

POLLUTION PREDICTION SYSTEM

Rohit Gupta | Raj Doshi | Tushita Gupta
Team 21

Abstract

Pollution prediction is the practice of predicting the value of air pollutants for a given location based on different weather parameters. Pollution prediction is done by gathering data about the current state of the atmosphere. Pollution information is essential in every facet of life like agriculture, tourism, airport system, mining industry, and power generation. Hence, predicting pollution adequately is very necessary. Here we are using different supervised machine learning techniques to process datasets that can learn and make predictions more effectively based on basic weather information. The pollution prediction system is modelled as a regression problem that is for different weather parameters, finding a continuous value of pollutants over time. A lot of factors affecting the pollutants were hidden, which were extracted to improve the prediction of pollutants.

1. Statement of contribution:

Raj Doshi: Performed EDA, cleaned data and built Random Forest Regressor implementation and optimized the models built. Prepared presentation and report.

Rohit Gupta: Performed EDA, cleaned data and built Neural Network model implementation and optimized the models built. Prepared presentation and report.

Tushita Gupta: Performed EDA, cleaned data and built Linear and Polynomial Regression models implementation and optimized the models built. Prepared presentation and report.

2. Introduction:

Air pollution has been a looming issue of the 21st century that has also significantly impacted the surrounding environment and societal health. Recently, previous studies have conducted extensive research on air pollution and air quality monitoring. Despite this, the fields of air pollution and air quality monitoring remain plagued with unsolved problems. In this project, the Pollution Prediction System is proposed to perform air pollution prediction for outdoor sites for various pollution parameters. Here in this project, we are predicting the values of air pollution measurements over time, based on basic weather information (temperature and humidity) and the input values of 5 sensors. The three target values to predict are carbon-monoxide, benzene, and nitrogen oxides.

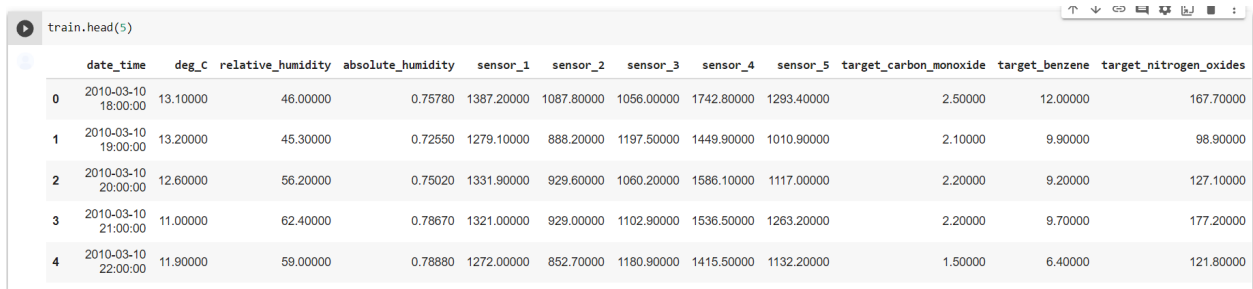
3. Feature Extraction and Preprocessing:

3.1 Dataset

The dataset used in this project provides the value of pollutants based on basic weather information (temperature and humidity) and the input values of 5 sensors. It is derived from Kaggle: <https://www.kaggle.com/c/tabular-playground-series-jul-2021/data>. There are 7111 records in the dataset. The data is cleaned and split into a training set (60%), a validation set (20%), a testing set (20%). There are a total of twelve features in the dataset.

The features are described below:

01. Date - Date and time of the record measurement
02. Deg_C - Temperature in Celcius
03. Relative Humidity - Relative humidity is the percentage of the current absolute humidity to the highest possible absolute humidity
04. Absolute Humidity - Absolute humidity is the mass of water vapour divided by the mass of dry air in a volume of air at a given temperature.
05. Sensor_1 - (tin oxide) hourly averaged sensor response (nominally CO targeted)
06. Sensor_2 - (titania) hourly averaged sensor response (nominally NMHC targeted)
07. Sensor_3 - (tungsten oxide) hourly averaged sensor response (nominally NOx targeted)
08. Sensor_4 - (tungsten oxide) hourly averaged sensor response (nominally NO2 targeted)
09. Sensor_5 - (indium oxide) hourly averaged sensor response (nominally O3 targeted)
10. Target_Carbon_Monooxide - Amount of Carbon Monoxide in air, which we need to predict
11. Target_Benzene - Amount of Benzene in air, which we need to predict
12. Target_nitrogen_oxide - Amount of Nitrogen Oxide in air, which we need to predict



	date_time	deg_C	relative_humidity	absolute_humidity	sensor_1	sensor_2	sensor_3	sensor_4	sensor_5	target_carbon_monoxide	target_benzene	target_nitrogen_oxides
0	2010-03-10 18:00:00	13.10000	46.00000	0.75780	1387.20000	1087.80000	1056.00000	1742.80000	1293.40000	2.50000	12.00000	167.70000
1	2010-03-10 19:00:00	13.20000	45.30000	0.72550	1279.10000	888.20000	1197.50000	1449.90000	1010.90000	2.10000	9.90000	98.90000
2	2010-03-10 20:00:00	12.60000	56.20000	0.75020	1331.90000	929.60000	1060.20000	1586.10000	1117.00000	2.20000	9.20000	127.10000
3	2010-03-10 21:00:00	11.00000	62.40000	0.78670	1321.00000	929.00000	1102.90000	1536.50000	1263.20000	2.20000	9.70000	177.20000
4	2010-03-10 22:00:00	11.90000	59.00000	0.78880	1272.00000	852.70000	1180.90000	1415.50000	1132.20000	1.50000	6.40000	121.80000

Figure 1: A table representing features in the dataset

3.2 EDA

EDA helps us in analyzing summary tables, and hypothesis testing to provide summary-level insight into a dataset, uncover underlying patterns and structures in the data, identify outliers, missing data, class balance, and other data-related issues, relate the available data to the business opportunity. The below *figure 2* gives us a basic insight into the distribution of the features in the dataset

```
[ ] train.describe()
```

	deg_C	relative_humidity	absolute_humidity	sensor_1	sensor_2	sensor_3	sensor_4	sensor_5	target_carbon_monoxide	target_benzene	target_nitrogen_oxides
count	7111.00000	7111.00000	7111.00000	7111.00000	7111.00000	7111.00000	7111.00000	7111.00000	7111.00000	7111.00000	7111.00000
mean	20.87803	47.56100	1.11031	1091.57210	938.06497	883.90330	1513.23835	998.33556	2.08622	10.23708	204.06678
std	7.93792	17.39873	0.39895	218.53755	281.97899	310.45636	350.18031	381.53770	1.44711	7.69443	193.92772
min	1.30000	8.90000	0.19880	620.30000	364.00000	310.60000	552.90000	242.70000	0.10000	0.10000	1.90000
25%	14.90000	33.70000	0.85590	930.25000	734.90000	681.05000	1320.35000	722.85000	1.00000	4.50000	76.45000
50%	20.70000	47.30000	1.08350	1060.50000	914.20000	827.80000	1513.10000	928.70000	1.70000	8.50000	141.00000
75%	25.80000	60.80000	1.40415	1215.80000	1124.10000	1008.85000	1720.40000	1224.70000	2.80000	14.20000	260.00000
max	46.10000	90.80000	2.23100	2088.30000	2302.60000	2567.40000	2913.80000	2594.60000	12.50000	63.70000	1472.30000

Figure 2: Basic statistics of the training dataset

The below figure 3 gives us an insight into the null values in the dataset.

```
[ ] data.isnull().sum()

date_time          0
deg_C              0
relative_humidity  0
absolute_humidity  0
sensor_1           0
sensor_2           0
sensor_3           0
sensor_4           0
sensor_5           0
target_carbon_monoxide  0
target_benzene     0
target_nitrogen_oxides 0
dtype: int64
```

Figure 3: Null values in the training dataset

Below figure 4 gives the correlation between all the features in the dataset.

<matplotlib.axes._subplots.AxesSubplot at 0x7f2e78d77750>

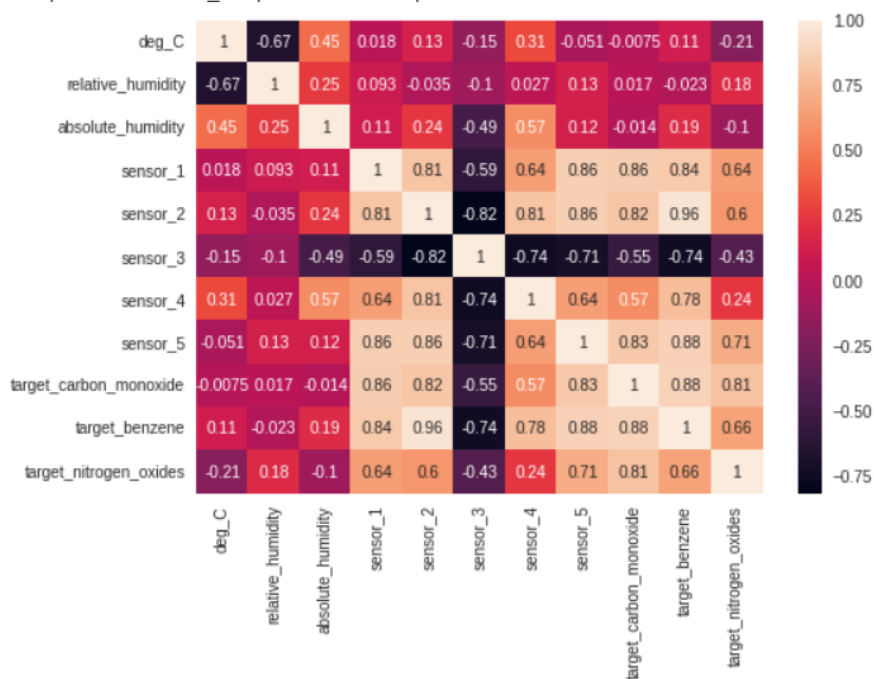
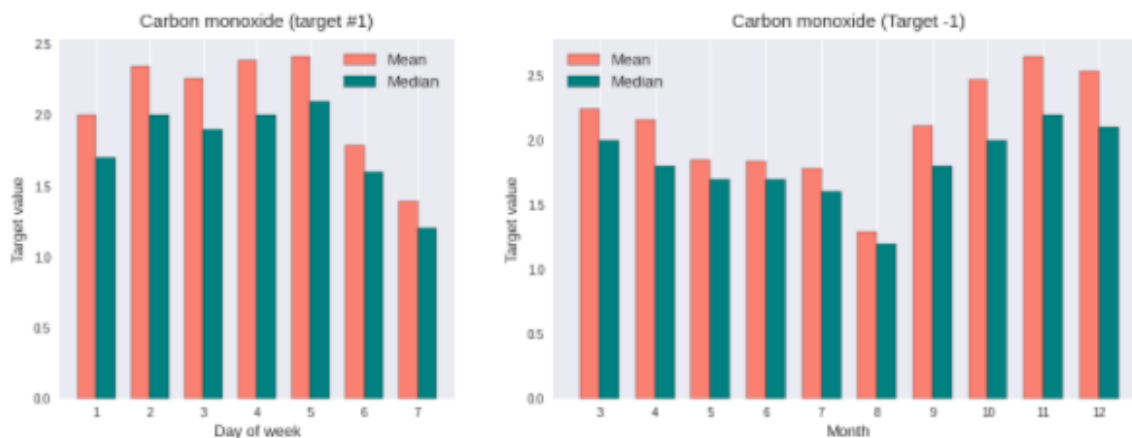


Figure 4: Correlation between the features

The below figure 5 gives the relation of pollutants and date(each day and month of the year). Here, we observed that the pollutants were high during weekdays as compared to weekends.



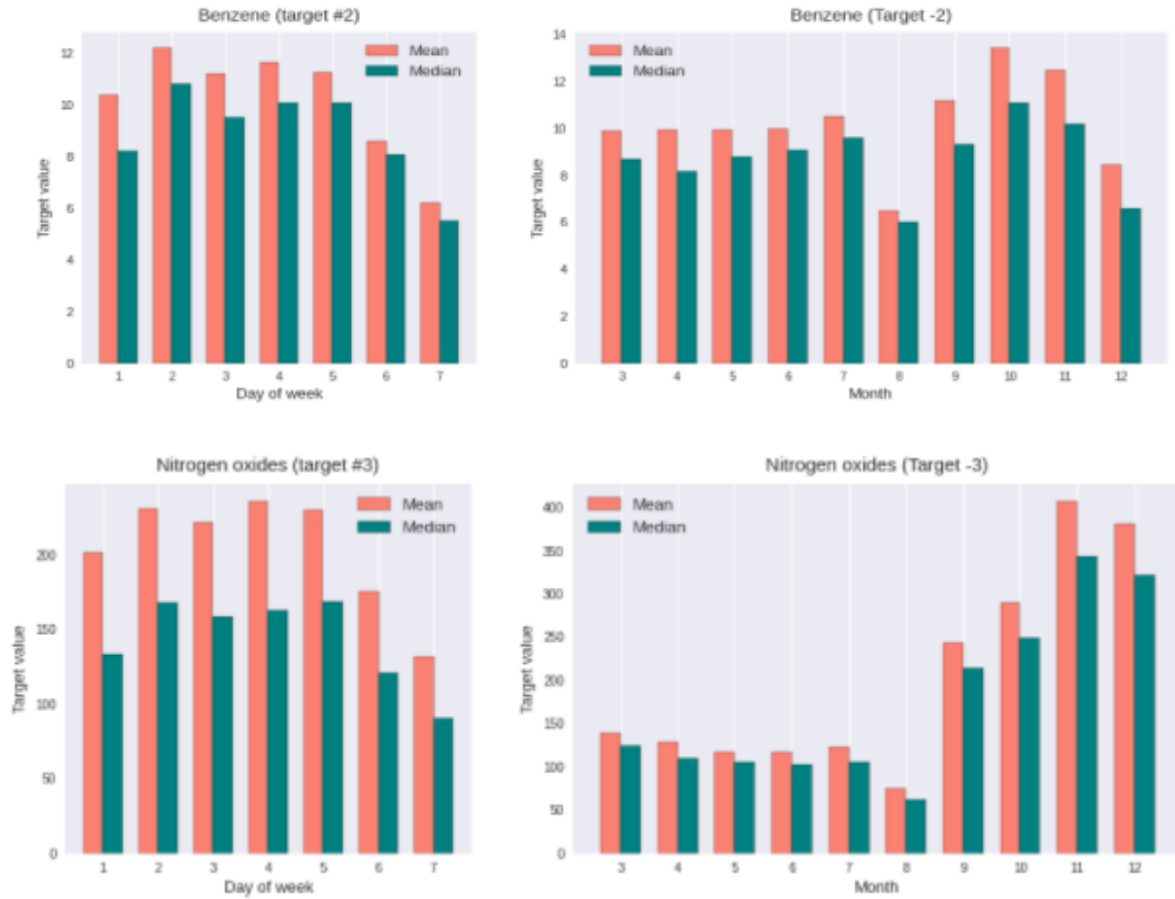
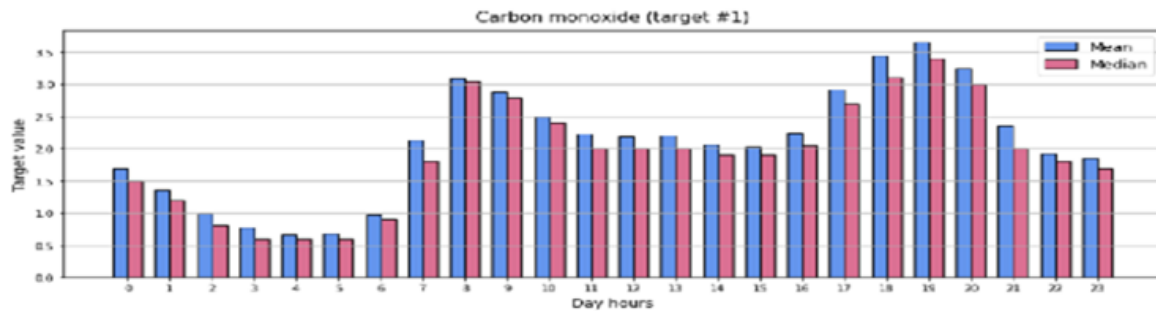


Figure 5: Value of Carbon Monoxide, Benzene and Nitrogen Oxide over each day and month of the year

The below figure 6 gives the relation of pollutants and time(time range during a day). Here, we observed that the pollutants were high during a particular time range in the day that is 8:00 am to 9:00 pm, and less during early morning time and late night.



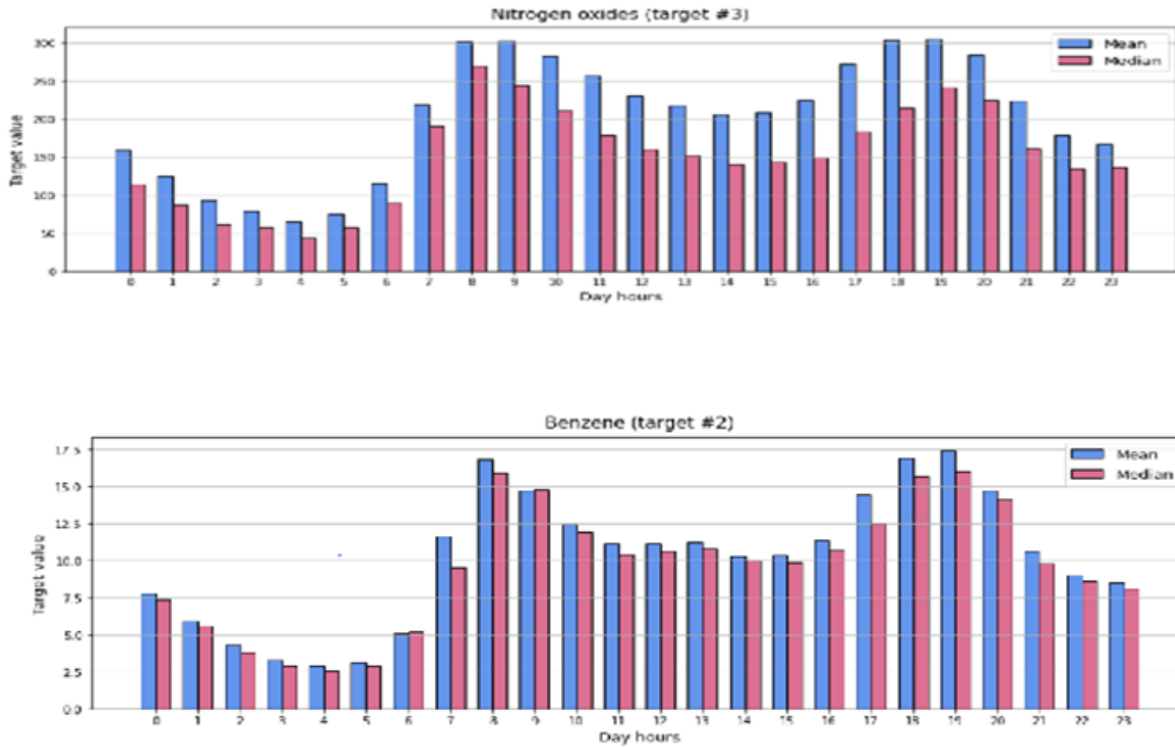


Figure 6: Value of Carbon Monoxide, Benzene and Nitrogen Oxide over time

3.3 Data Preprocessing:

From the EDA we observed the relation of pollutants with different days in a week and the time range in a day, how these features are affecting the amount of pollution.

We performed the below steps to use this information for predicting the data:

01. We extracted time and date separately from the date-time feature.
02. From the extracted date, we separated weekdays from weekends using one-hot encoding as pollutants were high during weekdays as compared to weekends.
03. From the extracted time, we separated the time range during which pollutants were high(8:00 am to 9:00 pm) from the other range using one-hot encoding.
04. After this, we added these two features to our dataset for the prediction of pollutants.

4. Proposed models:

The models which we have planned to implement for the prediction of air pollutants are:

1. Linear Regression
2. Multivariate Regression
3. Random Forest
4. Neural Network

5. Implementation

5.1 Linear Regression:

Linear regression is the simplest model used for statistical techniques for predictive modelling. It can be used to fit a predictive model to an observed data set of values of the response and explanatory variables. After developing such a model, if additional values of the explanatory variables are collected without an accompanying response value, the fitted model can be used to predict the response. It gives us an equation, where we have our features as independent variables, on which our target variable [nitrogen, benzene, carbon] is dependent.

Steps of implementation followed:

01. Use the changed dataset for the prediction of the three pollutants using a linear regression model
02. In our implementation, if we are getting any negative values of the pollutants on prediction, we are taking the absolute value for calculating the RMLSE error of the pollutants.
03. Ridge regression was applied for hyperparameter tuning

Upon implementation of Linear regression, we were getting high error values for all the pollutants but particularly for nitrogen, it was high. Results of different error metrics for the three pollutants on applying linear regression are as follows:

```
Linear Regression
Crabon-Monoxide: MAE Error:  0.37339479984478546
Carbon-Monoxide: RMSLE Error:  0.17913654943017446
Crabon-Monoxide: RMSE Error:  0.5696438319919122
Crabon-Monoxide: MAPE Error:  0.2616004290480654
Crabon-Monoxide: R-square:  0.8388906059441648
Benzene: MAE Error:  1.1775853475980091
Benzene: RMSLE Error:  0.22703513106907655
Benzene: RMSE Error:  1.5977933563981848
Benzene: MAPE Error:  0.41650379874271254
Benzene: R-square:  0.9561323734759828
Nitrogen-Oxide: MAE Error:  64.82380469366946
Nitrogen-Oxide: RMSLE Error:  0.9513804132459138
Nitrogen-Oxide: RMSE Error:  100.34884093967258
Nitrogen-Oxide: MAPE Error:  0.47305345066247606
Nitrogen-Oxide: R-square:  0.711598483598191
```

Fig 7: Test errors of all the three pollutants for linear regression

As we can observe that the error value for particularly nitrogen is high, we went for a more complex model for the prediction and implemented polynomial regression.

5.2 Polynomial Regression:

Polynomial Regression is a form of linear regression in which the relationship between the independent variable x and dependent variable y is modelled as an n th degree polynomial. Since we did not get the best results while performing linear regression as it was not able to linearly separate the data. To overcome this problem we used Polynomial regression. So here, To generate higher-order equations we can add powers of the original features as new features.

Steps of implementation followed:

01. Use the changed dataset for the prediction of the three pollutants using a polynomial regression model

02. We implemented the polynomial regression for different degrees, and through the different errors, metrics found the degree that is giving the best results for the dataset.
03. We observed that till degree 3 the errors were reducing and after that, they were increasing, so we choose degree 3 for prediction.
04. In our implementation, if we are getting any negative values of the pollutants on prediction, we are taking the absolute value for calculating the RMLSE error of the pollutants.

From the table below, we can see the required result obtained by performing polynomial regression on different degrees. We observe that at degree 3 we get the least error and as we go ahead increasing the complexity of the models we can see the pattern that the errors go on increasing. Thus, we concluded that degree 3 is the best model that can be used in polynomial regression to achieve the best accuracy and least RMSLE error amongst others.

Pollutant Errors(RMSLE)\Degree	2	3	4	5
Carbon-Monoxide	0.139	0.132	0.146	0.221
Benzene	0.112	0.114	0.121	0.223
Nitrogen-oxide	0.506	0.489	0.539	1.12

Fig 8: Table showing RMSLE error for different degrees for polynomial regression

5.3 Random Forests:

Decision Trees are a type of non-parametric supervised learning approach that is commonly used for classification and regression. The objective is to build a model that predicts the value of a target variable using basic decision rules derived from data attributes. A tree is an example of a piecewise constant approximation.

For example, as shown in the image below, decision trees learn from data to approximate a sine curve using a series of if-then-else decision rules. The deeper the tree, the more complicated the decision rules and the model's fit.

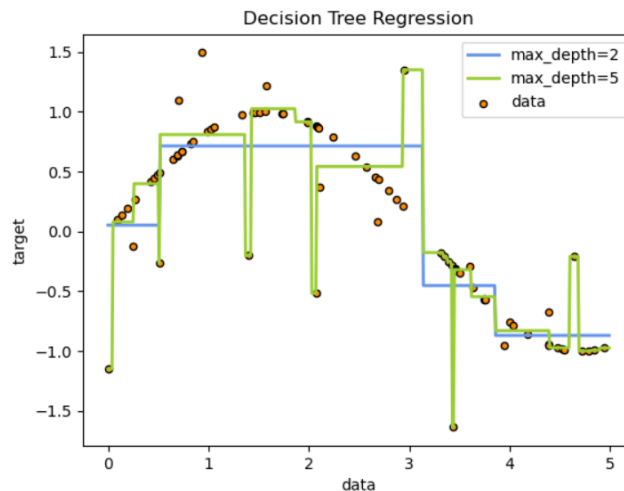


Fig 9: Decision Tree Regression

A random forest is a form of supervised learning algorithm that solves regression and classification problems using ensemble approaches (bagging). At training time, the algorithm constructs a large number of decision trees and outputs the mean/mode of prediction of the individual trees for a particular dataset, a single algorithm may not generate the best forecast. Machine learning algorithms have constraints, and creating a model with high accuracy is difficult. If we create and merge numerous models, we may be able to improve overall accuracy. The combination may be accomplished by aggregating the output from each model with two goals in mind: minimizing model error while retaining generalization. One of the ensemble learning techniques we used is Bootstrapping or bagging which is a type of resampling where we used large numbers of smaller samples of the same size, with replacement, from our original sample.

The diagram below shows the structure of a Random Forest. We can notice that the trees run in parallel with no interaction amongst them. A Random Forest operates by constructing several decision trees during training time and outputting the mean of all the decision trees and finally it can be used for further prediction. The basic design of how it performs in the backend is as follows:-

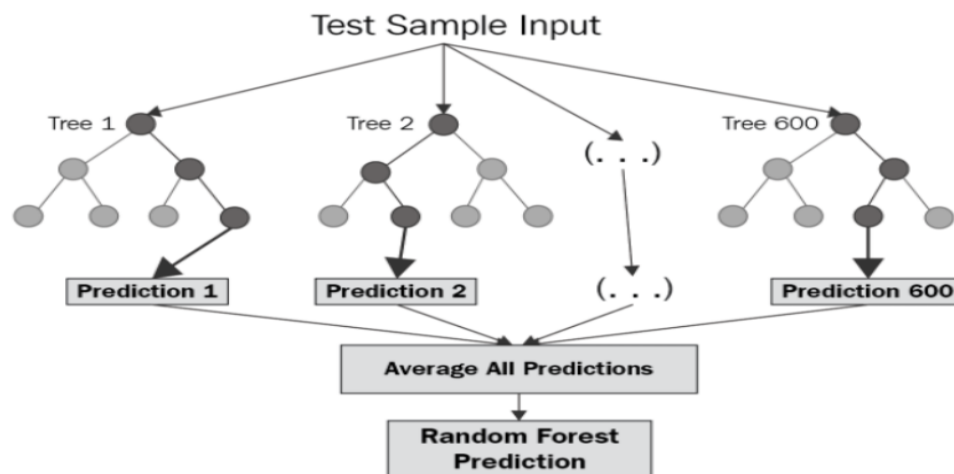


Fig 10:

Working of decision trees

Pseudo Code to Implement Random Forest:-

01. Pick at random k data points from the training set.
02. Build a decision tree associated with these k data points.
03. We choose the number N of trees we want to build and repeat steps 1 and 2.
04. For a new data point, make each one of your N -tree trees predict the value of y for the data point in question and assign the new data point to the average across all of the predicted y values.

Steps we performed to implement Random Forest:-

01. We'll figure out what variables are dependent (y) and independent I (X). The Target nodes (carbon monoxide, benzene, and nitrogen oxides) will be our dependent variables, while the remaining columns in the databases will be our independent variables.
02. To divide the data into train validation and testing datasets, we employ Train-Test Split
03. We import the class RandomForestRegressor from the sklearn package that contains ensemble learning, construct an instance of it, and assign it to a variable.
04. In the random forest, the parameter n_estimators create n number of trees, where n is the number we pass in. To estimate the appropriate number of trees required for the best accuracy model and minimize overfitting, we employed hyperparameter tweaking for the number of estimators. For validation testing, we employed cross validation methodologies.
05. Finally, we used the RandomForestRegressor.fit() function to train the model, modifying weights based on data values to gain better accuracy and lower error.

The following Diagram shows the Hyper-Parameter tuning based on the number of estimators, as we traverse from 100 to 1000. We observe the pattern as the accuracy increases and error decreases but after 800 estimators, we observe the negligible amount of changes and higher time consumption to process and build decision trees. Thus, 800 estimators give us the best probable result for our dataset.

```
FOR N_Estimators :- 700
Mean Absolute Error: target_carbon_monoxide    0.27
target_benzene          0.81
target_nitrogen_oxides  38.69
dtype: float64

Mean Absolute Percentage Error - Accuracy: target_carbon_monoxide    84.04
target_benzene          90.63
target_nitrogen_oxides  76.00
dtype: float64 %

RMSLE - 0.03647201189120783

FOR N_Estimators :- 800
Mean Absolute Error: target_carbon_monoxide    0.27
target_benzene          0.81
target_nitrogen_oxides  38.71
dtype: float64

Mean Absolute Percentage Error - Accuracy: target_carbon_monoxide    84.03
target_benzene          90.64
target_nitrogen_oxides  75.97
dtype: float64 %

RMSLE - 0.03651211791214198

FOR N_Estimators :- 900
Mean Absolute Error: target_carbon_monoxide    0.27
target_benzene          0.81
target_nitrogen_oxides  38.75
dtype: float64

Mean Absolute Percentage Error - Accuracy: target_carbon_monoxide    84.03
target_benzene          90.64
target_nitrogen_oxides  75.94
dtype: float64 %

RMSLE - 0.036568387649371197
```

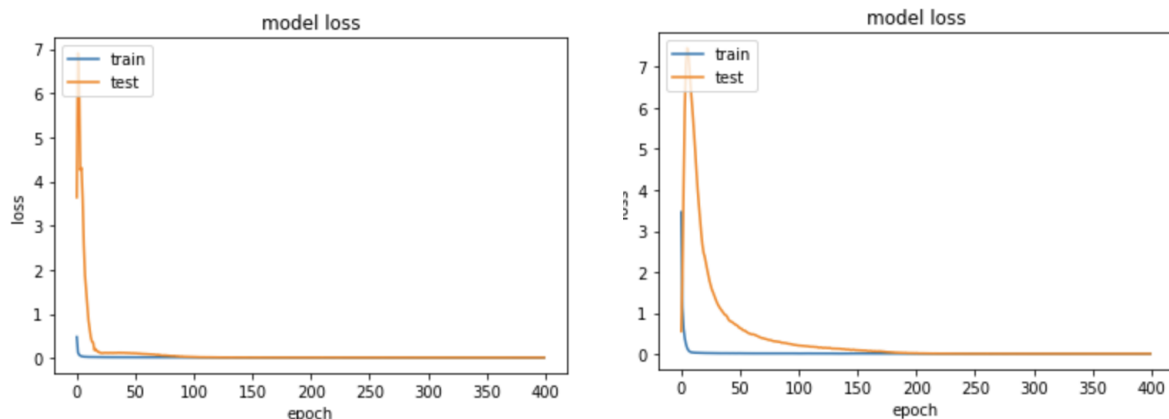
Fig 11: Test errors of all the three pollutants for Random Forest

5.4 Neural Network

Neural Networks are made of different tiny neurons connected to each other. We feed our data through the input layer. The data then passed through multiple hidden layers. These hidden layers perform most of the work. The inputs are multiplied by the weights between each layer. These layers are updated using backpropagation. Once they make a mistake they remember it and try not to repeat it.

Steps we performed to implement Neural Network:

01. We used the TensorFlow library to build our multi-layer perceptron
02. Since the data have different ranges, the neural network may give preference to data with values in higher ranges. So, we normalize the data using MinMaxScalar. We scaled it to between 0 to 1.
03. We started with 5 layers deep neural network, but we found it was underfitting. So we increased it up to 10 layers. We had 256 neurons initially, but we went up to 2048 neurons using trial and error.
04. For loss function, we used Root Mean Squared Log Error.
05. After training, we saw that Nitrogen's loss was very high compared to others. So we made separate models for each output data
06. We tried training our data with 200 epochs, but we saw there was potential for more improvement. So we increased it to 500 epochs.
07. But after increasing the epochs we saw that the training time increased. So to make sure we are not wasting time, we used ReduceLROnPlateau and EarlyStopping.
08. We set ReduceLROnPlateau such that it would halve the learning rate if there is no improvement in the validation loss for more than 50 epochs. We had a learning rate of 0.005 because the learning wasn't too slow or fast. With ReduceLROnPlateau it would get halved 2-3 times during training. [5]
09. EarlyStopping helps in retrieving us the best weights, whenever there is no improvement in the validation loss for 100 epochs[6]
10. After all of this, we saw that validation was not improving as much as the training loss. So we knew we were overfitting it.
11. So we used BatchNormalization and Dropout layer to tackle overfitting. We used different probabilities for different layers. Top layers had a probability of 0.3, as we went down, we reduced it to 0.05.
12. We also explored different activation, one such was LeakyReLU.[7]
13. Along with this, we used Adam Optimizer. We tried SGD, AdaGrad but we got the best result with Adam.[8]



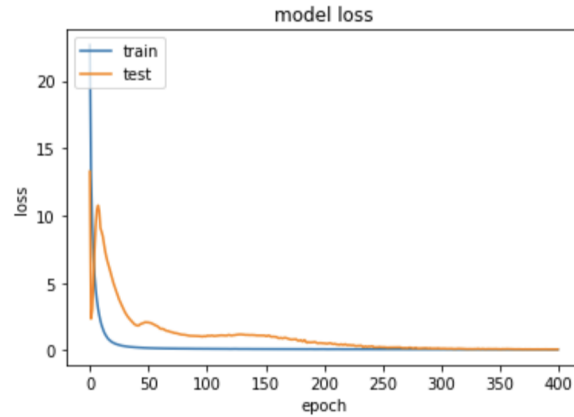


Fig 12: Training and testing loss graph(benzene, carbon and nitrogen)

To evaluate the models we used root mean squared error, mean absolute error, mean absolute percentage error, r-squared error and root mean squared error as metrics. Nitrogen had an RMSLE value around 0.3, almost thrice that of the other two. So to improve it, we added extra hidden layers. After this, we were able to improve it to around 0.22.

6. Results:

Github link for the project: <https://github.com/rohiiit/Pollution-prediction-system>

Table showing the error values for different metrics for the proposed models:

Model\Metric	RMSLE	MAE	MAPE	RMSE	R-square
LINEAR REGRESSION	0.4523	22.121	38%	34.16	0.8355
POLYNOMIAL REGRESSION (Degree=3)	0.245	15.5573	24.33%	25.2967	0.9089
RANDOM FOREST	0.04403	15	19%	45.020	0.90187
NEURAL NETWORK	0.1550	14.218	23.98%	26.853	0.8406

7. Conclusion:

We have used linear regression as our base model and we have compared all other models to it. From the above table, we can see that random forest has the best RMSLE value, neural network and polynomial regression have comparable values, and all the other metrics of all the models have a similar range. We can see that in the random forests the RMSE is quite high as compared to the linear regression model, this leads to the conclusion that

whenever the random forest model predicts most of the times it will hit the target dead centre but if it misses, it will miss the target by a mile.

8. References:

1. <https://www.kaggle.com/c/tabular-playground-series-jul-2021/data>
2. https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html
3. <https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.PolynomialFeatures.html>
4. <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html>
5. https://keras.io/api/callbacks/reduce_lr_on_plateau/
6. https://keras.io/api/callbacks/early_stopping/
7. <https://keras.io/api/layers/activations/>
8. <https://towardsdatascience.com/optimizers-for-training-neural-network-59450d71caf6>