EcoForecast: AI-powered prediction of carbon monoxide levels

Final Project Report

10.2. GitHub & Project Demo Link

1.INTRODUCTION

1.1 Project overview

Develop an AI-powered system to accurately predict carbon monoxide (CO) levels in the environment. This system aims to enhance public health and safety by providing timely and precise CO level forecasts, allowing for proactive measures to mitigate exposure risks.

1.2 Objectives

The objective of this project is to develop an AI-powered predictive system that accurately forecasts carbon monoxide (CO) levels in real-time. By leveraging historical data, real-time sensor inputs, and advanced machine learning algorithms, the system aims to:

Enhance Public Safety

Provide timely and accurate predictions of CO levels to enable preventive measures and reduce exposure risks.

Support Environmental Monitoring

Offer a reliable tool for environmental agencies to monitor and manage air quality effectively.

Raise Awareness

Inform the public and relevant stakeholders about potential CO hazards through an intuitive alert system and user-friendly dashboard.

Enable Proactive Actions

Assist local authorities, healthcare providers, and the general public in making informed decisions to mitigate the impact of high CO levels on health and well-being.

Improve Urban Planning

Provide data-driven insights for urban planners to design healthier and safer urban environments.

2. Project Initialization and Planning Phase

2.1 Define Problem Statement

Carbon monoxide (CO) is a hazardous pollutant produced by various sources, including vehicle emissions, industrial processes, and residential heating. As a colorless, odorless, and tasteless gas, it poses significant health risks, such as poisoning and respiratory issues, particularly in urban areas with high traffic and industrial activities. Traditional methods of monitoring CO levels often rely on fixed sensors that provide localized, real-time data but lack predictive capabilities.

2.2 Project Proposal (Proposed solution)

- Gather data from various sources, including CO sensors, meteorological stations, traffic monitoring systems, and industrial activity reports.
- Implement pipelines to continuously ingest and update data in real-time.
- Choose appropriate machine learning algorithms such as Time Series Analysis, Regression Models, and Neural Networks (e.g., LSTM, RNN) for predictive modeling.
- Train models using historical data and validate their performance with a separate dataset to ensure accuracy and reliability.

2.3 Initial Project Planning

- The initial project planning phase involves defining the project scope, establishing the team, gathering resources, and creating a detailed project plan with timelines and milestones. This phase sets the foundation for the project's success.

3. Data Collection and Preprocessing Phase

3.1 Data Collection Plan and Raw Data Sources Identified

- The dataset for "AI-Powered predicting carbon monoxide levels" is sourced from Kaggle.
- CO concentration levels.
- Meteorological data (temperature, humidity, day, hour).
- Traffic data (vehicle counts, types of vehicles, traffic flow).
- Industrial activity data (emission levels, operational hours).
- Geospatial data (location of sensors, industrial sites, and traffic zones).

3.2 Data Quality Report

- Dimensions and implementing the improvement plan, we aim to maintain high data quality, ensuring reliable and accurate predictions from the AI-powered CO level prediction system.

3.3 Data Exploration and preprocessing

- Check data types for each column to ensure they are appropriate (e.g., numerical, categorical, datetime).
- Replace missing values using mean, median, mode, or more advanced methods like K-nearest neighbors (KNN)imputation.
- For time series data, use linear or spline interpolation to fill in missing values.

4. Model Development Phase

4.1 Feature Selection Report

- Removing irrelevant or redundant features can reduce overfitting and improve generalization.

- Using fewer features can simplify model interpretation and implementation.
- Training models on a subset of relevant features can reduce computational time and resource requirements.

4.2 Model Selection Report

- Ability of the model to accurately predict CO levels.
- Ease of interpreting how the model makes predictions.
- Capability of the model to handle large volumes of data and real-time predictions.
- Manageability of model complexity and computational resources required.
- Model's ability to generalize well to unseen data and handle noise and outliers.

4.3 Initial Model Training Code, Model Validation and Evaluation Report

- **R-squared (R2)**: Indicates the proportion of the variance in the dependent variable (CO levels) that is predictable from the independent variables (features)

5.Model Optimization and Tuning Phase

Final Model Selection Justification

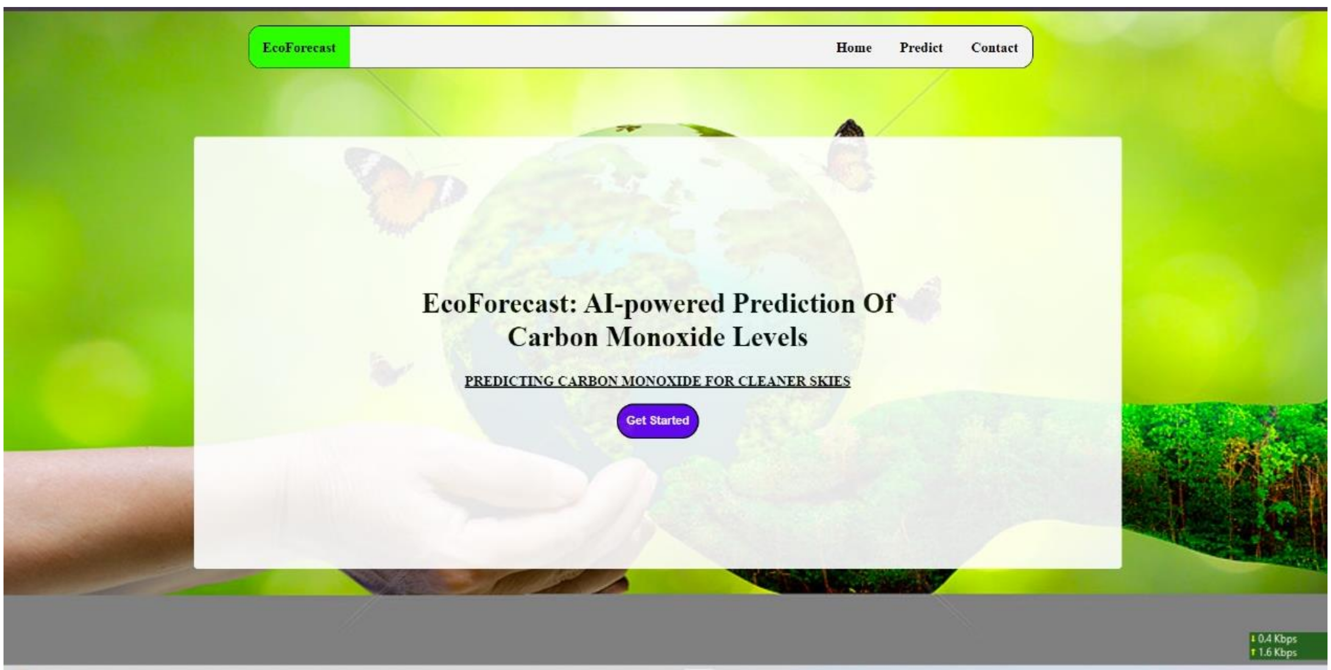
- The KNN algorithm is the final model chosen because of its best overall performance compared to the other models.

6. RESULTS

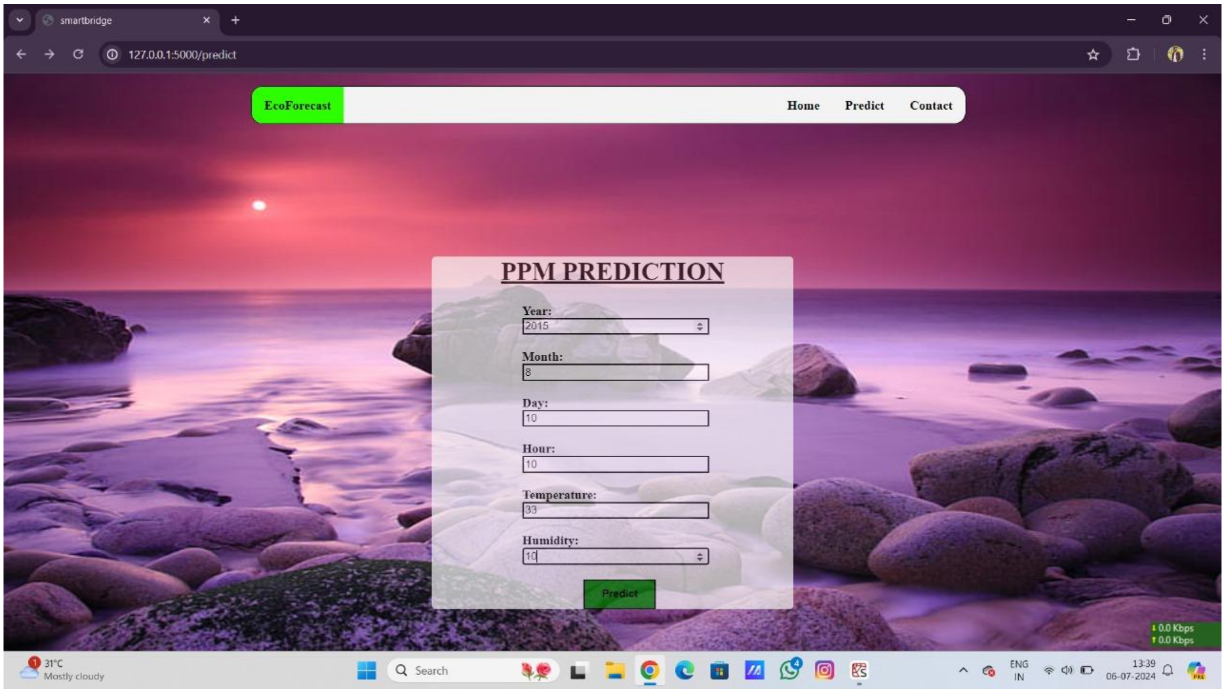
6.1 Output Screenshots

INDEX.HTML

HOME PAGE

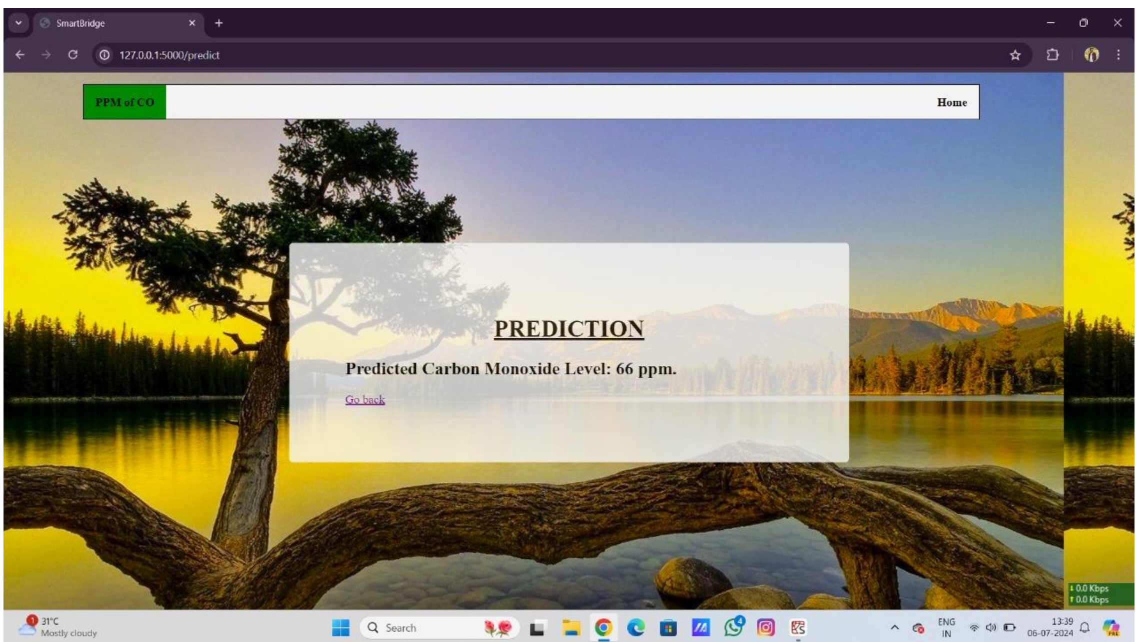


INNERPAGE.HTML:



OUTPUT PAGE:

LAST PAGE.HTML



7.ADVANTAGES AND DISADVANTAGES

ADVANTAGES:

1. **Industrial applications**
   - **Chemical Manufacturing**: CO is used as a reducing agent in the production of chemicals like methanol and acetic acid.
   - **Metal Extraction**: It is utilized in processes such as the extraction of iron from its ores in blast furnaces.
2. **Fuel and Energy**
   - **Syngas Production**: CO is a key component of synthesis gas (syngas), which is used to produce synthetic fuels and chemicals.
   - **Fuel Cells**: In some types of fuel cells, CO can be used as a fuel after being processed to reduce its harmful effects.
3. **Atmospheric Chemistry**
   - **Natural Occurrences**: CO is produced naturally in the environment through processes like wildfires and the oxidation of methane, playing a role in atmospheric chemistry.
4. **Medical Applications**
   - **Research**: In controlled environments, CO is used in medical research to study its effects on the human body and potential therapeutic uses.

Disadvantages:

1. **Health Hazards**
   - **Toxicity**: CO is a highly toxic gas that can cause serious health issues, including headaches, dizziness, nausea, and even death at high concentrations. It binds to hemoglobin in the blood more effectively than oxygen, preventing oxygen transport and leading to carbon monoxide poisoning.

- **Chronic Exposure**: Long-term exposure to low levels of CO can result in serious health problems such as cardiovascular and neurological damage.

2. **Environmental Impact**
    - **Air Pollution**: CO contributes to air pollution, particularly in urban areas with high traffic congestion. It is a major component of vehicle exhaust and industrial emissions.
    - **Greenhouse Gas**: Although not a major greenhouse gas, CO can indirectly contribute to climate change by affecting the levels of methane and other greenhouse gases in the atmosphere.
3. **Safety Risks**
    - **Combustion**: CO is a flammable gas, posing fire and explosion risks in environments where it is stored or used.
    - **Detection Difficulty**: CO is colorless, odorless, and tasteless, making it difficult to detect without specialized equipment. This increases the risk of accidental poisoning.
4. **Regulatory Concerns**
    - **Strict Regulations**: Due to its toxicity, there are stringent regulations on CO emissions and exposure in many countries. Compliance with these regulations can be costly and complex for industries.

8.Conclusion

Carbon monoxide (CO) has significant industrial and medical applications that make it valuable in certain contexts. However, its highly toxic nature and the associated health and environmental risks present serious disadvantages. Careful management, monitoring, and regulation are essential to mitigate the dangers of CO exposure and to ensure safe and beneficial use.

9. FUTURE SCOPE

1. **<u>Advanced Industrial Applications</u>**

- **Chemical Synthesis**: Continued advancements in catalysis could enhance the efficiency and selectivity of CO in the production of chemicals such as methanol, acetic acid, and hydrocarbons. Research into new catalysts and processes may open up new industrial applications for CO.
- **Metal Processing**: Innovations in metallurgical processes could further optimize the use of CO in metal extraction and refinement, potentially reducing energy consumption and environmental impact.

2. **<u>Sustainable Energy Solutions</u>**

- **Syngas and Renewable Fuels**: Development of sustainable methods for producing syngas (a mixture of CO and hydrogen) from renewable resources such as biomass or waste materials could provide a pathway to cleaner fuels and reduce reliance on fossil fuels.
- **Carbon Capture and Utilization (CCU)**: CO can play a role in CCU technologies, where it is captured from industrial emissions and converted into valuable products, thus contributing to carbon reduction efforts.

3. **<u>Environmental Monitoring and Management</u>**

- **Advanced Sensors**: Improvements in CO detection technologies, including more sensitive and accurate sensors, can enhance environmental monitoring, ensuring better control of air quality in urban and industrial areas.
- **Smart City Integration**: Integration of CO monitoring systems into smart city infrastructures could provide real-time data on air quality, enabling more responsive and effective measures to mitigate pollution.

4. **<u>Medical Research and Therapeutic Uses</u>**

- **Controlled Therapeutic Applications**: Research into the controlled use of CO as a therapeutic agent is ongoing. Low doses of CO have shown potential in treating conditions such as inflammation, ischemia-reperfusion injury, and organ transplantation.

- **Diagnostic Tools**: CO-based diagnostic tools could be developed to leverage its unique properties in detecting and monitoring certain medical conditions.

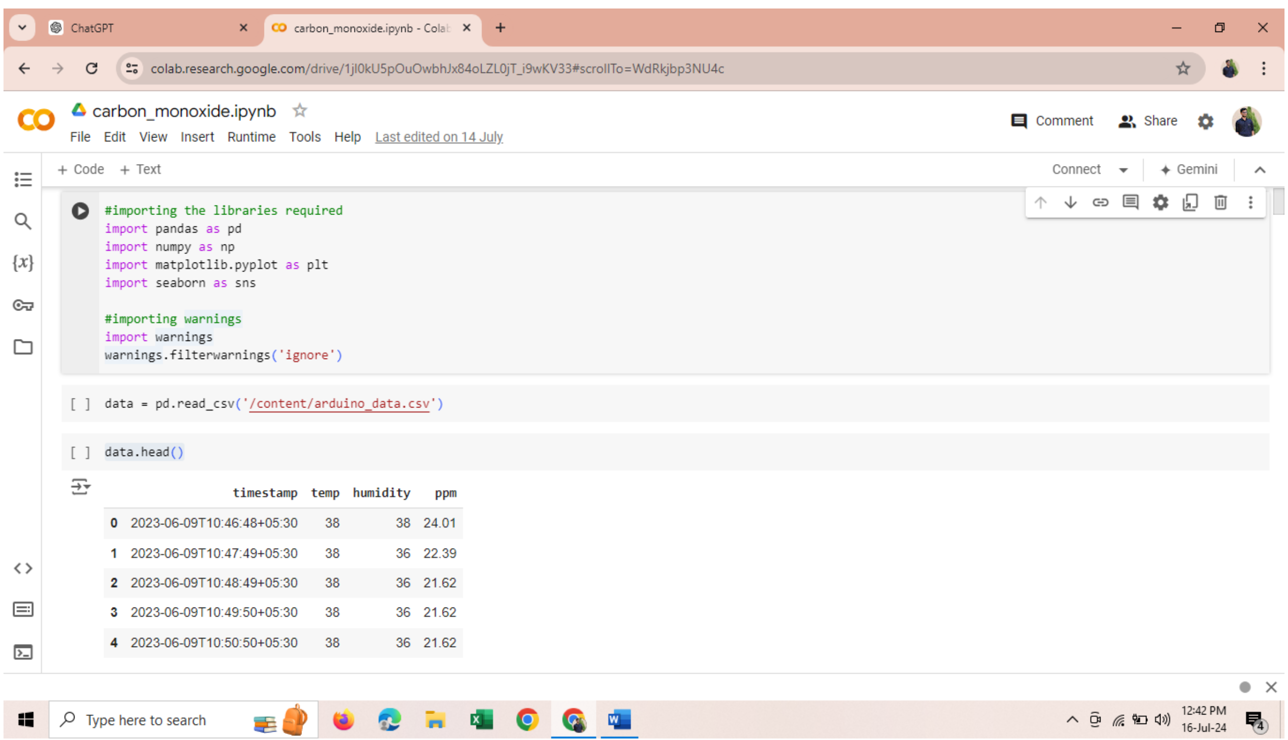
5. <u>**Combustion and Engine Technologies**</u>

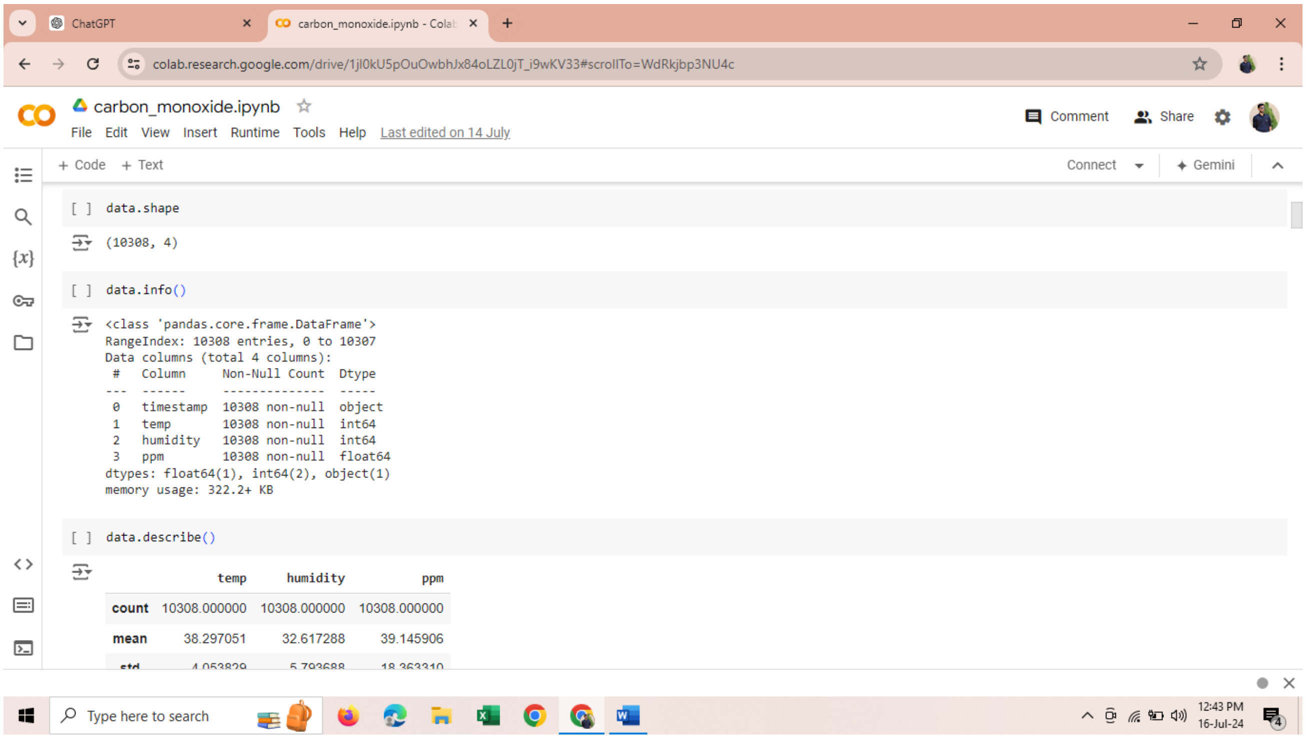
- **Clean Combustion**: Research into reducing CO emissions from combustion engines and industrial processes is critical. Innovations in engine design and fuel additives could lead to cleaner combustion with lower CO emissions.
- **Fuel Cell Technologies**: Advancements in fuel cell technologies, particularly those that can efficiently utilize CO, could enhance their viability as a clean energy source.

10.Appendix

10.1. Source Code

Code Snippets





```
data.shape
```

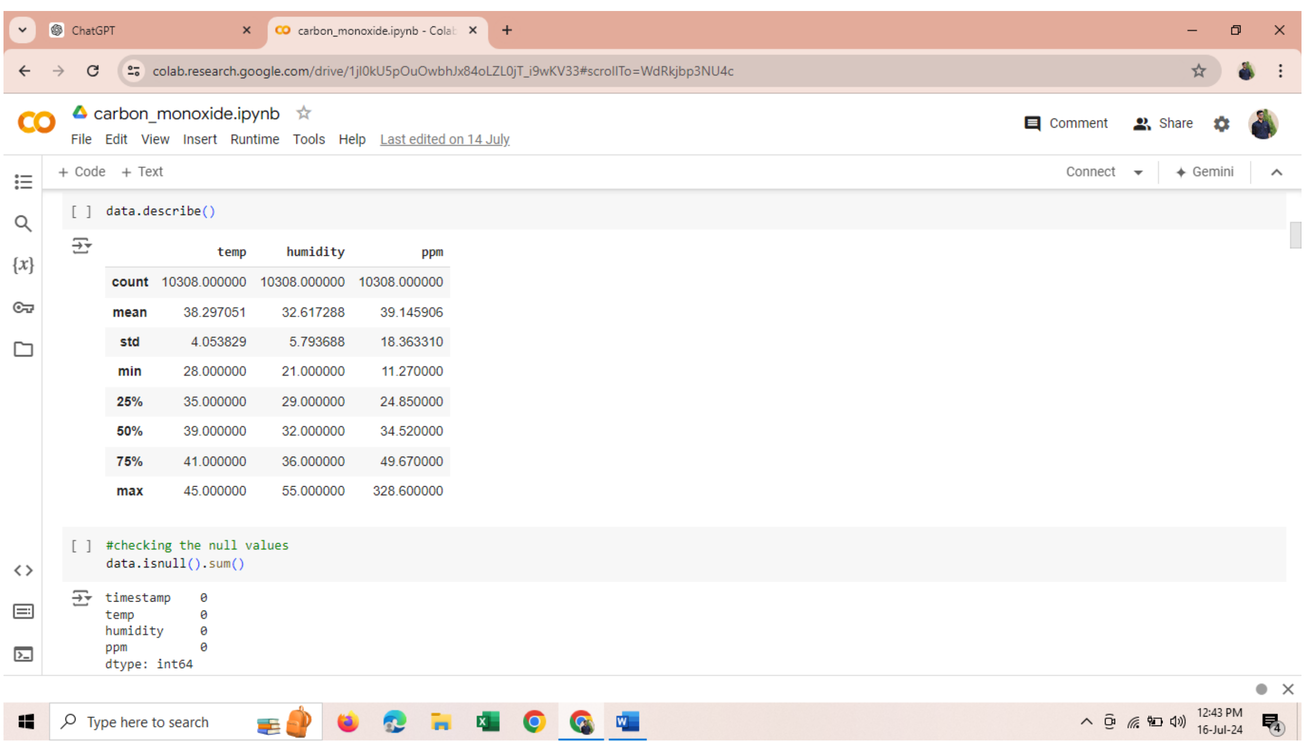
```
(10308, 4)
```

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10308 entries, 0 to 10307
Data columns (total 4 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   timestamp  10308 non-null  object
 1   temp       10308 non-null  int64
 2   humidity   10308 non-null  int64
 3   ppm        10308 non-null  float64
dtypes: float64(1), int64(2), object(1)
memory usage: 322.2+ KB
```

```
data.describe()
```

	temp	humidity	ppm
count	10308.000000	10308.000000	10308.000000
mean	38.297051	32.617288	39.145906
std	4.053829	5.793688	18.363310

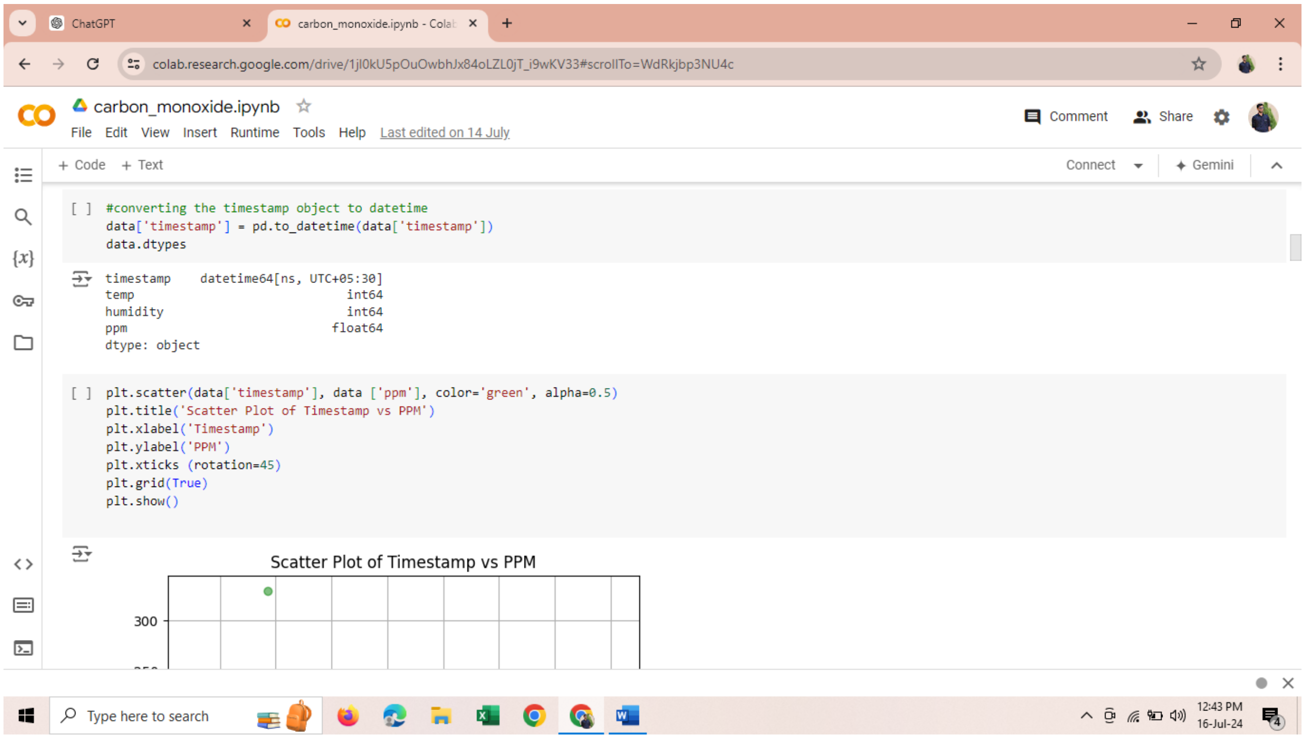


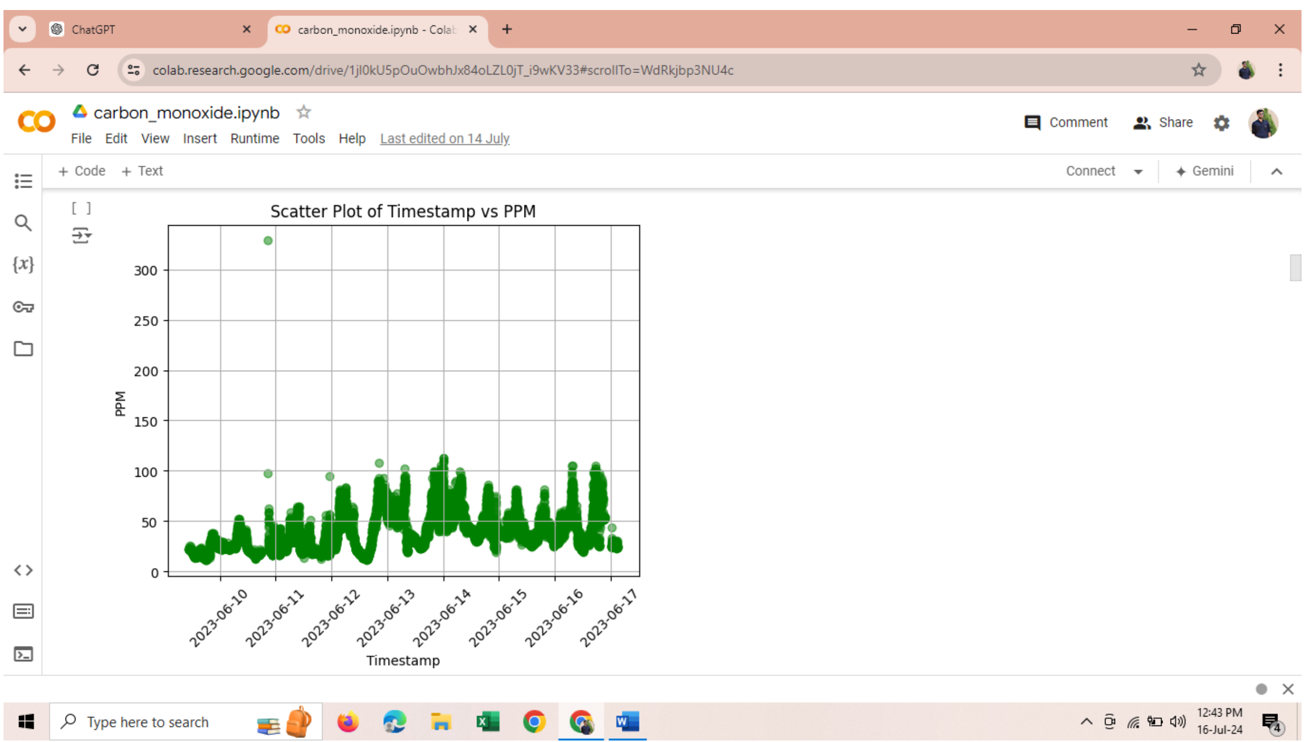
```
data.describe()
```

	temp	humidity	ppm
count	10308.000000	10308.000000	10308.000000
mean	38.297051	32.617288	39.145906
std	4.053829	5.793688	18.363310
min	28.000000	21.000000	11.270000
25%	35.000000	29.000000	24.850000
50%	39.000000	32.000000	34.520000
75%	41.000000	36.000000	49.670000
max	45.000000	55.000000	328.600000

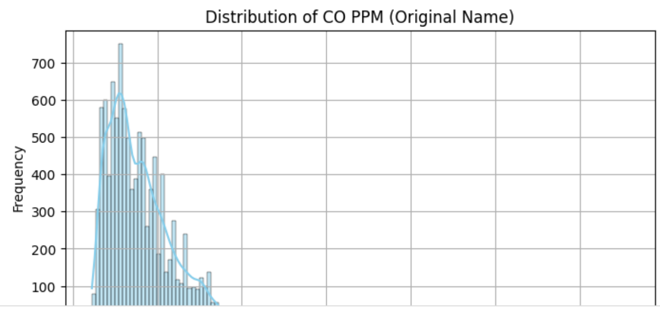
```
#checking the null values
data.isnull().sum()
```

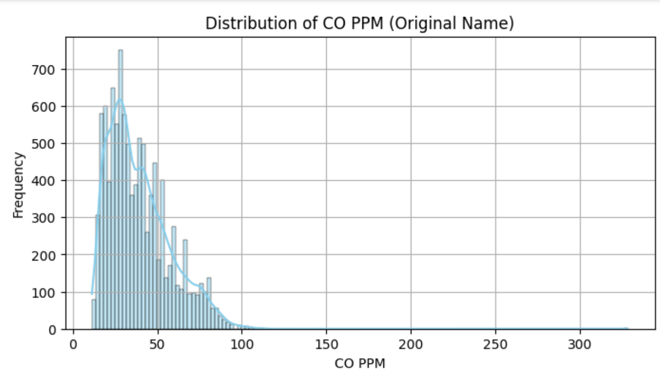
```
timestamp    0
temp         0
humidity     0
ppm          0
dtype: int64
```


ChatGPT
carbon_monoxide.ipynb - Colab
colab.research.google.com/drive/1jl0kU5pOuOwbhJx84oLZL0jT_i9wKV33#scrollTo=WdRkjbp3NU4c
carbon_monoxide.ipynb
File Edit View Insert Runtime Tools Help Last edited on 14 July
Comment Share
+ Code + Text
Connect
Gemini
#converting the timestamp object to datetime
data['timestamp'] = pd.to_datetime(data['timestamp'])
data.dtypes
timestamp datetime64[ns, UTC+05:30]
temp int64
humidity int64
ppm float64
dtype: object
plt.scatter(data['timestamp'], data ['ppm'], color='green', alpha=0.5)
plt.title('Scatter Plot of Timestamp vs PPM')
plt.xlabel('Timestamp')
plt.ylabel('PPM')
plt.xticks (rotation=45)
plt.grid(True)
plt.show()
Scatter Plot of Timestamp vs PPM
300
Type here to search
12:43 PM
16-Jul-24


ChatGPT
carbon_monoxide.ipynb - Colab
colab.research.google.com/drive/1jl0kU5pOuOwbhJx84oLZL0jT_i9wKV33#scrollTo=WdRkjbp3NU4c
carbon_monoxide.ipynb
File Edit View Insert Runtime Tools Help Last edited on 14 July
Comment Share
+ Code + Text
Connect
Gemini
Scatter Plot of Timestamp vs PPM
PPM
300
250
200
150
100
50
0
2023-06-10
2023-06-11
2023-06-12
2023-06-13
2023-06-14
2023-06-15
2023-06-16
2023-06-17
Timestamp
Type here to search
12:43 PM
16-Jul-24

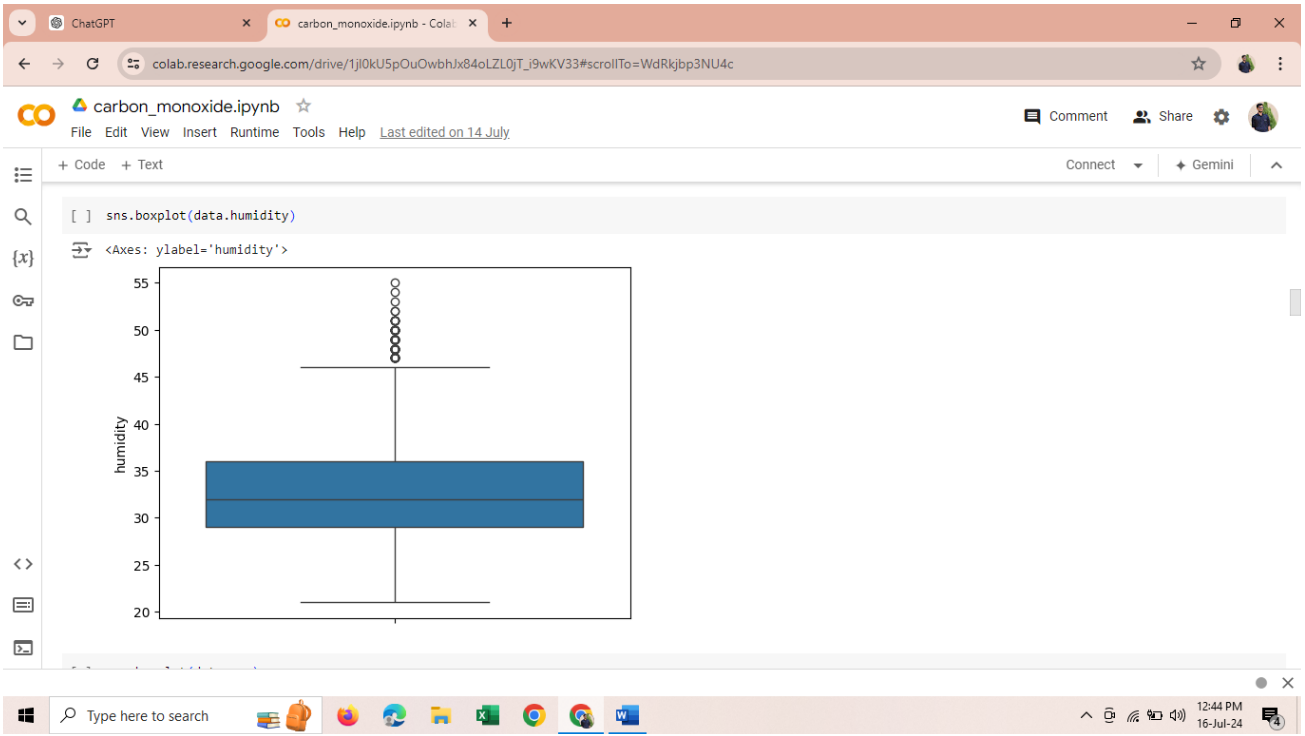
```
#histogram
plt.figure(figsize=(8,4))
sns.histplot(data['ppm'], kde=True, color='skyblue')
plt.title('Distribution of CO PPM (Original Name)')
plt.xlabel('CO PPM')
plt.ylabel('Frequency')
plt.grid(True)
plt.show()
```

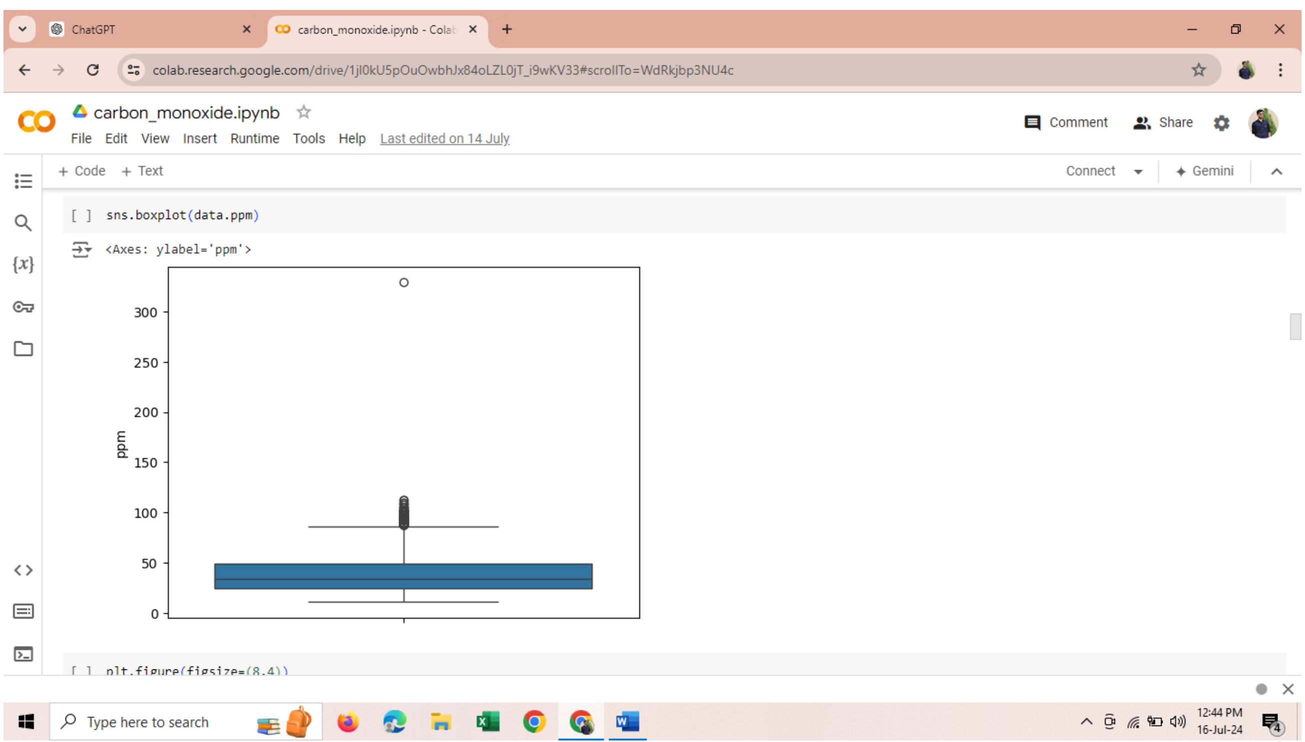




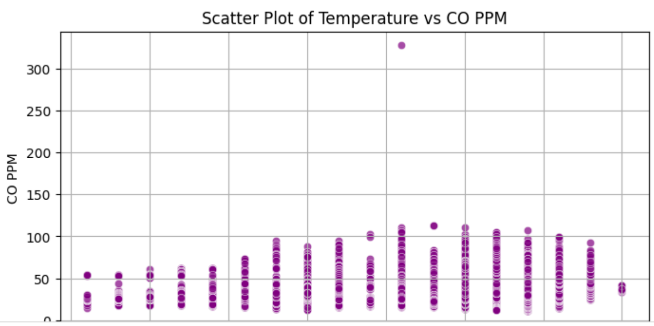

```
sns.boxplot(data.humidity)
```

```
<Axes: ylabel='humidity'>
```

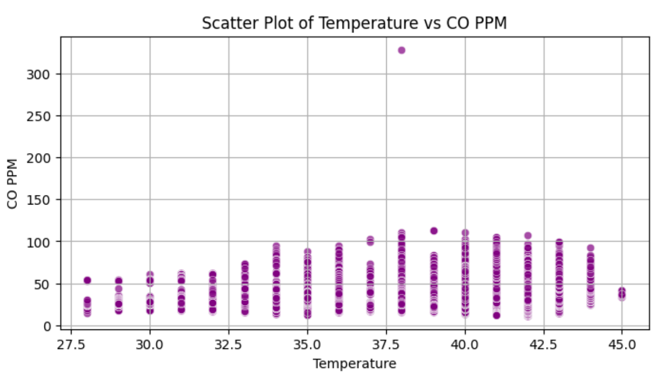

ChatGPT
carbon_monoxide.ipynb - Colab
colab.research.google.com/drive/1jl0kU5pOuOwbhJx84oLZL0jT_i9wKV33#scrollTo=WdRkjbp3NU4c
carbon_monoxide.ipynb
File Edit View Insert Runtime Tools Help Last edited on 14 July
Comment Share
+ Code + Text
Connect
Gemini
sns.boxplot(data.humidity)
<Axes: ylabel='humidity'>
humidity
55
50
45
40
35
30
25
20
Type here to search
12:44 PM
16-Jul-24


ChatGPT
carbon_monoxide.ipynb - Colab
colab.research.google.com/drive/1jl0kU5pOuOwbhJx84oLZL0jT_i9wKV33#scrollTo=WdRkjbp3NU4c
carbon_monoxide.ipynb
File Edit View Insert Runtime Tools Help Last edited on 14 July
Comment Share
+ Code + Text
Connect
Gemini
sns.boxplot(data.ppm)
<Axes: ylabel='ppm'>
ppm
300
250
200
150
100
50
0
plt.figure(figsize=(8,4))
Type here to search
12:44 PM
16-Jul-24

```
plt.figure(figsize=(8,4))
sns.scatterplot(x='temp', y='ppm', data=data, color='purple', alpha=0.7)
plt.title('Scatter Plot of Temperature vs CO PPM')
plt.xlabel('Temperature')
plt.ylabel('CO PPM')
plt.grid(True)
plt.show()
```




```
plt.show()
```




```
quant=data['humidity'].quantile(q=[0.75,0.25])
print(quant)
Q3=quant.loc[0.75]
```

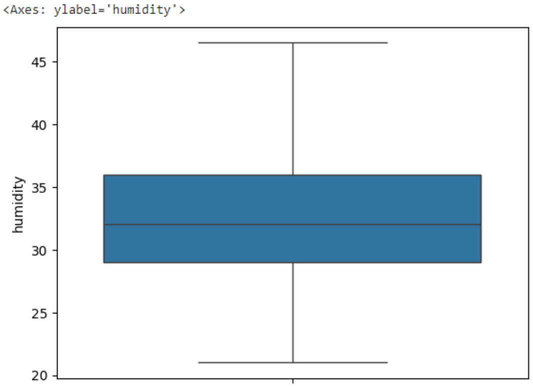
```
quant=data['humidity'].quantile(q=[0.75,0.25])
print(quant)
Q3=quant.loc[0.75]
print(Q3)
Q1=quant.loc[0.25]
print(Q1)
IQR=Q3-Q1
print(IQR)
maxwhisker=Q3+1.5*IQR
print(maxwhisker)
minwhisker=Q1-1.5*IQR
print(minwhisker)
```

```
0.75    36.0
0.25    29.0
Name: humidity, dtype: float64
36.0
29.0
7.0
46.5
18.5
```

```
data['humidity']=np.where(data['humidity']>46.5,46.5, data ['humidity'])
sns.boxplot(data['humidity'])
```


```
data['humidity']=np.where(data['humidity']>46.5,46.5, data ['humidity'])
sns.boxplot(data['humidity'])
```


```
<Axes: ylabel='humidity'>
```



```python
quant=data['ppm'].quantile(q=[0.75,0.25])
print(quant)
Q3=quant.loc[0.75]
print(Q3)
Q1=quant.loc[0.25]
print(Q1)
IQR=Q3-Q1
print(IQR)
maxwhisker=Q3+1.5*IQR
print(maxwhisker)
minwhisker=Q1-1.5*IQR
print(minwhisker)
```

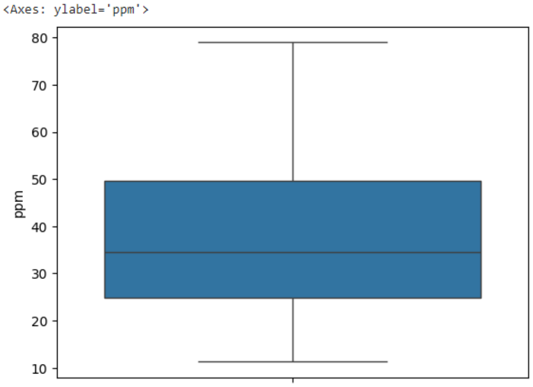
```
0.75    49.67
0.25    24.85
Name: ppm, dtype: float64
49.67
24.85
24.82
86.9
-12.380000000000003
```

```python
data['ppm'] = np.where(data['ppm'] > 78.975,78.975, data['ppm'])
sns.boxplot(data['ppm'])
```

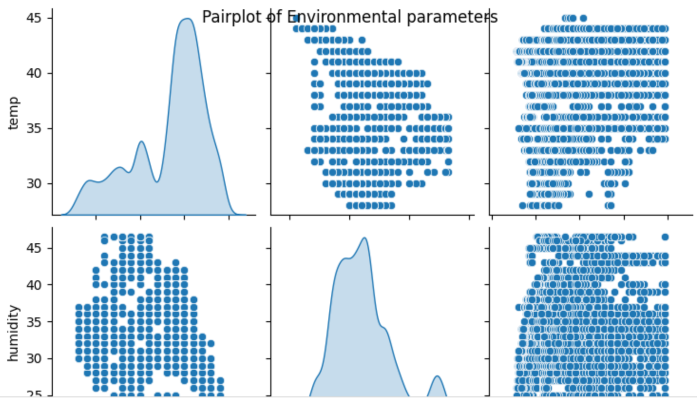
```
<Axes: ylabel='ppm'>
```

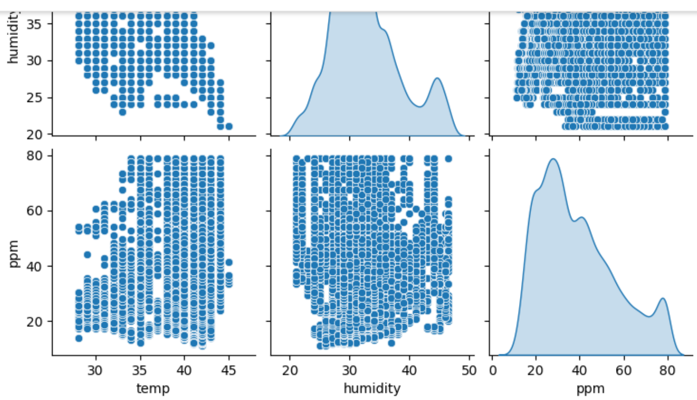
```python
data['ppm'] = np.where(data['ppm'] > 78.975,78.975, data['ppm'])
sns.boxplot(data['ppm'])
```

```
<Axes: ylabel='ppm'>
```

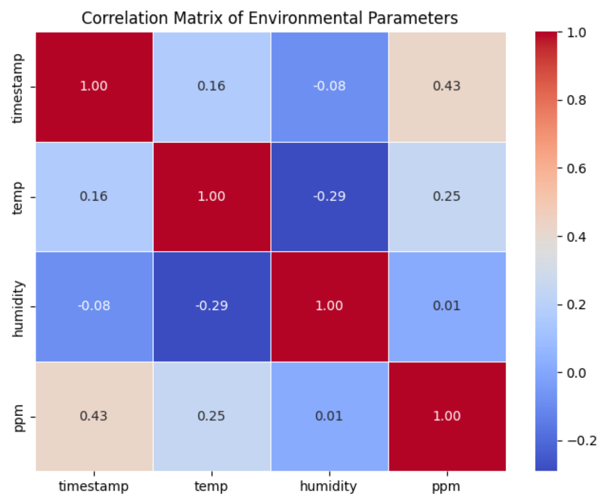


```
sns.pairplot(data, diag_kind='kde')
plt.suptitle('Pairplot of Environmental parameters')
plt.show()
```






```
correlation_matrix = data.corr()
#Heatmap of Correlation Matrix
plt.figure(figsize=(8, 6))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f', linewidths=0.5)
plt.title('Correlation Matrix of Environmental Parameters')
plt.show()
```



Handling Timestamp Feature

```python
data['timestamp'] = pd.to_datetime(data['timestamp'])

# Extracting year, month, and day
data['Year'] = data['timestamp'].dt.year
data['Month'] = data['timestamp'].dt.month
data['Day'] = data['timestamp'].dt.day
data['Hour'] = data['timestamp'].dt.hour

# Display the modified DataFrame
print(data [['timestamp', 'Year', 'Month', 'Day', 'Hour', 'temp', 'humidity', 'ppm']])
```

```
                      timestamp  Year  Month  Day  Hour  temp  humidity    ppm
0     2023-06-09 10:46:48+05:30  2023      6    9    10    38      38.0  24.01
1     2023-06-09 10:47:49+05:30  2023      6    9    10    38      36.0  22.39
2     2023-06-09 10:48:49+05:30  2023      6    9    10    38      36.0  21.62
3     2023-06-09 10:49:50+05:30  2023      6    9    10    38      36.0  21.62
4     2023-06-09 10:50:50+05:30  2023      6    9    10    38      36.0  21.62
...                         ...   ...    ...  ...   ...   ...       ...    ...
10303 2023-06-17 02:51:13+05:30  2023      6   17     2    28      32.0  25.71
10304 2023-06-17 02:52:13+05:30  2023      6   17     2    28      32.0  25.71
10305 2023-06-17 02:53:14+05:30  2023      6   17     2    28      32.0  25.71
10306 2023-06-17 02:54:14+05:30  2023      6   17     2    28      32.0  28.43
10307 2023-06-17 02:55:15+05:30  2023      6   17     2    28      32.0  30.36

[10308 rows x 8 columns]
```

```
#splitting independent and dependent variables
x = data [['Year', 'Month', 'Day', 'Hour', 'temp', 'humidity']]
```

```
x.head()
```

	Year	Month	Day	Hour	temp	humidity
0	2023	6	9	10	38	38.0
1	2023	6	9	10	38	36.0
2	2023	6	9	10	38	36.0
3	2023	6	9	10	38	36.0
4	2023	6	9	10	38	36.0

```
x.dtypes
```

```
Year          int32
Month         int32
Day           int32
Hour          int32
temp          int64
humidity    float64
dtype: object
```

```
y = data['ppm']
```

```
y = data['ppm']
```

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x,y, test_size=0.2, random_state=42)
```

```
from sklearn.preprocessing import StandardScaler
ss = StandardScaler()
x_train = ss.fit_transform(x_train)
x_test = ss.transform(x_test)
```

LinearRegression

```
from sklearn.linear_model import LinearRegression
LR = LinearRegression()
LR.fit(x_train, y_train)
```

```
▾ LinearRegression
LinearRegression()
```

```
y_pred = LR.predict(x_test)
y_pred
```

```
array([43.75486282, 42.38330191, 28.99622554, ..., 49.52443724,
```

LinearRegression

```
from sklearn.linear_model import LinearRegression
LR = LinearRegression()
LR.fit(x_train, y_train)
```

```
▾ LinearRegression
LinearRegression()
```

```
y_pred = LR.predict(x_test)
y_pred
```

```
array([43.75486282, 42.38330191, 28.99622554, ..., 49.52443724,
       21.4783531 , 42.3413951 ])
```

```
print("Training Accuracy=", LR.score(x_train, y_train))
print("Test Accuracy ", LR.score (x_test,y_test))
```

```
Training Accuracy= 0.2256505204635354
Test Accuracy  0.22101875220973688
```

```
from sklearn.metrics import r2_score
LR_r2score = r2_score(y_test,y_pred)
print("R-squared: ", LR_r2score)
```

```
R-squared:  0.22101875220973688
```

RandomForest Regressor

```
from sklearn.ensemble import RandomForestRegressor
RFR = RandomForestRegressor(n_estimators = 20, random_state = 0)
RFR.fit(x_train, y_train)
y_pred1 = RFR.predict(x_test)
RFR_r2score = r2_score(y_test,y_pred1)
```

```
from sklearn import metrics
print('R_squared: ', RFR_r2score)
```

```
R_squared:  0.935374935760041
```

```
print("Training Accuracy", RFR.score(x_train,y_train))
print("Test Accuracy", RFR.score(x_test,y_test))
```

```
Training Accuracy 0.9479937857412938
Test Accuracy 0.935374935760041
```

Decision Tree Regressor

```
from sklearn.tree import DecisionTreeRegressor
DTR = DecisionTreeRegressor()
DTR.fit(x_train, y_train)
```

Decision Tree Regressor

```python
from sklearn.tree import DecisionTreeRegressor
DTR = DecisionTreeRegressor()
DTR.fit(x_train, y_train)
```

```
▸ DecisionTreeRegressor
DecisionTreeRegressor()
```

```python
y_pred2 = DTR.predict(x_test)
y_pred2
```

```
array([26.34902439, 36.168     , 15.48909091, ..., 26.76      ,
       25.5935    , 60.15333333])
```

```python
DTR_r2score=r2_score(y_test,y_pred2)
print("R-squared:", DTR_r2score)
```

```
R-squared: 0.9345428377969989
```

```python
print("Training Accuracy= ", DTR.score(x_train,y_train))
print("Test Accuracy", DTR.score(x_test,y_test))
```

```
Training Accuracy=  0.948807397969692
Test Accuracy 0.9345428377969989
```

K-Nearest Neighbour:

```python
from sklearn.neighbors import KNeighborsRegressor
```

```python
knn_regressor = KNeighborsRegressor(n_neighbors=15)
knn_regressor.fit(x_train, y_train)
y_pred5 = knn_regressor.predict(x_test)
KNN_r2score = r2_score(y_test, y_pred5)
```

```python
print("R-squared:", KNN_r2score)
```

```
R-squared: 0.9165295650159341
```

```python
print("Training Accuracy", knn_regressor.score(x_train, y_train))
print("Test Accuracy", knn_regressor.score(x_test,y_test))
```

```
Training Accuracy 0.9296630341856954
Test Accuracy 0.9165295650159341
```

```python
#AccuracyScore DataFrame
accuracy_df = pd.DataFrame({'model': ['Linear Regression','Random Forest Regressor','Decision Tree Regressor','KNN'],
                            'R2_score': [LR_r2score,RFR_r2score,DTR_r2score, KNN_r2score]})
accuracy_df
```

	model	R2_score
0	Linear Regression	0.221019

```python
knn_regressor = KNeighborsRegressor(n_neighbors=15)
knn_regressor.fit(x_train, y_train)
y_pred5 = knn_regressor.predict(x_test)
KNN_r2score = r2_score(y_test, y_pred5)
```

```python
print("R-squared:", KNN_r2score)
```

```
R-squared: 0.9165295650159341
```

```python
print("Training Accuracy", knn_regressor.score(x_train, y_train))
print("Test Accuracy", knn_regressor.score(x_test,y_test))
```

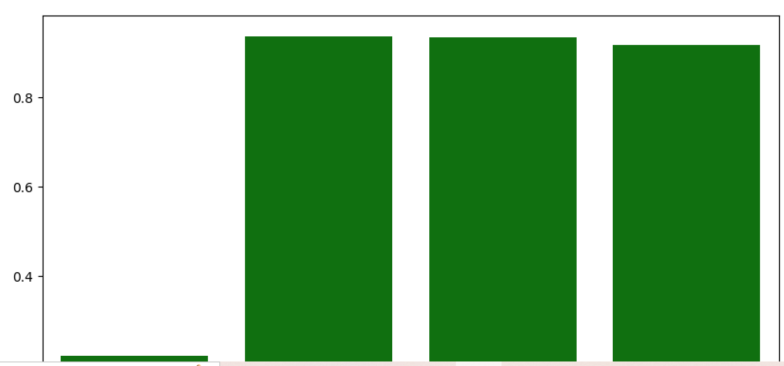
```
Training Accuracy 0.9296630341856954
Test Accuracy 0.9165295650159341
```

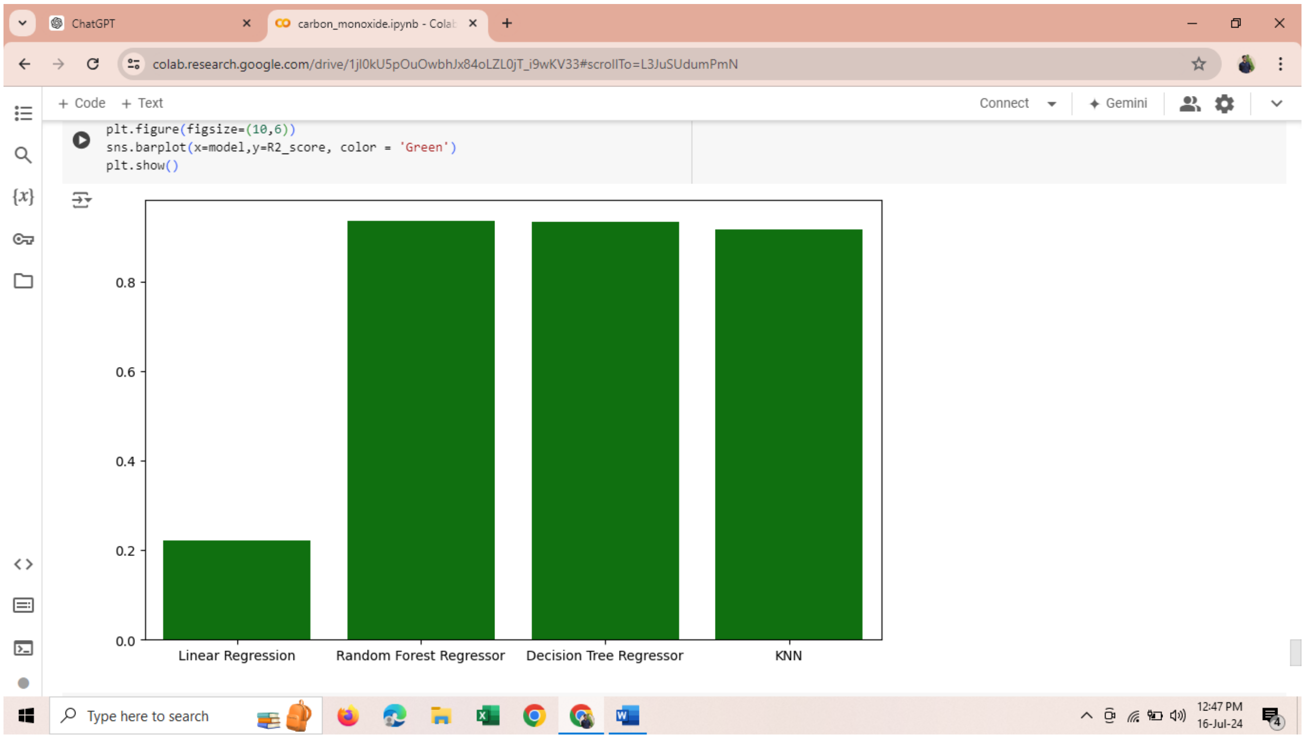
```python
#AccuracyScore DataFrame
accuracy_df = pd.DataFrame({'model': ['Linear Regression','Random Forest Regressor','Decision Tree Regressor','KNN'],
                           'R2_score': [LR_r2score,RFR_r2score,DTR_r2score, KNN_r2score]})
accuracy_df
```

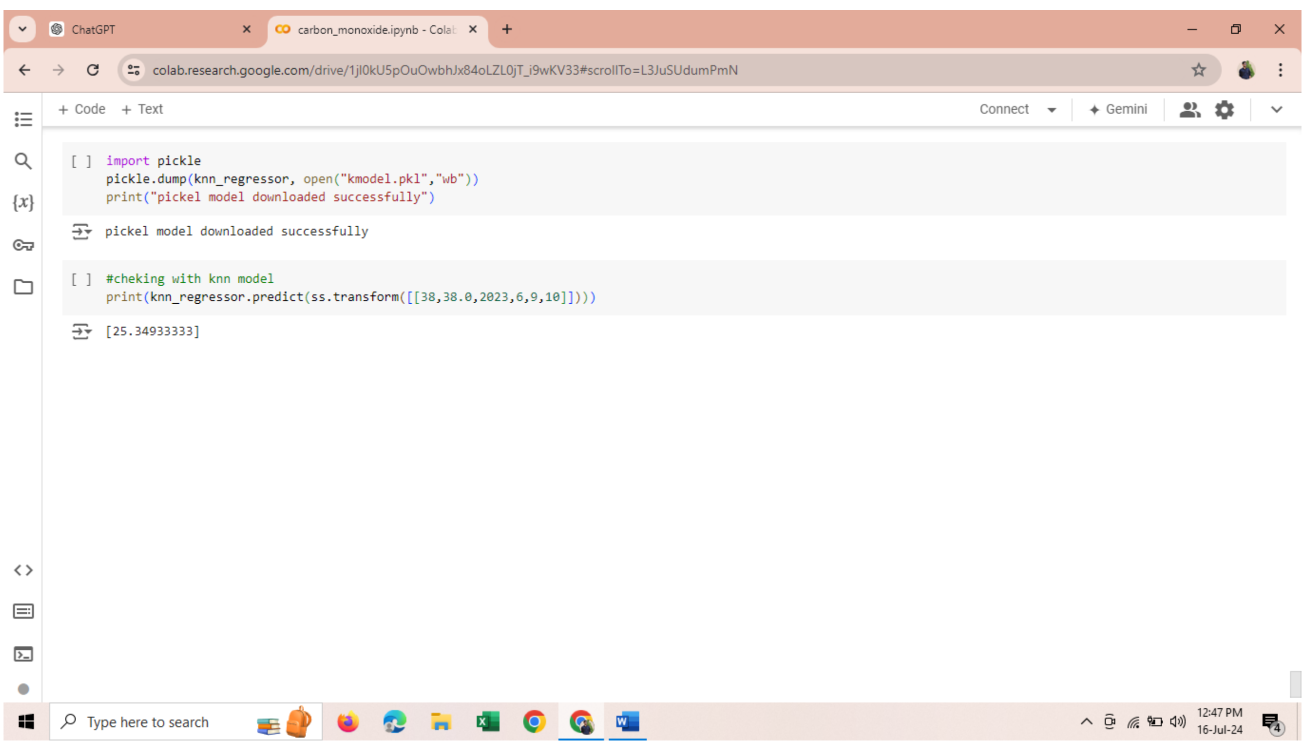
	model	R2_score
0	Linear Regression	0.221019
1	Random Forest Regressor	0.935375
2	Decision Tree Regressor	0.934543
3	KNN	0.916530

Graphical Representation Of The Model Comparison.

```python
#visual representation of models
model = ['Linear Regression', 'Random Forest Regressor', 'Decision Tree Regressor', 'KNN']
R2_score = [LR_r2score,RFR_r2score,DTR_r2score, KNN_r2score]
plt.figure(figsize=(10,6))
sns.barplot(x=model,y=R2_score, color = 'Green')
plt.show()
```




ChatGPT
carbon_monoxide.ipynb - Colab
colab.research.google.com/drive/1jl0kU5pOuOwbhJx84oLZL0jT_i9wKV33#scrollTo=L3JuSUdumPmN
+ Code + Text
Connect
Gemini
plt.figure(figsize=(10,6))
sns.barplot(x=model,y=R2_score, color = 'Green')
plt.show()
0.0
0.2
0.4
0.6
0.8
Linear Regression
Random Forest Regressor
Decision Tree Regressor
KNN
Type here to search
12:47 PM
16-Jul-24


ChatGPT
carbon_monoxide.ipynb - Colab
colab.research.google.com/drive/1jl0kU5pOuOwbhJx84oLZL0jT_i9wKV33#scrollTo=L3JuSUdumPmN
+ Code + Text
Connect
Gemini
import pickle
pickle.dump(knn_regressor, open("kmodel.pkl","wb"))
print("pickel model downloaded successfully")
pickel model downloaded successfully
#cheking with knn model
print(knn_regressor.predict(ss.transform([[38,38.0,2023,6,9,10]])))
[25.34933333]
Type here to search
12:47 PM
16-Jul-24

INDEX.HTML:

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>EcoForecast</title>
    <link rel="stylesheet" href="{{url_for('static',
filename='Style.css') }}">
</head>
<body>
    <ul>
        <li style="float:left"><a class="active"
href=""><b>EcoForecast</b></a></li>
        <li><a href="{{ url_for('contact') }}"><b>Contact</b></a></li>
        <li><a href="{{ url_for('submit') }}"><b>Predict</b></a></li>
        <li><a href="{{ url_for('index') }}"><b>Home</b></a></li>
    </ul>
    <div>
        <h1>EcoForecast: AI-powered Prediction Of<br>Carbon Monoxide
Levels</h1>
        <p><u><span style="color:red">PREDICTING CARBON MONOXIDE FOR
CLEANER SKIES</span></u></p>
        <a href="{{ url_for('submit') }}"><button><b>Get
Started</b></button></a>
    </div>
</body>
</html>
```

STYLES.CSS

```css
*{
  box-sizing: border-box;
}
body{
  background-image: url("carbon2.jpg");
```

```css
  background-size: 100vw 100vh;
}
ul {
  list-style-type: none;
  margin-left: 290px;
  padding: 0;
  overflow: hidden;
  border: 1px solid black;
  background-color: #f3f3f3;
  border-radius: 15px;
  width: 900px;
}
li {
  float: right;
}
li a {
  display: block;
  color:black;
  text-align: center;
  padding: 14px 16px;
  text-decoration: none;
}

li a:hover:not(.active) {
  background-color: gray;
}

li a.active {
  color:black;
  background-color: lawngreen;
}
div{
  background-color: white;
  opacity: 0.7;
  margin-left: 15%;
  margin-top: 5%;
  max-width: 70%;
  max-height: max-content;
  text-align: center;
  padding: 145px;
  color: black;
  font-weight: bold;
  border-radius: 4px;
}
p{
```

```
  font-size:large;
}
button{
  background-color:blue;
  color: white;
  padding: 10px;
  text-align: center;
  border-radius: 25px;
  border: 2px black solid;
}
button:hover{
  background-color: lawngreen;
  color: white;
}
```

INNERPAGE.HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>smartbridge</title>
    <link rel="stylesheet"
href="{{url_for('static',filename='Style1.css')}}">
</head>
<body>
    <ul>
        <li style="float:left"><a class="active" href="{{
url_for('index') }}">EcoForecast</a></li>
        <li><a href="{{ url_for('contact') }}">Contact</a></li>
        <li><a href="{{ url_for('submit') }}">Predict</a></li>
        <li><a href="{{ url_for('index') }}">Home</a></li>
    </ul>
    <div>
    <h1><u>PPM PREDICTION</u></h1>
    <form method="post">
        <label for="year">Year:</label>
        <input type="number" id="year" name="year" required><br><br>
        <label for="month">Month:</label>
        <input type="number" id="month" name="month" required><br><br>
        <label for="day">Day:</label>
        <input type="number" id="day" name="day" required><br><br>
```

```
        <label for="hour">Hour:</label>
        <input type="number" id="hour" name="hour" required><br><br>
        <label for="temp">Temperature:</label>
        <input type="number" id="temp" name="temp" required><br><br>
        <label for="humidity">Humidity:</label>
        <input type="number" id="humidity" name="humidity"
required><br><br>
        <input type="submit" value="Predict">
    </form>
    </div>
    <h2>{{resulty}}</h2>
</body>
</html>
```

STYLE1.CSS

```
body{
    background-image: url("carbon.jpg");
    background-size: 100vw 100vh;
}
        ul {
            list-style-type: none;
            margin-left: 20%;
            padding: 0;
            overflow: hidden;
            border: 1px solid black;
            background-color: #f3f3f3;
            border-radius: 15px;
            width: 900px;
        }
        li {
            float: right;
        }
        li a {
            display: block;
            color:black;
            text-align: center;
            padding: 14px 16px;
            text-decoration: none;
            font-weight: bold;
        }
        li a:hover:not(.active) {
            background-color: gray;
        }
        li a.active {
```

```
            color:black;
            background-color: lawngreen;
        }
        div{
            background-color: whitesmoke;
            opacity: 0.7;
            width: 30%;
            margin-top: 4%;
            margin-left: 35%;
            border-radius: 5px;
        }
        h1{
            text-align: center;
        }
        label{
            margin-left: 25%;
            font-weight: bold;
        }
        input{
            margin-left: 25%;
            border: 2px solid black;
            width: 50%;
        }
        input[type="submit"]{
            font-weight: bold;
            margin-left: 42%;
            padding: 2%;
            max-width:20%;
            color: black;
            background-color: green;
        }
```

LASTPAGE.HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>SmartBridge</title>
    <link rel="stylesheet" href="{{url_for('static',
filename='Style2.css')}}">
</head>
```

```
<body>
    <ul>
        <li style="float:left"><a class="active" href="{{
url_for('index') }}"><b>PPM of CO</b></a></li>
        <li><a href="{{ url_for('index') }}"><b>Home</b></a></li>
    </ul>
    <div>
        <h1><u>PREDICTION</u></h1>
        <h2>{{result}}</h2>
        <a href="{{ url_for('index') }}">Go back</a>
    </div>
</body>
</html>
```

STYLE2.CSS

```
body{
    background-image: url("carbon2.jpg");
    background-size:100vw 100vh;
}
div {
    background-color: whitesmoke;
    opacity: 0.8;
    width: 40%;
    margin-top: 11%;
    margin-left: 25%;
    border-radius: 5px;
    padding: 5%;
}
ul {
    list-style-type: none;
    margin-left: 100px;
    padding: 0;
    overflow: hidden;
    border: 1px solid black;
    background-color: #f3f3f3;
    border-radius: 0px;
    width: 80%;
}
li {
    float: right;
}
li a {
```

```css
    display: block;
    color: black;
    text-align: center;
    padding: 14px 16px;
    text-decoration: none;
}
li a:hover:not(.active) {
    background-color: gray;
}
li a.active {
    color: black;
    background-color: green;
}
h1{
    text-align: center;
    font-weight: bold;
}
```

PAVAN.PY:

```python
from flask import Flask, render_template, request
import pandas as pd
import pickle
import logging

app = Flask(__name__)

# Configure logging
logging.basicConfig(level=logging.DEBUG)

# Load the model
model_path = 'kmodel.pkl'
with open(model_path, 'rb') as file:
    model = pickle.load(file)

@app.route('/')
def index():
    return render_template('index.html')

@app.route('/predict', methods=["POST", "GET"])
def submit():
    if request.method == 'POST':
        try:
```

```
            # Reading the inputs given by the user
            input_feature = [request.form.get(name) for name in
['year', 'month', 'day', 'hour', 'temp', 'humidity']]
            logging.debug(f"Received input: {input_feature}")  # Log
input values

            # Ensure all inputs are present
            if not all(input_feature):
                return render_template("inner-page.html",
result="Error: Please provide all 6 input values.")

            # Convert to float
            input_feature = [float(x) for x in input_feature]

            # Define column names
            names = ['Year', 'Month', 'Day', 'Hour', 'temp',
'humidity']
            data = pd.DataFrame([input_feature], columns=names)

            # Predictions using the loaded model file
            prediction = model.predict(data)
            rounded_prediction = round(prediction[0])  # Round the
prediction to a whole number
            output = f"Predicted Carbon Monoxide Level:
{rounded_prediction} ppm."

        except Exception as e:
            logging.error(f"Error in prediction: {str(e)}")
            output = f"Error in prediction: {str(e)}"

        return render_template("last-page.html", result=output)
    return render_template("inner-page.html", result="")

@app.route('/contact')
def contact():
    return render_template('contact.html')  # Create a contact.html
and render it here

if __name__ == '__main__':
    app.run(debug=True, port=5000)
```

10.2 GitHub and project Demo link:

Github link:CLICK HERE

Project Demo link: **Click Here**