Air Quality Analysis & Prediction

Contents

Table of Contents

1. Introduction

1.1. Abstract

The world we live in is being rapidly automated and emerging technologies like Cloud, Internet of Things, and so forth are being continuously integrated into concepts such as Smart Cities to provide a high level of comfort to the residents with minimum human intervention [10]. A major challenge faced by corporations of developed cities is to control and regulate air quality. With the advent of modern air quality monitoring and pollution control systems, a novel prediction framework aids the process of finding effective solutions to complex problems. This project focuses on investigating the correlation between air quality and weather and building a prediction model based on the results of the exploratory analysis of historical weather and pollution data.

Air quality is assessed based on a banding system which measures the levels of pollutants, namely Ozone (O_3), Nitrogen dioxide (NO_2) and Particulate matter - PM_{10} and $PM_{2.5}$. The overall air quality index at any particular time is given as the maximum band for any pollutant. $PM_{2.5}$ is fine particulate matter of size less than 2.5 micrometres and is considered to have adverse impacts on health ranging from lung cancer to cardiovascular diseases. Although $PM_{2.5}$ is a crucial factor in deciding the overall air quality index, it is currently not included as a pollutant in the UK air quality banding system issued by the Committee on the Medical Effects of Air Pollutants (COMEAP). This is because extensive monitoring of PM2.5 levels using dedicated instruments has only started since 2015 and the presently available data is insufficient for conclusive analysis. [1]

This project aims to predict the air quality band for $PM_{2.5}$ using present and historical pollution data in combination with predicted weather data which is readily available. To solve this problem, firstly, exploratory data analysis will be conducted on available weather and pollution datasets to discover the correlation between different features. After employing suitable data cleaning and feature engineering methods based on the observations made, the feasibility of using different machine learning techniques such as classification and regression models will be analysed.

2. Description of Dataset

2.1. Data Retrieval

The dataset we have for this project was created by joining historical air pollution and weather datasets obtained from two different sources. The steps that were undertaken to obtain these datasets and creating the final dataset are detailed below:

The air pollution data was obtained from the London Air, the website of the London Air Quality Network (LAQN), which monitors air pollution in London and South East England. The LAQN was formed in 1993 to coordinate and improve air pollution monitoring in London. The website provides publicly available datasets that contain independent scientific measurements of various pollutants obtained from over 121 active monitoring sites [2].
The London Air website provides a data download tool which allows the user to download either data for one site or data for one species for up to six sites.

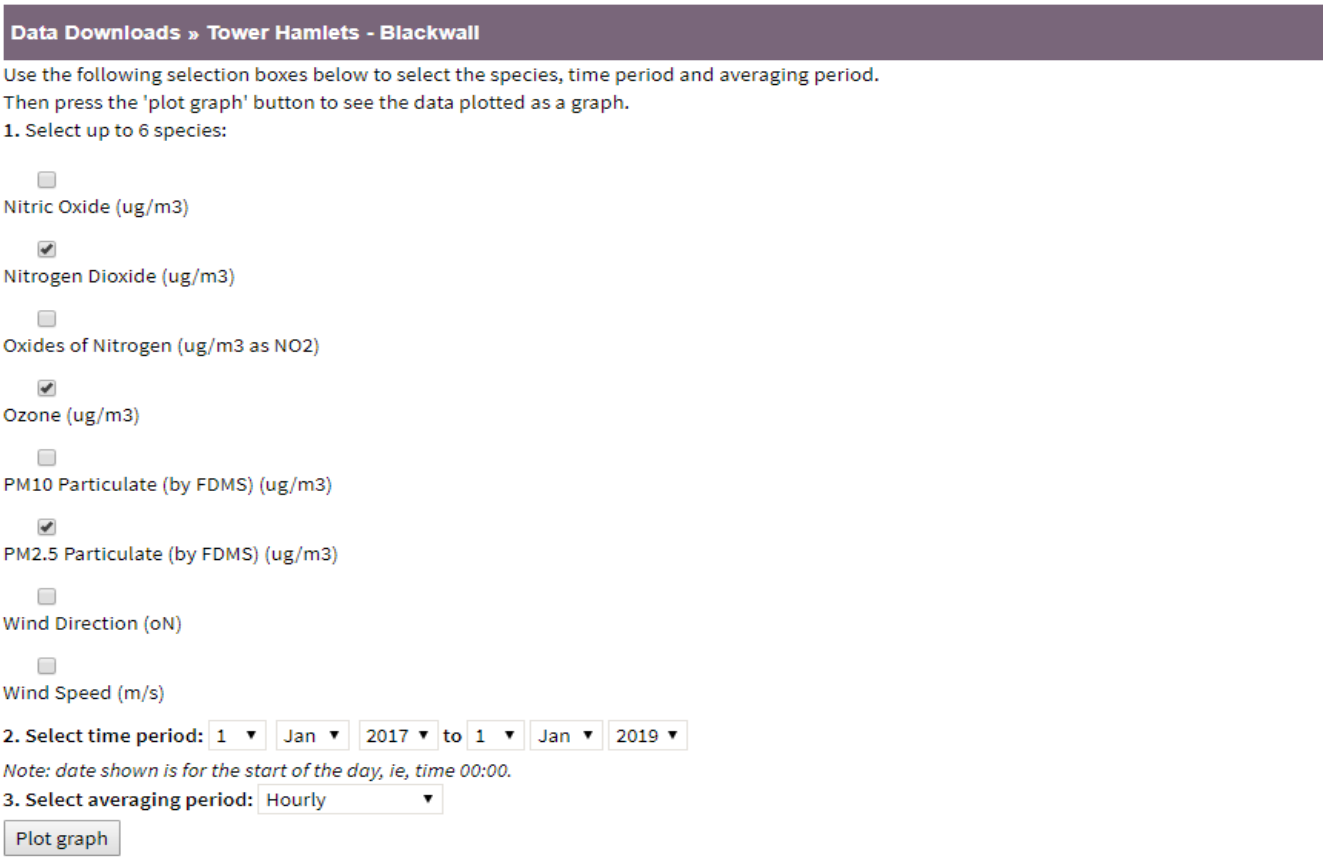

Figure 1: Pollution Data Collection

In this project, we have chosen to obtain the air pollution data for three specific species, namely Nitrogen Dioxide (NO_2), Ozone (O_3) and $PM_{2.5}$ Particulate Matter, from the Tower Hamlets monitoring station in Central London. The time period was chosen from January 1, 2017 to January 1, 2019. The sampling rate was chosen to be hourly which resulted in a total of 17520 samples for each pollutant during the chosen time period. The dataset was available to download as a CSV file.

The weather data was obtained from the Integrated Surface Global Hourly Dataset hosted by the National Oceanic and Atmospheric Administration (NOAA). The Integrated Surface Dataset is composed of worldwide surface weather observations from over 35,000 stations. Parameters included are atmospheric pressure, temperature/dew point, atmospheric winds, clouds, precipitation and more. ISD provides hourly weather data that is appropriate for the application in this project [3].

The Integrated Surface Dataset offers a data search tool that allows the user to choose the observed species, location and time period. In this project, we have chosen to collect temperature, humidity, wind speed and direction since these are expected to demonstrate a large correlation to air pollution. The dataset was available to download as a CSV file. The monitoring site was chosen to be City, London according to the geographical proximity to the Tower Hamlets station chosen in the air pollution dataset.

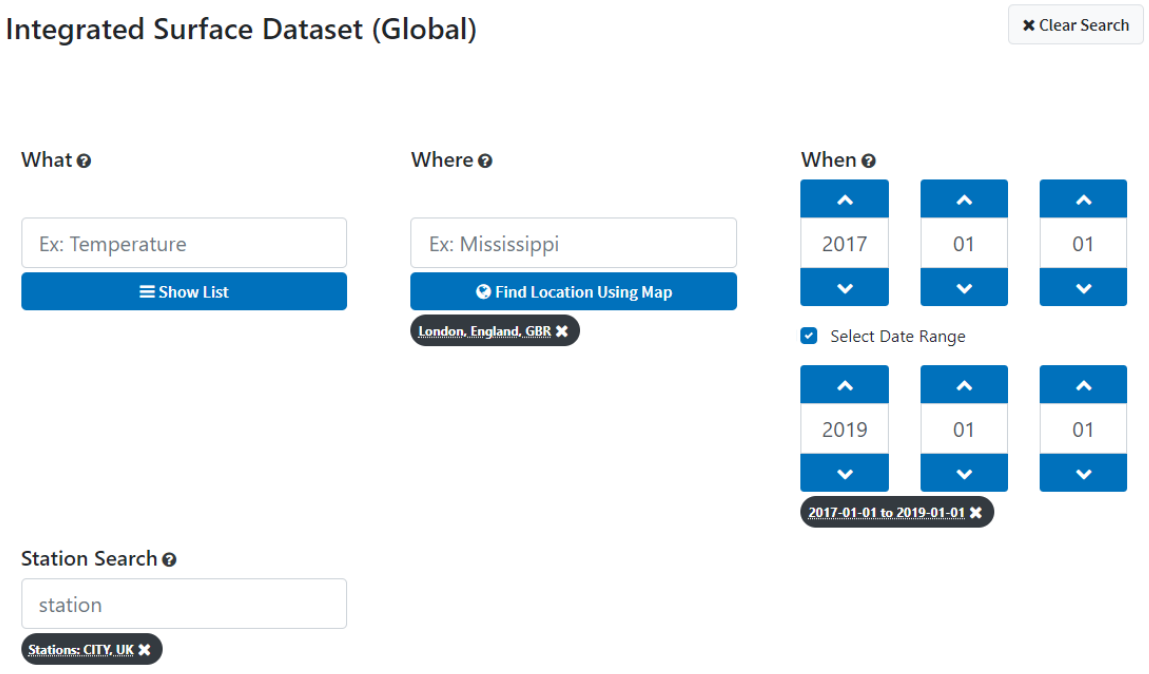

Figure 2: Weather Data Collection

2.2. Data Description

The pollution dataset had a size of 52560 rows x 6 columns and contained following fields of importance:

```
#Creating a pandas dataframe from the pollution dataset

pdata = pd.read_csv('data/pollution-1.csv')
pdata.head()
```

	Site	Species	ReadingDateTime	Value	Units	Provisional or Ratified
0	TH4	NO2	01/01/2017 00:00	38.9	ug m-3	R
1	TH4	NO2	01/01/2017 01:00	63.0	ug m-3	R
2	TH4	NO2	01/01/2017 02:00	47.9	ug m-3	R
3	TH4	NO2	01/01/2017 03:00	51.0	ug m-3	R
4	TH4	NO2	01/01/2017 04:00	42.8	ug m-3	R

Name	Description	Type
Site	A unique identifier for the monitoring site the data was obtained from.	string
Species	The name of the pollutant measured.	string
ReadingDateTime	The date and timestamp when the observation was made.	string
Value	The measured value of the pollutant.	float64
Units	The unit of the measured value of the pollutant.	string

Table 1: Pollution Dataset Description

The weather dataset had 34446 rows by 13 columns and contained following fields of importance:

```
#Creating a pandas dataframe from the weather dataset

wdata = pd.read_csv('data/weather-1.csv')
wdata.head()
```

	DATE	DEW	TMP	WND
0	2017-01-01T00:20:00	+0050,1	+0080,1	230,1,N,0046,1
1	2017-01-01T00:50:00	+0050,1	+0070,1	230,1,V,0031,1
2	2017-01-01T01:20:00	+0050,1	+0070,1	230,1,N,0026,1
3	2017-01-01T01:50:00	+0050,1	+0070,1	240,1,V,0031,1
4	2017-01-01T02:20:00	+0050,1	+0070,1	240,1,N,0046,1

Name	Description	Type
DATE	The date and timestamp when the observation was made.	String
DEW	The temperature to which a given parcel of air must be cooled at constant pressure and water vapor content in order for saturation to occur. The data is formatted as a string with string index ranges specifying different characteristics of the observation.	String
WND	The wind direction and wind speed. The data is formatted as a single string with string index ranges specifying both the wind speed and direction values.	String
TMP	The temperature of the air. The data is formatted as a string with string index ranges specifying different characteristics of the observation.	String

Table 2: Weather Dataset Description

2.3. Data pre-processing

The final data fed to the model is created by merging two datasets (pollution.csv and weather.csv) based on time. From observing the datasets, it is found that the two datasets have different number of observations per hour.

The weather data has 34446 records observed every 20 mins from 2017-01-01 00:20:00 to 2019-01-01 23:50:00.
The pollution data has 17520 observations per species observed every hour from 2017-01-01 00:00:00 to 2018-12-31 23:00:00 for 3 different species (NO_2, O_3, $PM_{2.5}$).

Step 1: Parsing Date Time
Since the data is read from a csv file, the datatype of ReadingDateTime was not in DateTime format, hence it was parsed into DateTime type while reading the csv. The best approach would be to store the datasets as a pickle file (.pkl) which would preserve the datatype of the features.

```
#Parsing date column of both datasets to be read by python

pdata = pd.read_csv('data/pollution-1.csv',parse_dates=['ReadingDateTime'])
wdata = pd.read_csv('data/weather-1.csv',parse_dates=['DATE'])
```

Step 2: Extracting numerical values
The features in the weather dataset are not explicitly numerical values; each value is a reference to a set of information through which the actual numerical values are to be extracted. This information is detailed in the following code block and table 3:

```
#Extracting numerical values of dew point, temperature, wind speed (converting from mps to kmph)
#and direction

wdata['DEW'] = wdata['DEW'].str[:-2].astype(np.float64)/10
wdata['TMP'] = wdata['TMP'].str[:-2].astype(np.float64)/10
wdata['DIR'] = wdata['WND'].str[:3].astype(np.float64)
wdata['SPD'] = (wdata['WND'].str[8:-2].astype(np.float64)/10)*3.6

#Calculating relative humidity from dew point and temperature

wdata['HUM'] = 100*(np.exp((17.625 * wdata['DEW'])/(243.04 +
wdata['DEW']))/np.exp((17.625 * wdata['TMP'])/(243.04 + wdata['TMP'])))

#Replacing missing wind direction observations with nulls

wdata.DIR.replace(999,np.nan,inplace=True)
```

Relative Humidity: It the ratio between the amount of water vapor at a given temperature to the maximum amount of water vapor the air can hold at that temperature. This is calculated from DEW point and TMP values.

Wind Values: The wind speed is converted from metres per second to kilometres per hour. The reference for the weather dataset also informs that wind direction = 999 indicates a missing value. These are replaced to be nulls.

Name	String Index	Description
DEW	[0-4]	The observed dew point temperature value with the following properties **Min:** -0982 **Max:** +0368 **Units:** Degrees Celsius **Scaling Factor:** 10
DEW	[5-6]	The code that denotes a quality status of the reported dew point temperature.
TMP	[0-4]	The observed temperature value with the following properties: **Min:** -0932 **Max:** +0618 **Units:** Degrees Celsius **Scaling Factor:** 10
TMP	[5-6]	The code that denotes a quality status of the reported temperature.
WIND	[0-2]	The angle, measured in a clockwise direction, between true north and the direction from which the wind is blowing. **Min:** 001 **Max:** 360 **Units:** Angular Degrees **Scaling Factor:** 1 **Missing:** 999
WIND	[4]	The code that denotes the quality status of a reported wind direction angle.
WIND	[6]	The code that denotes the character of the wind observation, which is a qualitative metric to indicate calm or strong winds.
WIND	[8-11]	The observed wind speed which is the rate of horizontal travel of air past a fixed point. **Min:** 0000 **Max:** 0900 **Units:** meters per second **Scaling Factor:** 10
WIND	[13]	The code that denotes the quality status of the reported wind speed.

Table 3 Numerical Value Extraction

Step 3: Down sampling the weather dataset

One of the major challenges faced while creating the final dataset was making the timestamps of the observed values of both datasets match. While the datasets spanned the same time period, the weather dataset had two samples every hour while the pollution dataset had strict hourly samples. It was also observed that there was a 20-minute discrepancy between the timestamps of both the datasets.

This was solved by down sampling the weather dataset to 17520 records by selecting records at a frequency of one hour. The down sampled records have the mean of the features over every hour as its values.

```
#Downsampling the weather dataset

wdata.index=wdata['DATE']
wdata_resampled = wdata.resample('1H').mean()[:17520]
wdata = wdata_resampled.reset_index()
```

Step 4: Merging the two datasets based on timestamp

The down sampled weather dataset is merged with the pollution dataset based on the timestamp by selecting the required features through concatenation. The features from the weather dataset are directly extracted, while the pollution dataset has 17520 records for each of the three pollutants spanning the chosen time period. The observed value for each pollutant was conditionally extracted from the pollution dataset and concatenated separately into the final constructed dataset.

```
#Joining the weather and pollution datasets

data = pd.concat([wdata['DATE'], wdata['DEW'],wdata['TMP'],wdata['DIR'],wdata['SPD'],
       wdata['HUM'],pdata.Value[pdata.Species=='NO2'].reset_index(drop=True).rename('NO2'),
       pdata.Value[(pdata.Species=='O3')].reset_index(drop=True).rename('O3'),
       pdata.Value[(pdata.Species=='PM2.5')].reset_index(drop=True).rename('PM25')], axis=1)
```

2.4. Data Visualization

The time series visualization of the final dataset was plotted to construct initial impressions about the collected data.

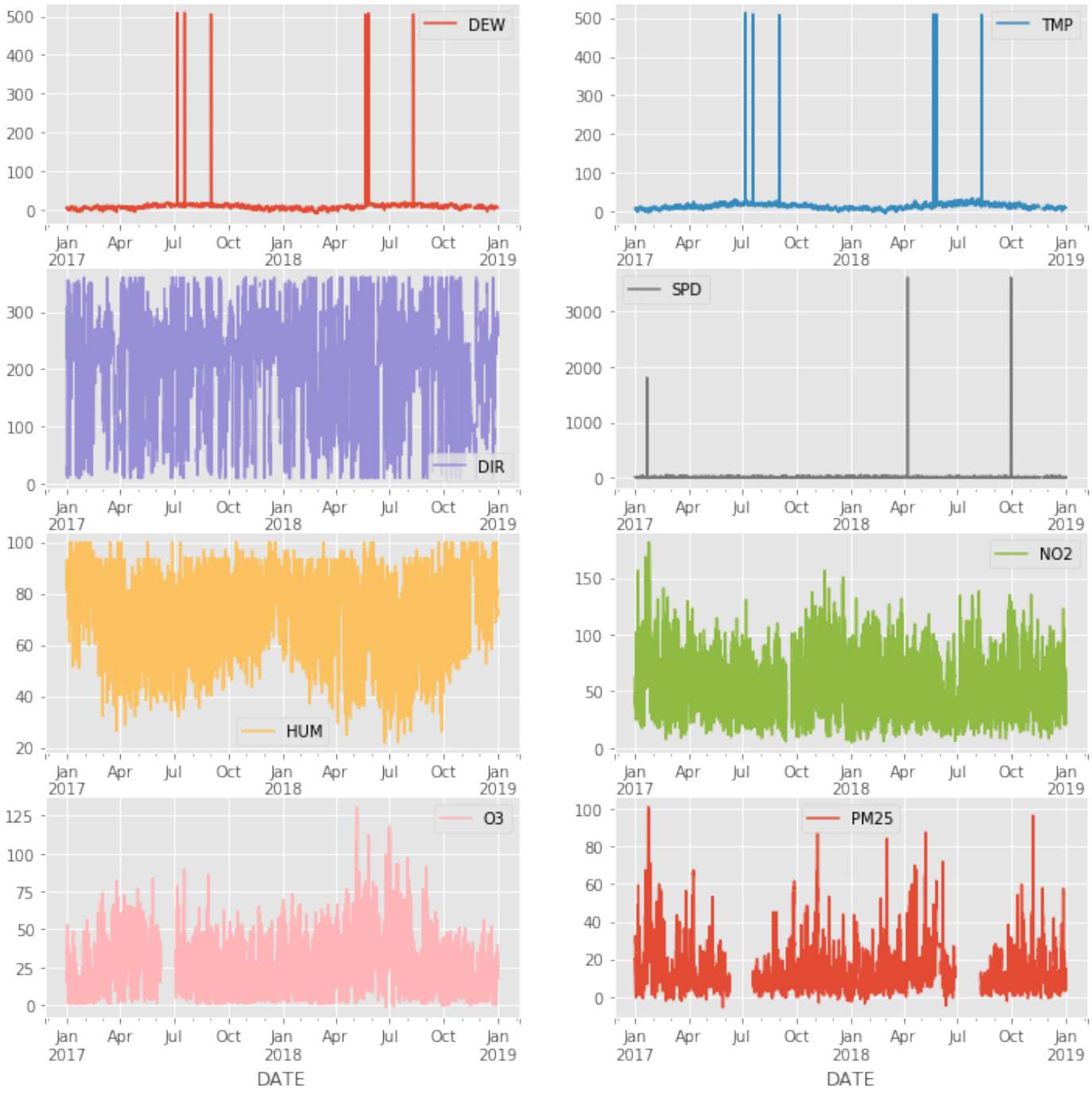

Figure 3: Timeseries visualisation

It is clear from the initial visualizations that many features of the dataset contain null values and outliers. Given the scientific nature of the dataset, it is not ideal to handle these issues generically by employing statistical methods. The outliers and null values in the dataset have to be handled in context with available scientific information, and the methodological approach that was undertaken to perform these operations is detailed in the next section.

3. Data Cleaning

3.1. Handling Outliers

From Table 4, the maximum and minimum possible values for the weather dataset can be inferred as shown below:

Band	Dew (°C)	Temperature (°C)	Wind Direction (°)	Wind Speed (Km/Hour)	Humidity (%)
Minimum	-98.2	-93.2	1	0	0
Maximum	36.8	61.8	360	324	100

Table 4: Limits for weather features

Since these values are obtained from the reference for the weather dataset, any value outside these limits can be considered as outliers and confidently discarded.
The London Air Quality Network (LAQN) specifies the following values for the different bands of pollutants [4]:

Band	Index	O_3 ($\mu g/m^3$)	NO_2 ($\mu g/m^3$)	PM2.5 ($\mu g/m^3$)
Low	1-3	0-100	0-200	0-35
Moderate	4-6	101-160	201-400	36-53
High	7-9	161-240	401-600	54-70
Very High	10	241 or more	601 or more	71 or more

Table 5: Limits for pollution features

The corresponding pollutant concentrations for the highest band indicate the possible outliers for the pollution dataset. The dataset has to be examined closely to determine if a value could be an outlier. The number of total possible outliers based on these thresholds and their respective upper and lower limits were observed:

```
#Number of possible outilers
print('DEW:',((data['DEW'] <= -99) | (data['DEW'] >= 37)).sum())
print('TMP:',((data['TMP'] <= -94) | (data['TMP'] >= 62)).sum())
print('DIR:',((data['DIR'] < 1) | (data['DIR'] > 360)).sum())
print('SPD:',((data['SPD'] < 0) | (data['SPD'] > 324)).sum())
print('HUM:',((data['HUM'] < 0) | (data['HUM'] > 100)).sum())
print('NO2:',((data['NO2'] < 0) | (data['NO2'] > 601)).sum())
print('O3:',((data['O3'] < 0) | (data['O3'] > 241)).sum())
print('PM25:',((data['PM25'] < 0) | (data['PM25'] > 71)).sum())
```

Species	No. of Outliers
Dew	9
Temperature	9
Wind Direction	0
Wind Speed	10
Humidity	0
NO_2	0
O_3	9
$PM_{2.5}$	128

Table 6: Number of outliers

	Species	Min	Max
0	DEW	-9.000000	508.95000
1	TMP	-5.000000	513.45000
2	DIR	10.000000	360.00000
3	SPD	0.000000	3599.64000
4	HUM	21.789608	100.00000
5	NO2	4.300000	181.39999
6	O3	-1.600000	130.70000
7	PM25	-5.800000	101.20000

Table 7: Min and Max values of features

Based on the above observations, it can be concluded that the outliers for the weather dataset can be safely discarded. In the pollution dataset, O_3 has 9 outliers, but the maximum value is 130.7 $\mu g/m^3$ which is well within the threshold of 241 $\mu g/m^3$. This indicates that all the outliers are negative values and can safely be discarded. $PM_{2.5}$ has 128 possible outliers and the maximum observed value is 101.2 $\mu g/m^3$ which is still a feasible value and the range of observed positive values cannot be considered as outliers. Hence only the negative values are discarded.

```
#Replacing outliers with nulls
#Ozone cannot be negative
#PM2.5 positive outlier is a possible value
dew_out = data.DEW[(data['DEW'] <= -99) | (data['DEW'] >= 37)]
tmp_out = data.TMP[(data['TMP'] <= -94) | (data['TMP'] >= 62)]
spd_out = data.SPD[(data['SPD'] < 0) | (data['SPD'] > 324)]
o3_out = data.O3[(data['O3'] < 0) | (data['O3'] > 241)]
pm25_out = data.PM25[(data['PM25'] < 0)]
data.DEW.replace(dew_out,np.nan,inplace=True)
data.TMP.replace(tmp_out,np.nan,inplace=True)
data.SPD.replace(spd_out,np.nan,inplace=True)
data.O3.replace(o3_out,np.nan,inplace=True)
data.PM25.replace(pm25_out,np.nan,inplace=True)
```

Since the total number of outliers is not alarmingly high, they were replaced with null values, which will be handled in the next section.

The time series visualization of the dataset was plotted after removing the outliers.

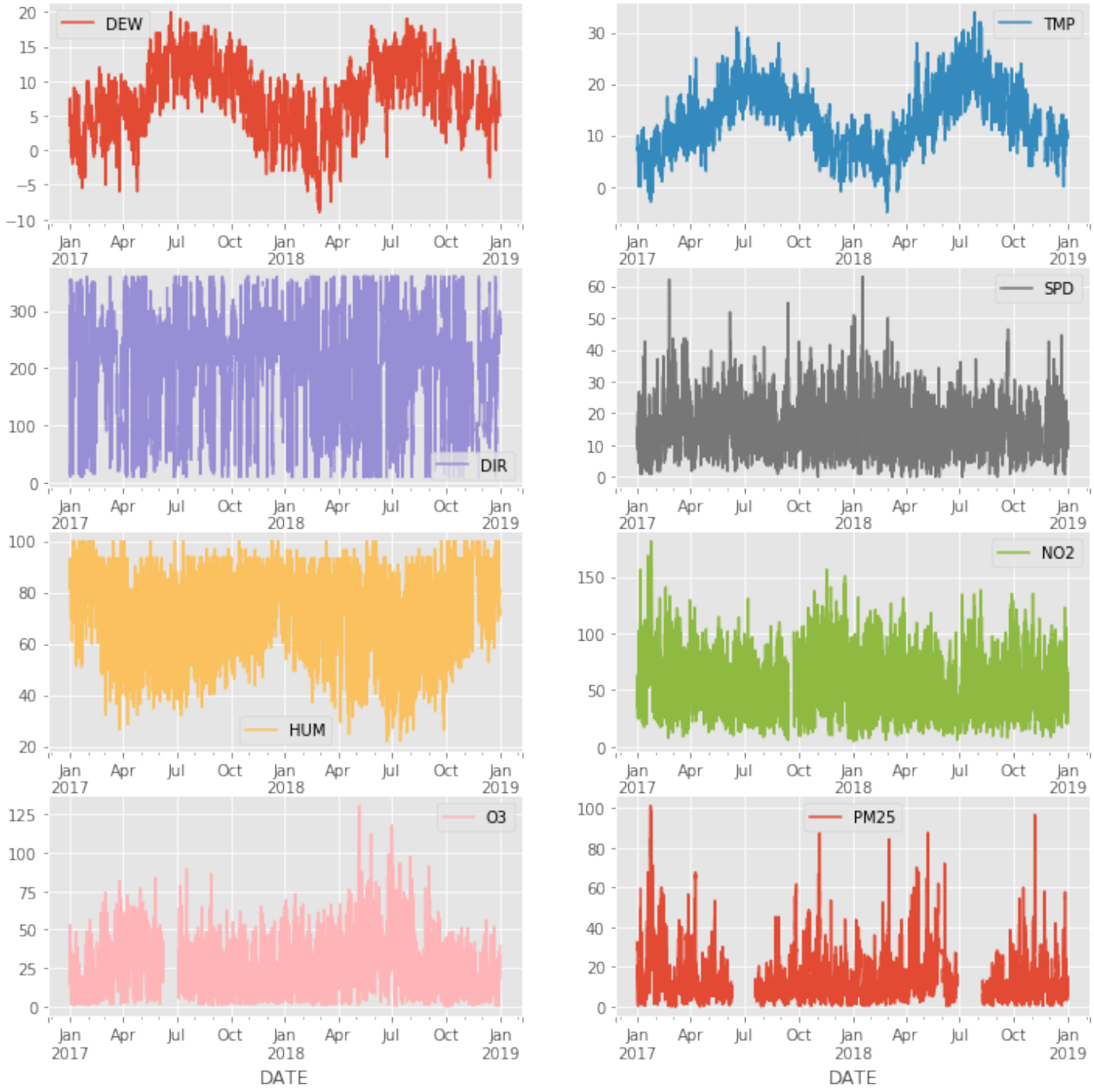

Figure 4: Timeseries without outliers

This proves that removing the outliers methodically had a huge impact on the dataset, as is evident from the plotted waveforms, particularly when DEW, TMP and SPD are juxtaposed.

3.2. Handling null values

This section deals with handling of missing null values. Different approaches were implemented in order to handle and impute the missing values. This is a crucial part of pre-processing as it could lead to wrong prediction and classification of any model being used in the future.

The fundamental approach is to understand the reason of missing values and observe the distribution of null values.

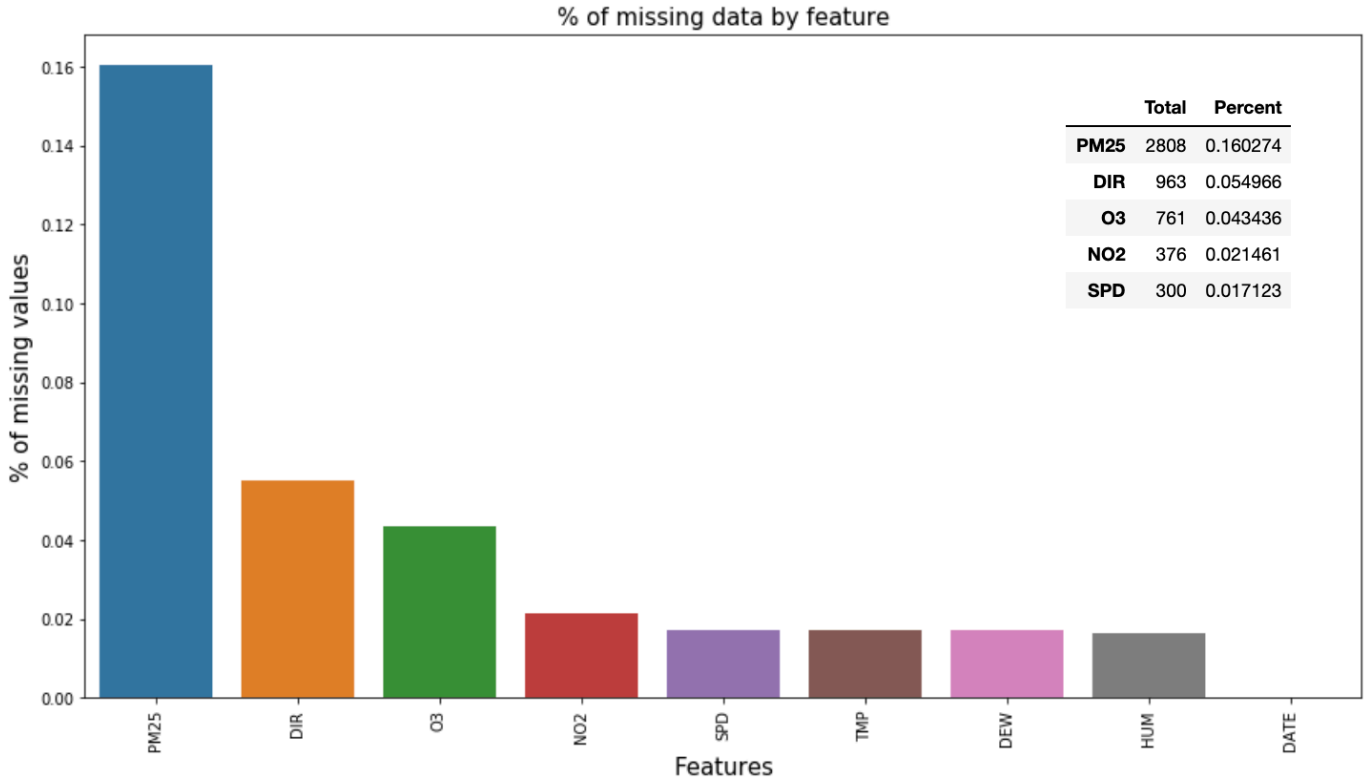

Figure 5: Number of null values for different features

From the previous time series visualization and exploring the percentage of missing data by feature, it is obvious that $PM_{2.5}$ has a large amount of missing data in account of no observations being made in the late summer months. Wind direction has the next higher percentage of missing values on account of the conversion of '999' missing values from Section **2.3**. Ozone has no observations throughout the month of July, 2017 as apparent from the time series visualization. Several imputation methods were tested and the advantages & disadvantages of each of these methods were extensively studied.

3.2.1. Do Nothing

Some algorithms are designed to handle the missing data based on the calculation of data loss in the training data. XGBoost for example is capable of this scenario while others simply choose to ignore the null values. There are also algorithms (SciKit – Linear Regression) that demands handling of null beforehand. The prediction models chosen in this project could not work with the presence of null values in the dataset.

3.2.2. Deleting records

This is a popular method of handling null values in case the amount of missing data is relatively low. The decision on deleting the entire record can either depend on a particular feature; if it is missing a value, or a range of features missing values simultaneously. This method is however only advised when there are enough samples in the given dataset. Also, it has to be observed that deletion of records doesn't bias the dataset and influence the output.

3.2.3. Replacing with mean

This approach can be applied with features that are independent of each other. Though this can be extended only in a numerical dataset, it depends on the nature of that particular feature. All the missing values are replaced with the mean of the respective features

3.2.4. Imputing using most recent values

Imputing using the frequent values is another statistical strategy and is highly effective when missing values in a dataset are completely random and scattered. The Pandas library has a built-in function to fill the null values using recent observations. The 'fillna' function in combination with the forward fill method propagates the last valid observation forward to next valid observation.

3.2.5. Using Algorithms which support missing values

KNN is a machine learning algorithm which works on the principle of distance measure. While the algorithm is applied, KNN considers the missing values by taking the majority of the K nearest values. [5] Unfortunately, the SciKit Learn library for the K – Nearest Neighbor algorithm in Python does not support the presence of the missing values. Another algorithm which can be used here is Random Forest. This model produces a robust result because it works well on non-linear and the categorical data. It adapts to the data structure taking into consideration of the high variance or the bias, producing better results on large datasets.

Approach	Pros	Cons
Deleting Rows	• Removal of entire null values may result in better accuracy of the model. • Removal of records which doesn't have weightage improves the quality of model.	• Loss of information. • Not suitable if a large of portion of dataset are null values.
Replacing with Mean	• Suitable for smaller datasets. • May perform better than removal of records.	• Can result in more variance in the dataset. • Not suitable for categorical values and induces uncertainty.
Forward Fill	• Suitable for categorical values.	• Doesn't take correlation into consideration. • Can increase bias in the dataset.
KNN Algorithm	• Does not require creation of a predictive model for each attribute with missing data in the dataset • Correlation of the data is neglected.	• It is a very time-consuming process and it can be critical in data mining where large databases are being extracted. • Choice of distance functions can be Euclidean, Manhattan etc. which does not yield a robust result.

Table 8: Different approaches to handling nulls

Since the dataset is large enough, it was decided that deleting all records with missing values was the best approach.

3.3. Feature Engineering

Feature engineering is the methodical process of using available knowledge about the data to create or modify features to aid the machine learning prediction model. From the time series visualizations, it is apparent that the date-time feature has a huge impact on both the datasets. Since the date feature is a timestamp, the required features such as time of day, weekday and month have to be extracted. Weather and pollution have seasonality and trend, since the dataset that is fed to the model is composed of weather and pollution it is important that we train the model by considering day of the week and month of the year. Day of the week-based seasonality is important to be considered for pollution features, since the change is pollution is more erratic. Month of the year seasonality is important to be considered for weather-related features as weather has characteristics that change more sluggishly than pollution.

```
#Adding date features

data['MONTH'] = pd.DatetimeIndex(data['DATE']).month
data['HOUR'] = pd.DatetimeIndex(data['DATE']).hour
data['DAY'] = data['DATE'].dt.weekday_name
data['DAY_CAT'] = data['DATE'].dt.dayofweek
```

The day of week is extracted as the weekday name for visualization and plotting purposes and also as a numerical value for feeding it into the prediction model.
It is also apparent from the time series visualization of the dataset that all the observations constantly rise and fall, which indicates that all features depend highly upon the previous values. Time shifted features were created for all the current weather and pollution features using the Pandas shift method. The optimum shift period was found to be up to 5 hours, which will be demonstrated using a correlation matrix in the next section. The time shifting is done before dropping the null values because the shift function requires a continuous dataset.

```
# Adding time shifted features
## This is done before removing nulls so that the dataset is continuous

data['DEW_1'] = data.DEW.shift(periods=1)
data['TMP_1'] = data.TMP.shift(periods=1)
data['DIR_1'] = data.DIR.shift(periods=1)
data['SPD_1'] = data.SPD.shift(periods=1)
data['HUM_1'] = data.HUM.shift(periods=1)
data['NO2_1'] = data.NO2.shift(periods=1)
data['O3_1'] = data.O3.shift(periods=1)
data['PM25_1'] = data.PM25.shift(periods=1)
```

3.4. Correlation between features

A correlation matrix is constructed to study the relationship between different features of the final dataset.

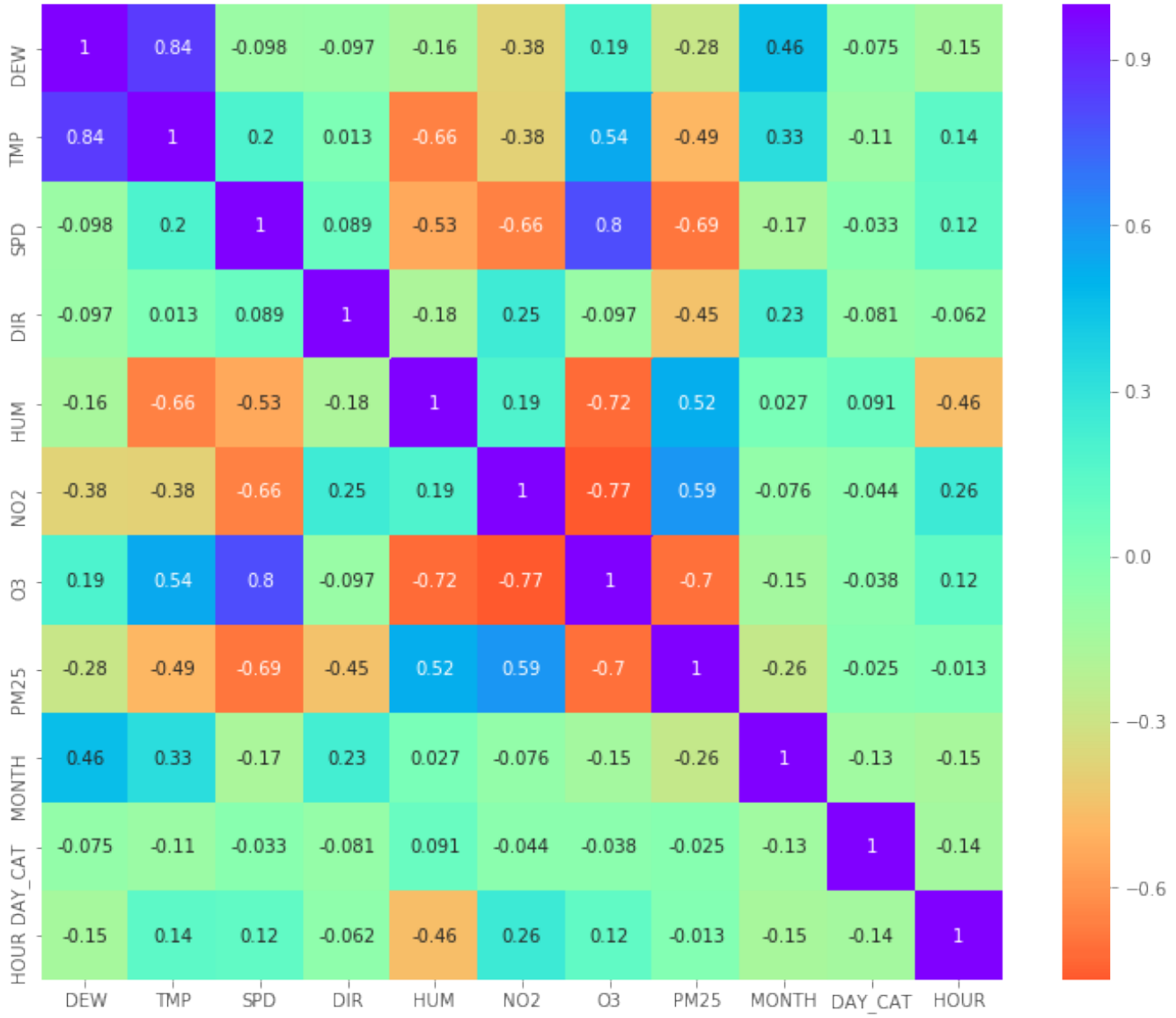

Figure 6: Correlation matrix for the dataset

From the matrix, the following inference are made:

Positive Correlation	Negative Correlation
• DEW-TMP • SPD-O3 • PM25-NO2 • MONTH-DEW • MONTH-TMP • PM25-HUM	• TMP-HUM • SPD-NO2 • SPD-PM25 • HUM-O3 • NO2-O3 • O3-PM25 • HOUR-HUM

Table 9: Inference from correlation matrix

These inferences can be further confirmed through the following visualizations:

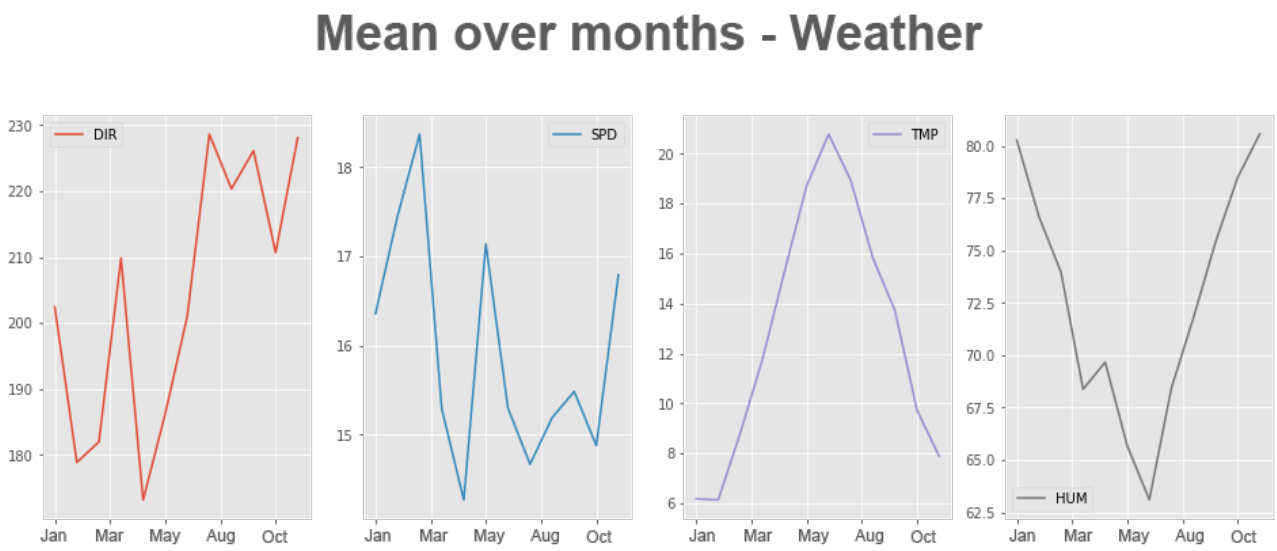

Figure 7: Mean over months - weather

The seasonal trend of the weather data can be clearly observed, with temperature going up during summer months, which in turn affects the humidity inversely.

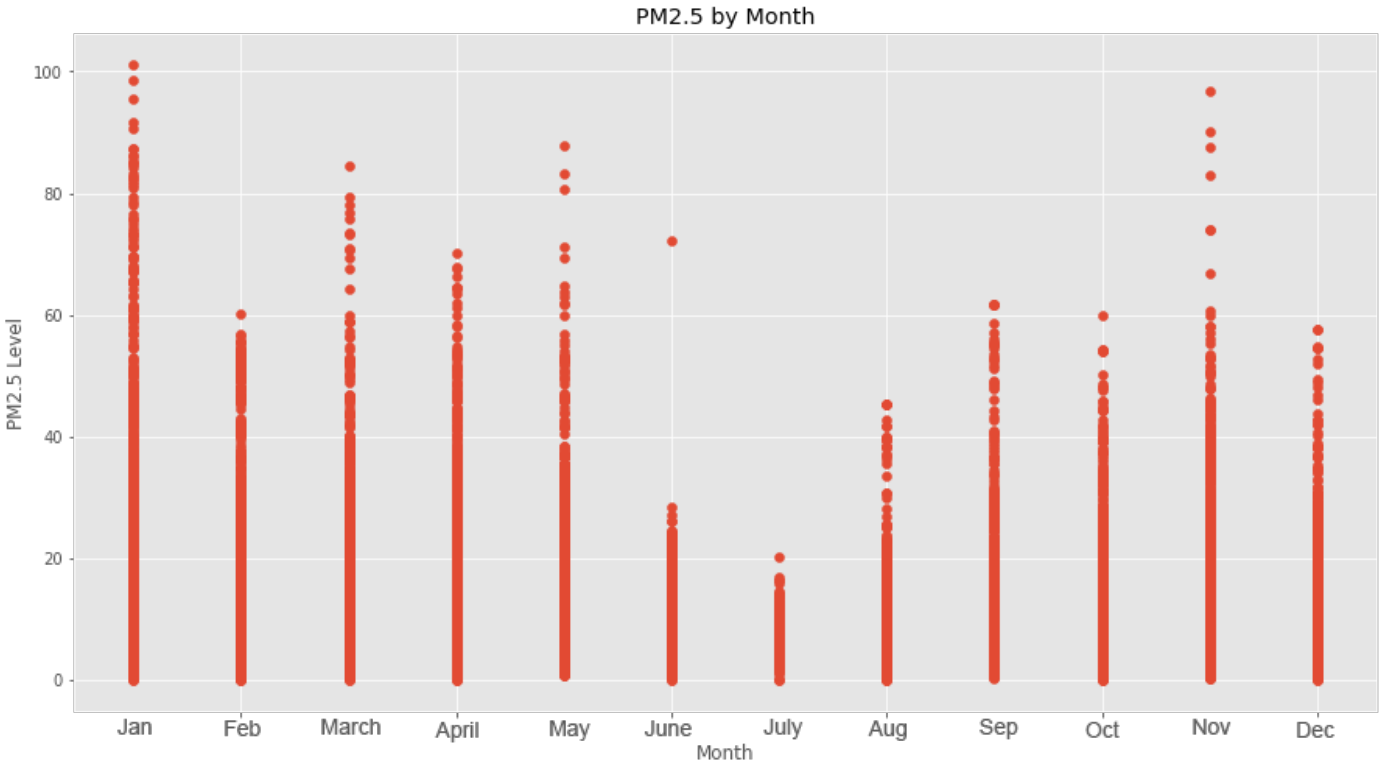

Figure 8: PM2.5 by month

From the scatter plot above, a clear relationship can be seen between the weather data and $PM_{2.5}$ levels. It is expected that the pollution levels would rise due to the cold temperatures in the winter months. Another interesting observation is the strong negative correlation between wind speed and $PM_{2.5}$ values. It is observed that the month of July has the strongest wind speeds and the least $PM_{2.5}$ levels.

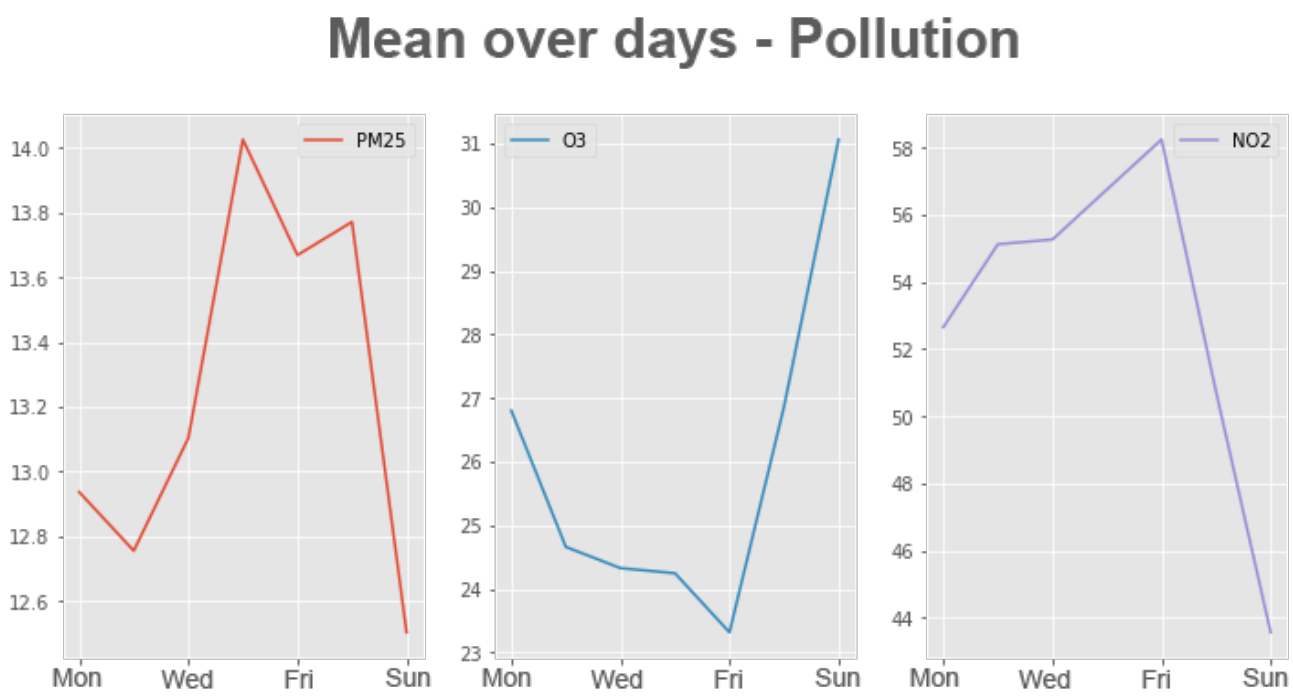

Figure 9: Mean over days - Pollution

The weekly seasonal trend in the pollution data demonstrates that there are high levels of $PM_{2.5}$ during the weekdays and it falls rapidly during the weekend. Ozone has an inverse relationship with the NO_2 and $PM_{2.5}$ levels.

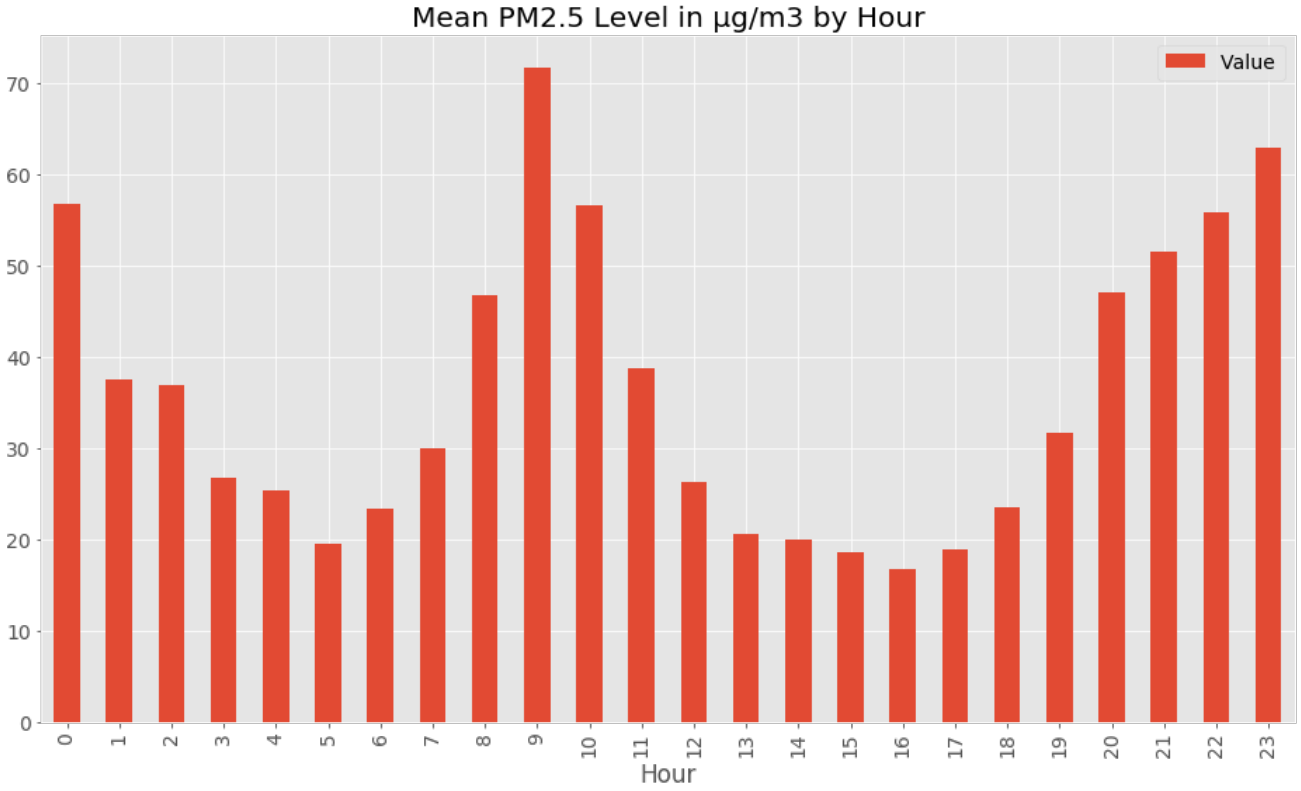

Figure 10: PM2.5 by hour

The $PM_{2.5}$ levels constantly rise and fall throughout the day, which indicates a strong correlation between previously observed and current values. This was the motivation behind adding time shifted values for all the seasonal features in the dataset.

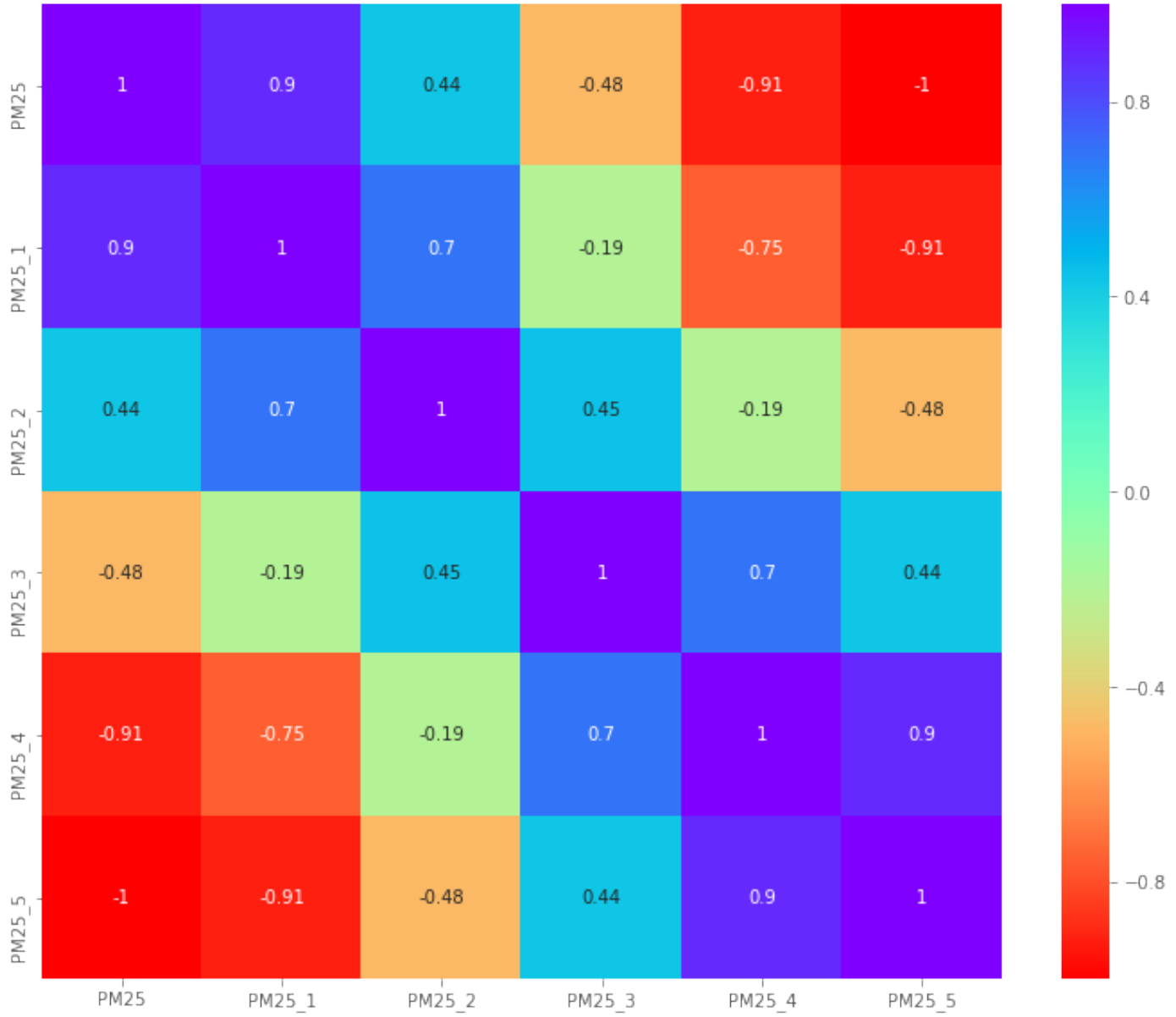

Figure 11: Correlation matrix for shifted features of PM2.5

It is seen from the correlation matrix between the observed $PM_{2.5}$ values and the time shifted values for up to 5 hours that the correlation goes from strongly positive to strongly negative over the chosen period of time. This was decided as the optimal shift period as adding extra hours would be redundant and have no noticeable performance increase and fewer hours would obviously lower the model performance due to incomplete data.

4. Prediction Models

4.1. Data preparation

Before the final dataset is fed into the prediction model, some preparation has to be done. This includes deleting all rows containing null values, which was decided to be the best approach to deal with missing values within the context of the dataset used. The final dataset before dropping null values has a total of 17520 records, which is an hourly observation for each of the feature for the time period of two year. There are 53 features in total after the feature engineering steps detailed in the previous section. The date and day of week features are to be dropped because the relevant date features such as hour, month and numerical values for day of week have already been extracted.

```
#Shape of final dataset

data.shape
```

```
(17520, 53)
```

```
# Show rows where any cell has a NaN
data[data.isnull().any(axis=1)].shape
```

```
(6314, 53)
```

```
#Deleting rows with nulls, dropping unwanted features
data = data.dropna(axis=0).reset_index(drop=True)
data = data.drop(['DATE','DAY'], axis=1)
data_final.shape
```

```
(11206, 51)
```

After the data preparation steps, there are a total of 11206 records and 51 features in the final dataset.

The next step is to separate the output, which is the observed value of $PM_{2.5}$ from the input features.

```
#Extracting inputs and output

y = data[['PM25']]
X = data.drop(['PM25'], axis=1)
```

The data is split into testing data and training data using the train_test_split function in Sci-Kit Learn. A 70-30 split is chosen considering the size of the dataset in order to avoid over or underfitting.

```
#Splitting into test and training
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.3, random_state=1234)
```

4.2. Performance metrics

The following performance metrics are used to evaluate the performance of the models discussed in this project.

RMSE – Root mean square error is an aggregation of the deviation of predicted values from the actual values over multiple observations. Lower the RMSE the better
MAE - Mean absolute error represents the average distance between the predicted and actual values. Lower the MAE the better.
R^2 - Coefficient of determination is used to measure how close the predicted data points are to the fitted regression line. It is a measure of the goodness of fit. A higher R^2 value indicates better model performance.

4.3. Classification vs Regression

One of the main aims of this project is to compare the performance of classification and regression in the field of air quality prediction. This problem is domain-specific since a classification model would only predict the pollution band given a set of conditions. A regression model on the other hand, would give a numerical value that could be used for further analysis and visualization. The domain specificity of this problem arises from the fact that the general public is only concerned about the general levels of pollution, and their health implications. A classification model would be expected to perform well in this case since numerical analysis of the predicted data is not required.
Regression models are rarely used in the domain of air quality prediction because the general use-cases do not warrant specific value prediction. Although, using a regression model would provision means for further detailed data analytics. [6]
A field for the band of air quality is created according to the thresholds defined for $PM_{2.5}$ by the LAQN. This is done using the cut function available in Pandas.

```
##### Classifying AQI based on PM2.5 values
data['BAND'] = pd.cut(data['PM25'],[-1,11.9,23.9,35.9,41.9,47.9,53.9,58.9,64.9,70.9,102],
                      labels=[1,2,3,4,5,6,7,8,9,10])
```

Band	Index	PM2.5 Particles
		µg m-3
Low		
	1	0-11
	2	12-23
	3	24-35
Moderate		
	4	36-41
	5	42-47
	6	48-53
High		
	7	54-58
	8	59-64
	9	65-70
Very High		
	10	71 or more

Table 10: PM2.5 bands [4]

4.4. Linear regression

The first regression model used for predicting the $PM_{2.5}$ levels is linear regression. It is generally used for predictive analysis and is used as the base model for performance comparison.

Linear regression is used for finding the relationship between

1. Independent variable called predictor.
2. Dependent variable known as response.

Linear regression can be represented as

$$Y = B_0 + B_i * X_i$$

Where,

Y - Dependent variable

B_0 - Bias

B_i – Weights for 'i' features

X_i – 'i' independent variables

Linear model library from Sci-Kit learn is used for implementing linear regression.

```
#Linear Regression

from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

# Create linear regression object
regr = LinearRegression()
```

The fit method is used to train the model using the regressor object.

```
# Train the model using the training sets
regr.fit(X_train, y_train)
```

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
         normalize=False)
```

The predicted values of the model are generated by running the predict function on the test input.

```
# Make predictions using the testing set
lin_pred = regr.predict(X_test)
```

Performance Evaluation:

Root mean squared error	3.01
Mean absolute error	2.07
R-squared	0.92

Table 11: Performance evaluation - Linear regression

The graph below shows the distance between predicted and actual values for the Linear Regression model.

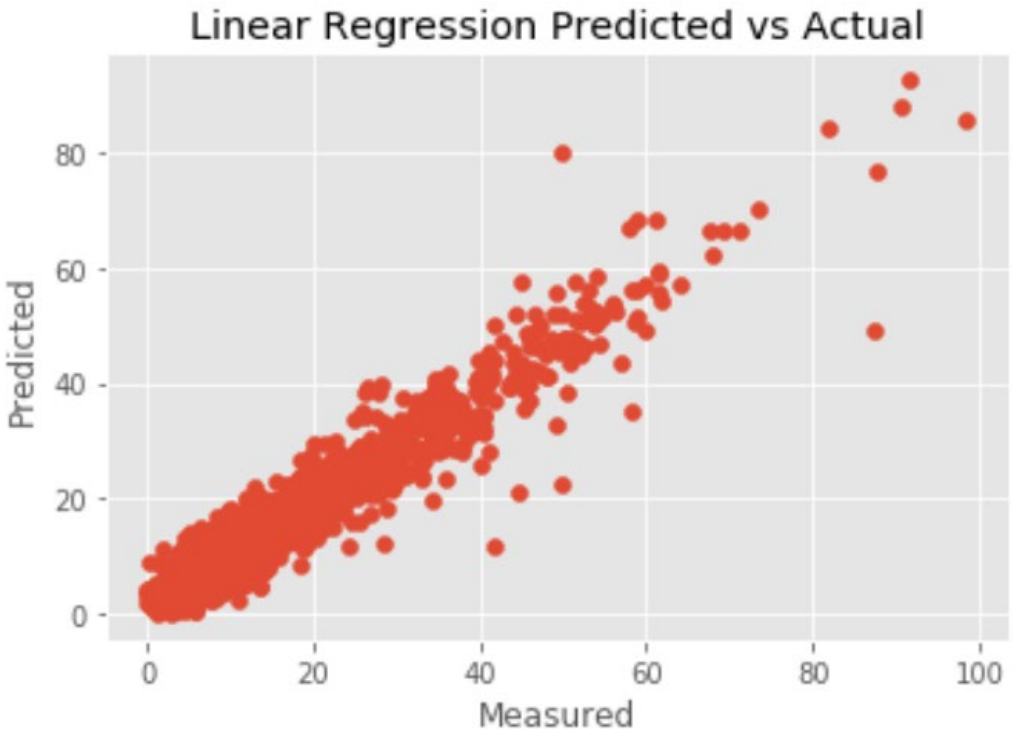

Figure 12: Linear Regression Predicted vs Actual

4.5. XGBOOST

A decision tree contains all possible consequence for their respective decisions. This includes cost of the resource, outcomes and utilities. Whereas, gradient boosting decision tree is more suitable technique for regression and classification problems. It builds decision trees in a stage wise fashion and optimizes using differentiable loss function. Although both these models were tested for prediction, the best performance was observed with the XGBoost algorithm, which is detailed below:
XGBoost is an end-to-end lifting tree system which is a scalable learning system for tree boosting [7]. This comparatively performs better than the GBDT algorithm. [8]
The model runs effectively run ten times faster when on a single node and is capable of being scaled to distributed and other memory limited settings. [3]. It is possible due to the algorithmic optimizations. A novel tree learning algorithm is designed to handle incomplete datasets.

$$\hat{y}_i = \phi(\mathbf{x}_i) = \sum_{k=1}^{K} f_k(\mathbf{x}_i), \quad f_k \in \mathcal{F},$$

For a given data set with n examples and m features
D = {(xi, yi)} (|D| = n, xi ∈ Rm, yi ∈ R)
The tree ensemble model uses K additive functions to predict the output. yˆi = φ(xi) = XK k=1 fk(xi), fk ∈ F, (1) where F = {f(x) = wq(x)} (q: R m → T, w ∈ R T) is the space of regression trees (also known as CART). Here q represents the structure of each tree that maps an example to the corresponding leaf index. T is the number of leaves in the tree. Each fk corresponds to an independent tree structure q and leaf weights w. Unlike decision trees, each regression tree contains a continuous score on each of the leaf, we use wi to represent score on ith leaf.
Since the model contains a fraction of missing values, XGBoost could be an ideal algorithm to handle it. According to [3] when the data is parse, an instance is supposed to be filled in a default direction. In every branch, there are two directions of right and left split. The optimal directions are learned from the training dataset.

4.5.1. XGBoost Regression

The XGBRegressor function from Sci-Kit Learn is used to make prediction on the test data.

```
#XGBoost

from xgboost.sklearn import XGBRegressor

#Fitting XGB regressor
xboost = XGBRegressor(n_estimators=200)
```

```
xboost.fit(X_train, y_train)
```

```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
       colsample_bytree=1, gamma=0, importance_type='gain',
       learning_rate=0.1, max_delta_step=0, max_depth=3,
       min_child_weight=1, missing=None, n_estimators=200, n_jobs=1,
       nthread=None, objective='reg:linear', random_state=0, reg_alpha=0,
       reg_lambda=1, scale_pos_weight=1, seed=None, silent=True,
       subsample=1)
```

Performance Evaluation:

Root mean squared error	2.93
Mean absolute error	1.98
R-squared	0.92

Table 12: Performance evaluation - XGBoost Regression

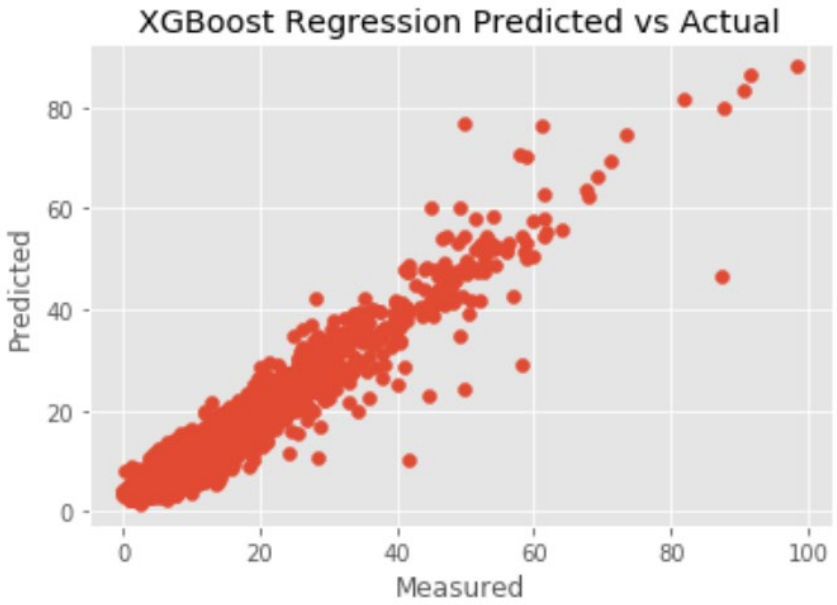

Figure 13: XGBoost Regression Predicted vs Actual

4.5.2. XGBoost Classification

This project aims to compare the performance of a regression model against a classification model. Hence, the entire data set is fed into the XGBoost Regression model detailed in the previous section to obtain predicted PM2.5 values which are then categorized into pollution bands. This is compared with a direct XGBoost classifier model which is detailed below:

```
#Classification model
y = data['BAND'].astype('int64')
X = data.drop(['BAND'], axis=1)
```

```
#XGBoost

from xgboost.sklearn import XGBClassifier

#Fitting XGB regressor
xboost = XGBClassifier(max_depth=3, n_estimators=300, learning_rate=0.05)
```

```
seed = 7
test_size = 0.33
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=test_size, random_state=seed)
```

```
xboost.fit(X_train, y_train)
```

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
       colsample_bytree=1, gamma=0, learning_rate=0.05, max_delta_step=0,
       max_depth=3, min_child_weight=1, missing=None, n_estimators=300,
       n_jobs=1, nthread=None, objective='multi:softprob', random_state=0,
       reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
       silent=True, subsample=1)
```

A high R^2 of 0.9997 is achieved with the confusion matrix shown below:

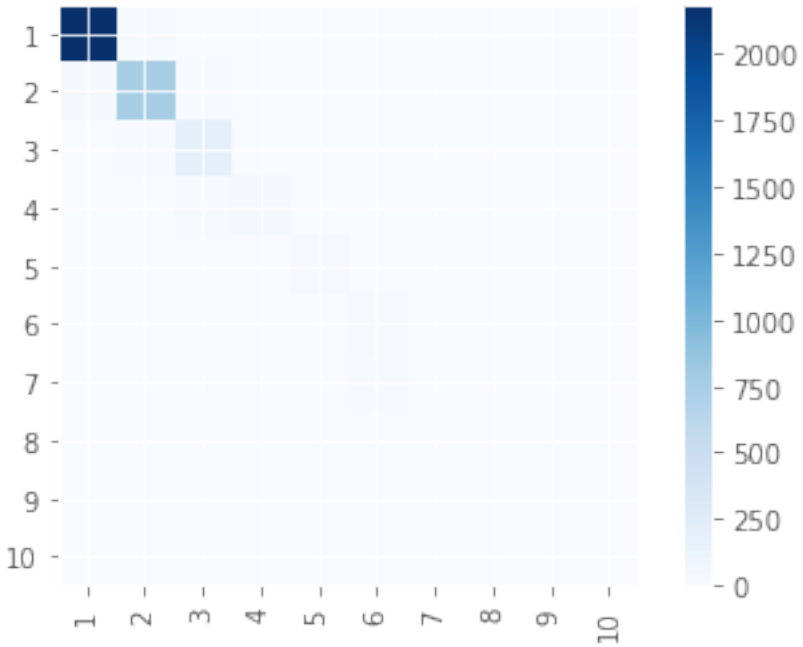

Figure 14: Confusion matrix - XGBoost Classifier

4.5.3. XGB Regressor as a Classifier

The predicted results from the XGB regressor are cut into 10 different bands representing the 10 levels of AQI and these bands are compared with the actual values by cutting the PM2.5 values in the ground truth in the same way and the below confusion matrix is obtained.

```
#Predicting values of the entire input set using XGBRegressor
reg_pred = xboost.predict(X)
#Assigning bands to predicted values
regr_pred['BAND'] = pd.cut(regr_pred['0'],[-1,11.9,23.9,35.9,41.9,47.9,53.9,58.9,64.9,70.9,102],labels=[1,2,3,4,5,6,7,8,9,10])
#Setting ground truth and predictions
gt = data['BAND'].astype('int64')
pred = regr_pred['BAND'].astype('int64')
accuracy_score(gt,pred)
```

This approach yielded an R^2 value of 0.8769 and the confusion matrix is presented below:

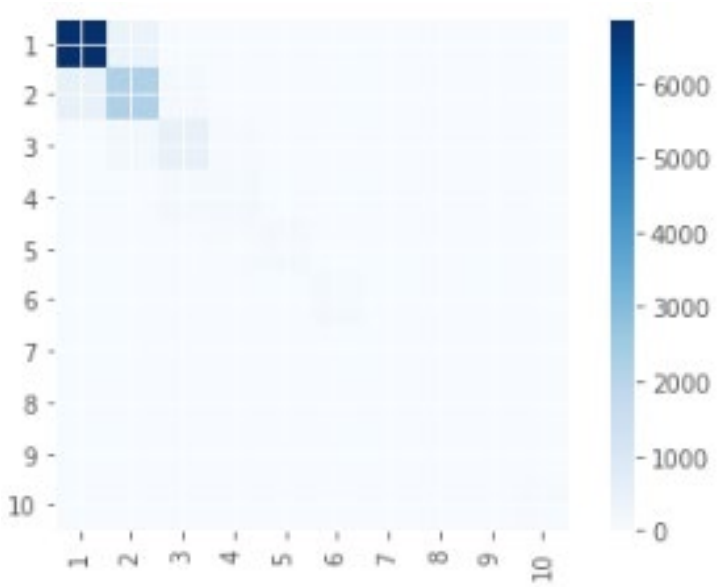

Figure 15: Confusion matrix - XGBoost Regressor as a classifier

4.6. Performance Comparisons

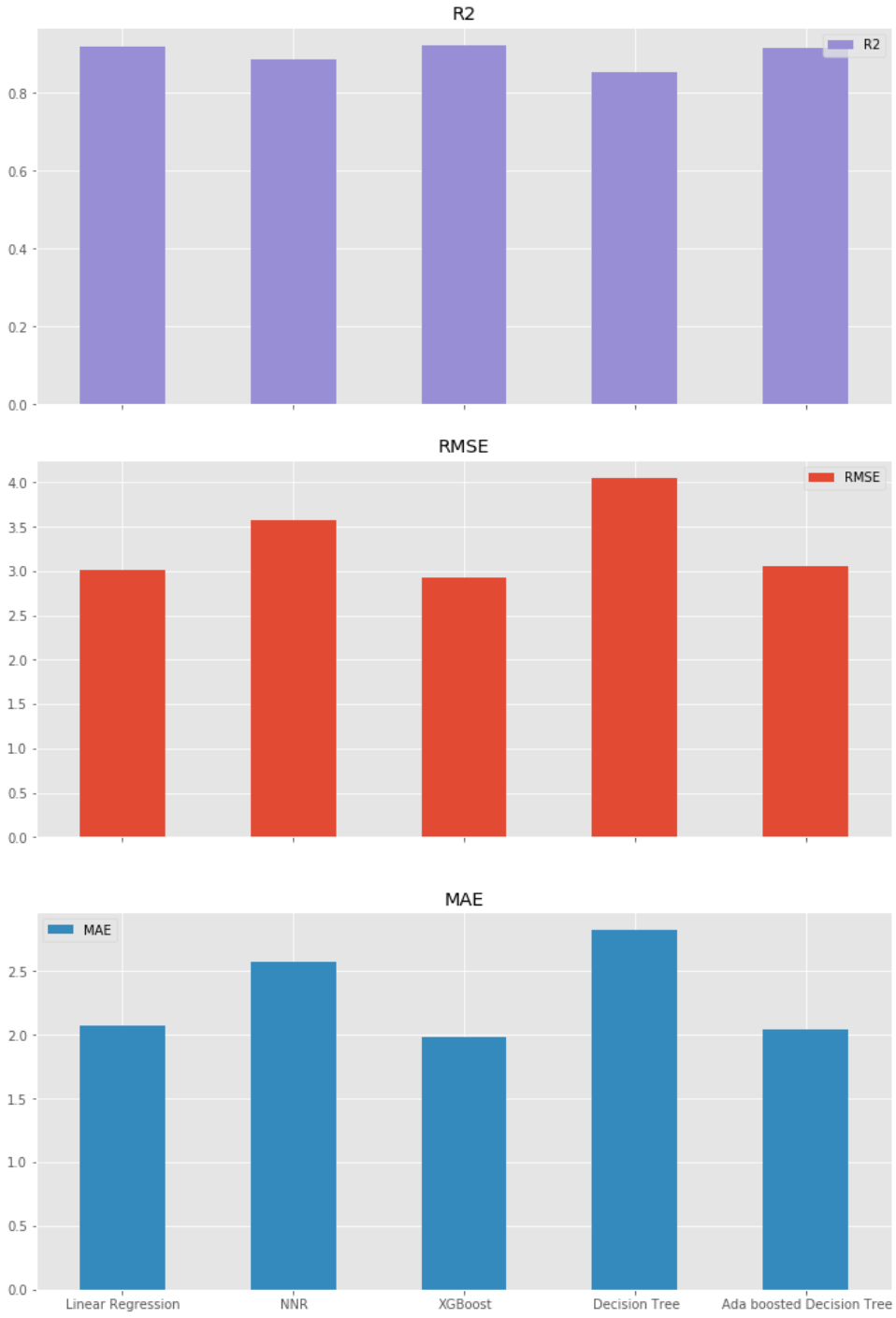

Figure 16: Performance comparison of different models

A performance comparison of the different regression models tested in this project is made and their performance metrics are presented in the form of bar plots. It can be seen that, there is no drastic difference in the performance of the different models and all the models performs more or less equally with good numbers. Although XGBoost and Ada Boosted Decision tree had a close call, XGBoost topped the table with a lesser RMSE and better score.

5. Conclusion

Throughout this project, several models which can predict Pm2.5 levels and classify them into different pollution bands were experimented and their performance was successfully evaluated. The exploratory data analysis and feature engineering methods implemented for the prediction models revealed interesting correlations between weather and pollution data. We obtained several notable outcomes from the predictive models that are worth being discussed.

Different approaches to handle null values yielded varied performance from each of the models, however simply dropping the records that had null values seemed to be the best approach. Between obtaining the AQI by predicting the PM2.5 values and using a classifier to predict the AQI band straight away, the classifier seemed to perform better. A regression model could be used for applications in data analytics, but it is concluded that classifier models perform better for air quality prediction.

6. Future Scope

Combining the above model with a real-time API would help in predicting the pollution bands almost in real-time. Further research can be done on Geographic Information System (GIS) on Air Pollution which can help make decisions to geography based on human thinking patterns. It can also organize geographic data, to select and use specific task or project like checking Air quality at location. We can also improve our project by working with Live API and keep predicting the future values automatically.

Realtime data ingestion for air quality:
As mentioned in the challenges, the suitable form of collecting data is to obtain from the source itself. When it is obtained from the IoT devices, it is quite common to the data is delivered in the form of API. The IoT devices send data to cloud which in turn provides the API capability. This readily available API service also allows the predictions of AQI in real time.

GIS for air quality:
A geographic information system (GIS) is a technological tool for comprehending geography and making intelligent decisions. By understanding geography and people's relationship to location, we can make informed decisions about the way we live on our planet. [9]
A good GIS program can process geographic data from a variety of sources and integrate it into a map project. Many countries have an abundance of geographic data for analysis, and governments often make GIS datasets publicly available.

7. Appendix

References

[1] GOV.UK. (2019). *COMEAP: review of the UK air quality index*. [online] Available at: https://www.gov.uk/government/publications/comeap-review-of-the-uk-air-quality-index [Accessed 31 Mar. 2019].
[2] Anon, (2019). [online] Available at: https://www.londonair.org.uk/london/asp/datadownload.asp [Accessed 31 Mar. 2019].
[3] "https://www1.ncdc.noaa.gov/pub/data/ish/ish-format-document.pdf"
NOAA (2016). *Integrated surface dataset*. NOAA Centres for environmental information.
[4] Great Britain. Committee on medical effects of air pollution. (2012). *Air quality band calculation*. Available at: https://www.londonair.org.uk/london/asp/airpollutionindex.asp?la_id=&region=0&bulletin=hourly&site=&bulletindate=18/03/2019%2017:00:00&level=All&MapType=Google&VenueCode=
[5] Sen, S., Das, M. and Chatterjee, R. (2016). A Weighted kNN approach to estimate missing values. *2016 3rd International Conference on Signal Processing and Integrated Networks (SPIN)*.
[6] Martinez, N., Montes, L., Mura, I. and Franco, J. (2018). Machine Learning Techniques for PM<inf>10</inf> Levels Forecast in Bogotá. *2018 ICAI Workshops (ICAIW)*.
[7] T. Q. Chen, and C. Guestrin, XGBoost:A Scalable Tree Boosting System, in Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2016: 785-794.
[8] J. Ye, J. Chow, and J. Chen, Stochastic gradient boosted distributed decision trees, in Proceedings of the 18th ACM conference on Inf
[9] Esri.com. (2019). [online] Available at: https://www.esri.com/library/bestpractices/gis-and-science.pdf [Accessed 31 Mar. 2019].
[10] Postranecky, M. and Svitek, M. (2017). Smart city near to 4.0 — an adoption of industry 4.0 conceptual model. 2017 Smart City Symposium Prague (SCSP).

GitHub Repository: https://github.com/jamesjojijacob/data-analytics-cw