Date'], format='%Y-%m-%d')

df.drop('Time', axis=1, inplace=True, errors = 'ignore')

df.head()

df = df[['Date','Month', 'Weekday','DateTime', 'Hour', 'CO(GT)','PT08.S1(CO)', 'C6H6(GT)', 'PT08.S2(NMHC)',

      'NOx(GT)', 'PT08.S3(NOx)', 'NO2(GT)', 'PT08.S4(NO2)', 'PT08.S5(O3)', 'T', 'RH', 'AH']]

df.head()

df.shape

df.columns

df.duplicated().sum()

sns.set_theme(style="whitegrid")

# Boxplots of columns

fig, axes = plt.subplots(nrows=4, ncols=3, figsize=(15, 20))

axes = axes.flatten()


for i, column in enumerate(df.columns[5:]):

  if column != 'Date':

     df[column] = pd.to_numeric(df[column], errors='coerce')


  sns.boxplot(x=df[column], ax=axes[i])

  axes[i].set_title(f'Boxplot of {column}')


for j in range(len(df.columns) - 2, len(axes)):
```

```
    fig.delaxes(axes[j])

plt.tight_layout()
plt.show()



cols = df.columns[5:]
Q1 = df[cols].quantile(0.25)
Q3 = df[cols].quantile(0.75)
IQR = Q3 - Q1

scale = 1.4

lower_lim = Q1 - scale * IQR
upper_lim = Q3 + scale * IQR

condition = ~((df[cols] < lower_lim) | (df[cols] > upper_lim)).any(axis=1)
df_new = df[condition]

outliers_removed = len(df) - len(df_new)
print(f"Number of outliers removed: {outliers_removed}")
df_new.shape
df_new.drop(['CO(GT)']  ,axis=1, inplace=True, errors = 'ignore')
df_new.drop(['NOx(GT)'] ,axis=1, inplace=True, errors = 'ignore')
df_new.drop(['C6H6(GT)'],axis=1, inplace=True, errors = 'ignore')
df_new.drop(['NO2(GT)'] ,axis=1, inplace=True, errors = 'ignore')
df_new.head()
df_new.shape
# @title PT08.S2(NMHC)
```

```
from matplotlib import pyplot as plt
df_new['PT08.S2(NMHC)'].plot(kind='hist', bins=20, title='PT08.S2(NMHC)')
plt.gca().spines[['top', 'right',]].set_visible(False)
month_df_list = []
day_df_list   = []
hour_df_list  = []

months = ['January','February','March', 'April', 'May','June',
        'July', 'August', 'September', 'October', 'November', 'December']

days = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday']

# Filtering based on Month
for month in months:
   temp_df = df_new.loc[(df_new['Month'] == month)]
   month_df_list.append(temp_df)

# Filtering based on Day
for day in days:
   temp_df = df_new.loc[(df_new['Weekday'] == day)]
   day_df_list.append(temp_df)

# Filtering based on Hour
for hour in range(24):
   temp_df = df_new.loc[(df_new['Hour'] == hour)]
   hour_df_list.append(temp_df)
# Barplot of Monthly CO emissions
sns.barplot(x = 'Month', y = 'PT08.S1(CO)', data = df_new)
```

```
plt.title('Average CO Values Per Month')
plt.xticks(rotation=90)
plt.show()

# Barplot of Weekly CO emissions
sns.barplot(x = 'Weekday', y = 'PT08.S1(CO)', data = df_new)
plt.title('Average CO Values Per Day of the Week')
plt.xticks(rotation=90)
plt.show()

# Barplot of Hourly CO emissions
sns.barplot(x = 'Hour', y = 'PT08.S1(CO)', data = df_new)
plt.title('Average CO Values Per Hour')
plt.xticks(rotation=90)
plt.show()
# Final DataFrame
df_final = df_new.iloc[:, 5:]
df_final
# Columns in the Final DataFrame
df_final.columns
# Calculate correlation matrix
pollutants = ['PT08.S1(CO)', 'PT08.S2(NMHC)', 'PT08.S3(NOx)', 'PT08.S4(NO2)','PT08.S5(O3)', 'T', 'RH', 'AH']
pollutant_data = df_new[pollutants]

correlation_matrix = pollutant_data.corr()

# Visualize correlation matrix as a heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap="coolwarm", fmt=".2f")
```

```
plt.title("Correlation Matrix of Pollutant Variables")
plt.show()
# Visualization as Pairplots
sns.pairplot(pollutant_data)
plt.show()
# Drop 'T', 'RH', and 'AH' columns ( Removal after Feature Selection for Modelling)
df_final.drop(columns=['T','RH','AH'], inplace=True)
df_final.head(5)
co_concentration = df_final['PT08.S1(CO)']
predictors = ['PT08.S2(NMHC)', 'PT08.S3(NOx)', 'PT08.S4(NO2)', 'PT08.S5(O3)']
predictor_data = df_final[predictors]

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(predictor_data, co_concentration, test_size=0.2, random_state=42)

# Standardize features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Initialize models
models = {
  'Linear Regression': LinearRegression(),
  'Decision Tree': DecisionTreeRegressor(),
  'Random Forest': RandomForestRegressor(),
  'Gradient Boosting': GradientBoostingRegressor(),
  'SVR': SVR()
}
```

```
# Results storage
results = []

# Train and evaluate models
for name, model in models.items():
    # Fit the model
    model.fit(X_train_scaled, y_train)

    # Predictions
    y_train_pred = model.predict(X_train_scaled)
    y_test_pred = model.predict(X_test_scaled)

    # RMSE
    rmse = np.sqrt(mean_squared_error(y_test, y_test_pred))

    # R-squared
    r2_test = r2_score(y_test, y_test_pred)

    # Store results
    results.append({
        'Model': name,
        'RMSE': rmse,
        'R-squared': r2_test,
    })

# Display results
results_df = pd.DataFrame(results)
results_df
```

```
rfr = RandomForestRegressor()
rfr.fit(X_train_scaled, y_train)



y_pred = rfr.predict(X_test_scaled)
# Scatter plot of actual vs. predicted values
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred, alpha=0.5)
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red', linestyle='--')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.title('Actual vs. Predicted Values- Random Forest')
plt.grid(True)
plt.show()



# Feature importance from Random forest
importances = rfr.feature_importances_
features = X_train.columns
# Sort feature importances in descending order
indices = np.argsort(importances)[::-1]

# Print the feature ranking
print("Feature ranking:")
for i, idx in enumerate(indices):
   print(f"{i + 1}. Feature '{features[idx]}' ({importances[idx]})")

# Plot the feature importances
```

```
plt.figure(figsize=(10, 6))
plt.title("Feature importances")
plt.bar(range(X_train.shape[1]), importances[indices], align="center")
plt.xticks(range(X_train.shape[1]), [features[idx] for idx in indices], rotation=90)
plt.xlim([-1, X_train.shape[1]])
plt.show()



# Tuned MLP
random.seed(42)

data = df_final.copy()

# Define features (X) and target variable (y)
X = data.drop(columns=['PT08.S1(CO)'])
y = data['PT08.S1(CO)']

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Standardize the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Define hyperparameters for grid search
param_grid = {
  'hidden_layer_sizes': [(64, 32), (128, 64, 32),(256,128,64,32)],
  'activation': ['relu', 'tanh'],
```

```
  'solver': ['adam', 'lbfgs'],
}

# Build the MLP model
model = MLPRegressor(random_state=42)

# Perform grid search for hyperparameter tuning
grid_search = GridSearchCV(model, param_grid, cv=5, scoring='neg_mean_squared_error')
grid_search.fit(X_train_scaled, y_train)

# Get the best model from grid search
best_model = grid_search.best_estimator_

# Train the best model
fit=best_model.fit(X_train_scaled, y_train)

# Predict on the test set
y_pred = best_model.predict(X_test_scaled)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
rmse = sqrt(mse)
r2 = r2_score(y_test, y_pred)

print(f"Best Parameters: {grid_search.best_params_}")
print(f"R2 Score: {r2}")
print(f"Root Mean Squared Error: {rmse}")

# Scatter plot of actual vs. predicted values
```

```
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred, color='blue', alpha=0.5)
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red', linestyle='--')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.title('Actual vs. Predicted Values - MLP')
plt.grid(True)
plt.show()
```