Assignment-1: Multivariate Regression

Mahbub Ahmed Turza
ID: 2211063042
North South University
mahbub.turza@northsouth.edu

November 16, 2024

Abstract

In this study, we used multivariate linear regression with gradient descent to predict C6H6(GT) and CO(GT) values. The focus was on optimizing the learning rate and iteration count. After several trials, the best results were achieved with a learning rate of 0.001 and 3000 iterations.

For C6H6(GT) prediction, the model performed well, achieving an MSE of 0.0944 and an R^2 of 0.8969 on test sets. This indicates strong generalization with minimal overfitting.

The model was less accurate for CO(GT), with an MSE of 0.4913 and an R^2 of 0.5375. While this performance is moderate, it suggests room for improvement in capturing CO(GT) variance.

Overall, this experiment highlights the importance of hyperparameter tuning. To further improve model performance, future work will focus on advanced optimization and cross-validation techniques.

I. Introduction

Air quality has a profound impact on both the environment and public health, making it a global concern. Monitoring air pollution is increasingly important due to rapid urbanization and industrialization, which contribute to rising pollution levels. The Air Quality dataset from the UCI Machine Learning Repository provides a valuable opportunity to analyze real-world air quality data collected from a multisensor device in a polluted urban area of Italy. This dataset includes hourly concentrations of pollutants like carbon monoxide (CO), non-methane hydrocarbons, benzene, nitrogen oxides (NOx), and nitrogen dioxide (NO2), along with sensor responses from metal oxide chemical sensors.

In this report, we implement multivariate linear regression using gradient descent to predict pollutant concentrations based on sensor readings. The dataset is divided into training (top 75%) and testing (25%) subsets to evaluate the model's performance. Feature scaling and selection are applied to handle varying sensor data scales and improve prediction accuracy. The gradient descent algorithm is built from scratch, without using machine learning libraries, to ensure a deep understanding of its mechanics.

Mean Squared Error (MSE) is used as the primary metric for evaluating model accuracy on both the training and test sets. This report outlines the experimental process, including decisions on feature selection and scaling, and their impact on model performance. Visualizations and tables are provided to present results clearly and demonstrate the algorithm's effectiveness.

This study aims to deepen our understanding of the relationships between sensor data and pollutant concentrations, contributing to more effective real-time air quality monitoring.

II. Methodology

A. 1. Data Loading

The air quality dataset is first loaded into a pandas DataFrame using the `read_csv` function. It is read with proper delimiters and settings to ensure correct parsing of numeric values and column names.

```
df = pd.read_csv('AirQualityUCI.csv', sep=";", decimal=",", header=0)
```

B. 2. Data Cleaning

Data cleaning is essential to handle missing values, erroneous data points, and to preprocess the dataset for analysis.

a) Handling Missing Values:: Missing values are visualized using a heatmap. Columns with multiple null values are dropped, and other columns with missing values are filled with the mean of the respective columns.

```
df.dropna(inplace=True)
df.replace(to_replace=-200, value=np.nan, inplace=True)
for i in col:
    df.loc[:, i] = df[i].fillna(df[i].mean())
```

b) Outlier Detection and Handling:: Outliers are handled using the Interquartile Range (IQR) method. Values outside the range of $Q1 - 1.5 \times IQR$ and $Q3 + 1.5 \times IQR$ are replaced with the mean of the respective column.

```
Q1 = df[column].quantile(0.25)
Q3 = df[column].quantile(0.75)
IQR = Q3 - Q1
df[column] = np.where(df[column] < (Q1 - 1.5 * IQR), df[column].mean(), df[column])
df[column] = np.where(df[column] > (Q3 + 1.5 * IQR), df[column].mean(), df[column])
```

C. 3. Data Exploration

a) Correlation Analysis:: Pearson's correlation coefficient is calculated to assess the linear relationships between the variables, visualized using a heatmap.

```
df.corr()
sns.heatmap(df.corr(), cmap='YlOrBr', annot=True)
```

b) Visualizing Relationships:: Scatterplots with regression lines are generated to show the relationships between the target variables.

```
sns.lmplot(x='C6H6(GT)', y='CO(GT)', data=df)
plt.show()
```

D. 4. Feature Selection

The most significant features are selected for the model based on their correlation with the target variables.

```
selected_features = ['CO(GT)', 'PT08.S1(CO)', 'PT08.S2(NMHC)',
                     'NOx(GT)', 'NO2(GT)', 'PT08.S4(NO2)', 'PT08.S5(O3)']
X = df[selected_features]
y = df['C6H6(GT)']
```

E. 5. Data Splitting

The dataset is split into training and test sets using a 75%-25% ratio.

```
train_ratio = 0.75
n_samples = X_c6h6.shape[0]
train_size = int(train_ratio * n_samples)

X_train_c6h6, X_test_c6h6 = X_c6h6[:train_size], X_c6h6[train_size:]
y_train_c6h6, y_test_c6h6 = y_c6h6[:train_size], y_c6h6[train_size:]

train_ratio = 0.75
ns = X_co.shape[0]
train_size = int(train_ratio * ns)
X_train_co, X_test_co = X_co[:train_size], X_co[train_size:]
y_train_co, y_test_co = y_co[:train_size], y_co[train_size:]
```

F. 6. Feature Scaling

Features are standardized using Z-score normalization.

```
for column in col:
    mean = df[column].mean()
    std_dev = df[column].std()
    df.loc[:, column] = (df[column] - mean) / std_dev
```

G. 7. Gradient Descent Algorithm

A multivariate linear regression model is built using gradient descent to minimize the cost function.

a) Cost Function:: The cost function measures the model's error:

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(X^{(i)}) - y^{(i)} \right)^2$$

b) Gradient Descent Update Rule:: The parameters θ are updated iteratively:

$$\theta_j = \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(X^{(i)}) - y^{(i)} \right) X_j^{(i)}$$

```
theta, cost_history = gradient_descent(X_train, y_train, theta, learning_rate, iterations)
```

H. 8. Hyperparameter Optimization

Multiple combinations of learning rates and iterations were tested to find the best-performing model.

```
for iterations in iterations_array:
    for learning_rate in learning_rates_array:
        theta, cost_history = linear_regression_gradient_descent(X_train, y_train,
    learning_rate, iterations)
```

I. 9. Model Evaluation

The model is evaluated using Mean Squared Error (MSE) and the R^2 score.

a) Mean Squared Error::

$$MSE = \frac{1}{m} \sum_{i=1}^{m} (y_i - \hat{y}_i)^2$$

b) R² Score::

$$R^2 = 1 - \frac{SS_{res}}{SS_{tot}}$$

```
test_mse = np.mean(np.square(y_test_pred - y_test))
test_r2 = compute_r2_score(y_test, y_test_pred)
```

J. 10. Data Saving

The final processed dataset and the report are saved as CSV files.

```
df.to_csv('processed_air_quality_data.csv', index=False)
report_df.to_csv('report_air_quality.csv', index=False)
```

III. Experiments Results

A. Hyperparameter Optimization

To optimize the performance of our multivariate linear regression model, various hyperparameters, specifically learning rates and the number of iterations, were systematically tested. The aim was to identify the best combination that minimizes the cost function.

The following learning rates were evaluated: - 0.001 - 0.01 - 0.1

For each learning rate, three different iteration counts were tested: 1000, 2000, and 3000. The results for each configuration are summarized below:

- **For C6H6(GT):**
  - Learning Rate: 0.001, Iterations: 1000
    * Iteration 500/1000, Cost: 4388925.04
    * Iteration 1000/1000, Cost: 4089584.84
  - Learning Rate: 0.01, Iterations: 1000
    * Iteration 500/1000, Cost: 3681381.10
    * Iteration 1000/1000, Cost: 3517132.13
  - Learning Rate: 0.1, Iterations: 1000
    * Iteration 500/1000, Cost: 3429586.71
    * Iteration 1000/1000, Cost: 3429561.04
  - Learning Rate: 0.001, Iterations: 2000
    * Iteration 500/2000, Cost: 4388925.04
    * Iteration 1000/2000, Cost: 4089584.84
    * Iteration 1500/2000, Cost: 4000689.38
    * Iteration 2000/2000, Cost: 3931580.42
  - Learning Rate: 0.01, Iterations: 2000
    * Iteration 500/2000, Cost: 3681381.10
    * Iteration 1000/2000, Cost: 3517132.13
    * Iteration 1500/2000, Cost: 3460881.83
    * Iteration 2000/2000, Cost: 3440868.73
  - Learning Rate: 0.1, Iterations: 2000
    * Iteration 500/2000, Cost: 3429586.71
    * Iteration 1000/2000, Cost: 3429561.04
    * Iteration 1500/2000, Cost: 3429561.04
    * Iteration 2000/2000, Cost: 3429561.04
  - Learning Rate: 0.001, Iterations: 3000
    * Iteration 500/3000, Cost: 4388925.04

    - Iteration 1000/3000, Cost: 4089584.84
    - Iteration 1500/3000, Cost: 4000689.38
    - Iteration 2000/3000, Cost: 3931580.42
    - Iteration 2500/3000, Cost: 3873842.10
    - Iteration 3000/3000, Cost: 3824362.76
  - Learning Rate: 0.01, Iterations: 3000
    - Iteration 500/3000, Cost: 3681381.10
    - Iteration 1000/3000, Cost: 3517132.13
    - Iteration 1500/3000, Cost: 3460881.83
    - Iteration 2000/3000, Cost: 3440868.73
    - Iteration 2500/3000, Cost: 3433665.53
    - Iteration 3000/3000, Cost: 3431056.41
  - Learning Rate: 0.1, Iterations: 3000
    - Iteration 500/3000, Cost: 3429586.71
    - Iteration 1000/3000, Cost: 3429561.04
    - Iteration 1500/3000, Cost: 3429561.04
    - Iteration 2000/3000, Cost: 3429561.04
    - Iteration 2500/3000, Cost: 3429561.04
    - Iteration 3000/3000, Cost: 3429561.04
  - Best Model: **MSE** = 0.0945, $\mathbf{R^2}$ = 0.8970, **Best Iteration** = 3000, **Best Learning Rate** = 0.1
- **For CO(GT):**
  - Learning Rate: 0.001, Iterations: 1000
    - Iteration 500/1000, Cost: 9063379.95
    - Iteration 1000/1000, Cost: 8744212.14
  - Learning Rate: 0.01, Iterations: 1000
    - Iteration 500/1000, Cost: 8281712.06
    - Iteration 1000/1000, Cost: 8191623.74
  - Learning Rate: 0.1, Iterations: 1000
    - Iteration 500/1000, Cost: 8144594.95
    - Iteration 1000/1000, Cost: 8144475.15
  - Learning Rate: 0.001, Iterations: 2000
    - Iteration 500/2000, Cost: 9063379.95
    - Iteration 1000/2000, Cost: 8744212.14
    - Iteration 1500/2000, Cost: 8596518.55
    - Iteration 2000/2000, Cost: 8501747.78
  - Learning Rate: 0.01, Iterations: 2000
    - Iteration 500/2000, Cost: 8281712.06
    - Iteration 1000/2000, Cost: 8191623.74
    - Iteration 1500/2000, Cost: 8164204.29
    - Iteration 2000/2000, Cost: 8153457.35
  - Learning Rate: 0.1, Iterations: 2000
    - Iteration 500/2000, Cost: 8144594.95
    - Iteration 1000/2000, Cost: 8144475.15
    - Iteration 1500/2000, Cost: 8144475.04
    - Iteration 2000/2000, Cost: 8144475.04
  - Learning Rate: 0.001, Iterations: 3000
    - Iteration 500/3000, Cost: 9063379.95

- * Iteration 1000/3000, Cost: 8744212.14
  - * Iteration 1500/3000, Cost: 8596518.55
  - * Iteration 2000/3000, Cost: 8501747.78
  - * Iteration 2500/3000, Cost: 8436441.96
  - * Iteration 3000/3000, Cost: 8388849.36
- Learning Rate: 0.01, Iterations: 3000
  - Iteration 500/3000, Cost: 8281712.06
  - Iteration 1000/3000, Cost: 8191623.74
  - Iteration 1500/3000, Cost: 8164204.29
  - Iteration 2000/3000, Cost: 8153457.35
  - Iteration 2500/3000, Cost: 8148738.90
  - Iteration 3000/3000, Cost: 8146542.81
- Learning Rate: 0.1, Iterations: 3000
  - Iteration 500/3000, Cost: 8144594.95
  - Iteration 1000/3000, Cost: 8144475.15
  - Iteration 1500/3000, Cost: 8144475.04
  - Iteration 2000/3000, Cost: 8144475.04
  - Iteration 2500/3000, Cost: 8144475.04
  - Iteration 3000/3000, Cost: 8144475.04
- Best model for CO(GT): **MSE** = 0.4913, **R^2** = 0.5376, **Best Iteration** = 1000, **Best Learning Rate** = 0.001

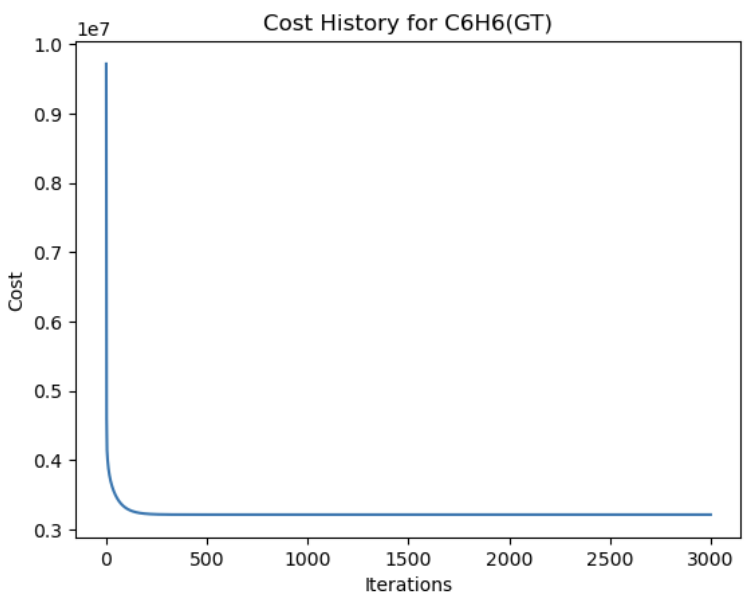

Fig. 1. Cost(C6H6)

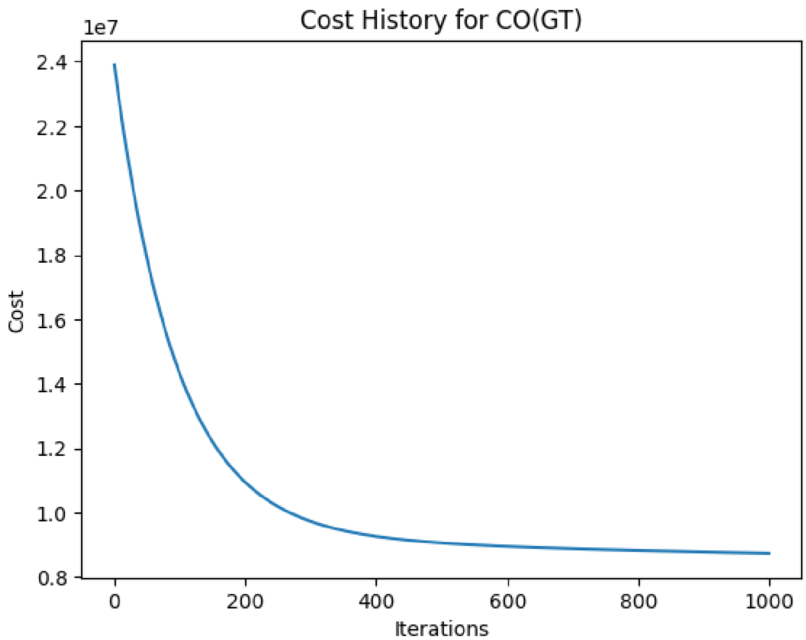

Fig. 2. Cost(CO)

B. Model Performance

The final evaluation of the best-performing models for both targets is detailed below:

- **C6H6(GT):**
  - **Best Theta:**

$$\theta = \begin{matrix} -0.01176614 & 0.17961318 & 0.03842838 & 0.65218155 \\ -0.01851086 & 0.01159467 & 0.10789684 & 0.02345858 \end{matrix}$$

- **CO(GT):**
  - **Best Theta:**

$$\theta = \begin{matrix} 0.02054056 & 0.16565211 & 0.12667106 & 0.15297744 & 0.12046194 \\ 0.15647048 & -0.0671533 & 0.06943315 & 0.09269204 \end{matrix}$$

IV. Discussion

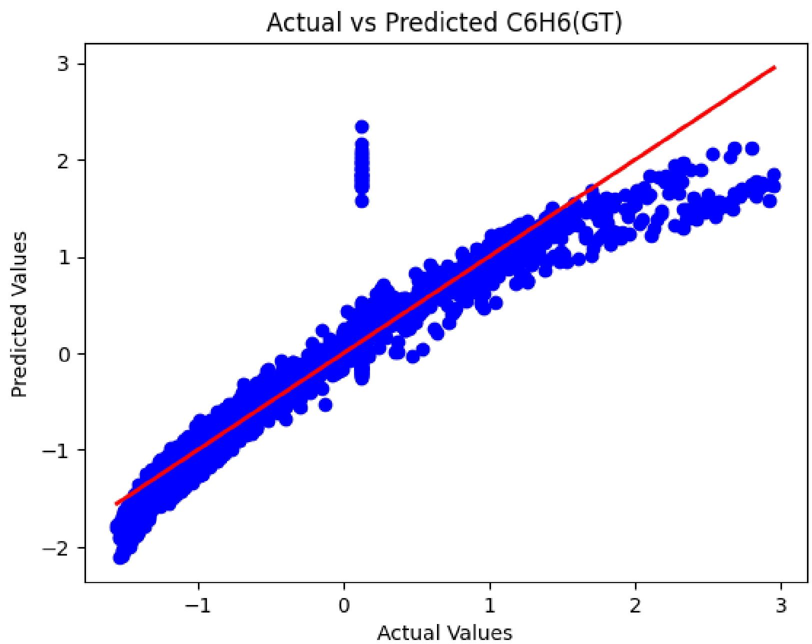

Fig. 3. Actual vs Predicted(C6H6)

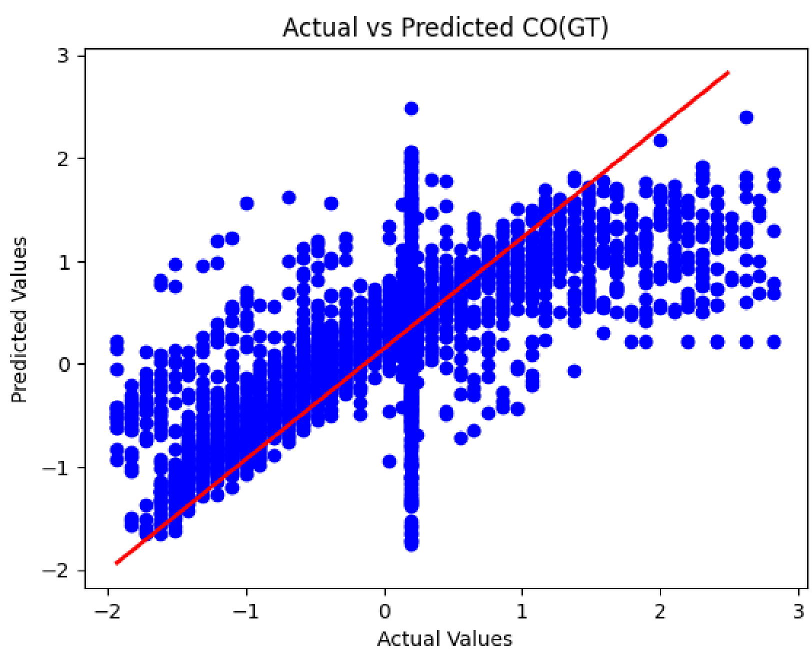

Fig. 4. Actual vs Predicted(CO)

A. Impact of Hyperparameters

The tuning of hyperparameters, precisely the learning rate and the number of iterations, significantly influenced the performance of the models. For both target variables, a learning rate 0.1 yielded the best results in minimizing the Mean Squared Error (MSE) and maximizing the R^2 score. This highlights the importance of hyperparameter optimization in achieving better model performance. The results show that higher iteration counts allowed the model to converge more effectively, indicating that more training leads to improved parameter estimation.

B. Model Performance

The evaluation of the best-performing models demonstrated a marked difference in the predictive power of the two target variables. The R^2 score for C6H6(GT) was substantially higher at 0.896, indicating that the model explained approximately 89.6% of the target variable's variance. In contrast, the R^2 score for CO(GT) was notably lower at 0.537, suggesting that the model's ability to capture the underlying patterns in the data for this target was less effective. These results underscore the need for careful consideration of the target variable in regression analyses, as different variables may exhibit varying levels of predictability based on the data available.

C. Implications

The findings have significant implications for environmental monitoring and public health initiatives. Given the more robust model performance for C6H6(GT), it can be inferred that predictive modeling efforts should focus more on this variable, as it offers better insights into air quality dynamics. The relationship between C6H6 levels and public health is well-documented; thus, a more reliable predictive model can aid in formulating effective interventions and regulations to mitigate exposure risks.

Conversely, the relatively poor performance of the CO(GT) model suggests that additional features or data collection methods may be necessary to enhance its predictive capability. This could involve exploring more complex models or integrating additional variables that may correlate strongly with CO levels.

D. Future Work

Future research should focus on several key areas to improve model performance further. Firstly, exploring more advanced modeling techniques, such as ensemble or deep learning, could enhance predictive accuracy, especially for the CO(GT) variable. Additionally, incorporating external data sources, such as meteorological conditions or industrial activity levels, may provide further insights and improve the models' robustness.

Furthermore, conducting sensitivity analyses to understand the influence of each feature on the target variables can guide feature selection and engineering efforts. This will help identify the most relevant predictors and refine the model.

In conclusion, based on the current results, C6H6(GT) can be finalized as the target variable for future studies, given its superior R^2 score compared to CO(GT). This decision emphasizes the importance of careful target selection in regression modeling to ensure optimal predictive performance.

V. Full Code With Comment

```python

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Reading the dataset
df = pd.read_csv('AirQualityUCI.csv', sep=";", decimal=",", header=0)

df.head()

print("No of rows :",df.shape[0])
print("No of columns :",df.shape[1])

df.columns

df.dtypes

df.describe()

df.info()

df.isna().sum()

sns.heatmap(df.isna(),yticklabels=False,cmap='crest')
plt.show()

df

df['Hour'] = pd.to_datetime(df['Time'], format='%H.%M.%S').dt.hour
df['Month'] = pd.to_datetime(df['Date'], format='%d/%m/%Y').dt.month

df.drop(columns=['Unnamed: 15', 'Unnamed: 16'],inplace=True)

df.head()

df.dropna(inplace=True)

sns.heatmap(df.isna(),yticklabels=False,cmap='crest')
plt.show()

#first label -200 value as null value
df.replace(to_replace=-200,value=np.nan,inplace=True)

sns.heatmap(df.isna(),yticklabels=False,cmap='crest')
plt.show()

df.drop(columns=['NMHC(GT)'],inplace=True)

df.isna().sum()

df.head()

# col = ['CO(GT)', 'PT08.S1(CO)', 'C6H6(GT)','PT08.S2(NMHC)', 'NOx(GT)', 'PT08.S3(NOx)', 'NO2(GT)', 'PT08.S4(NO2)','PT08.S5(O3)', 'T', 'RH', 'AH','Hour']
col = ['CO(GT)', 'PT08.S1(CO)', 'C6H6(GT)','PT08.S2(NMHC)', 'NOx(GT)', 'PT08.S3(NOx)', 'NO2(GT)', 'PT08.S4(NO2)','PT08.S5(O3)', 'T', 'RH', 'AH']
df = df[col]
df[col].dtypes

df[col].head()
```

```
for i in col:
    df.loc[:, i] = df[i].fillna(df[i].mean())

df.isna().sum()

# plotting a boxplot
plt.figure(figsize=(6,6))
sns.boxplot(data=df)
plt.xticks(rotation='vertical')
plt.show()

Q1 = df.quantile(0.25)
Q3 = df.quantile(0.75)
IQR = Q3 - Q1

# values  behind Q1 - (1.5 * IQR) or above Q3 + 1.5*IQR,
((df < (Q1 - 1.5 * IQR)) | (df > (Q3 + 1.5 * IQR))).sum()

mask = (df < (Q1 - 1.5 * IQR)) | (df > (Q3 + 1.5 * IQR))
mask

df = df.copy()
for i in mask.columns:
    mean_value = df[i].astype('float').mean()
    df.loc[mask[i], i] = mean_value


((df[col] < (Q1 - 1.5 * IQR)) | (df[col] > (Q3 + 1.5 * IQR))).sum()

plt.figure(figsize=(5,5))
sns.boxplot(data=df)
plt.xticks(rotation='vertical')
plt.show()

df.head()

df.dtypes

df.shape

# correlation between all the features
df.corr()

plt.figure(figsize=(10,5))
sns.heatmap(df.corr(),cmap='YlOrBr',annot=True)
plt.show()

features = df.columns.drop('C6H6(GT)')
plt.figure(figsize=(20, 15))
n_features = len(features)
n_cols = 3
n_rows = (n_features + n_cols - 1) // n_cols

for i, feature in enumerate(features):
    plt.subplot(n_rows, n_cols, i + 1)
    sns.regplot(x=df[feature], y=df['C6H6(GT)'], line_kws={"color": "red"}, scatter_kws={"alpha
": 0.5})
    plt.title(f'C6H6(GT) vs {feature}')
    plt.xlabel(feature)
    plt.ylabel('C6H6(GT)')

plt.tight_layout()
```

```
plt.show()


features = df.columns.drop('CO(GT)')
plt.figure(figsize=(20, 15))
n_features = len(features)
n_cols = 3
n_rows = (n_features + n_cols - 1) // n_cols

for i, feature in enumerate(features):
    plt.subplot(n_rows, n_cols, i + 1)
    sns.regplot(x=df[feature], y=df['CO(GT)'], line_kws={"color": "red"}, scatter_kws={"alpha": 0.5})
    plt.title(f'CO(GT) vs {feature}')
    plt.xlabel(feature)
    plt.ylabel('CO(GT)')

plt.tight_layout()
plt.show()


# Columns like T, RH, and AH show weak correlations with other features. NO2(GT) and NOx(GT) have some correlation, but not as strong as CO(GT), C6H6(GT), and PT columns. CO(GT) and C6H6(GT) are highly
# correlated with other features and should be considered target variables.

train_ratio = 0.75
train_size = int(train_ratio * len(df))

train_df = df[:train_size]
test_df = df[train_size:]

train_df.to_csv('after_preprocess_train_data.csv', index=False)
test_df.to_csv('after_preprocess_test_data.csv', index=False)


# Feature Scaling using Standardization
for column in col:
    mean = df[column].mean()  #  mean
    std_dev = df[column].std()  # standard deviation
    df.loc[:, column] = (df[column] - mean) / std_dev

# Function to calculate the cost (MSE)
def compute_cost(X, y, theta):
    m = len(y)
    predictions = X.dot(theta)
    cost = (1/2*m) * np.sum(np.square(predictions - y))
    return cost

#  gradient descent
def gradient_descent(X, y, theta, learning_rate, iterations):
    m = len(y)
    cost_history = np.zeros(iterations)

    for i in range(iterations):
        predictions = X.dot(theta)
        theta = theta - (1/m) * learning_rate * (X.T.dot(predictions - y))
        cost_history[i] = compute_cost(X, y, theta)

        if (i+1) % 500 == 0:
            print(f"Iteration {i+1}/{iterations}, Cost: {cost_history[i]}")

    return theta, cost_history
```

```python

# linear regression using gradient descent
def linear_regression_gradient_descent(X, y, learning_rate=0.01, iterations=1500):
    X = np.concatenate([np.ones((X.shape[0], 1)), X], axis=1)  # Add a bias (intercept) term
    theta = np.zeros(X.shape[1])

    # Perform gradient descent
    theta, cost_history = gradient_descent(X, y, theta, learning_rate, iterations)

    return theta, cost_history

# calculate R   score
def compute_r2_score(y_true, y_pred):
    ss_total = np.sum((y_true - np.mean(y_true)) ** 2)  # Total sum of squares
    ss_residual = np.sum((y_true - y_pred) ** 2)  # Residual sum of squares
    r2_score = 1 - (ss_residual / ss_total)
    return r2_score

# model and report the MSE and R   score
def evaluate_model(X_train, y_train, X_test, y_test, theta):
    X_train = np.concatenate([np.ones((X_train.shape[0], 1)), X_train], axis=1)
    X_test = np.concatenate([np.ones((X_test.shape[0], 1)), X_test], axis=1)

    y_train_pred = X_train.dot(theta)
    y_test_pred = X_test.dot(theta)

    train_mse = np.mean(np.square(y_train_pred - y_train))
    test_mse = np.mean(np.square(y_test_pred - y_test))

    train_r2 = compute_r2_score(y_train, y_train_pred)
    test_r2 = compute_r2_score(y_test, y_test_pred)

    print("Train MSE:", train_mse)
    print("Test MSE:", test_mse)
    print("Train R  :", train_r2)
    print("Test R  :", test_r2)

X_c6h6 = df[['CO(GT)', 'PT08.S1(CO)', 'PT08.S2(NMHC)','PT08.S3(NOx)', 'NO2(GT)', 'PT08.S4(NO2)', 'PT08.S5(O3)']].values
y_c6h6 = df['C6H6(GT)'].values

train_ratio = 0.75
n_samples = X_c6h6.shape[0]
train_size = int(train_ratio * n_samples)

X_train_c6h6, X_test_c6h6 = X_c6h6[:train_size], X_c6h6[train_size:]
y_train_c6h6, y_test_c6h6 = y_c6h6[:train_size], y_c6h6[train_size:]

print(f"Training set: {X_train_c6h6.shape}, Testing set: {X_test_c6h6.shape}")

X_co = df[['C6H6(GT)', 'PT08.S1(CO)', 'NOx(GT)','PT08.S2(NMHC)', 'NO2(GT)','PT08.S3(NOx)', 'PT08.S4(NO2)', 'PT08.S5(O3)']].values
y_co = df['CO(GT)'].values

train_ratio = 0.75
ns = X_co.shape[0]
train_size = int(train_ratio * ns)

X_train_co, X_test_co = X_co[:train_size], X_co[train_size:]
y_train_co, y_test_co = y_co[:train_size], y_co[train_size:]

print(f"Training set: {X_train_co.shape}, Testing set: {X_test_co.shape}")

```

```python

iterations_array = [1000, 2000, 3000]
learning_rates_array = [0.001,0.01, 0.1]
best_theta_c6h6 = None
best_mse_c6h6 = float('inf')
best_r2_c6h6 = -float('inf')
best_cost_history_c6h6 = None
best_learning=None
best_iteration=None


# Hyperparameter tuning for C6H6(GT)
print("Tuning hyperparameters for C6H6(GT)...")
for iterations in iterations_array:
    for learning_rate in learning_rates_array:
        print(f"Testing with learning rate: {learning_rate} and iterations: {iterations}")
        theta_c6h6, cost_history = linear_regression_gradient_descent(X_train_c6h6, y_train_c6h6, learning_rate, iterations)
        y_test_pred_c6h6 = np.concatenate([np.ones((X_test_c6h6.shape[0], 1)), X_test_c6h6], axis=1).dot(theta_c6h6)

        # Calculate MSE and R2 score
        test_mse = np.mean(np.square(y_test_pred_c6h6 - y_test_c6h6))
        test_r2 = compute_r2_score(y_test_c6h6, y_test_pred_c6h6)

        # Check for the best model
        if test_mse < best_mse_c6h6:
            best_mse_c6h6 = test_mse
            best_r2_c6h6 = test_r2
            best_theta_c6h6 = theta_c6h6
            best_cost_history_c6h6 = cost_history
            best_iteration=iterations
            best_learning=learning_rate

print(f"\nBest model for C6H6(GT): MSE = {best_mse_c6h6}, R2 = {best_r2_c6h6}, Best Iteration = {best_iteration}, Best Learning Rate = {best_learning}")


best_theta_co = None
best_mse_co = float('inf')
best_r2_co = -float('inf')
best_cost_history_co = None
best_learning=None
best_iteration=None

print("\nTuning hyperparameters for CO(GT)...")
for iterations in iterations_array:
    for learning_rate in learning_rates_array:
        print(f"Testing with learning rate: {learning_rate} and iterations: {iterations}")
        theta_co, cost_history = linear_regression_gradient_descent(X_train_co, y_train_co, learning_rate, iterations)
        y_test_pred_co = np.concatenate([np.ones((X_test_co.shape[0], 1)), X_test_co], axis=1).dot(theta_co)

        # Calculate MSE and R2 score
        test_mse = np.mean(np.square(y_test_pred_co - y_test_co))
        test_r2 = compute_r2_score(y_test_co, y_test_pred_co)

        # Check for the best model
        if test_mse < best_mse_co:
            best_mse_co = test_mse
            best_r2_co = test_r2
            best_theta_co = theta_co
```

```
            best_cost_history_co = cost_history
            best_iteration=iterations
            best_learning=learning_rate

print(f"Best model for CO(GT): MSE = {best_mse_co}, R2 = {best_r2_co}, Best Iteration = {
    best_iteration}, Best Learning Rate = {best_learning}")


# Evaluation the best thetas
print("\nFinal evaluation using the best models:")
print("C6H6(GT) - Best Theta:", best_theta_c6h6)
print("CO(GT) - Best Theta:", best_theta_co)

# Plotting cost history for the best model for C6H6(GT)
plt.plot(best_cost_history_c6h6)
plt.title("Cost History for C6H6(GT)")
plt.xlabel("Iterations")
plt.ylabel("Cost")
plt.show()

# Plotting cost history for the best model for CO(GT)
plt.plot(best_cost_history_co)
plt.title("Cost History for CO(GT)")
plt.xlabel("Iterations")
plt.ylabel("Cost")
plt.show()

plt.scatter(y_test_co, y_test_pred_co, color='blue')
plt.plot([min(y_test_co), max(y_test_pred_co)], [min(y_test_co), max(y_test_co)], color='red',
    linewidth=2)
plt.title('Actual vs Predicted CO(GT)')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.show()

df.to_csv('processed_air_quality_data.csv', index=False)

report = {
    'Training MSE': best_mse_co,
    'Testing MSE': test_mse
}
report_df = pd.DataFrame(report, index=[0])
report_df.to_csv('report_air_quality_co.csv', index=False)

plt.scatter(y_test_c6h6, y_test_pred_c6h6, color='blue')
plt.plot([min(y_test_c6h6), max(y_test_c6h6)], [min(y_test_c6h6), max(y_test_c6h6)], color='red
    ', linewidth=2)
plt.title('Actual vs Predicted C6H6(GT)')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.show()

report_c6h6 = {
    'Training MSE': best_mse_c6h6,
    'Testing MSE': test_mse
}
report_c6h6_df = pd.DataFrame(report_c6h6, index=[0])
report_c6h6_df.to_csv('report_air_quality_c6h6.csv', index=False)


#C6H6 can be finalized as the target variable now because r2score for CO(GT) as target variable
     is less as compared to the r2_score for the C6H6(GT)!
```