Decentralized Air quality Monitoring and Prediction (DAMP) - Product Report

Uppsala University - Projekt DV/Project CS (1DT054)
Fall 2020

Last Revision Date: **2021-01-17**

Borja González Farías

Egemen Yiğit Kömürcü

Ahmet Can Turgut

Jad Ali Daoud

Laleh Aftabi

Nithesh Chandher Karthikeyan

Raheel Ali

Reuben Sajith R

Saradh Tiwari

Uvais Karni

Vaidehi Vipra

Abstract

Air pollution is an important issue these days that continues to bring more and more concern to the overall population. For this reason, several air quality stations measure the pollutants in the air, and numerous prediction models have been developed to predict the quantity of these pollutants within a specified time (in the next hour, for example). First, a centralized model was developed to predict the number of certain pollutants (PM10 and NO2, for example) within the next hour. After verifying the accuracy of the centralized model (comparing the predictions with the real values) the federated model was built next, in which a server distributes a specific model to all the clients participating in the federation for training. The Tensorflow library and its Tensorflow Federated implementation were used for this model to carry out the experiments necessary to test the federated learning approach.

Acknowledgements

The team would like to thank both the Uppsala University and Ericsson for giving us the opportunity to work on a real project that will not only benefit us in our education and experience but also may lead to a new system that may enhance the lives of many people in Stockholm. We would like to show our gratitude to our course coordinator Vera van Zoest for her support and guidance on the concerns of our group, for keeping us informed about the various activities, meetings and assignments throughout the course, and for her willingness to help us when needed.

We would also like to share our acknowledgement to Tobias Mages, the course assistant assigned to our group, who has helped and guided us every week (through weekly meetings) with our doubts and questions about the implementation of our models, and gave us plenty of advice and possible solutions to follow. This project and the research behind it would not have been possible without his support, guidance, and knowledge.

Working on the project seemed challenging in the initial stages and we gratefully acknowledge the support of Konstantinos Vandikas from Ericsson, who helped us address our questions, and doubts about the specification of the project.

The competition for the centralized model was a positive approach which elevated both the team and the competition spirit and we thank Shenghui Li for setting it up.

Finally, we would like to thank Mikael Lundgren for his lecture on Agile, which allowed us to develop a deeper understanding of how to work with this approach and how to tackle long and complicated tasks alongside a team.

Index

Introduction **8**

Motivation 8

Project overview 8

Project goals 9

Organisation **9**

What is agile 9

Working With Agile 10

Why do we need to work with agile? 10

Working with agile with research project 10

Agile process flow 11

Where we failed to apply agile 11

Background **11**

Air quality and its monitoring 11

Problems with traditional Machine learning 12

Why Federated Learning 12

Knowing the dataset 12

What is time series data? 13

Why LSTM 14

Methodology **14**

Description of the data: 14

Dataset behaviour: 16

Anomalies in Data: 17

Missing Values 17

Outliers in data 18

Negative values 20

Data Preprocessing: 21

Feature Engineering 21

Feature Selection 22
Correlation 22
Feature Importance 24
Feature Importance using XGBoost (Extreme Gradient Boosting) 24
Permutation based Feature Importance using XGBoost (Extreme Gradient Boosting) 25
Feature importance using SHAP values 26
Data Imputation 28
Mean 28
Median 28
Interpolation 29
Data imputation using GAIN 30
Outliers Removal 31
Threshold truncation 31
Normalization 32
Centralized Model 35
Generalized model: 35
LSTM Model: 35
Hyperparameter tuning: 37
Experiments 37
Experiment 1: 37
Results Obtained for Experiment: 38
Experiment 2: 39
Results Obtained for Experiment: 40
Experiment 3: 42
Results Obtained for Experiment: 43
Federated Learning 45
Strength 47
Weakness 47
Federated Learning Implementations 48

Tensorflow Federated 48
What is it? 48
Implementation 48
FEDn 48
What is it? 48
Implementation 49
Experiment 1: 50
Experiment 2: 50
Experiment 3: 50
Experiment 4: 51
Experiment 5: 51
Results 51
Evaluation 52
Quantization of the model 53
STM32CubeMX 53
Tensorflow Lite 54
Experiment 1: 54
Results Obtained for Experiment: 55
Experiment 2: 55
Results Obtained for Experiment: 55
Experiment 3: 56
Results Obtained for Experiment: 56
Turn Into a Product 57
Why we decided to turn it into a product 57
Conclusions 59
Appendix A: Installation instructions 60
Appendix B: Future work 61
Appendix C: Tools 62
TensorFlow 62

TensorFlow Lite 62

TensorFlow Federated 62

FEDn 62

Flutter 62

Firebase 62

Trello 62

Git 62

Docker 63

References **64**

Introduction

Motivation

Air pollution is considered by many a threat to not only the environment but also to us, humans. There are two kinds of pollutants: primary (such as Volatile Organic Compounds or VOCs, PM or particulate matter, NOX, etc.) and secondary pollutants (pollutants that come from the mix of other chemicals, e.g. NO2 and sunlight produce ozone).

These pollutants come primarily from three sources: natural (volcanic eruptions, wildfires, etc.), stationary (industrial factories) and mobile (cars). Since we live in a still-growing world in terms of technology, population, etc., more and more pollution is being generated from the last two sources.

This air pollution can cause heart and lung diseases, as well as increase the risk of cancer. As a result, the elderly, children and people with lung/heart conditions are the most vulnerable in this situation. Global warming is also a current and ongoing issue that affects everyone living not only in Sweden, but in the world as well, and this project can be a minor contribution to fight against the issue.

We also need to take into account atmospheric patterns such as rain, air pressure, temperature, etc., that affects the volume of each pollutant in the air.

Project overview

The system will take data from both weather stations and air quality stations. Then, it will correlate all input variables through neural networks and give a prediction of how safe the air will be for the next hour.

For this, two types of machine learning have been used: centralized learning and federated learning. In the first one, all the data was gathered in one place and trained an LSTM model there, and in the second one, the large data sets rely on different devices and the model is trained in each of them.

Following this, the accuracy of both learning methods will be compared and will then be decided which of the methods is better.

Finally, a mobile app will be developed to show in a user-friendly way the predictions of the air quality for the next hour.

Project goals

There are mainly two goals for this project:

1. Help the population's health with informing about the pollution's situation so that the required precautions can be taken into account.
2. Use federated learning to preserve the privacy of the data and take advantage of the computational power of the independent devices.

Organisation

Agile software delivery approach is used in this project to divide the work amongst the team members to reach the desired goal.

What is agile

Agile is an approach for software delivery. With an agile approach, projects are handled in small pieces which are called sprints. Each sprint contains its own small tasks. Compared with waterfall, the aspect of agile is reducing planning time. Open source agile approach manifesto is as follows .

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

As the manifesto said, agile working principle focused with customers and releasing features fast. Agile requires huge communication and daily, weekly evaluation meetings.

Since when you create small projects among the big projects, lots of dependencies are created among the team members.

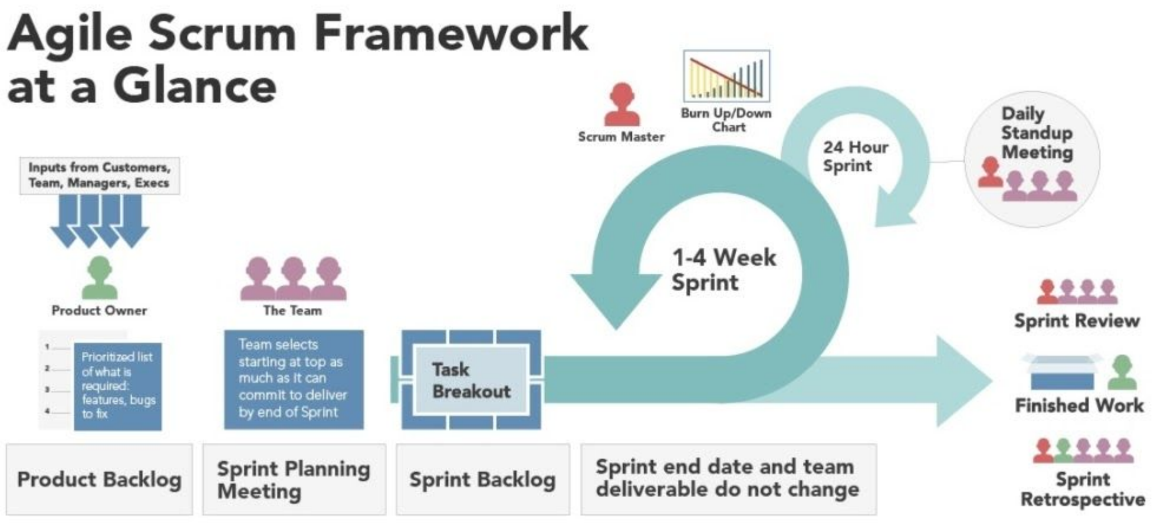

Agile Scrum Framework at a Glance
Inputs from Customers, Team, Managers, Execs
Product Owner
Prioritized list of what is required: features, bugs to fix
The Team
Team selects starting at top as much as it can commit to deliver by end of Sprint
Task Breakout
Scrum Master
Burn Up/Down Chart
24 Hour Sprint
Daily Standup Meeting
1-4 Week Sprint
Sprint Review
Finished Work
Sprint Retrospective
Product Backlog
Sprint Planning Meeting
Sprint Backlog
Sprint end date and team deliverable do not change

Table: Agile workflow

[https://www.planview.com/resources/guide/agile-methodologies-a-beginners-guide/basics-benefits-agile-method/]

Working With Agile

Why do we need to work with agile?

We selected agile since we had a limited time. However the project was a research project so we didn’t know how the project would evolve. It makes the project impossible to plan completely. Agile allows us to create small tasks to complete every sprint. This creates the opportunity to provide tools for specifications.

Working with agile with research project

When we look at the agile manifesto we can see that there are some conflicts between research projects. Research projects depend on the documentation and process steps. These conflicts make it hard to adapt these rules to the research project.

Agile process flow

We decided to split the team into 4 different groups, research group, machine learning group, federated learning group and organization group. Each team has 2-3 people. Each team decides their sprints individually. Every week we met on Fridays to observe any dependencies between teams. We followed simple agile flow in this project. We applied requirements, design, develop, test, deploy circles in each iteration.

Where we failed to apply agile

For many of us we have no past with agile. It is sometimes hard to adopt this kind of work style. Another problem is organizing teams. We initially randomly separate teams and it just made progress slow.

Background

Air quality and its monitoring

Air quality monitoring stations and small air quality monitoring devices are now becoming smaller and more efficient with the advancements in the IoT field, generating a plethora of data everyday that is being used to monitor the ambient air quality in real-time. Air quality is being personalized. Most air quality monitoring networks are built to promote human health objectives, and monitoring stations are established in population centers. They may be near busy roads, in city centers, or at locations of particular concern (e.g., a school, hospital, park, particular emissions sources). Monitoring stations also may be established to determine background pollution levels, away from urban areas and emissions sources.[15]

From raging traffic to blazing fires, and playful firecrackers the air quality of a given region can be greatly adversely affected. Organizations in this age are therefore using IoT for air quality sensing and these devices are compact, mobile, and can be carried virtually everywhere for air monitoring, allowing us to better understand (with the aid of machine learning) the air quality and its underlying variations in a given region, and to better forecast the air quality for local residents and travellers.

Problems with traditional Machine learning

With so much data being generated, air quality being personalized, and varying conditions affecting the air quality locally; conventional machine learning approach where all of the data is stored in one place and a model is deployed to predict the air quality will just not work, as it can get fairly complex to predict the air quality specific to a particular region in a city. Let alone the facts that these mobile devices are low-powered, and that local data samples from different air quality monitoring stations are not identically distributed.

Common problems with traditional machine learning approaches (involving data privacy, power consumption, large storage, etc.) led us to use machine learning in a decentralized approach known as Federated Learning.

Why Federated Learning

Federated learning is a novel approach to decentralized machine learning, where the model is trained across multiple devices or servers on the local data obviating the need to gather the data in a common/central location, and the assumption that the local data is identically distributed. It also helps protect sensitive data as it is never shared with the server and employs secure aggregation protocols that govern the safety and anonymity of the devices participating in the federation.

In our approach we do not need to address the secure aggregation protocols as it is assumed that a relationship of trust already exists between the participants of the federation (as mentioned in the project specification).

Knowing the dataset

The dataset chosen for this experiment is taken from the Swedish Meteorological and Hydrological Institute (SMHI) and consists of numerous features that partake in determining air quality. These features include (and are not limited to) NO2, NOX as NO2, PM10, PM2.5, Black Carbon, CO, and O3. In this project we are trying to forecast the particulate matter PM10. All the features in the dataset may have a correlation with PM10, hence we pre-process the data to find these correlations and to better understand the dataset.

Meteorological factors such as air pressure, relative humidity, wind speed, etc may also impact air quality and may contribute to PM10 formation. The integration of these meteorological factors into the dataset, which is also available on the SMHI website, was axiomatic.

The preliminary analysis shows that the data is gathered in a sequence of successive hourly intervals starting from 00:00 hours and ending at 23:00 hours of any given day. The availability of the dataset is in a time-series manner and a deep time-series analysis is crucial to understanding the varying conditions that affect the air quality at a given time of the day. For example, our analysis showed that the pollution levels were higher in the day (04:00 to 16:00) from Monday to Friday compared to the weekends, thereby depicting the number of people working, driving, etc. Cases on any given day that may have irregular levels of pollution are flagged as outliers. For example, excessive use of firecrackers on New Year's Eve contributes to an elevated degree of air contamination. Irregular levels of pollution can also be due to sensor malfunction in the monitoring stations, or inclement weather.

What is time series data?

Time series data is basically a sequence of measurements that are collected over time. These measurements are taken with respect to a predetermined variable and at regular time intervals. One of the main characteristics of time series data is that the ordering matters!

The list of observations that we collect is ordered on a timeline, and the order in which they appear tells a great deal about underlying patterns. This will entirely change the context of the data, should the order be changed [17].

Components of the time series data:

- Secular trend, which describe the movement along the term;
- Seasonal series, are such that resemblances take place in identical sections of the period [25].
- Cyclical fluctuations, which correspond to periodical but not seasonal variations;
- Irregular variations, which are other non-random sources of variations of series. [24]

Why LSTM

In order to work with time-series data, Long Short-Term Memory (LSTM) was chosen as these networks have feedback connections which can process the entire data sequence instead of single data points. LSTMs are an improvement over traditional RNNs which faced the problem of vanishing gradients. A gradient is a vector whose value is used to update a neural network's weight. The vanishing gradient problem is when a gradient shrinks while back propagating and becomes so minimal that it contributes to less or no learning. The RNN layers receiving this small gradient value do not learn. Hence, in longer sequences, an RNN will forget what it has seen. It has, in other words, a short-term memory.

LSTMs, on the other hand, have a longer short-term memory (as the name suggests), and are able to learn and remember longer sequences. LSTMs' internal mechanisms include cell states and gates that control the information flow. The cell states carry relevant information throughout the sequence processing. Theoretically, it is possible to carry the data from an earlier time step to the last time step, allowing LSTMs to recall longer sequences and preventing the short-term memory problems faced by RNNs.

Methodology

Description of the data:

According to WHO reports, air pollution accounts to 4.2 million deaths per year due to heart disease, stroke, lung cancer, reduced lung functions, chronic respiratory disease and cardiovascular disease. These are due to the presence of toxic pollutants like Carbon monoxide (CO), Nitrogen dioxide (NO2) and many more. In the city of Stockholm, Nitrogen dioxide (NO2) and particulate matter (PM10) are two most air pollutants that are concerning. The rise in concentration of NO2 is due to the exhaust from traffic. Also, NO2 shows strong spatial and temporal variability. Similarly the increase in the concentration of particulate matter is due to the wearing of tyres while braking in vehicles. These particles can spread over a large area in a lesser time.

Swedish Meteorological and Hydrological Institute (SMHI) is a Government agency in Sweden, which has installed air quality monitoring stations across the city of Stockholm. List of air pollutants measured in the air quality monitoring station in the city of Stockholm are as follows:

- Black Carbon
- Carbon Monoxide (CO)
- Ozone (O3)
- Nitrogen dioxide (NO2)
- (NO2 as NOX)
- Particulate Matter (PM10)
- (PM2.5)
- Sulfur dioxide (SO2)

Note: Not all the stations in Stockholm monitor all of the above listed pollutants.

SMHI has also installed weather stations across the city of Stockholm which measures the weather data which can be used as an external feature to predict the concentration of PM10.

Weather data includes the following features:

- Air Temperature
- Precipitation Amount
- Air Pressure
- Relative Humidity
- Wind speed and direction

Initially, the hourly data is collected from the following station for the period of 00:00:00 01 January 2015 to 23:00:00 31 January 2019.

Station Number	**Station Name**	**Air pollutants**
Station 1	Stockholm Sveavägen 59 Street	CO, NO2, NOX as NO2, PM10, PM2.5
Station 2	Stockholm Hornsgatan 108 Gata	Black Carbon, O3, CO, NO2, NOX as NO2, PM10, PM2.5
Station 3	Stockholm Torkel Knutssonsgatan	Black Carbon, O3, CO, NO2, NOX as NO2, PM10, PM2.5, SO2
Station 4	Stockholm E4 / E20 Lilla Essingen	NO2, NOX as NO2, PM10, PM2.5

Table 1. List of stations in Stockholm

Note: Stations are selected in such a way that all the stations contain at least the following pollutants (NO2, NOX as NO2, PM10 and PM2.5)

Dataset behaviour:

For time series data analysis, it is hard to predict non stationary data which is defined by the data showing no periods, seasons or trends. So, the data should be processed so that it would become stationary in order to make successful predictions using statistical or deep learning models. Therefore, outlier detection techniques such as IQR are used to remove the outliers in order to make it more stationary.

For example the following graph shows the time series data visualisation of the PM10 values in 2019 where it contains PM10 gas values for approximately 800 hours.

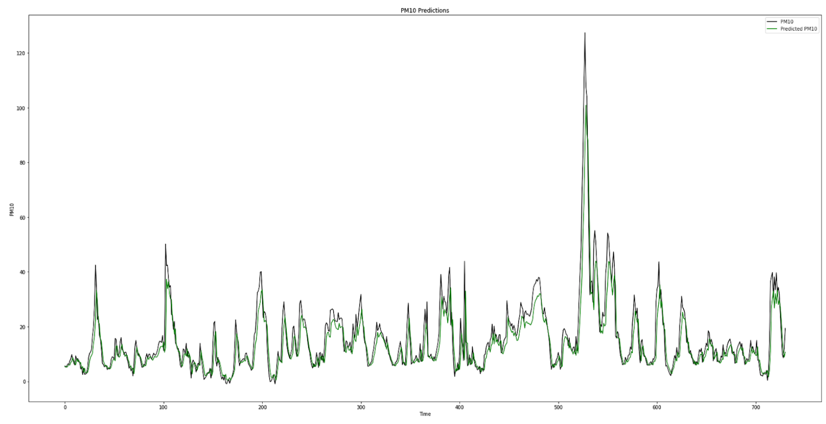


Fig 16. Visualizing predicted and actual PM10 values

The data contains some period that repeats over time like up and downs but with the random walk which can not be predicted as well. The dataset does not include any trend because it does not increase or decrease constantly, the last and the first values are a bit close to each other. It can also be observed that there is a huge value at time step 600 which is never seen before and it should be removed otherwise the values after that value will be predicted as the ones that have so many errors.

Anomalies in Data:

SMHI uses high quality air quality monitoring station and weather station data. Also, some data cleaning and preprocessing has been done by the SMHI before sharing the data to the public. However, it is advisable to preprocess the data before creating the model as the data are recorded by the sensor network and there is a large possibility for the error to happen. For example: the concentration of air pollutants can't be negative. Also, there might be a possibility of Outliers which may not be sensor errors but it can be due to some events like fireworks during New year's eve. These errors need to be addressed before moving onto creating a model to predict the concentration of PM10.

Missing Values

Missing values is one of the most common types of error that occurs in a dataset. The dataset is a continuous collection of the level of pollutants in the atmospheric air for every hour. But there are a few values missing in between the continuous dataset. This can be due to various reasons such as sensor malfunctioning, communication error, storage corruption, etc. The missing values are in the form of an empty string in the dataset.

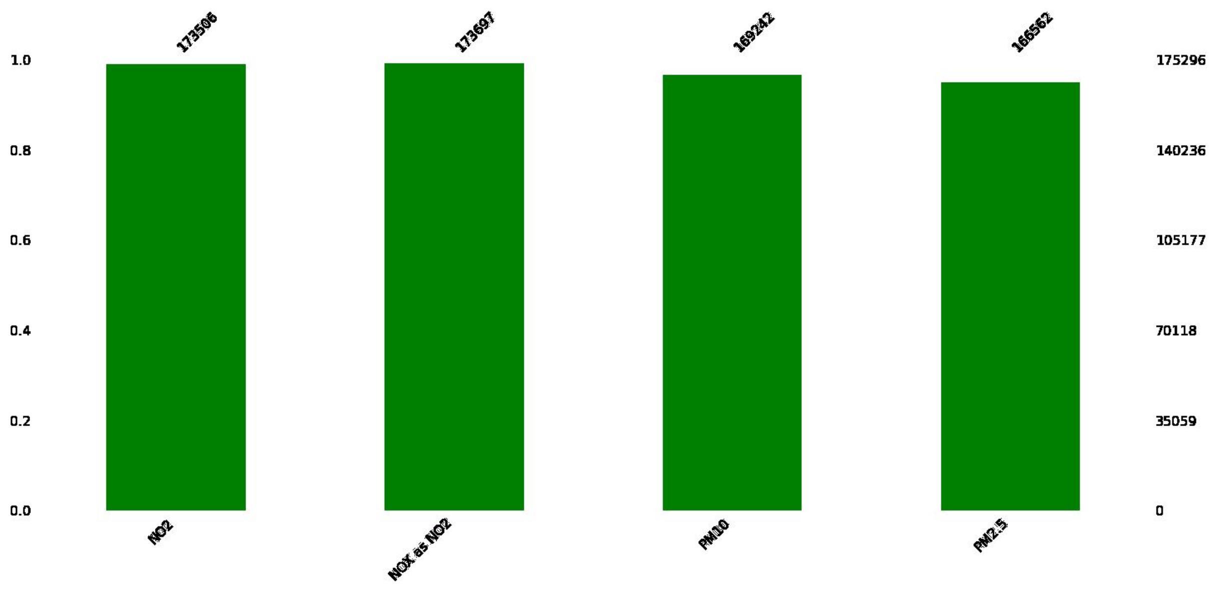


Fig 1. Bar graph that shows the number of entries(without NAN) each feature has out of total entries (175296).

From the above graph, it is interpreted that the number of missing values in

NO2 is 1790 (1.02%), PM10 is 6054 (3.45%) and

NOX as NO2 is 1599 (0.91%), PM2.5 is 8734 (5.0%).

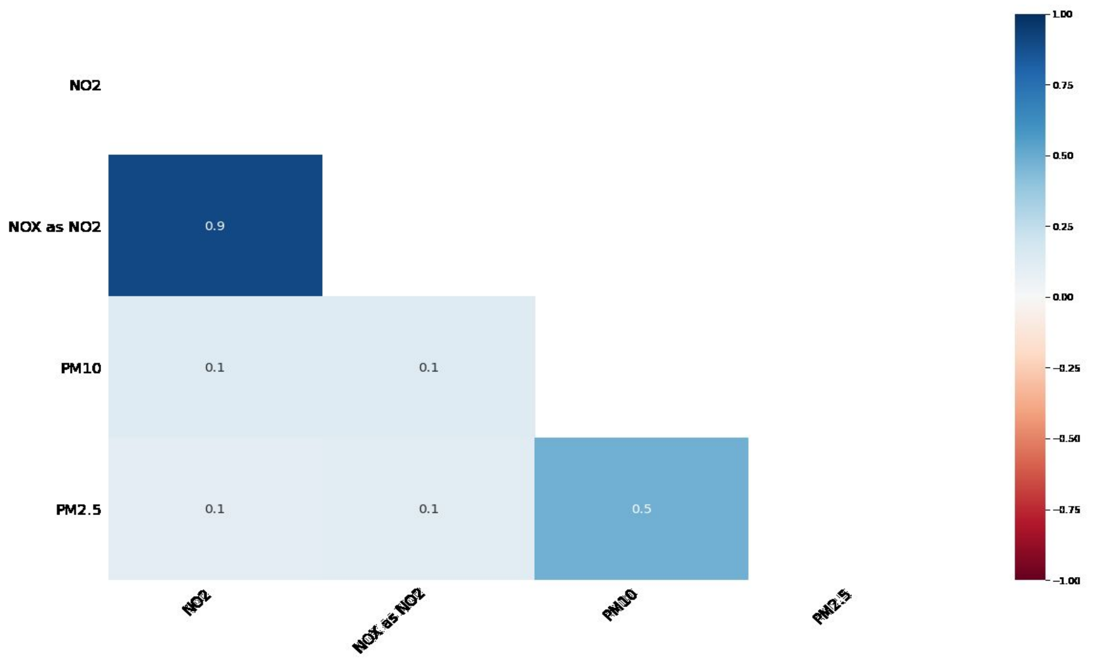


Fig 2. Heat map that shows the correlation between the null values by feature.

Outliers in data

Various studies were carried out to analyze the data gathered in successive hourly intervals. Some inspirations were taken from the reference paper published in Springer Link on 08 March 2018, "Outlier Detection in Urban Air Quality Sensor Networks."[23]

In this study, the researchers aggregated the NO2 concentrations to hourly values. Using 10-min data, the outlier detection method would give more detailed instances of outlier compared to using hourly data. The results of 10-min outlier detection were interpreted differently from the results of hourly outlier detection. In hourly outlier detection, peaks occurring as a result of a strongly emitting vehicle passing by are more likely to be averaged out as they may occur every hour. In 10-min data, such peaks are more likely to be considered outliers. Hourly outliers give a better overview of hours in which there is an abnormal number of peaks rather than showing individual peaks, as in the case of 10-min outlier detection[23].

Widely used traditional outlier detection methods are as follows:

- Functional outlier detection - common type of temporal outlier detection that compares various function curves of fixed time periods. So, in the past this method was used to detect PM10, SO2, NO, NO2, CO and O3 to detect months with unusually high air pollutant concentrations. Or, working days and non-working days with outlying NOX levels[23].

- Functional outlier detection is a method used to compare an entire vector of measurements like for example, all observations in a month, so therefore, it is less suitable[23].

The data get cleaned before being used. Negative concentration values occurred when the concentrations were below the limit of detection and were removed from the dataset (1.5%). Zeroes in the data indicated a sensor failure and were removed from the dataset (1%). High peaks in NO2 concentrations can occur in 10-min data if the sensor is exposed to a high concentration peak for a short period of time. Similar peaks in hourly concentration data however are more likely to be caused by sensor failure and influence the outlier detection[23].

The above study was helpful in detecting the outliers in air quality data. In a low cost sensor network, there is a high possibility of having outliers. Outlier detection is the method to find the data that are statistically different from the expected value at given location and time. There are two kinds of outliers that are possible in a low cost air quality monitoring sensor network. Errors that are due to sensor faults/ malfunction, human handling of sensors, positioning of sensor in harsh weather conditions and many more. These errors need to be removed as it affects the data analysis and adds bias to the data[23].

Another type of outliers is due to events. Events are true observations of immensely high and low concentrations of air pollutants compared to the expected concentration of air pollutants at the particular location and time (Zhang et al. 2007). For example, the concentration of air pollutants are observed high during the night of december 31st (New year eve) every year due to firework events happening across the city [11]. True events can be related to very local sources (e.g., a small fire, truck idling within meters of a monitor) or to very unusual weather circumstances such as low mixing height and high atmospheric stability resulting in poor dispersion of emitted pollutants.

Outliers can be classified into two types based on the location and time. Spatial Outliers are data whose values are different from the values of the spatial neighbourhood. Since there are many air quality stations in the city of Stockholm, there is a high possibility of spatial outliers in the data. For example, air quality stations near to the industries or road traffic can have a higher concentration of air pollutants when compared to other stations. Temporal Outliers are data whose values are significantly different from the expected values over time. Firework events during the new year eve is one example that affects the behavioural pattern of the data across time [12].

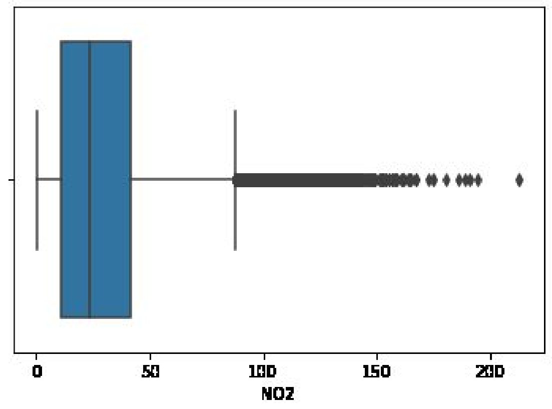


Fig 3. Box plot visualization of NO2

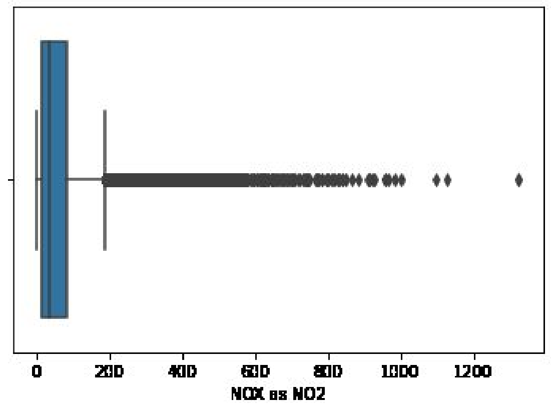


Fig 4. Box plot visualization of NOX as NO2

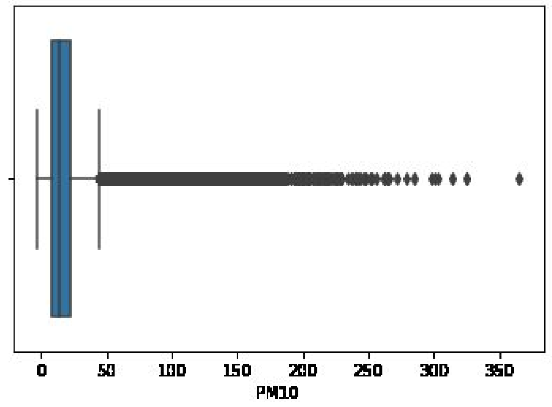


Fig 5. Box plot visualization of PM10

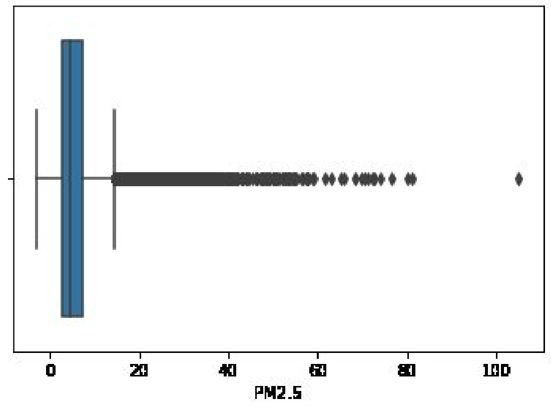
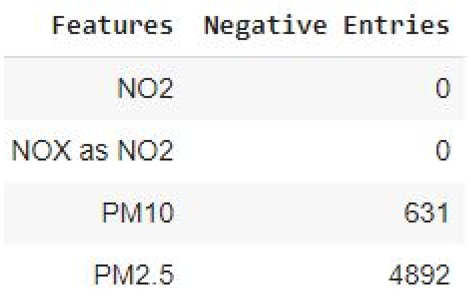


Fig 6. Box plot visualization of PM2.5

Negative values

The level of pollutants cannot be negative, hence a negative value in the data is erroneous. The level of impurities in the atmospheric air is measured as the difference between the atmospheric air and a sample clean air. At times, the atmospheric air could be cleaner than the sample air. In these cases we would end up having negative values slightly below zero. A larger value below zero would be due to sensor error.



Features	Negative Entries
NO2	0
NOX as NO2	0
PM10	631
PM2.5	4892

Fig: Table that shows the number of negative values in the data

Since there is no much difference between negative values and no value (NaN) as the real value is unknown, the negative values are considered as NaN like missing values.

Data Preprocessing:

Data preprocessing is an important step in data science and machine learning problems. If the data is not properly analysed, there is a high risk of producing false results. Also, knowledge discovery will be difficult if there is any irrelevant information or noise present in the data. Hence, the analysis of data is the first and foremost method after data collection in any machine learning problem. Data preprocessing method includes data preparation, data cleaning, dimensionality reduction and transformation of data, outliers detection and removal, and normalisation. The results obtained after the preprocessing methods will be the final dataset which will be used for further machine learning algorithms [13].

In the previous section, we have looked at the different anomalies that are found in the data. These anomalies are resolved using the following data preprocessing techniques.

Feature Engineering

Feature engineering is a method where new features are developed from the raw data using data mining techniques. In time series data, different time features can be created to check whether the data follows any trends. For example, are the concentration of the air pollutants higher during weekdays when compared to weekends? or how the concentration of air pollutants varies depending on the four seasons (Summer, Winter, Autumn and Spring) in a year. Similarly, there can be hourly trends like at what period of time in a day where the concentration of air pollutants are seen high. This can be due to traffic during peak hours of the day which can lead to increase in pollution. These time features can be introduced into the data through one hot encoding.

Many of the machine learning algorithms require numerical features as Input. When there are categorical features in the data like seasons in time series data, it needs to be encoded into numerical features using feature engineering so that the machine learning can read and understand the feature. One hot encoding is the most commonly used encoding method. For example, different seasons in a year can be encoded as follows.[10]

Season	**One hot encoding**
Summer	[1,0,0,0]
Winter	[0,1,0,0]
Autumn	[0,0,1,0]
Spring	[0,0,0,1]

Feature Selection

The performance of a machine learning model is hugely influenced by the data features used to train it, therefore the feature selection method used to select the data feature is vital. Feature selection is usually carried out before training the machine learning model as it provides benefits like reducing training duration, increasing accuracy and also avoiding overfitting as unwanted data features are removed. The feature selection methods such as Correlation and Feature Importance were used.

Correlation

Correlation is used to check how close two variables are linearly related with each other. The Correlation value ranges from +1 to -1, High positive correlation value or high negative correlation value indicates a good linear relation between the two variables and correlation closer to zero indicates no linear correlation between the variables. The issues with correlation are that it is hard to convey some useful information from the interpretation and it does not interpret any non-linear relationships between the variables which is also important for choosing Features. Correlation can be only applied to continuous data variables. Apart from vanilla correlation or Pearson correlation, Spearman correlation can be used to identify non linear relation between variables, it is possible because as spearman correlation assesses relation using monotonic function. Spearman correlation can identify three cases if it is monotonically increasing, decreasing or finally not monotonic. Therefore Spearman is better than Pearson correlation, figure below provides the results from each method..

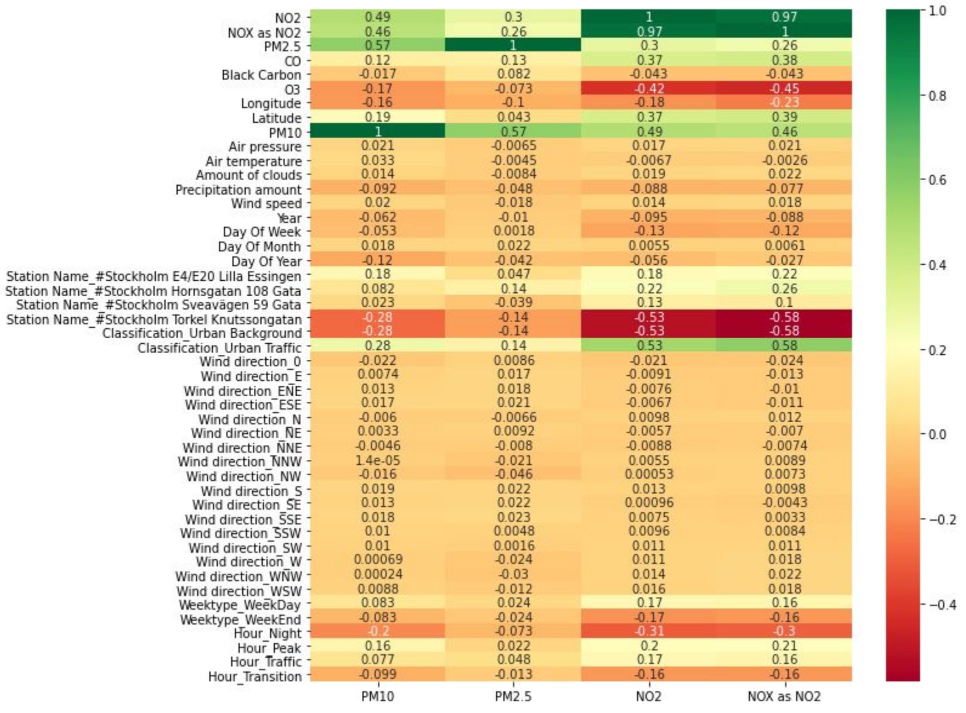


Fig 7.(i). Heat Map Plot Interpretation of Pearson Correlation

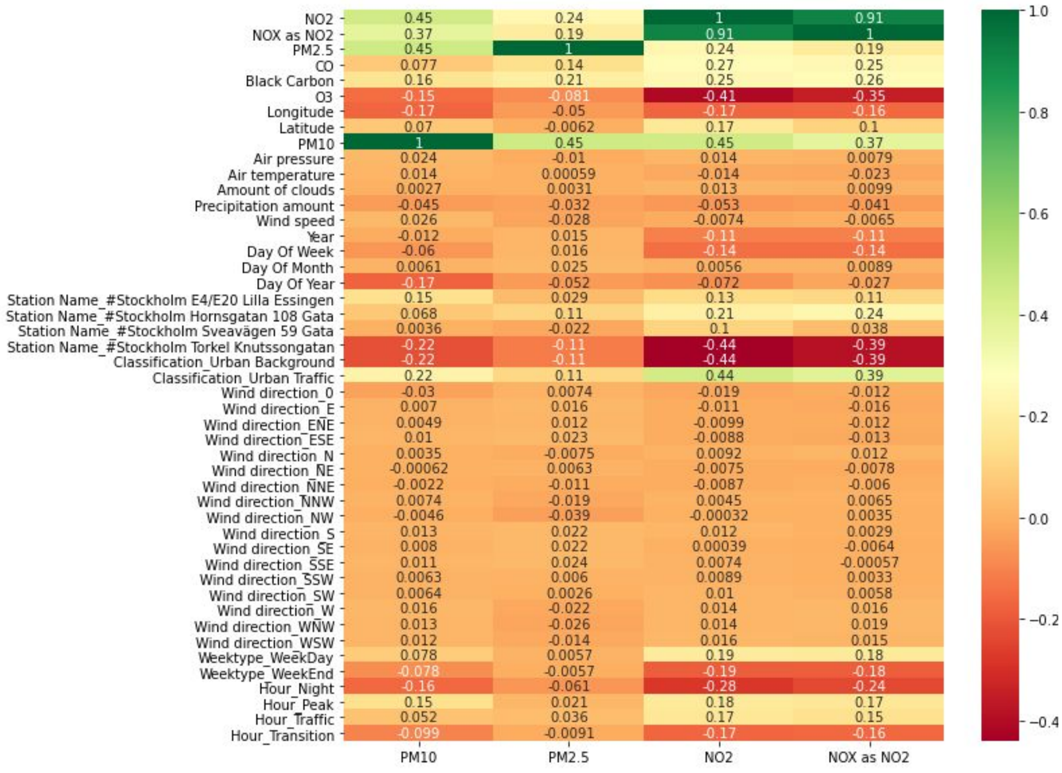


Fig 7.(ii). Heat Map Plot Interpretation of Spearman Correlation

Feature Importance

Another popular method is Feature Importance, as the name denotes it provides a score of how important each feature is in predicting the target feature or prediction. It allows you to identify data features that contain the most information denoting the distribution of the target variable. The importance of the features is determined by the model's logic, therefore the model performance was validated on a test set to ensure its a good model and so the results obtained can be accepted to be valid. Feature importance overcomes the disadvantages of correlation such as not considering categorical features and not considering non-linearity between input and target features.

Feature Importance using XGBoost (Extreme Gradient Boosting)

XGBoost is a variant of gradient boosted decision trees that is used because it is straight-forward method to obtain an importance score for each feature using decision trees. We went for XGBoost over other gradient boosted tree methods as it is efficient, fast and finally has a built-in function to carry feature importance. It uses n weak learner regression trees to accurately predict target features. There are three possible importance types that can be used for the mentioned method weight, gain and cover.

- In weight-based the importance score for features is simply determined on basis of how many times a feature is picked while generating trees.
- In gain-based the importance score for features is based on how features affect the accuracy.
- In cover-based methods, a different approach of splitting the dataset based on the node split is used to determine the importance score.

But out of these three important types, weight was chosen just on the basis that it provides the importance score based on how many times a feature is deemed useful . Below figure showing the score result obtained.

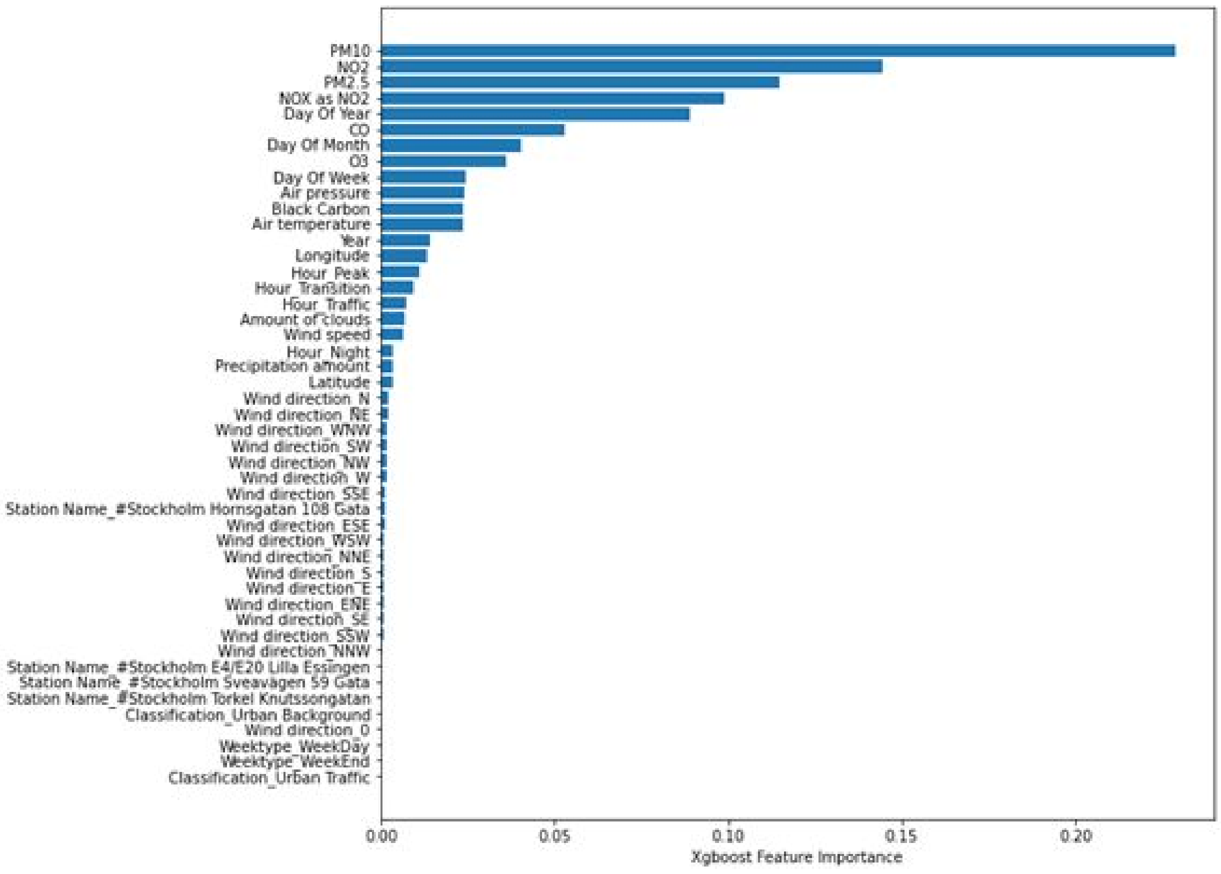


Fig 8. Feature Importance Score using XGBoost for predicting PM10 (target feature), importance type is weight

We know that feature importance using XGBoost does provide a sensible means for variable selection in many applications, but it is not reliable in cases where the target feature has varying continuous features scales or several categories features. [18]

Permutation based Feature Importance using XGBoost (Extreme Gradient Boosting)

The permutation-based importance overcomes the issues of the vanilla feature importance, it is possible as the model is computed after each feature is randomly shuffled which allows to evaluate the features that are important for the performance. In order to overcome the issue of attaining misleading results from a bad model, we evaluated the performance of the XGB model on the test data and it got a rmse of 9.2329. This method gives a breakdown of the relationship between input and target features and is suitable for all rectangular data [19]. Below figure showing the score result obtained.

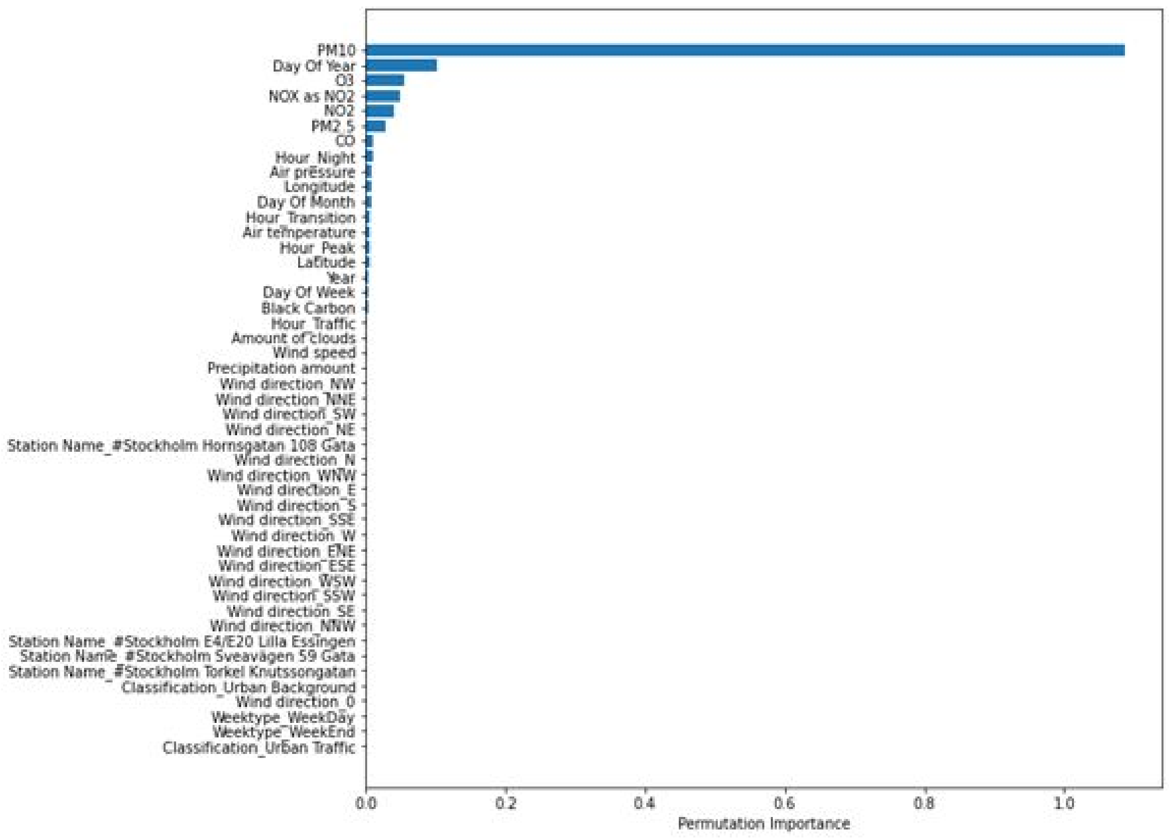


Fig 9. Permutation based Feature Importance Score using XGBoost for predicting PM10

Even though Permutation based feature importance is really a lot of the use cases, it is computationally expensive as you need to run several iterations

Feature importance using SHAP values

Shapley Additive Explanations (SHAP) [20] provides the explanation of how input features contribute to the prediction of the target features and SHAP is based on Shapley values from game theory. It has a vast collection of explainers and plots that could be used to interpret models and understand every aspect of how the solution is derived using the model. We tried using Deep Explainer, an explainer function for Deep Neural Networks that would help to get an insight of the working of LSTM based model training but we were not successful at implementing it but it worked absolutely fine with the previous used XGBoost model. SHAP provides detailed model explanation, which therefore simplifies the process to select preferred features and train model to attain good performance, below are figure providing (1) Feature importance using SHAP and (2) Breakdown on what rank of values contribute the most. Features selected using SHAP method are been used for the training final model.

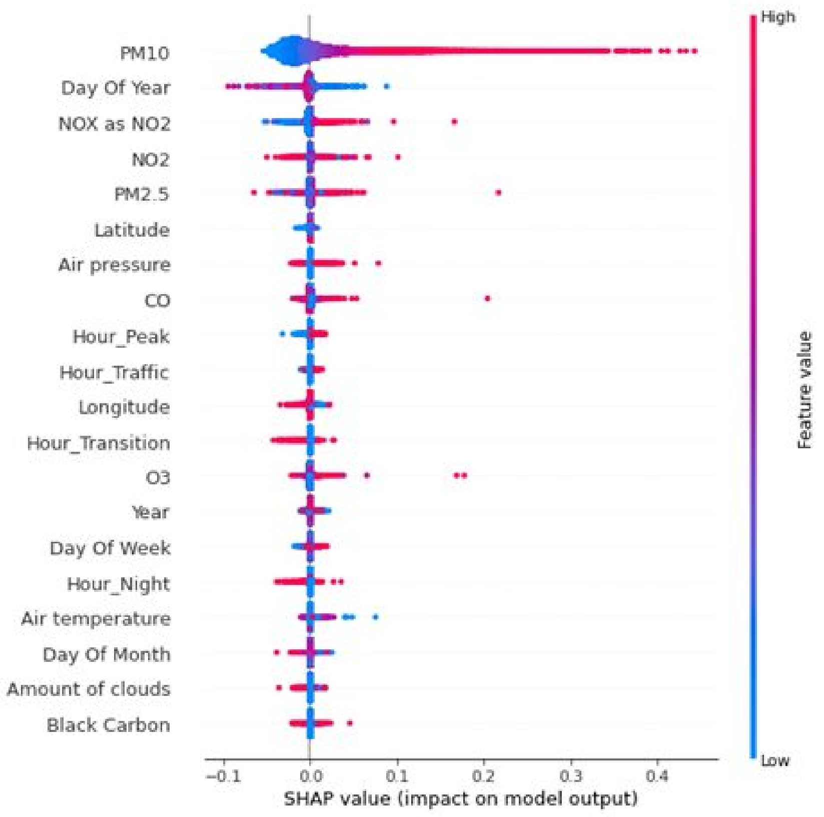


Fig 10. SHAP Feature value-based impact for predicting target feature PM10

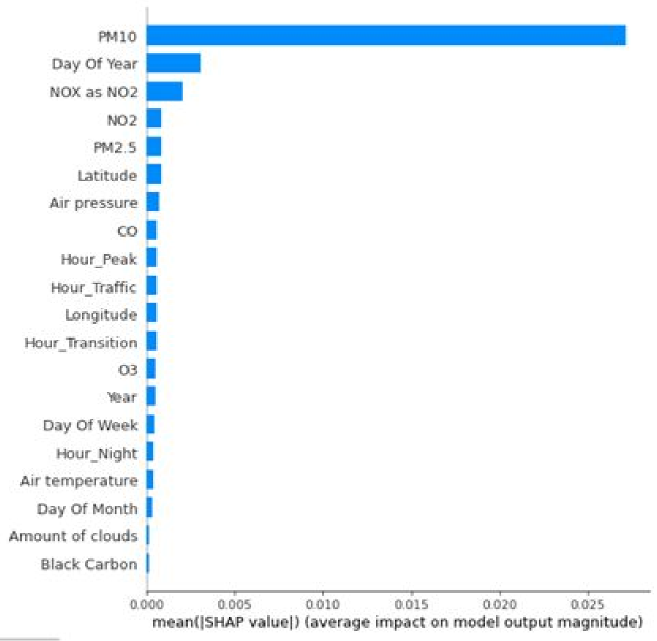


Fig 11. SHAP based Feature Importance Score for contributing features using XGBoost for predicting PM10

Data Imputation

Missing data and negative values are two major anomalies found in the data. To handle missing data, the data imputation method is introduced. Data imputation is a method that uses statistics to introduce data into the dataset in such a way that it does not affect the behavioural pattern of the dataset. However, data imputation may introduce bias into the dataset [14]. Higher bias can decrease model complexity leading to models being underfit. Hence, data imputation can be used in case the dataset has marginally lesser missing values so that it doesn't introduce much bias in the dataset. From the missing value stats, it is observed that NO2 and NOX as NO2 has only 1% of missing values. Thus, these missing values can be handled using data imputation. There are different methods to impute data.

Mean

One method of imputing values is by finding the mean of non-missing values and replacing the missing values with the mean value. It is one of the easiest ways to impute missing values however, it can introduce the bias in the data.

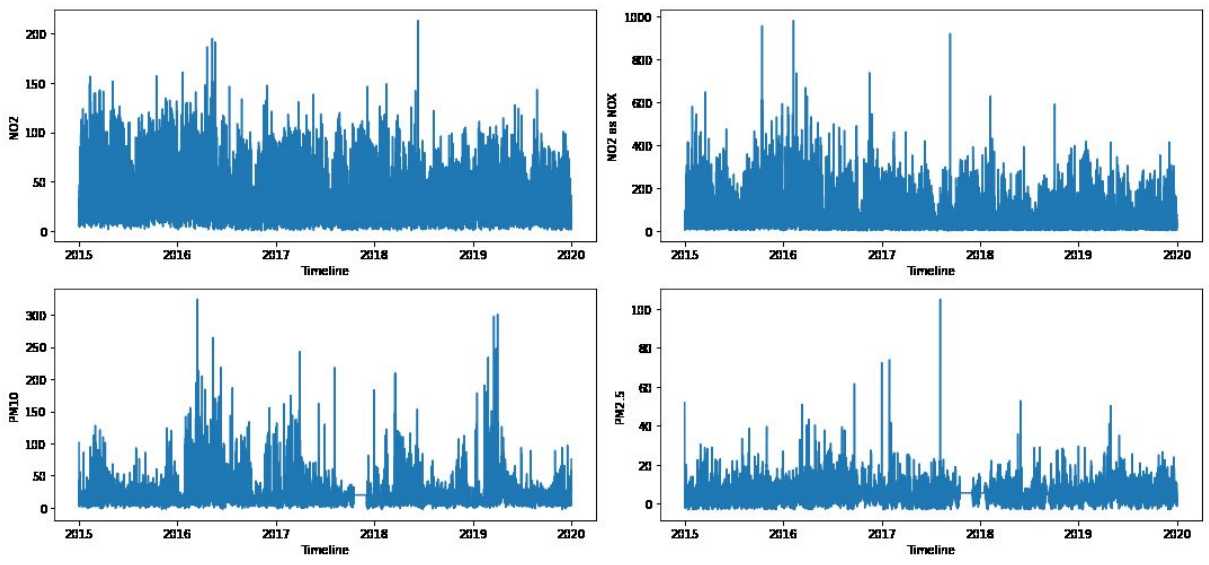


Fig 12. Distribution of Station 1 temporal data after mean imputation

Median

Another method of imputing values is by finding the median of non-missing values and replacing the missing values with the median value. The method is similar to the previous one where the median of non-missing values is taken instead of mean. Similar to the mean imputation, the method can increase the bias in the data.

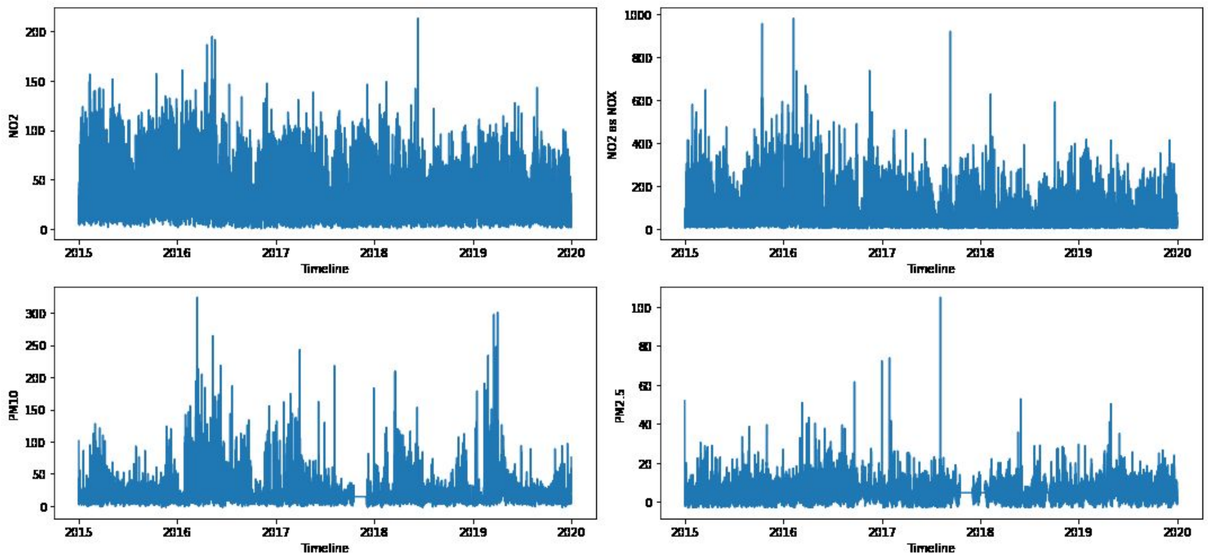


Fig 13. Distribution of Station 1 temporal data after median imputation

From the figure, it is observed that there is not much difference in the distribution of temporal data.

Interpolation

Interpolation is a method that finds new data from the range of already known data. Linear Interpolation is the most commonly used method where the missing value is found using the data points of previous value and the next value.

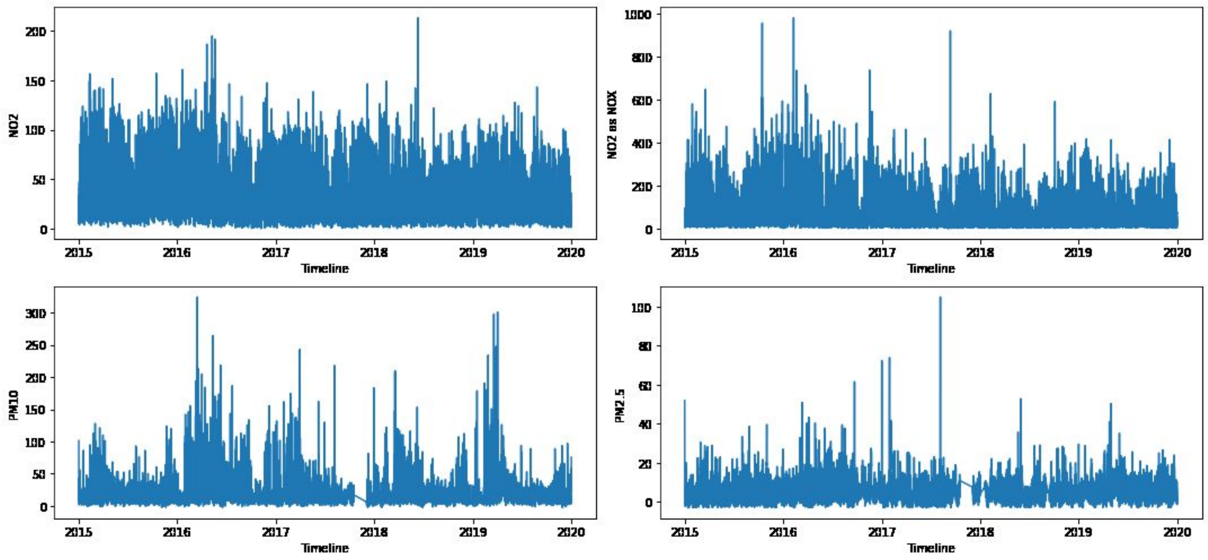


Fig 14. Distribution of Station1 temporal data after Linear Interpolation

Data imputation using GAIN

Generative Adversarial Imputation Nets (GAIN) is adapted from popular Generative Adversarial Nets (GAN) to impute missing data points present in rectangular data. Similar to vanilla GAN, GAIN consists of generators and discriminators but has additional input elements such as mask matrix to generator and hint matrix to the discriminator. Our main aim out of implementation was to impute missing data in cases of random missing values across the four air pollutants ('NO2', 'NOX as NO2', 'PM2.5', 'PM10'). Both Generator and Discriminator are fully connected neural nets, Generator is used to generate plausible data points very close to original or expected value while discriminator validates the generated sample to be near the actual value or not. GAIN is basically a MinMax problem where the Generator tries to generate a good enough value that discriminator assumes it to be the actual value. Hint matrix used in the Generator is to prevent overfitting of discriminator and is just disclosing a portion of the data as illustrated in the figure 15. In order to train a data imputation network a data sample with least missing data is required, by means of data distribution analysis across years, months and stations we choose Stockholm St Eriksgatan 83 Station and time slice was 2018 January to October which had only 18 rows of missing data. We therefore ended with a sample size of 7295 rows and 4 columns ('NO2', 'NOX as NO2', 'PM2.5', 'PM10'). Based on the chosen missing rate a Mask Matrix is generated with binary values where 1 represents data present and zero represents missing values. Based on the aforementioned Mask matrix the Original data is augmented. As illustrated in figure 15 the model was trained for 100000 iteration with batch size of 32 and hint rate of 20%. We attained really skewed values and were not able to figure out the reason for the poor performance and possible reasons later identified were disregarding temporal information and difficulty in identifying convergence of the model.

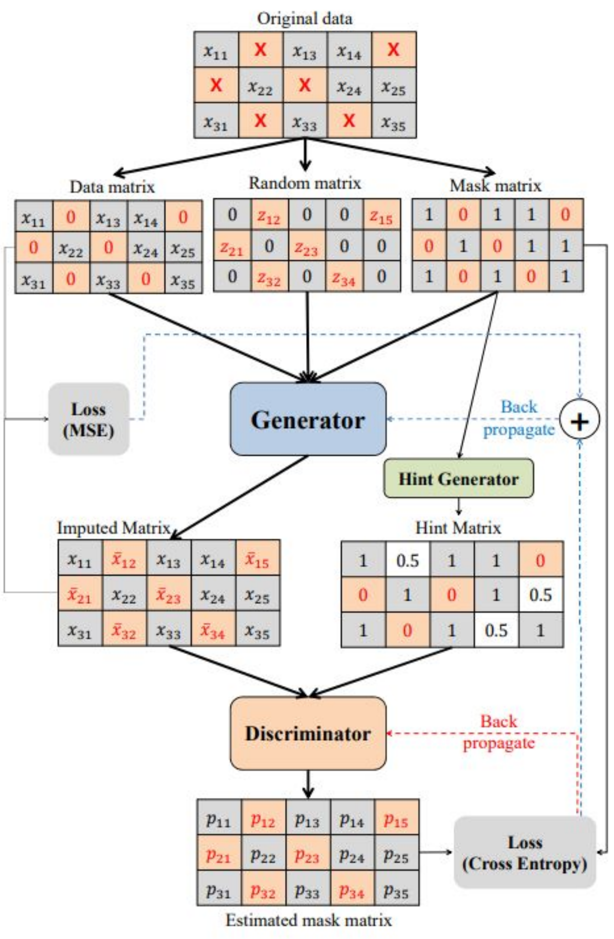


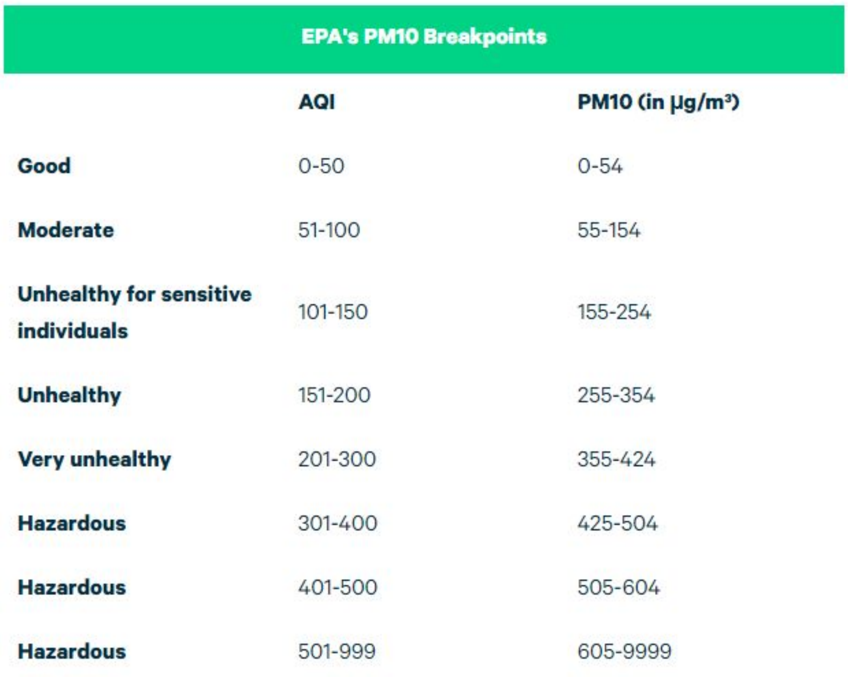
Fig 15. GAIN Architecture[22]

Outliers Removal

Any value that does not comply logically with the required series would be an outlier. In this data the outliers are high values caused due to events like new year or fire in the locality. In some cases the values would get extremely high due to sensor errors. Negative values are also outliers since pollution values cannot be negative.

Threshold truncation

Threshold truncation is the preliminary step in outlier removal. As we can see from the figure below the moderate levels range from 55-154. Anything above 255 is unhealthy and it indicates that it would not be a normal circumstance. Hence we shall fix a threshold value and remove the values above the threshold. On studying the collected data a value of 200 is fixed as the threshold and values about this value were truncated. The removed values were treated as missing data. Similarly, the negative values were also removed and were treated as missing data. These missing data are later interpolated to fill the sequence.



EPA's PM10 Breakpoints

	AQI	PM10 (in µg/m³)
Good	0-50	0-54
Moderate	51-100	55-154
Unhealthy for sensitive individuals	101-150	155-254
Unhealthy	151-200	255-354
Very unhealthy	201-300	355-424
Hazardous	301-400	425-504
Hazardous	401-500	505-604
Hazardous	501-999	605-9999

Fig 15. Table showing the PM10 risk levels and thresholds retrieved from [21]

Normalization

Normalization is the method in which data of different dimensions are transformed into values of comparable ranges. Each feature has data in different ranges of values and there is a high possibility that the feature with higher range of values can influence the result of the model due to its larger value. Features with a larger range of values can dominate other features even though the relative variations of the other features are more significant. Therefore to avoid these issues, data is normalized before developing the model. The most simple way to normalize the data is to find the mean and standard deviation of each feature and subtract each value in the feature by its respective mean and divide by its respective standard deviation. This method of normalisation is called Z-transformation or Standardization.

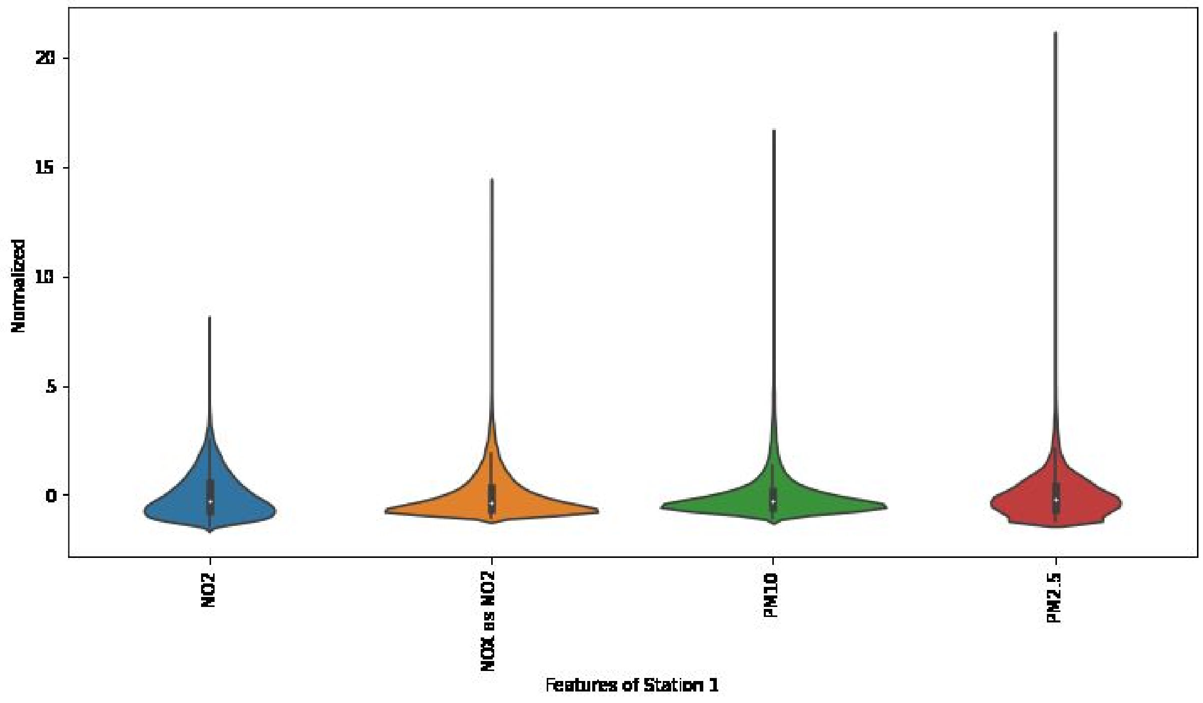


Fig: Violin plot of Station 1 data after Standardization

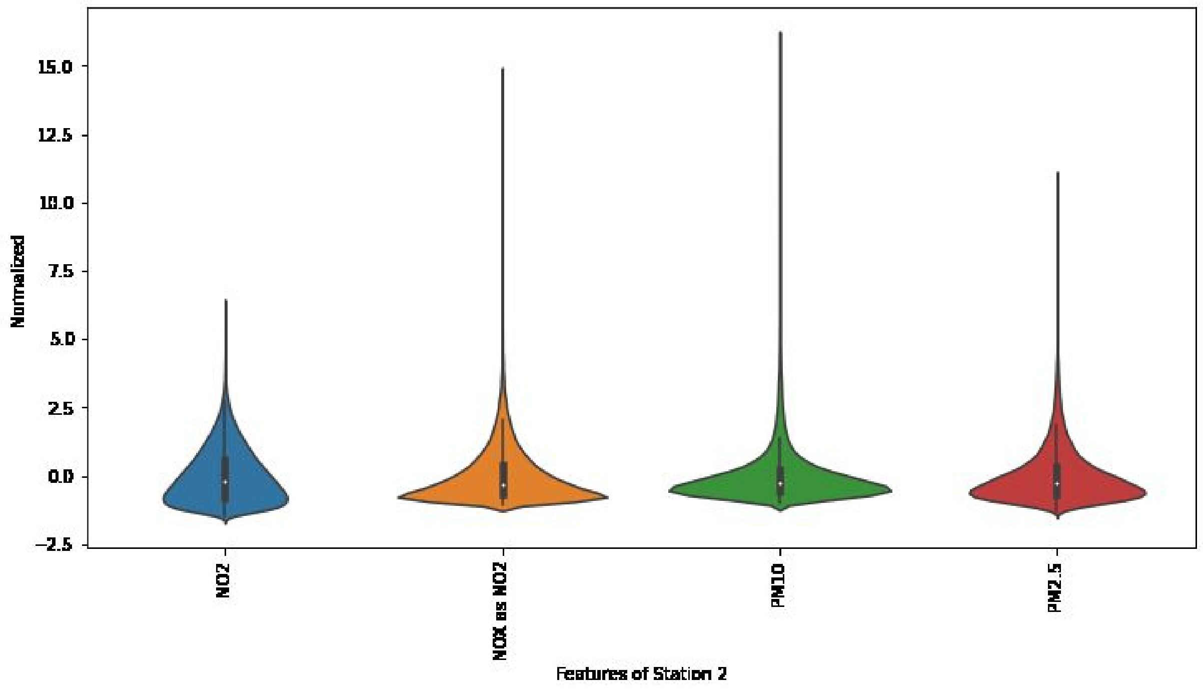


Fig: Violin plot of Station 2 data after Standardization

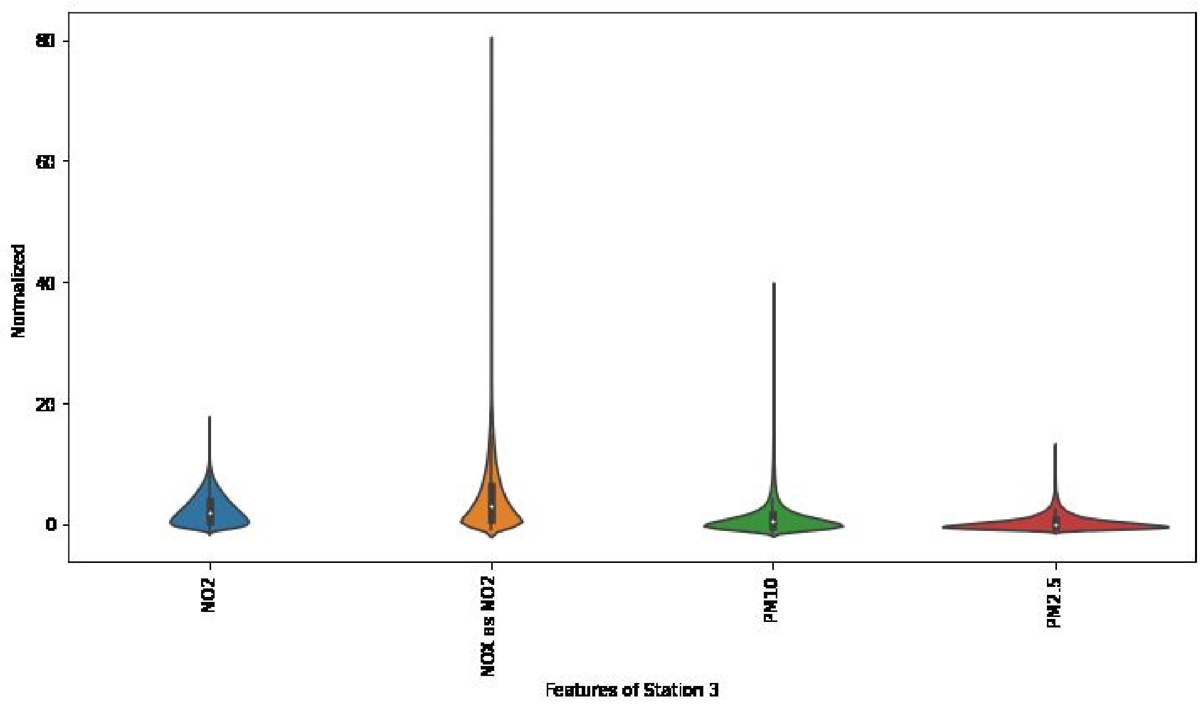


Fig: Violin plot of Station 3 data after Standardization

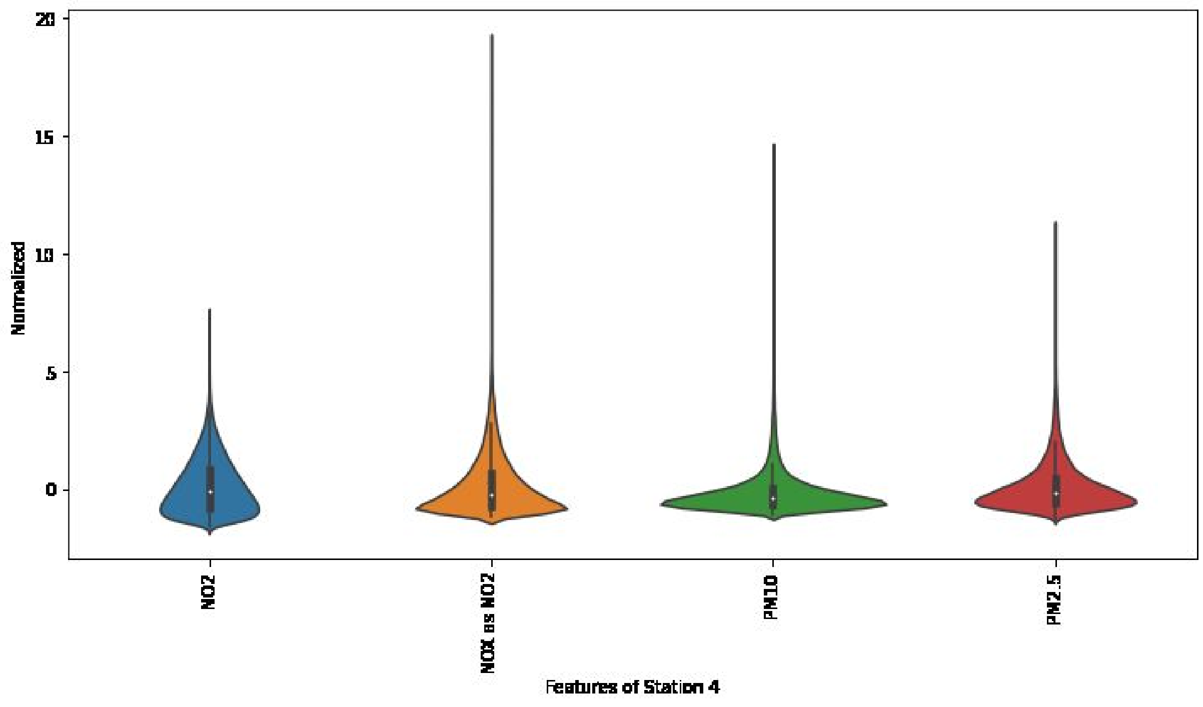


Fig: Violin plot of Station 4 data after Standardization

The above results show the data distribution after normalisation (simple normalisation method discussed above) through violin plot. There is another way of normalising the data. In this method, the min value (Xmin) and the max value (Xmax) of a feature is found. For every value X in the feature, replace X with (X - Xmin)/ (Xmax - Xmin). This process is repeated for every feature in the dataset. This method of normalisation is called min-max rescaling where all the values in the dataset are in the range [0,1].

Centralized Model

Generalized model:

There exists a competition about the centralized model accuracy within two groups where it is tried to predict 4 gases values such as CO, NO2, PM10, NOX, etc. Even though the main objective of this project is to predict the PM10 gas values, the model may evolve to an overfitted model when the loss value is tried to be minimised more and more. In addition to minimising the metric with a competition atmosphere, it is also useful for acquiring a generalised model so that an overfitted model with only PM10 values as output may not perform well on different datasets. Therefore predicting 4 different gase values within the approximately equal SMAPE metrics are acceptable for the model to be considered as generalized.

LSTM Model:

The centralized LSTM model that performs the best is the one that is used in the competition where it predicts the 4 gases values with approximately 0.5 SMAPE values for each one. Therefore, this model is suitable for a generalized one and can behave well enough for every gase for different datasets.

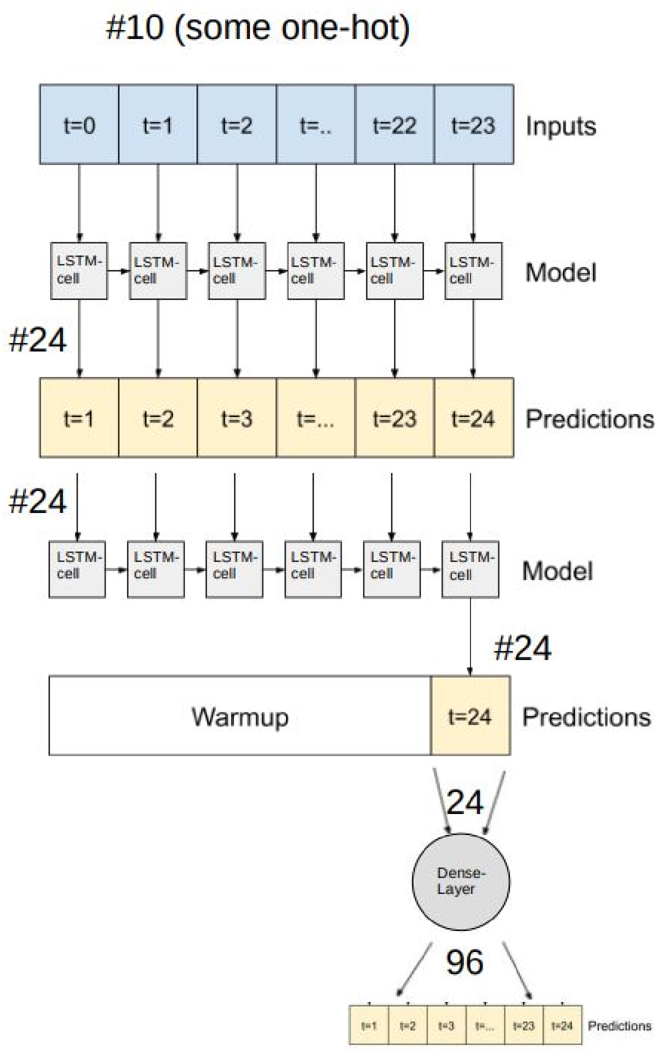


Fig 17. General architecture of our 24 hours prediction model

The LSTM model contains 3 layers: First layer is the LSTM-sequence Layer which takes the 24 hours sequence of the all features coming from the one hot encoding and outputs the 24 hours sequence of the future values.

Secondly, the middle hidden layer also is an LSTM layer but it performs sequence to vector predictions where instead of sequence to sequence predictions. Since the outputs are future hourly values for only the next hour, the output shape is only 24 which come from the last LSTM cell as output.

Lastly, it contains the dense layer getting the next 24 hours values as predictions for all 4 gases therefore, the number of output values are 24 x 4 = 96 values.

Hyperparameter tuning:

Different parameters are tried with the centralized model in order to get the best MAE score. Also, early stopping and model checkpoint techniques are leveraged to save the model by setting up a patience metric. It means that if the model loss metric does not decrease after a patience amount of epochs then the training stops. Consequently, a better model can be gathered with less time. Also, the best model is saved with the best parameters so that the training with the same parameters is not performed every time when the prediction with test data is being achieved. This also can save up time. The parameters that are used in the LSTM model are Epochs = 100, Batch size = 24, Optimizer = ADAM, Loss = 'mae'.

There is a lot of disagreement on whether or not drop out layers are useful, it can simply be stated as a layer that helps generalize the model and also assist with forestalling overfitting by disregarding arbitrarily chosen neurons during preparation, and hence diminishes the affectability to the particular loads of individual neurons.

CPU	Intel i3 4030u
GPU	Nvidia 830M
RAM	8GB
Tensorflow Version	2.2.0 GPU with 10.1 Cuda Libraries

Table: Hardware specification used to run the below experiments:

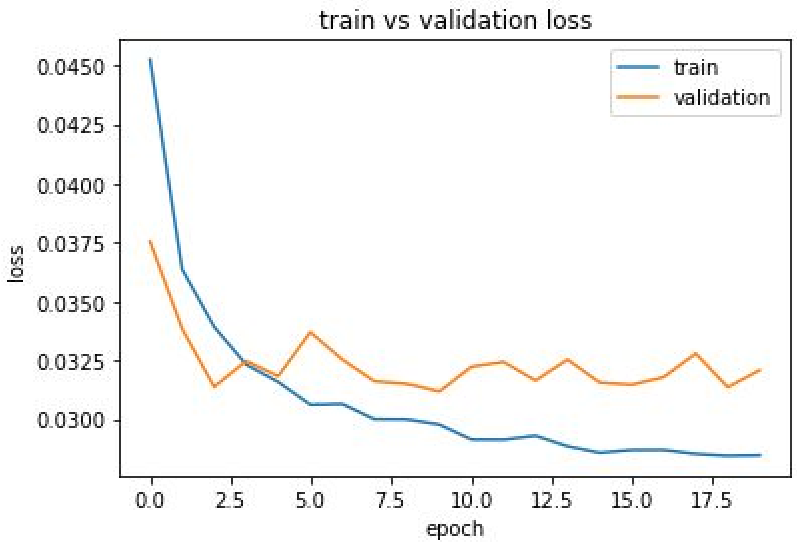
Experiments

The following three experiments were carried out to analyze how the feature selected affects the performance and also to verify if the selected feature using SHAP provide better performance.

Experiment 1:

In this experiment only four features: PM10, NO2, NOX, PM2.5 are used. The previous 24 hours values are utilised to predict the next 24 hours of the PM10. Based on features availability across the different stations the following 4 stations were used for the experiments and they are #8779, #8780, #8781 and #18644. 2017 is used as Training data used, 2018 as validation and finally 2019 as Testing data. The data is normalized using Min Max Normalisation between range 0 and 1. All NaNs and negative values are replaced with zeros, the reason we did not carry one with any of the aforementioned imputation methods is that they all introduced biases. Regarding the model structure it is sequential with two LSTM layers each with 24 LSTM units followed by a dropout layer with 0.2 dropout value. Along with adam as optimizer, Mean absolute error as loss, Early stopping with patience 10 is used to auto terminate training.

Results Obtained for Experiment:



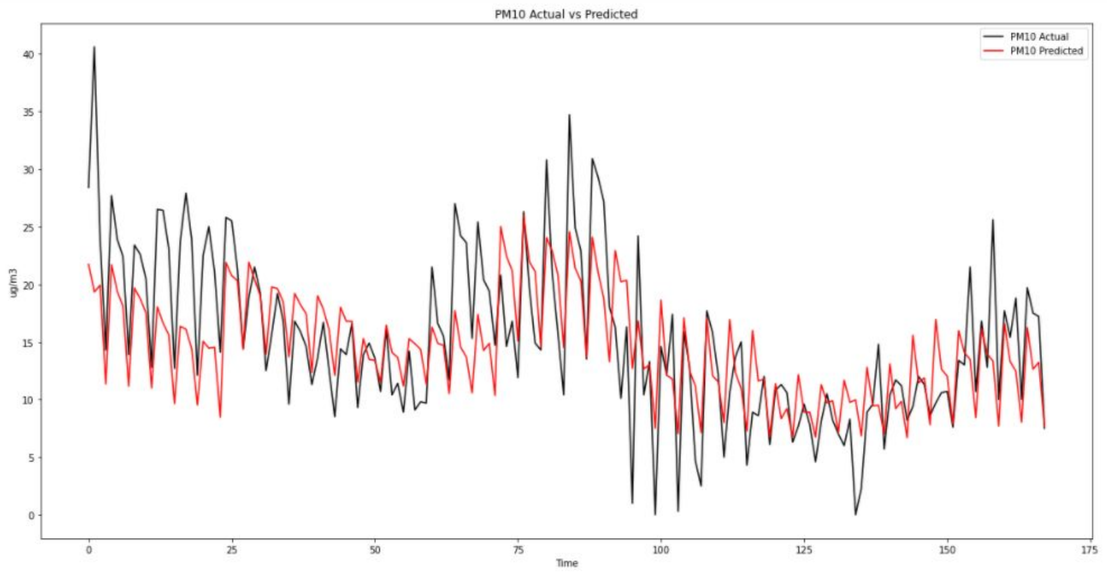


Fig 19. Train vs validation loss plot for experiment 1 and actual vs predicted value plot

Training time: 37.88 seconds

Epochs Run before convergence: 20 epochs

Metrics	Training Data:	Testing Data:
RMSE	20.0753	9.4730
MAE	3.1558	2.4320
SMAPE	0.4306	0.3984

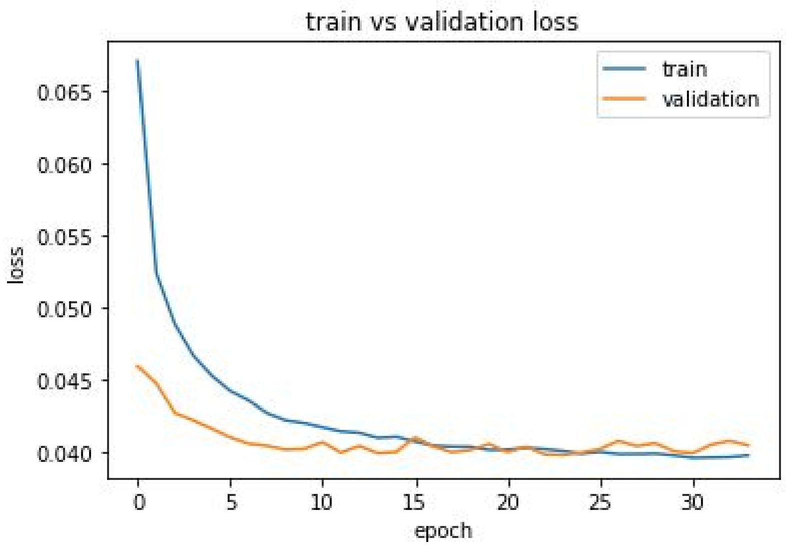
Table 2. Results from Experiment 1

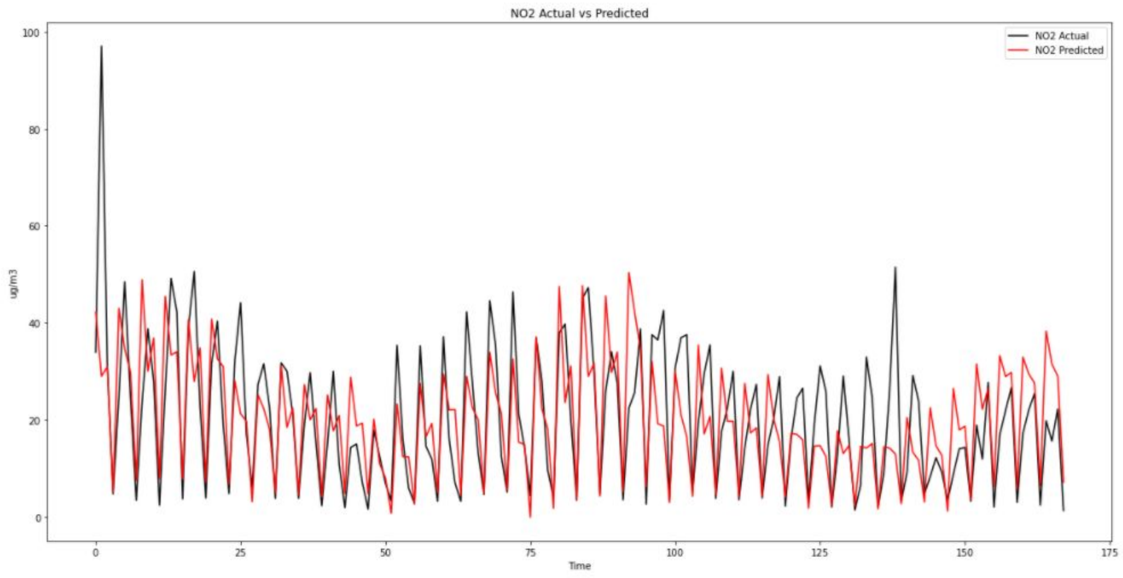
The above experiment was carried out as a baseline to verify how the model performed with just predicting PM10, so that its performance can be used to validate other experiments' performance. The metrics table shows that the model performed better on the test data than on the training data, the same behaviour can be seen across the experiments. The reason for this behavior could be due to the variation across data, the usual solution to the issue is to randomise the data set but our data is time series so it cannot be used here.

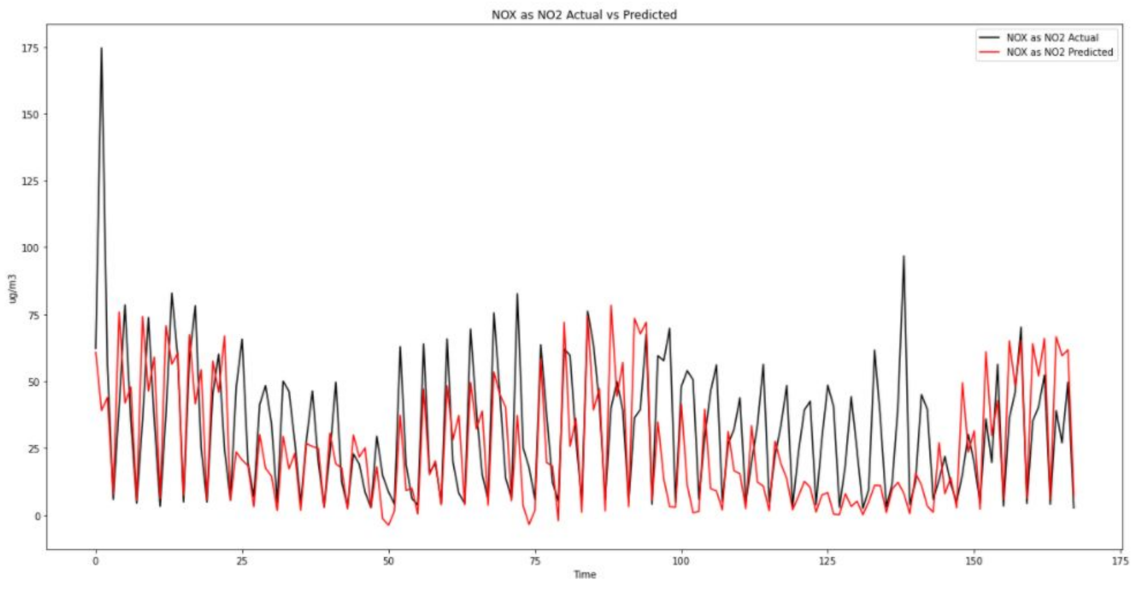
Experiment 2:

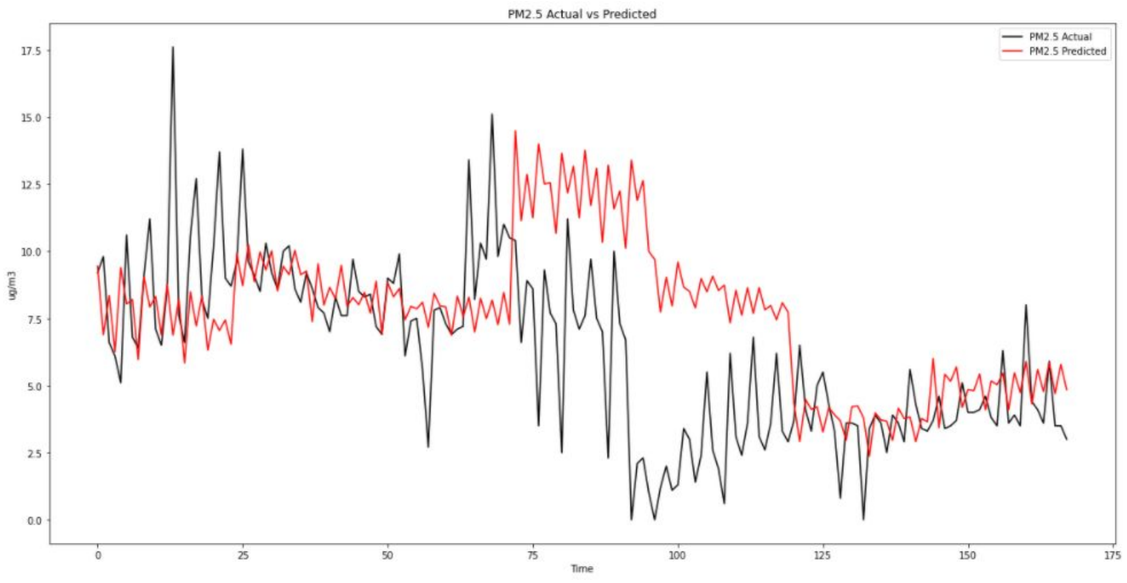
In this experiment only four features: PM10, NO2, NOX, PM2.5 are used. The previous 24 hours values are utilised to predict the next 24 hours of the PM10, NO2, NOX, PM2.5. Based on features availability across the different stations the following 4 stations were used for the experiments and they are #8779, #8780, #8781 and #18644. 2017 is used as Training data used, 2018 as validation and finally 2019 as Testing data. The data is normalized using Min Max Normalisation between range 0 and 1. All NaNs and negative values are replaced with zeros. Regarding the model structure it is sequential with two LSTM layers each with 24 LSTM units followed by a dropout layer with 0.2 dropout value. Along with adam as optimizer, Mean absolute error as loss, Early stopping with patience 10 is used to auto terminate training.

Results Obtained for Experiment:






NOX as NO2 Actual vs Predicted
NOX as NO2 Actual
NOX as NO2 Predicted
ug/m3
Time


PM2.5 Actual vs Predicted
PM2.5 Actual
PM2.5 Predicted
ug/m3
Time

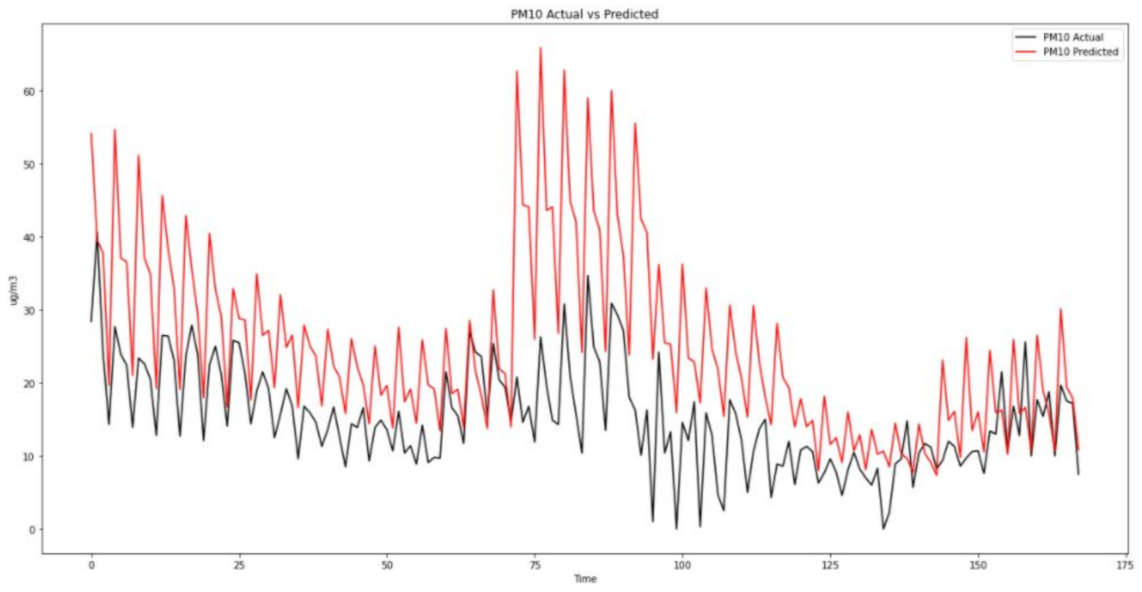


Fig 20. Train vs validation loss plot for experiment 2 and actual vs predicted value plot

Training time: 56.84 seconds

Epochs Run before convergence: 34 epochs

Metrics	Training Data:	Testing Data:
RMSE	40.3515	29.5073
MAE	4.0794	3.7689
SMAPE	0.4605	0.5190

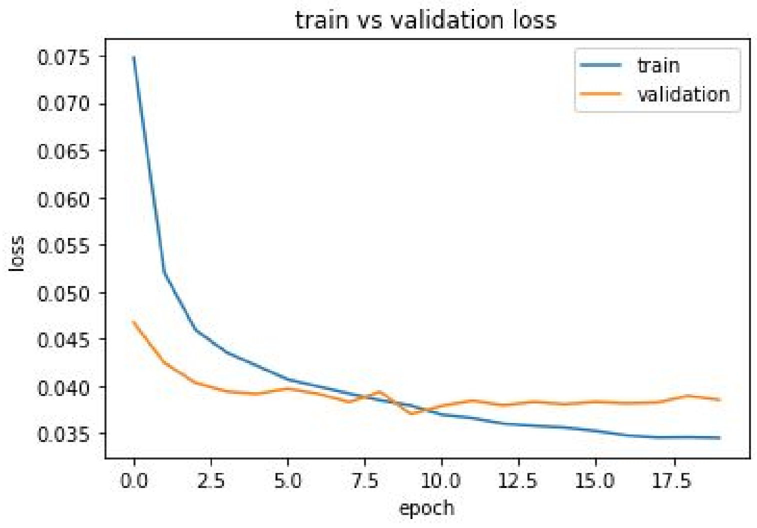
Table 3. Results from Experiment 2

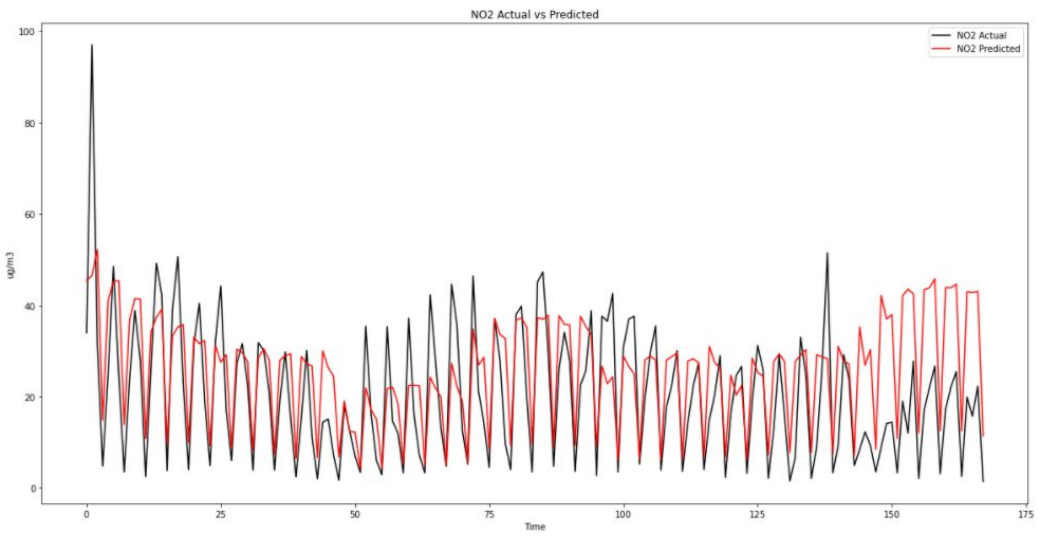
This experiment was carried out to verify how the performance changed when more features are predicted using the same input features. We expected the performance to drop and get worse results compared to previous experiments and that was exactly what was attained.

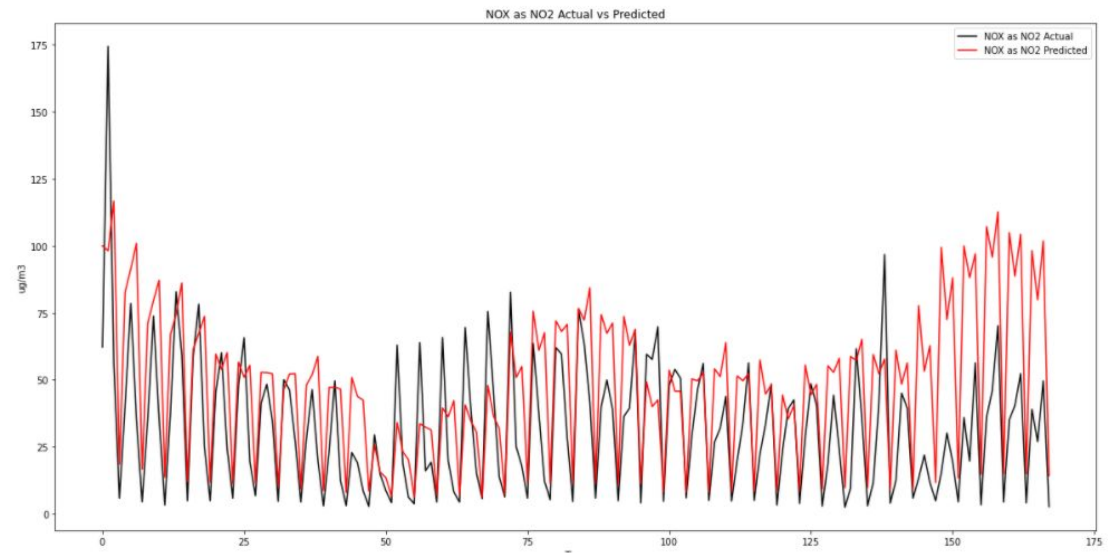
Experiment 3:

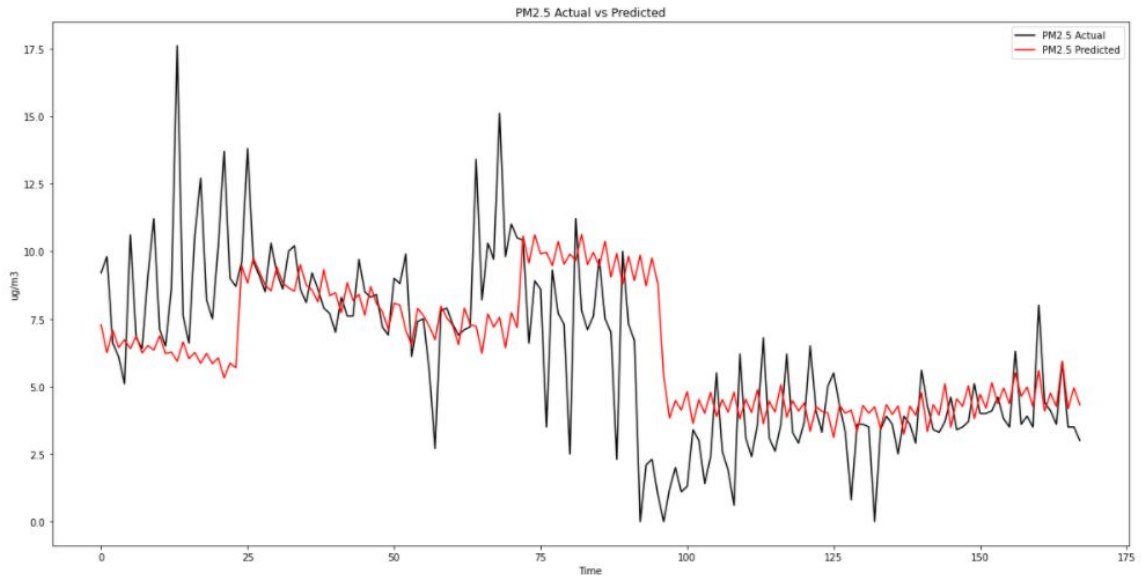
In this experiment we will use features selected using Feature Importance: PM10, NOX as NO2, NO2, PM2.5, Latitude, Air pressure, CO, Longitude, O3, Year, Amount of clouds, Black Carbon, Day Of Week, Day Of Month, Day Of Year and Hour . The previous 24 hours values are utilised to predict the next 24 hours of the PM10, NO2, NOX, PM2.5. Based on features availability across the different stations the following 4 stations were used for the experiments and they are #8779, #8780, #8781 and #18644. 2017 is used as Training data used, 2018 as validation and finally 2019 as Testing data. The data is normalized using Min Max Normalisation between range 0 and 1. All NaNs and negative values are replaced with zeros. Regarding the model structure it is sequential with two LSTM layers each with 24 LSTM units followed by a dropout layer with 0.2 dropout value. Along with adam as optimizer, Mean absolute error as loss, Early stopping with patience 10 is used to auto terminate training. We just want to verify whether there is an increase in performance.

Results Obtained for Experiment:






NOX as NO2 Actual vs Predicted
NOX as NO2 Actual
NOX as NO2 Predicted
ug/m3
0
25
50
75
100
125
150
175


PM2.5 Actual vs Predicted
PM2.5 Actual
PM2.5 Predicted
ug/m3
Time
0
25
50
75
100
125
150
175

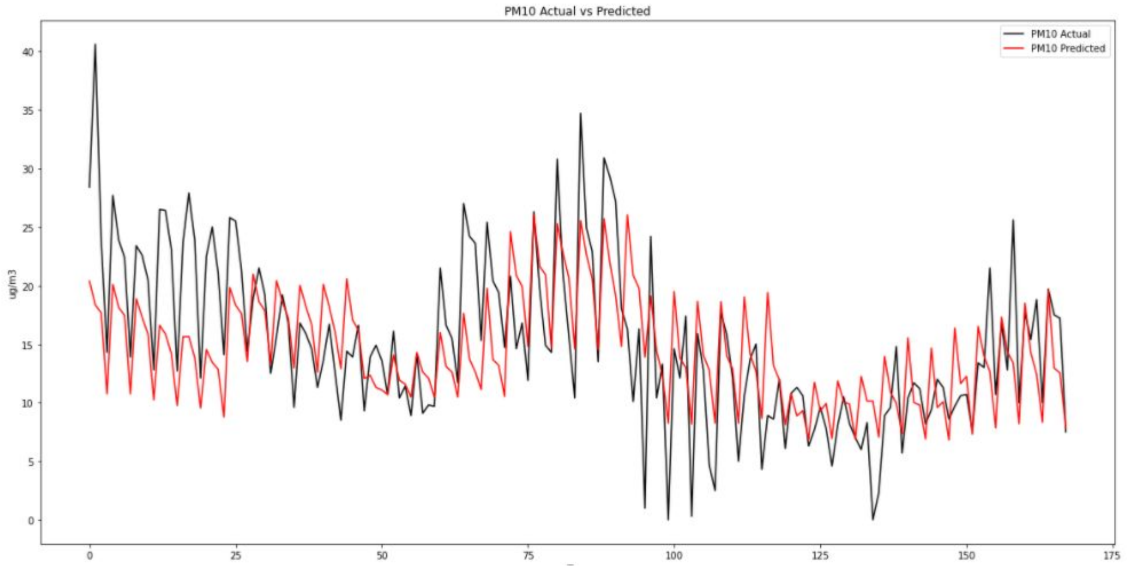

PM10 Actual vs Predicted
PM10 Actual
PM10 Predicted
ug/m3
0
25
50
75
100
125
150
175

Fig 21. Train vs validation loss plot for experiment 3and actual vs predicted value plot

Training time: 37.12 seconds

Epochs Run before convergence: 20 epochs

Metrics	Training Data:	Testing Data:
RMSE	30.3667	28.1107
MAE	3.6937	3.7631
SMAPE	0.4220	0.5029

Table 4. Results from Experiment 3

This experiment was carried out using the input features selected using SHAP based feature importance, we wanted to verify how the performance of the model is affected using more features. From the above experiments we can conclude that by using the inputs selected based on our feature selection methods we see slightly better results and importantly the model converged much more quickly around 20 seconds faster than the experiment 2 using 4 selected input features. Comparing the predicted vs actual values plot for all the pollutants especially for PM2.5 and PM10 we can notice that the experiment 3 model performed better over experiment 2.

Federated Learning

Federated learning[1](FL) is a machine learning distributed approach to training on a completely decentralized dataset that lives on inside different clients or devices such as mobile phones. What happens is that clients normally generate large datasets, instead of uploading these datasets to the server for further training of a model, the server sends its own model to these clients where a separate individual training process will occur on the separate data sets. For every specified number of epochs x, each model will contain their own set of weights. These weights will then be sent back into the server where an averaging process will aggregate all the weights into one global model which can be used to perform accurate predictions[1]. If the training is not done, the global model will be sent back onto the clients for further training. See Figure 1 for an Illustration of the process.

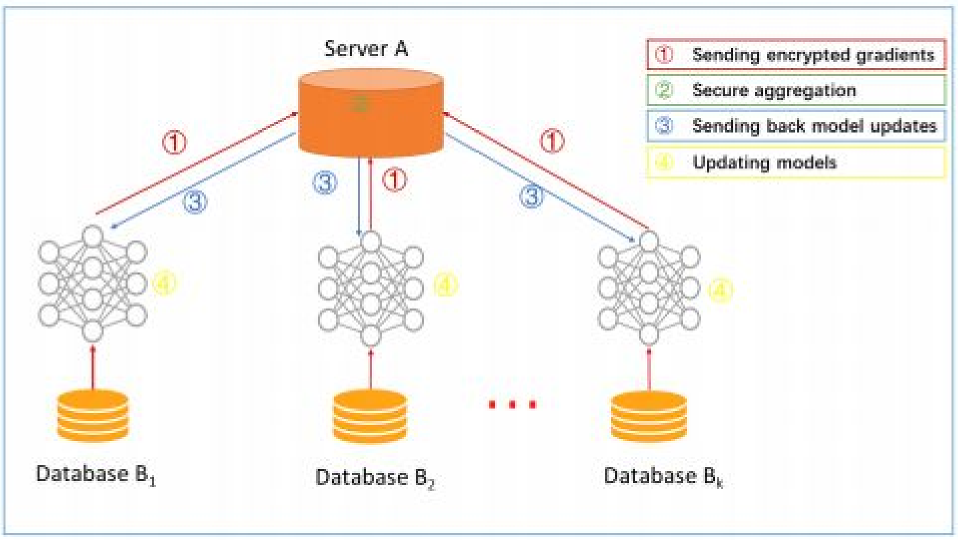


Fig 21. An illustration of a federated learning process from Ref.[2]

In the context of our project. A model *m* is initialized on the server with weights set to a random state, *m* is then distributed across *c* clients (represented as an air quality monitoring station) where they will perform the training on the model individually using their own dataset *d* which share the same feature space between all clients, this is known as horizontal federated learning (see Fig 21.). Then all the weights will be returned after a specified number of epochs to the server where the averaging of the weights happens in order to generate the global model. The global model can then be redistributed to the clients for further training. One iteration of this process qualifies as a training round *t*. The entire process can be encapsulated in the following equation[1] :

$$\sum_{c=1}^{c} \frac{d_c}{D} m_{t+1}^{c}$$

Where *D* is the sum of the entire datasets, m_{t+1}^{c} is the updated model of client *c*. This would lead us to a global model *M*.

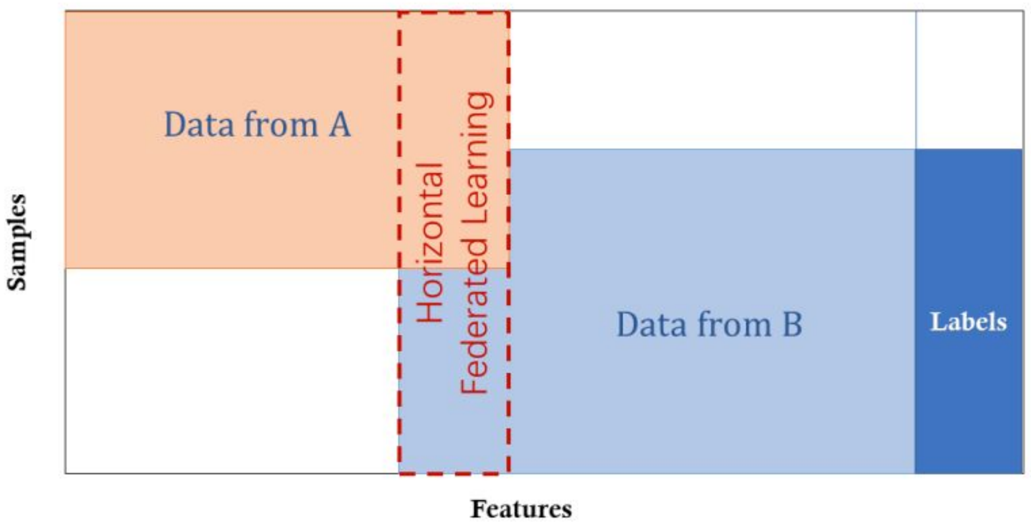


Fig 22. An illustration of data composition during a federated learning process from Ref.[2]

Strength

Privacy: One of the main strengths of a federated learning process is the preservation of privacy [1]. This quality stems from the fact that the server never gets a hold of the client's data thus giving the clients more control over their data. The model updates themselves are also ephemeral [1], all processing happens in memory and nothing (besides the weights which get put in a weight vector) gets stored. Keith Bonawitz et al. developed a protocol that will allow secure aggregation of weight vectors without learning each user's individual contribution[3]. However, the clients will still have to put trust into the server to not dig deeper into the weight updates so as to not infer any additional information.

Distributed data: When data is inherently distributed or also too big to store on a centralized server, it is more convenient to use distributed frameworks such as federated learning because of the convenience and increase in parallelized computation [4].

Weakness

Cost efficiency: Distributed machine learning systems such as federated learning suffer from requiring additional resources and computational power in order to achieve higher training speed which is something to aim for when performing training. In the case where the model created does not save money more than it did while training, it would probably be better to look into other alternatives [4].

Federated Learning Implementations

Tensorflow Federated

What is it?

Created by the Google Brain team, Tensorflow is an open-source library for numerical computation and large scale machine learning which mostly heavily incorporates deep learning methods and techniques. For our project we make use of the Federated implementation of Tensorflow known as Tensorflow Federated (TFF). TFF focuses on computations on decentralized data as well as federated deployments of deep learning and neural network models across multiple clients. As it stands TFF only allows simulated experiments of the federated settings which we will make full use of.[5]

Implementation

For the tensorflow federated implementation we chose the same 4 stations that we have chosen for the centralized model. Next we set up our data sets and clients, in our case we consider one dataset to belong to one client. Therefore this would leave us with 4 clients to run the models on and aggregate the weight from. To note that the clients will not run on a remote server as this is not a functionality yet supported by TFF as far as we can tell, therefore this whole scenario is simulated to act like it was.

FEDn

What is it?

FEDn is an open source modular framework for federated learning created by an Uppsala based company called Scaleout Systems [6]. FEDn allows both simulated federated learning as well as distributed. The architecture into three core parts:

- The client node which will be holding the data, they receive the code to be executed from combiners and requests for model updates and so on.

- The combiner node aggregates the weight updates received from the client nodes during a training round. One or more combiners can be set up, each with their own set of client networks.

- The reducer serves three roles, it carries out the global training strategy to the combiners. It maintains the global updates of the entire FEDn network. It mediates communication between client and combiner via a REST api.

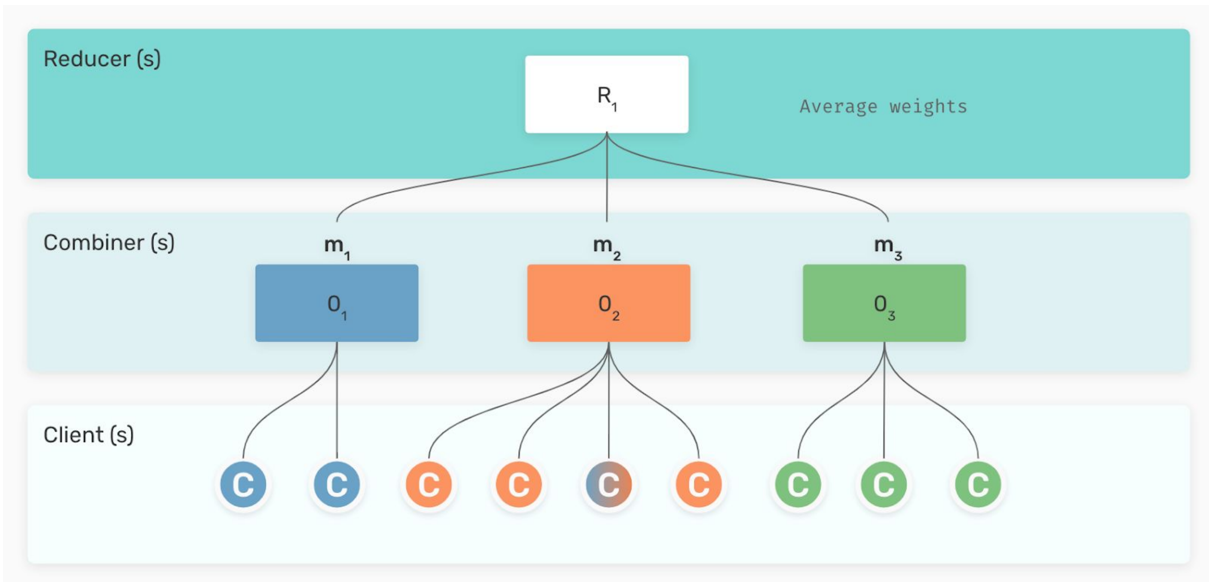


Fig 23. Figure showing the general architecture FEDn from ref. [6]

Implementation

For the FEDn implementation, we first set up our different python scripts that will be distributed by the combiner onto the clients. The scripts are the training and validation ones. These should be similar to the ones used inside the TFF implementation. A seed model is generated which will be uploaded onto the FEDn interface. This seed model will then be sent from the combiners onto the clients for training.

The preprocessing that we performed is simply replacing the NaN and negative values with zeros. Similarly to Experiment 1, Testing data is taken from all four different stations from a date range of 2019-01-01 00:00:00 to the last recorded timestamp, anything previous to that data is considered training. We distribute the training and testing data into different folders on four different clients which will then be read at runtime. Finally we launch the reducer, combiners and clients and start training. The FEDn implementation can be found in experiment 4.

We set up a scenarios of experiments as follows:

CPU	AMD Ryzen 7 3700X
RAM	16 GB
Tensorflow Version	2.3.0

Table: Specification of system and libraries used

Note: Experiment 1-4 were implemented with TFF and experiment 5 with FEDn

Experiment 1:

We use only four features: PM10, NO2, NOX, PM2.5. Using the previous 5 time steps to predict the PM10s next hour value. Testing data is taken from all four different stations from a date range of 2019-01-01 00:00:00 to the last recorded timestamp, anything previous to that data is considered training. The data is unnormalized. We replace the NaNs and negative values with 0s.

The model we set up contains 5 lstm units and followed by a dense layer. We use SGD as an optimizer and mean absolute error as a loss function. We decided to run the model for 10 rounds as it seemed sensible not to overfit on our training data. For the aggregation, we used TFFs internal averaging process.

Experiment 2:

Similar to the previous experiment, however we try to run the model for 100 rounds to see if the model overfits or further improves.

Experiment 3:

We use the same four features. However we try to use 24 previous timesteps in order to predict the 4 features for the next 24 time steps. For the model we use 24 LSTM units. The number of training rounds chosen is 10. The rest is similar to the previous setups.

Experiment 4:

This experiment was carried out using input features selected based on correlation with target feature PM10, the selected input features are NO2, NOX as NO2, PM2.5, PM10, CO, Black Carbon, Air Temperature, Classification, WeekType and HourType. Based on features availability across the different stations the following 4 stations were used for the experiments and they are #8779, #8780, #8781 and #18644. 2016, 2017 and 2018 is used as Training data used and finally 2019 as Testing data. The data is normalized using Min Max Normalisation between range 0 and 1. All NaNs and negative values are replaced with zeros. Regarding the model structure it is sequential with single LSTM layers with 24 LSTM units followed by a dropout layer with 0.2 dropout value. Along with Adam as optimizer, Mean squared error as loss function, Early stopping was not supported, so the model was trained for 10 iterations. The idea was to validate if adding correlated features will improve performance.

Experiment 5:

For the following experiment, we use the FEDn implementation of federated learning. We try to benchmark this experiment to experiment 1. This means that we try to predict the hourly PM10 using the previous 5 time steps of the 4 features (NO2, PM10, NOX, PM2.5). Our model is also similar with 24 lstm units followed by a dense layer. SGD is used as the optimizer and means absolute error as a loss function. We run FEDn for 10 training rounds. We run with 2 combiners and 4 client nodes following the data-center example provided. The results can be observed in table 4 as well as other extra information that FEDn provides. The aggregation is provided at the combiner level and happens on a built in averaging process provided by FEDn.

Results

Note: These values are all normalized.

Training MAE:	Testing MAE:	Training time:
0.02046	0.02114	69.94 Seconds

Table 5. Results from Experiment 1

Training MAE:	Testing MAE:	Training time:
0.01596	0.01700	631.88 Seconds

Table 6. Results from Experiment 2

Training MAE:	Testing MAE:	Training time:
0.05406	0.05187	137.87

Table 7. Results from Experiment 3

Training MAE:	Testing MAE:	Training time:
0.01411	0.01337	N/A

Table 7. Results from Experiment 4

Training MAE:	Testing MAE:	Training time:
0.01301	0.01521	872.76

Table 8. Results from Experiment 5

Evaluation

To benchmark the federated experiments to the centralized one, we perform the same procedure and compare our results. For experiment 1 the testing MAE is 0.02114, however it seems to underperform compared to the centralized one which had an MAE of 0.0168. This can be due to the aggregation procedure which doesn't perform as optimally as having the model trained on the entire dataset as in the centralized one.

For experiment 2 we try to run the federated model for 100 rounds, to see if it will eventually converge. We notice that the model takes a very long time to converge and the results improve over the 10 rounds version of the model. However, if we compare the training results 0.01596 and the testing result 0.017. We notice that the model is starting to overfit. In conclusion, trial and error should show the optimal number of training rounds to choose.

In experiment 3, we try to predict 24 hours into the future the 4 different chosen features to see how our model performs when we have to predict more values into the future. We notice that the training time takes longer as we are including more data to predict, it almost doubles in time compared to predicting 1 feature for the next one hour. Finally we obtain a high MAE of 0.05406 in comparison to experiment 1 0.02114.

In experiment 4, we use more features to perform hourly predictions to see if our model improves. This results in the model significantly improving with an MAE of 0.01337 compared to experiment 1 where we used only 4 features to predict hourly PM10 and had an MAE of 0.02114. This is due to the features added that have high correlation with PM10, thus leading the model to perform better predictions since it has more information.

For experiment 5, another framework for decentralized machine learning was used (FEDn) in order to see if there is improvement over the tensorflow federated library implementation in experiment 1. We notice that it produces a much better testing result of 0.01524 which is lower than experiment 1's 0.02114. To be noted, the training time greatly increased to almost 15 minutes. This can be due to the framework's architecture and the communication between different elements of the fedN network. Whereas in TFF everything seems to exist on the same level and there is no need for network communication between different components.

Quantization of the model

In order to test how our model performs on embedded devices, we opted to run experiments by quantizing our model. We chose two different ways of doing so, however we failed to do so because of the lack of tools and the scope being outside of our knowledge

STM32CubeMX

STM32CubeMX is a graphical tool, created by STMicroelectronics, which allows configuration of STM32 embedded devices and micro-processors. Thus allowing us to generate C code for the device's initialization. ST provides AI oriented solutions through the X-CUBE-AI module which will allow us to run our own .h5 pretrained model on said microprocessors. [7], [8].

The process is as follows: We first run our centralized model and then save it into an .h5 format. Following that, we launch the GUI which allows us to select the microprocessor we would like to work with, a lot of the configuration is automatically handled by the GUI tool. Choices of the microprocessor can be selected according to their prices, memory capacity and computing capabilities. After having made our selection, we provide our pretrained model and then generate the corresponding C code automatically through the interface.

Unfortunately, due to our lack of expertise in embedded devices and setups as well as the lack of a physical microprocessor to run our experiment with, we couldn't proceed further to run our experimentation on such devices. We did not manage to find a way to run a simulation using the GUI, however this could be because of our lack of orientation around the tool and more time will be needed to further discover these things.

Tensorflow Lite

Tensorflow lite is a tool provided by the developers of Tensorflow that allows converting models into lite format to run on embedded devices, IoT devices and mobile phones etc. It enables performance of machine learning methods on the edge of the network instead of having an intermediary (server) with data being sent back and forth.[9]

Conversion of tensor flow model to lite version is straightforward step of passing model into tf lite converter function but in order to predict target feature use of an intermediate interpreter is required. The input features using Interpreter needs to be converted as a tensor and then only be used to predict target feature again through the means of Interpreter.

Experiment 1:

In this experiment only four features: PM10, NO2, NOX, PM2.5 are used. The previous 24 hours values are utilised to predict the next 24 hours of the PM10. Based on features availability across the different stations the following 4 stations were used for the experiments and they are #8779, #8780, #8781 and #18644. 2017 is used as Training data used, 2018 as validation and finally 2019 as Testing data. The data is normalized using Min Max Normalisation between range 0 and 1. All NaNs and negative values are replaced with zeros. Regarding the model structure it is sequential with two LSTM layers each with 24 LSTM units followed by a dropout layer with 0.2 dropout value. Along with adam as optimizer, Mean absolute error as loss, Early stopping with patience 10 is used to auto terminate training. Only difference is that the model is converted to lite form using TF-lite.

Results Obtained for Experiment:

Metrics	Testing Data	Testing Data using Tensorflow Lite
RMSE	9.473020043472582	9.473020064550532
MAE	2.43200867558147	2.432008640919287
SMAPE	0.39844135519908547	0.398441338762834

Table 9. Results from Experiment 1

Experiment 2:

In this experiment only four features: PM10, NO2, NOX, PM2.5 are used. The previous 24 hours values are utilised to predict the next 24 hours of the PM10, NO2, NOX, PM2.5. Based on features availability across the different stations the following 4 stations were used for the experiments and they are #8779, #8780, #8781 and #18644. 2017 is used as Training data used, 2018 as validation and finally 2019 as Testing data. The data is normalized using Min Max Normalisation between range 0 and 1. All NaNs and negative values are replaced with zeros. Regarding the model structure it is sequential with two LSTM layers each with 24 LSTM units followed by a dropout layer with 0.2 dropout value. Along with adam as optimizer, Mean absolute error as loss, Early stopping with patience 10 is used to auto terminate training. Only difference is that the model is converted to lite form using TF-lite.

Results Obtained for Experiment:

Metrics	Testing Data	Testing Data using Tensorflow Lite
RMSE	29.507331084705083	29.507322493061057
MAE	3.7689495546475817	3.7689490052370203
SMAPE	0.5190059631434402	0.5190059025249859

Table 10. Results from Experiment 2

Experiment 3:

In this experiment we will use features selected using Feature Importance: PM10, NOX as NO2, NO2, PM2.5, Latitude, Air pressure, CO, Longitude, O3, Year, Amount of clouds, Black Carbon, Day Of Week, Day Of Month, Day Of Year and Hour . The previous 24 hours values are utilised to predict the next 24 hours of the PM10, NO2, NOX, PM2.5. Based on features availability across the different stations the following 4 stations were used for the experiments and they are #8779, #8780, #8781 and #18644. 2017 is used as Training data used, 2018 as validation and finally 2019 as Testing data. The data is normalized using Min Max Normalisation between range 0 and 1. All NaNs and negative values are replaced with zeros. Regarding the model structure it is sequential with two LSTM layers each with 24 LSTM units followed by a dropout layer with 0.2 dropout value. Along with adam as optimizer, Mean absolute error as loss, Early stopping with patience 10 is used to auto terminate training. We just want to verify whether there is an increase in performance. Only difference is that the model is converted to lite form using TF-lite.

Results Obtained for Experiment:

Metrics	Testing Data	Testing Data using Tensorflow Lite
RMSE	28.11073612437721	28.110728319463
MAE	3.763110016365047	3.763109483150674
SMAPE	0.5029585802343842	0.5029557702475171

Table 11. Results from Experiment 3

The above experiments are the same as the one carried out using vanilla TensorFlow models. The result obtained using TensorFlow lite is almost the same with negligible difference in performance. The only noticeable difference identified between these models is the duration taken to predict the target feature, for prediction target feature with 1460 samples vanilla TensorFlow model needed 0.31307 seconds will TensorFlow lite needed 3.1112 seconds. This drastic difference is probably due to the limitation that it could only predict target features at the phase of one sample at a time.

Turn Into a Product

Why we decided to turn it into a product

In this project, not all of the team members had expertise in machine learning. This was a good opportunity to learn ML, but team size is huge. We decided to move the workforce to a different part which was not mandatory for the project.

While turning into a product we used a general production flow.

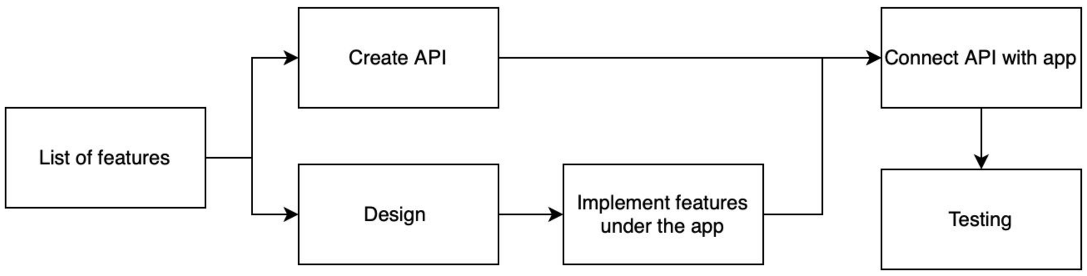


Table 12. General production flow.

Features of the app as follows:

- Create an API for fetch and store data from stations.
- List several stations around Stockholm inside of the app.
- List data of stations inside of the app.
- Managed to run a model which converted to **Tensorflow Lite** inside of the app.
- Display predicted data inside of the app

For the design part, we only designed the database. When creating the app, we decided to make the app with flutter. We planned to provide API with firebase cloud functions.

As the first step, we planned to build an API. Firebase was a good candidate to create our API since Firebase already hosts and manages our JavaScript functions which are responsible for parsing or communicating with third party APIs for retrieving hourly sensor data. So far we used data between 1959 - 2019. Obviously, we cannot use this data for our app that's why we need to find another data source. We checked how other air quality web pages find data. After a short search we found 52 Degree North Sensor Observation Service API. The first problem we faced with 52 Degree North SOS API's, is that their REST API documentation gives "404 Not Found" error message. We tried to go through with their developer guide book but the results were not satisfying since we couldn't find any way to get raw data. Moreover, we found another documentation from GitHub but this also gives problems during installation steps. We, therefore, decided to continue this API idea as future work.

API was designed for real-time data fetching. This basic API design also contains our database communication design.

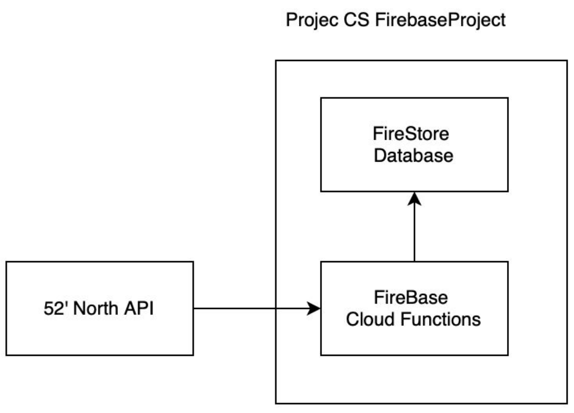


Table 13. Basic API Design

As a last resort, we decided to fill the database by hand. For database design, we thought about two different collections. The first collection is trained data and the second one is hourly data to compare and draw graphs on the app.

For creating mobile app we used the flutter framework which is created by google. We initially thought it was super easy but we significantly wasted our time with the firebase custom_ml library to run into the flutter app which we created. We decided to give up on using that library and putting a model inside of the app. This restricts us to update models remotely. After several tests with real data which entered the firebase by hand. We got similar results when running in other environments.

Conclusions

We have experimented with multiple techniques regarding preprocessing and feature engineering to extract the most optimal features. Surprisingly the results showed that weather features don't contribute as much as days of the weeks, months and year as well as other pollutants. Imputation techniques such as interpolation and mean propagation were used to analyze how closely the imputed data will resemble the actual data, end results showed that we couldn't provide an imputation method that can accurately imitate the actual missing values. We made use of imputing missing values with zeros as a replacement to dismissing NaN values since the latter approach would connect different distanced dates of the time series into each other which would ruin the successivity of the data.

Some experiments for a more centralized model were ran, where we tried both hourly prediction of one pollutant (PM10) and 4 pollutants for the next 24 hours (PM10, PM2.5, NOX, NO2), the results of the latter were worse as we had expected. We tried to benchmark our federated implementation of the same experiments we did with the centralized model. The federated implementation performed slightly worse than the centralized one, this was to be expected as all the data was distributed rather than having a single source of truth as in the centralized model. Experiments for the federated model displays that increased training rounds have improved the model's ability to perform prediction however it also took a long time to converge. Quantization of the model did not show a very large degrading of our model's performance. Running the federated model in a pseudo-distributed environment such as in FEDn showed promising results. However, an actual integration inside of a fully distributed environment still needs to be made as well as further experimentation with time series data inside of a distributed machine learning models environment to see if we can fully seize the potential of such predictive models.

Appendix A: Installation instructions

We used Docker to install all the required libraries for federated machine learning models. Docker is a tool that is designed to make it easier to create, deploy and run applications by using containers. We can re-use the image of the docker container to initialize it as many times as we want to, by using docker compose, in which we can scale up and scale down the docker containers. We used flask to run the model. Flask is a web framework that acts as a web server that will always listen to a defined port which runs and returns a response upon receiving a HTTP request. Basically we place the model into a flask application. In this project, we installed docker on the machine locally and then built the container with docker image. On running the federated model, we got the results quite fast then running without the docker file. We then uploaded the container image on the docker hub, we can say docker hub acts like a GitHub where we pull and push our code, moreover, it is a repository for the users to store data like container images. By creating and logging accounts then pushing through IP commands we can make the dockerhub active for the container. We can use the curl command to show the results on the browser.

The github repository link is https://github.com/RaheelTheDeveloper/damp.git where installation, execution and development instructions can be found.

Appendix B: Future work

As with almost all the projects, there still remains improvements that can be done. However, due to lack of time, it was not applicable. One of the interesting things to do in the future is to use a different and more powerful model such as the Transformer model. Transformer model was first proposed in a paper called *Attention Is All You Need* [16]. This model can handle the long-term dependencies, which is an obstacle in an LSTM model. Therefore, the team is very interested to implement it and see the difference of outcomes of the two models.

Non-simulated settings can be used to implement the federated learning part of the project to see whether the results might differ if the data was distributed in a cluster as in this project the whole experiment was done on a single machine. Furthermore, trying out various weight averaging algorithms and implementations of federated learning can be experimented to get a better insight of what can be improved and specifically what would work the best when it comes to dealing with the time-series data.

For the app, because of the lack of time, we couldn't find other working APIs to create our own in order to fetch real-time hourly data, so we made a design on how that could be made so we can have it in mind for a future where implementing it could be a very valuable option.

Appendix C: Tools

TensorFlow

Tensorflow is an open-source library for numerical computation and large scale machine learning which mostly heavily incorporates deep learning methods and techniques.

TensorFlow Lite

A lightweight machine learning framework for deploying models on Android, iOS, and embedded systems like Raspberry Pi and Edge TPUs.

TensorFlow Federated

TensorFlow Federated(TFF) is an open source machine learning framework or other computations on decentralized data.

FEDn

FEDn is an open source modular framework for federated learning created by an Uppsala based company called Scaleout Systems [6].

Flutter

Flutter is an API for creating apps. Flutter uses the Dart programming language.

Firebase

Firebase is a platform developed by Google with the purpose of mobile and web application development. Firebase contains cloud functions, NOSQL database and pretrained ML models and App analytics.

Trello

Trello is a task management app.

We used already created agile templates. Which contains. Backlog, Sprint, Doing, Done, Deployed titled table. Every team member is responsible with creating, moving and testing the tasks as trello cards.

Git

Git is a version control system. A software which keeps track of the changes that has been made on a file.

Docker

Docker is an open source containerization platform.which enables developers to package applications into containers. It allows containers to be portable to any system running a Linux or Windows operating system.

References

[1] A. Hard *et al.*, “Federated Learning for Mobile Keyboard Prediction,” Nov. 08, 2018. p.48

[2] Q. Yang, Y. Liu, T. Chen, and Y. Tong, “Federated Machine Learning,” *ACM Transactions on Intelligent Systems and Technology*, vol. 10, no. 2. pp. 1–19, 2019, doi: 10.1145/3298981. p.48

[3] Keith Bonawitz, Vladimir Ivanov, Benjamin Kreuter, Antonio Marcedone, Hugh Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, Karn Seth, “Practical Secure Aggregation for Privacy-Preserving Machine Learning,” *CCS ’17: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pp. 1175–1191, Oct. 2017. p.49

[4] J. Verbraeken, M. Wolting, J. Katzy, J. Kloppenburg, T. Verbelen, and J. S. Rellermeyer, “A Survey on Distributed Machine Learning,” 2020, Accessed: Dec. 04, 2020. [Online]. Available: http://dl.acm.org/citation.cfm?id=3377454. p.50

[5] “TensorFlow Federated.” https://www.tensorflow.org/federated (accessed Dec. 29, 2020).

[6] scaleoutsystems, “scaleoutsystems/fedn.” https://github.com/scaleoutsystems/fedn (accessed Dec. 29, 2020).

[7] “STM32CubeMX.” https://www.st.com/en/development-tools/stm32cubemx.html (accessed Dec. 30, 2020).

[8] “X-CUBE-AI.” https://www.st.com/en/embedded-software/x-cube-ai.html (accessed Dec. 30, 2020).

[9] “TensorFlow Lite guide.” https://www.tensorflow.org/lite/guide (accessed Jan. 08, 2021).

[10] Cerda, Patricio, Gaël Varoquaux, and Balázs Kégl. "Similarity encoding for learning with dirty categorical variables." *Machine Learning* 107, no. 8-10 (2018): 1477-1494.

[11] Van Zoest, V. M., A. Stein, and G. Hoek. "Outlier detection in urban air quality sensor networks." *Water, Air, & Soil Pollution* 229, no. 4 (2018): 111. p.23

[12] Kut, Alp, and Derya Birant. "Spatio-temporal outlier detection in large databases." *Journal of computing and information technology* 14, no. 4 (2006): 291-297.

[13] García, Salvador, Julián Luengo, and Francisco Herrera. *Data preprocessing in data mining*. Cham, Switzerland: Springer International Publishing, 2015.

[14] Zhang, Zhongheng. "Missing data imputation: focusing on single imputation." *Annals of translational medicine* 4, no. 1 (2016).

[15] United States Environmental Protection Agency “*Managing Air Quality - Ambient Air Monitoring*” from https://www.epa.gov/air-quality-management-process/managing-air-quality-ambient-air-monitoring

[16] Ashish Vaswani and Noam Shazeer and Niki Parmar and Jakob Uszkoreit and Llion Jones and Aidan N. Gomez and Lukasz Kaiser and Illia Polosukhin (2017). Attention Is All You

NeedCoRR, abs/1706.03762.

[17] Avila, J. (2017). Scikit-Learn Cookbook - Second Edition: Over 80 Recipes for Machine Learning in Python with Scikit-Learn. Packt Publishing.

[18] Strobl, C., Boulesteix, AL., Zeileis, A. et al. Bias in random forest variable importance measures: Illustrations, sources and a solution. BMC Bioinformatics 8, 25 (2007). https://doi.org/10.1186/1471-2105-8-25

[19] "4.2. Permutation feature importance — scikit-learn 1.0.dev0 documentation", Scikit-learn.org, 2021. [Online]. Available: https://scikit-learn.org/dev/modules/permutation_importance.html. [Accessed: 13- Jan- 2021].

[20] "Welcome to the SHAP documentation — SHAP latest documentation", Shap.readthedocs.io, 2021. [Online]. Available: https://shap.readthedocs.io/en/latest/index.html. [Accessed: 13- Jan- 2021].

[21] D. Smith, "PM10: How Do Coarse Particles (Particulate Matter) Affect Air Quality?", Learn.kaiterra.com, 2021. [Online]. Available: https://learn.kaiterra.com/en/air-academy/pm10-particulate-matter-pollutes-air-quality. [Accessed: 13- Jan- 2021].

[22] Jinsung Yoon and James Jordon and Mihaela van der Schaar (2018). GAIN: Missing Data Imputation using Generative Adversarial NetsCoRR, abs/1806.02920. (edited)

[23] Van Zoest et al. (2018) "*Outlier Detection in Urban Air Quality Sensor Networks. Water, Air, & Soil Pollution*" 229 (4): 111 https://link.springer.com/article/10.1007/s11270-018-3756-7

[24] (2008) Time Series. In: The Concise Encyclopedia of Statistics. Springer, New York, NY. https://doi.org/10.1007/978-0-387-32833-1_401(2008) Time Series. In: The Concise Encyclopedia of Statistics. Springer, New York, NY. https://doi.org/10.1007/978-0-387-32833-1_401

[25] Box, George EP, Steven C. Hillmer, and George C. Tiao. "Analysis and modeling of seasonal time series." *Seasonal analysis of economic time series*. NBER, 1978. 309-344.