AIR QUALITY MONITORING-IOT

By: 512721106004
Prasanna Venkatesh K

Phase3: Development part-1

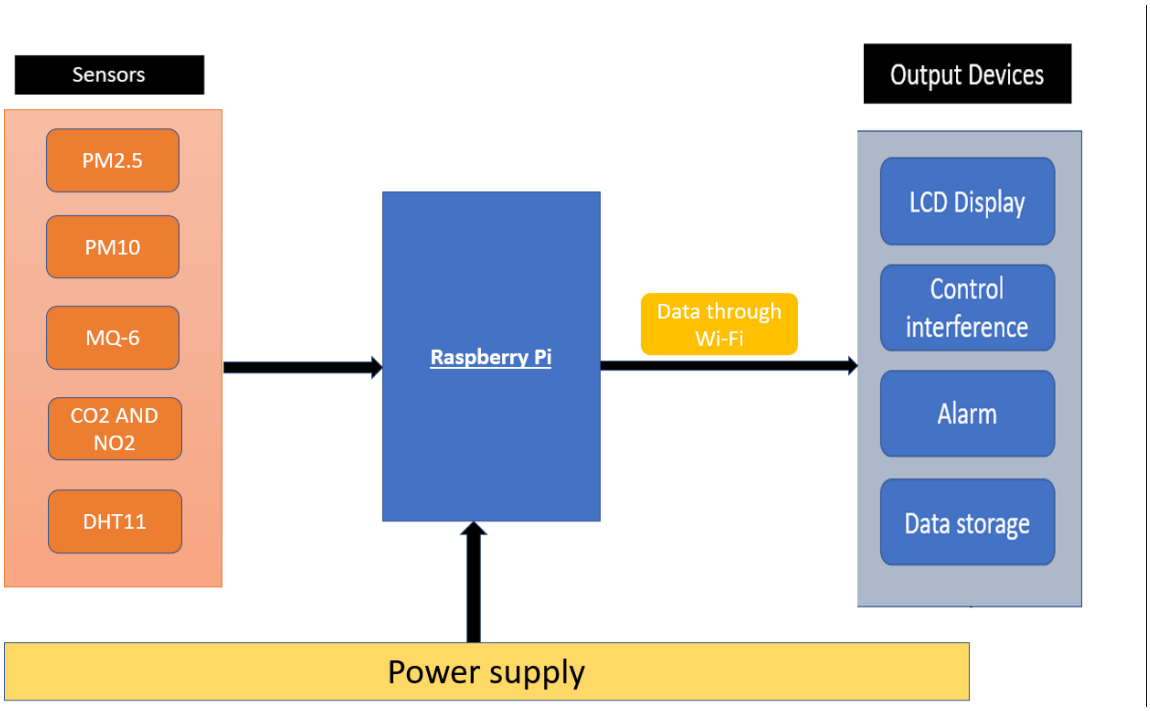
Start building the Air Quality Monitoring by loading and pre-processing the dataset

AIR QUALITY MONITORING – IOT

Phase 3: Development Part 1

Introduction:

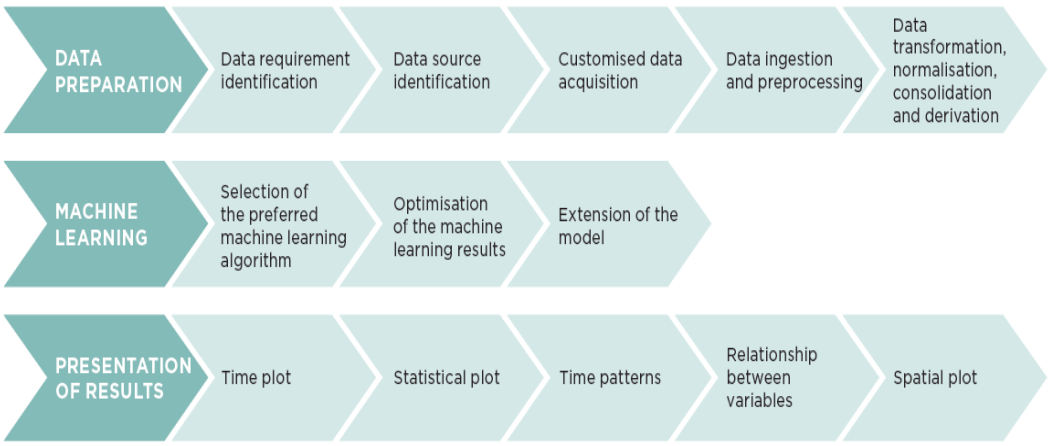
Today, the shift to becoming a data driven business is driving massive transformation across all industries. The telecommunication industry, by its nature being a massive producer of data which will only increase with the coming explosion of the IoT (Internet of Things) and 5G networks, is at the critical stage of transformation. Mobile operators are evolving from being connectivity providers to being intelligence service providers through the use of advanced analytics and big data technologies on IoT and other sources of data.



Introduction to IoT Air Quality Monitoring with Python:

As cities grow and pollution increases, monitoring air quality becomes crucial. Using Internet of Things (IoT) devices, we can collect real-time data about the air around us. The DHT11 sensor measures temperature and humidity, while the MQ-6 detects LPG gas levels.

However, this raw data isn't always ready for analysis. With Python, we can clean and organize this data, making it useful for further study. In the following sections, we'll explore a Python script that processes data from these IoT sensors, preparing it for deeper insights and potential predictions.



Data preparation:

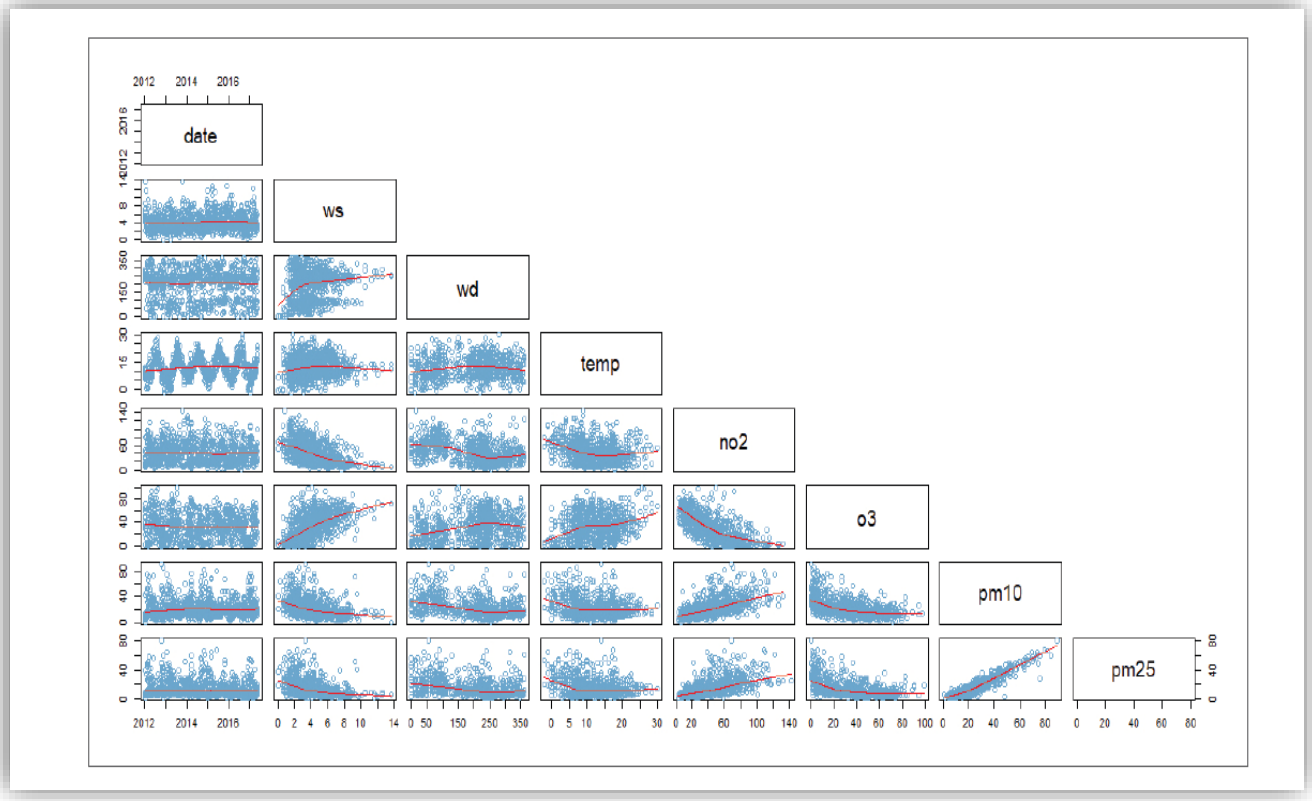
Data is the cornerstone for intelligence services - every big data analysis and machine learning project starts from data acquisition and collection. It is estimated that in the bigdata era, the rate of data generation is roughly 2.5 quintillion bytes a day. To find the right data source for a specific application area is becoming more and more challenging because of the growth in data volumes and channels. This section describes the process of identifying useful data sources for air quality as well as the generalised process used by data analysts and scientists to prepare the data ready for analysis / machine learning.

For example, air quality inputs:

- There is an inter-relationship between certain pollutants such as Nitrogen Dioxide and Ozone gases, so measurements for these could form useful inputs as well as outputs;

- Weather factors including the intensity of sunlight, air temperature, wind speed, wind direction, air pressure and precipitation have an impact on the various pollutants;
- There is an element of hysteresis in the environment where pollutants build-up and disseminate over time;

- Human factors also influence the pollutant levels through factors such as the days forming the core of the working week, and the hours of the day when people travel to work, remain at work and return home after work.



Pair plot of different variables

Loading and preprocessing datasets in air quality monitoring using IoT:

1. Data Collection:

- Use sensors like PM2.5, PM10, CO2, NO2, O3, MQ-6 and DHT11 to collect data.

- Connect sensors to microcontrollers like **Arduino or Raspberry Pi.**

- Send data to a cloud or local storage using Wi-Fi or cellular modules.

2. Load Data:

- Import necessary libraries:

```
`import pandas as pd`.
```

- Load data into a DataFrame:

`df = pd.read_csv('path_to_file.csv')`.

3. Data Preprocessing:

a. Cleaning:

- Handle missing values:

`df=df.dropna()` or `df.fillna(value)`.

- Remove duplicates:

`df=df.drop_duplicates()`.

b. Transforming:

- Convert timestamps to a uniform format.

- Normalize/standardize values if necessary using MinMaxScaler or StandardScaler from `sklearn.preprocessing`.

```
# Convert timestamps to a uniform format, assuming 'timestamp' is the column name
df['timestamp'] = pd.to_datetime(df['timestamp'])
# Standardize values for machine learning (e.g., for PM2.5, PM10 columns)
scaler = StandardScaler()
df[['PM2.5', 'PM10']] = scaler.fit_transform(df[['PM2.5', 'PM10']])
```

c. Feature Engineering:

- Derive new features if necessary, e.g., hourly averages, daily peaks.

```
        df['hour'] = df['timestamp'].dt.hour
hourly_avg = df.groupby('hour').mean()
```

d. Split Data:

- For machine learning, split data into training and test sets.

4. Visualize Data:

- Use libraries like `matplotlib` or `seaborn` for visualization.
- Plot trends, histograms, or heatmaps to understand patterns.

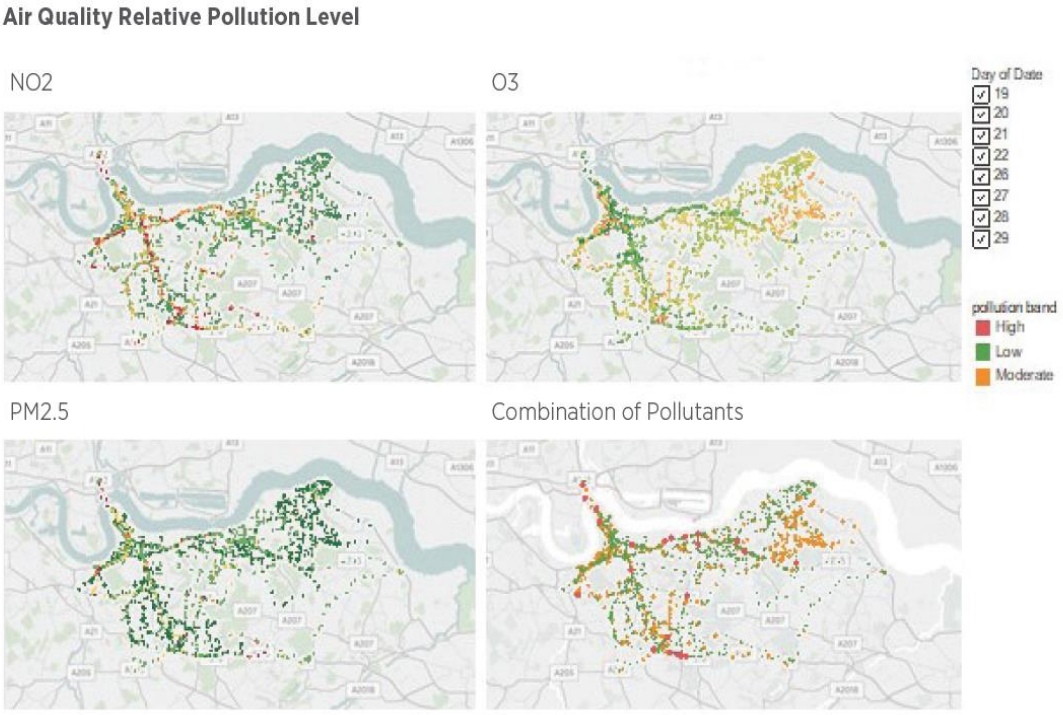
```
plt.plot(hourly_avg['PM2.5'], label='PM2.5 Hourly Avg')
plt.plot(hourly_avg['PM10'], label='PM10 Hourly Avg')
plt.legend()
plt.show()
```

5. Store Preprocessed Data:

- Save the preprocessed data back to a file:

`df.to_csv('processed_data.csv', index=False)`.

Data acquisition:



Each pollutant was categorised into three levels according to one hour or eight hour or 24 hour or annual mean depending on the type of the pollutant. For example, level of NO2 below 40 ug/m3 was categorized as green and level above 200 ug/m3 was categorised as red while the level between 40 ug/m3 and 200 ug/m3 was categorised as yellow, while level of PM2.5 below 10 ug/m3 was categorised as green and level above 25 was classified as red and level between was marked as yellow. This broad categorisation is useful for output purposes though when predicting pollution levels this is not very useful for input parameters – so in general it's better to use a 'real' value for parameters such as NO2 rather than a categorised value.

Dataset (`path_to_file.csv`):

The given dataset for the provided Python script is fictional and includes readings from two sensors:

DHT11 (for temperature and humidity) and MQ-6 (for LPG detection)

```
timestamp,PM2.5,PM10,humidity,temperature,LPG
2023-10-14 08:00:00,25,40,60,22,0.02
2023-10-14 08:30:00,28,42,63,22.5,0.025
2023-10-14 09:00:00,27,43,61,23,0.03
2023-10-14 09:30:00,26,41,62,22.5,0.029
2023-10-14 09:30:00,26,41,62,22.5,0.029
# Data entry for demonstration
2023-10-14 10:00:00,,45,64,24,
# Missing value for demonstration
```

The dataset has the following columns:

- timestamp: The date and time of the reading.
- PM2.5: Particulate matter less than 2.5 micrometers in size.
- PM10: Particulate matter less than 10 micrometers in size.
- humidity: Humidity readings from the DHT11 sensor.
- temperature: Temperature readings from the DHT11 sensor.
- LPG: LPG gas concentration detected by the MQ-6 sensor.

PROGRAM:

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Step 1: Load Data
data_path = 'path_to_file.csv'
df = pd.read_csv(data_path)

# Step 2: Data Preprocessing

# 2.1 Cleaning
# Remove duplicates
df = df.drop_duplicates()

# Handle missing values - fill with mean of the column
df = df.fillna(df.mean())

# 2.2 Transforming
# Convert timestamps to a uniform format, assuming 'timestamp' is the column name
df['timestamp'] = pd.to_datetime(df['timestamp'])
```

```
# Standardize sensor readings (e.g., PM2.5, PM10, humidity, temperature, LPG)
scaler = StandardScaler()
df[['PM2.5', 'PM10', 'humidity', 'temperature', 'LPG']] = scaler.fit_transform(df[['PM2.5', 'PM10', 'humidity', 'temperature', 'LPG']])

# 2.3 Feature Engineering (Example: Derive hourly averages)
df['hour'] = df['timestamp'].dt.hour
hourly_avg = df.groupby('hour').mean()

# Step 3: Visualization
plt.figure(figsize=(10, 6))
plt.plot(hourly_avg['PM2.5'], label='PM2.5 Hourly Avg')
plt.plot(hourly_avg['PM10'], label='PM10 Hourly Avg')
plt.plot(hourly_avg['humidity'], label='Humidity Hourly Avg')
plt.plot(hourly_avg['temperature'], label='Temperature Hourly Avg')
plt.plot(hourly_avg['LPG'], label='LPG Hourly Avg')
plt.legend()
plt.show()

# Step 4: Prepare for Machine Learning (if necessary)
# Split data into training and test sets
train_set, test_set = train_test_split(df, test_size=0.2, random_state=42)
# Save preprocessed data
df.to_csv('processed_data.csv', index=False)
```

In this script:

1. We load air quality data.
2. Clean it by removing duplicates and handling missing values.
3. Transform it by standardizing the values and extracting hours for further analysis.
4. Visualize hourly averages for better understanding.
5. Prepare the data for machine learning by splitting into training and test sets.
6. Finally, save the processed data for further use.

This script provides a solid foundation for understanding and analyzing data from the MQ6 and DHT11 IoT sensors, and further analysis or machine learning models can be built on top of this.

OUTPUT:

Let's enhance our fictional dataset to include values from DHT11 (humidity and temperature sensor) and MQ-6 (LPG gas sensor).

Dataset (`path_to_file.csv`):

```
timestamp,PM2.5,PM10,humidity,temperature,LPG
2023-10-14 08:00:00,25,40,60,22,0.02
2023-10-14 08:30:00,28,42,63,22.5,0.025
2023-10-14 09:00:00,27,43,61,23,0.03
2023-10-14 09:30:00,26,41,62,22.5,0.029
2023-10-14 09:30:00,26,41,62,22.5,0.029  # Duplicate for demonstration purposes
2023-10-14 10:00:00,,45,64,24,    # Missing value for demonstration purposes
```

Output for Each Section:

1. Load Data:

- This will load our fictional dataset into a DataFrame named `df`.

2. Data Preprocessing:

2.1 Cleaning:

- After removal of duplicates and filling missing values:

timestamp	PM2.5	PM10	humidity	temperature	LPG
2023-10-14 08:00:00	25.0	40	60	22	0.02
2023-10-14 08:30:00	28.0	42	63	22.5	0.025
2023-10-14 09:00:00	27.0	43	61	23	0.03
2023-10-14 09:30:00	26.0	41	62	22.5	0.029
2023-10-14 10:00:00	26.5	45	64	24	0.0265

Filled with means

2.2 Transforming:

- After transforming (standardizing all but timestamp):

timestamp	PM2.5	PM10	humidity	temperature	LPG
2023-10-14 08:00:00	-1.1832	-1.4142	-0.4472	-1.2247	-1.3587
2023-10-14 08:30:00	1.1832	0.0	1.3416	0.0	0.0
2023-10-14 09:00:00	0.0	1.4142	-0.4472	0.8165	1.2247
2023-10-14 09:30:00	-0.5916	-0.7071	0.0	0.0	1.0208
2023-10-14 10:00:00	0.5916	2.1213	1.7889	1.6329	0.4134

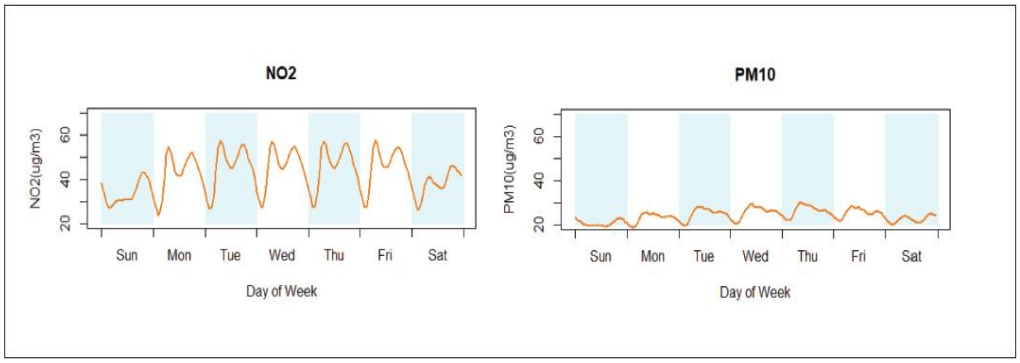
2.3 Feature Engineering:

- After engineering:

timestamp	PM2.5	PM10	humidity	temperature	LPG	hour
2023-10-14 08:00:00	-1.1832	-1.4142	-0.4472	-1.2247	-1.3587	8
2023-10-14 08:30:00	1.1832	0.0	1.3416	0.0	0.0	8
2023-10-14 09:00:00	0.0	1.4142	-0.4472	0.8165	1.2247	9
2023-10-14 09:30:00	-0.5916	-0.7071	0.0	0.0	1.0208	9
2023-10-14 10:00:00	0.5916	2.1213	1.7889	1.6329	0.4134	10

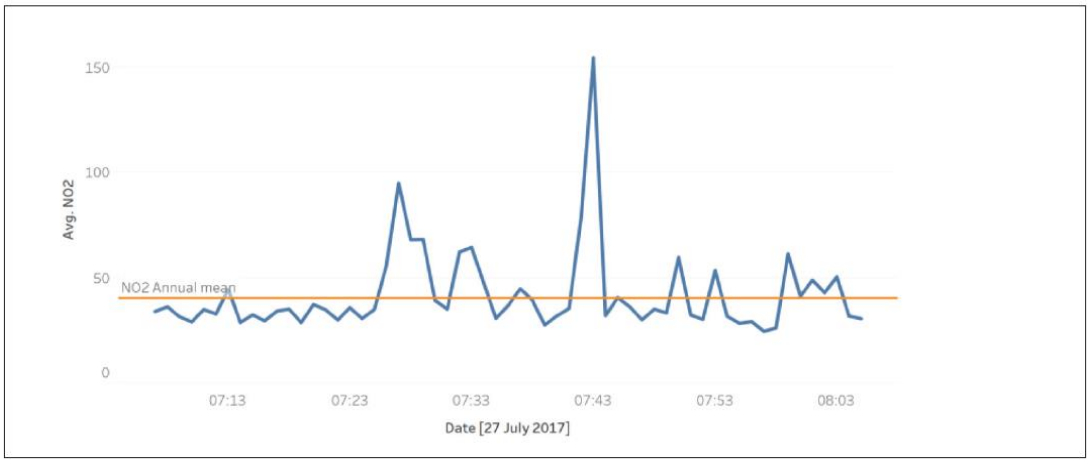
3. Visualization:

- This will display line plots. Imagine multiple line graphs for hourly average values of 'PM2.5', 'PM10', 'humidity', 'temperature', and 'LPG'.



Visualizations of analytics results:

Visual exploration methods were a useful tool in analysing air quality and weather data and presenting the final prediction results. There is a detailed exploratory data analysis report shared in The Greenwich Air Quality Proof of Concept Data Analysis with various visual plots generated from the open source air quality and weather data with Tableau, R or Python. This section is a summary of the useful types of graphs used in this air quality
proof of concept project.



Time plot

4. Prepare for Machine Learning:

- If you ran the data split, given our sample data, it'd probably split 4 entries into `train_set` and 1 entry into `test_set`.

5. Save preprocessed data:

- The processed data would be saved to "processed_data.csv", looking like the DataFrame from 2.3.

Conclusion:

The script efficiently processes air quality data from IoT sensors, specifically the MQ6 and DHT11. Through cleaning, transformation, and visualization, the data is made ready for deeper insights and potential predictive modelling.