Project Title: Air Quality Monitoring and Pollution Prediction

Name:Kibon Kiprono Solomon

Date:29/08/2023

Problem Statement: Air pollution is a growing concern in urban areas, with adverse effects on public health and the environment. Developing an air quality monitoring and pollution prediction system using machine learning can help raise awareness, inform policy decisions, and enable citizens to take protective measures.

Project goal: The aim of this project is to Create a machine learning system that monitors air quality in real-time, predicts pollution levels, and provides actionable insights for individuals and authorities to mitigate the impact of air pollution.

In [3]:

```
#import necessary libraries
import numpy as np
import pandas as pd
import plotly.express as px
import seaborn as sns
from sklearn.model_selection import train_test_split, cross_val_score, GridSearchCV, Rand
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.preprocessing import PolynomialFeatures, StandardScaler
from sklearn.svm import SVR
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from scipy.stats import randint
```

In [4]:

```
#load the dataset
data=pd.read_csv('Air_quality_data.csv')
# Display the first few rows of the dataset
data.head(10)
```

Out[4]:

	City	Date	PM2.5	PM10	NO	NO2	NOx	NH3	CO	SO2	O3	Benze
0	Ahmedabad	2015-01-01	NaN	NaN	0.92	18.22	17.15	NaN	0.92	27.64	133.36	0.
1	Ahmedabad	2015-01-02	NaN	NaN	0.97	15.69	16.46	NaN	0.97	24.55	34.06	3.
2	Ahmedabad	2015-01-03	NaN	NaN	17.40	19.30	29.70	NaN	17.40	29.07	30.70	6.
3	Ahmedabad	2015-01-04	NaN	NaN	1.70	18.48	17.97	NaN	1.70	18.59	36.08	4.
4	Ahmedabad	2015-01-05	NaN	NaN	22.10	21.42	37.76	NaN	22.10	39.33	39.31	7.
5	Ahmedabad	2015-01-06	NaN	NaN	45.41	38.48	81.50	NaN	45.41	45.76	46.51	5.
6	Ahmedabad	2015-01-07	NaN	NaN	112.16	40.62	130.77	NaN	112.16	32.28	33.47	0.
7	Ahmedabad	2015-01-08	NaN	NaN	80.87	36.74	96.75	NaN	80.87	38.54	31.89	0.
8	Ahmedabad	2015-01-09	NaN	NaN	29.16	31.00	48.00	NaN	29.16	58.68	25.75	0.
9	Ahmedabad	2015-01-10	NaN	NaN	NaN	7.04	0.00	NaN	NaN	8.29	4.55	0.

In [5]:

```
# Check the distribution of the target variable (AQI_Bucket)
data['AQI_Bucket'].value_counts()
```

Out[5]:

```
Moderate        8829
Satisfactory    8224
Poor            2781
Very Poor       2337
Good            1341
Severe          1338
Name: AQI_Bucket, dtype: int64
```

In [6]:

```
#Remove null values from the target variable
data=data[data['AQI'].notna()]
data.head()
```

Out[6]:

	City	Date	PM2.5	PM10	NO	NO2	NOx	NH3	CO	SO2	O3	Benzen
28	Ahmedabad	2015-01-29	83.13	NaN	6.93	28.71	33.72	NaN	6.93	49.52	59.76	0.0
29	Ahmedabad	2015-01-30	79.84	NaN	13.85	28.68	41.08	NaN	13.85	48.49	97.07	0.0
30	Ahmedabad	2015-01-31	94.52	NaN	24.39	32.66	52.61	NaN	24.39	67.39	111.33	0.2
31	Ahmedabad	2015-02-01	135.99	NaN	43.48	42.08	84.57	NaN	43.48	75.23	102.70	0.4
32	Ahmedabad	2015-02-02	178.33	NaN	54.56	35.31	72.80	NaN	54.56	55.04	107.38	0.4

In [7]:

```
# Display data types and non-null counts of columns
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 24850 entries, 28 to 29530
Data columns (total 16 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   City        24850 non-null  object 
 1   Date        24850 non-null  object 
 2   PM2.5       24172 non-null  float64
 3   PM10        17764 non-null  float64
 4   NO          24463 non-null  float64
 5   NO2         24459 non-null  float64
 6   NOx         22993 non-null  float64
 7   NH3         18314 non-null  float64
 8   CO          24405 non-null  float64
 9   SO2         24245 non-null  float64
 10  O3          24043 non-null  float64
 11  Benzene     21315 non-null  float64
 12  Toluene     19024 non-null  float64
 13  Xylene      9478 non-null   float64
 14  AQI         24850 non-null  float64
 15  AQI_Bucket  24850 non-null  object 
dtypes: float64(13), object(3)
memory usage: 3.2+ MB
```

In [8]:

```
#sum of missing values in each column
data.isna().sum()
```

Out[8]:

```
City              0
Date              0
PM2.5           678
PM10           7086
NO              387
NO2             391
NOx            1857
NH3            6536
CO              445
SO2             605
O3              807
Benzene        3535
Toluene        5826
Xylene        15372
AQI               0
AQI_Bucket        0
dtype: int64
```

In [9]:

```
# Calculate the percentage of missing values in each column
missing_percentage = (data.isnull().sum() / len(data)) * 100
missing_percentage
```

Out[9]:

```
City           0.000000
Date           0.000000
PM2.5          2.728370
PM10          28.515091
NO             1.557344
NO2            1.573441
NOx            7.472837
NH3           26.301811
CO             1.790744
SO2            2.434608
O3             3.247485
Benzene       14.225352
Toluene       23.444668
Xylene        61.859155
AQI            0.000000
AQI_Bucket     0.000000
dtype: float64
```

In [10]:

```
#remove columns with more than 50% missing values
data.drop('Xylene',axis=1,inplace=True)
```

In [11]:

```
#fill the rest of missing values with median
data.fillna(data.median(),inplace=True)
data.head()
```

```
C:\Users\hp\AppData\Local\Temp\ipykernel_15272\3294820205.py:2: FutureWarn
ing: The default value of numeric_only in DataFrame.median is deprecated.
In a future version, it will default to False. In addition, specifying 'nu
meric_only=None' is deprecated. Select only valid columns or specify the v
alue of numeric_only to silence this warning.
  data.fillna(data.median(),inplace=True)
```

Out[11]:

	City	Date	PM2.5	PM10	NO	NO2	NOx	NH3	CO	SO2	O3	Benzer
28	Ahmedabad	2015-01-29	83.13	96.18	6.93	28.71	33.72	16.31	6.93	49.52	59.76	0.(
29	Ahmedabad	2015-01-30	79.84	96.18	13.85	28.68	41.08	16.31	13.85	48.49	97.07	0.(
30	Ahmedabad	2015-01-31	94.52	96.18	24.39	32.66	52.61	16.31	24.39	67.39	111.33	0.:
31	Ahmedabad	2015-02-01	135.99	96.18	43.48	42.08	84.57	16.31	43.48	75.23	102.70	0.4
32	Ahmedabad	2015-02-02	178.33	96.18	54.56	35.31	72.80	16.31	54.56	55.04	107.38	0.4

In [12]:

```
#descriptive statitstics
data.describe()
```

Out[12]:

	PM2.5	PM10	NO	NO2	NOx	NH3
count	24850.000000	24850.000000	24850.000000	24850.000000	24850.000000	24850.000000
mean	66.966637	112.102860	17.502312	28.870163	31.645675	21.865639
std	62.283431	76.325808	22.266346	24.447523	29.629575	22.460343
min	0.040000	0.030000	0.030000	0.010000	0.000000	0.010000
25%	29.560000	71.780000	5.720000	12.090000	14.030000	11.280000
50%	48.785000	96.180000	9.910000	22.100000	23.680000	16.310000
75%	79.507500	122.957500	19.710000	37.910000	38.170000	24.710000
max	914.940000	917.080000	390.680000	362.210000	378.240000	352.890000

In [13]:

```
# Remove outliers from the data
# Calculate the IQR for each column
Q1 = data.quantile(0.25)
Q3 = data.quantile(0.75)
IQR = Q3 - Q1

# Define a threshold to identify outliers
outlier_threshold = 1.5

# Identify outliers using the IQR method
outliers = ((data < (Q1 - outlier_threshold * IQR)) | (data > (Q3 + outlier_threshold * I

# Remove outliers by replacing them with NaN
data_no_outliers = data[~outliers]

# Alternatively, you can choose to drop rows with any NaN values
data_no_outliers = data_no_outliers.dropna()

# Save the cleaned dataset
data_no_outliers.to_csv('cleaned_data.csv', index=False)
```

```
C:\Users\hp\AppData\Local\Temp\ipykernel_15272\4198190489.py:3: FutureWarn
ing: The default value of numeric_only in DataFrame.quantile is deprecate
d. In a future version, it will default to False. Select only valid column
s or specify the value of numeric_only to silence this warning.
  Q1 = data.quantile(0.25)
C:\Users\hp\AppData\Local\Temp\ipykernel_15272\4198190489.py:4: FutureWarn
ing: The default value of numeric_only in DataFrame.quantile is deprecate
d. In a future version, it will default to False. Select only valid column
s or specify the value of numeric_only to silence this warning.
  Q3 = data.quantile(0.75)
C:\Users\hp\AppData\Local\Temp\ipykernel_15272\4198190489.py:11: FutureWar
ning: Automatic reindexing on DataFrame vs Series comparisons is deprecate
d and will raise ValueError in a future version. Do `left, right = left.al
ign(right, axis=1, copy=False)` before e.g. `left == right`
  outliers = ((data < (Q1 - outlier_threshold * IQR)) | (data > (Q3 + outl
ier_threshold * IQR)))
```

In [14]:

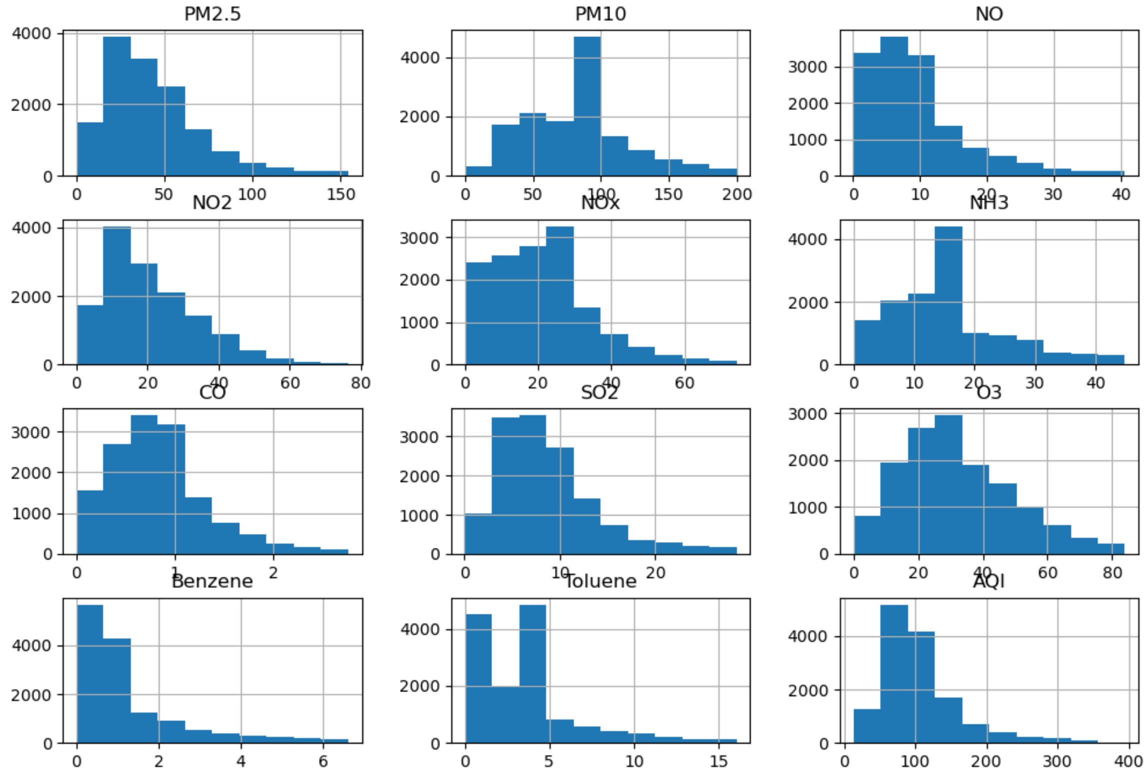
```
#load the clean data
data=pd.read_csv('cleaned_data.csv')
#display the first ten entries
data.head(10)
```

Out[14]:

	City	Date	PM2.5	PM10	NO	NO2	NOx	NH3	CO	SO2	O3	Benzene	T
0	Ahmedabad	2015-02-04	80.65	96.18	2.37	22.83	24.00	16.31	2.37	25.73	47.30	0.00	
1	Ahmedabad	2015-05-23	38.88	96.18	2.18	16.28	17.74	16.31	2.18	26.40	12.47	3.00	
2	Ahmedabad	2015-07-16	65.37	96.18	0.64	8.07	8.57	16.31	0.64	16.31	8.02	1.63	
3	Ahmedabad	2015-07-17	56.28	96.18	0.60	8.02	8.51	16.31	0.60	18.93	6.20	1.52	
4	Ahmedabad	2015-07-18	48.17	96.18	0.65	8.00	8.55	16.31	0.65	18.99	7.97	1.23	
5	Ahmedabad	2015-07-19	33.56	96.18	0.72	7.89	8.56	16.31	0.72	11.28	10.18	0.61	
6	Ahmedabad	2015-07-20	31.30	96.18	0.55	8.10	8.49	16.31	0.55	10.29	7.44	0.52	
7	Ahmedabad	2015-07-22	57.11	96.18	0.55	8.28	8.42	16.31	0.55	14.58	2.60	2.81	
8	Ahmedabad	2015-07-23	34.72	96.18	0.10	9.73	7.88	16.31	0.10	13.62	4.87	1.44	
9	Ahmedabad	2015-07-24	29.22	96.18	0.06	9.57	7.77	16.31	0.06	13.58	5.08	1.37	

In [15]:

```
# Visualize the data using histograms
import matplotlib.pyplot as plt
data.hist(figsize=(12, 8))
plt.show()
```



In [16]:

```
# Visualize the data using a correlation matrix heatmap
corr_matrix = data.corr(numeric_only=True)
fig=px.imshow(corr_matrix, title='Correlation Matrix')
fig.show()
```

In [17]:

```
#split the data into training and testing
X=data.drop(['City','Date','AQI','AQI_Bucket'],axis=1)
y=data['AQI']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=12)
```

In [18]:

```
#train the models
# Evaluation function
def evaluate_model(model, X_test, y_test):
    y_pred = model.predict(X_test)
    mae = mean_absolute_error(y_test, y_pred)
    mse = mean_squared_error(y_test, y_pred)
    rmse = np.sqrt(mse)
    r2 = r2_score(y_test, y_pred)

    return mae, mse, rmse, r2

# 1. Simple Linear Regression (using PM2.5 feature)
simple_linear_reg = LinearRegression()
simple_linear_reg.fit(X_train[['PM2.5']], y_train)
simple_linear_reg_metrics = evaluate_model(simple_linear_reg, X_test[['PM2.5']], y_test)


# 2. Multiple Linear Regression
multiple_linear_reg = LinearRegression()
multiple_linear_reg.fit(X_train, y_train)
multiple_linear_reg_metrics=evaluate_model(multiple_linear_reg,X_test,y_test)

# 3. Polynomial Regression
poly_features = PolynomialFeatures(degree=2)
X_train_poly = poly_features.fit_transform(X_train)
X_test_poly = poly_features.transform(X_test)
poly_reg = LinearRegression()
poly_reg.fit(X_train_poly, y_train)
poly_reg_metrics=evaluate_model(poly_reg,X_test_poly,y_test)

# 4. Ridge Regression
ridge_reg = Ridge(alpha=1.0)
ridge_reg.fit(X_train, y_train)
ridge_reg_metrics=evaluate_model(ridge_reg,X_test,y_test)

# 5. Lasso Regression
lasso_reg = Lasso(alpha=1.0)
lasso_reg.fit(X_train, y_train)
lasso_reg_metrics=evaluate_model(lasso_reg,X_test,y_test)

# 6. Support Vector Regression
svr_reg = SVR(kernel='linear')
svr_reg.fit(X_train, y_train)
svr_reg_metrics=evaluate_model(svr_reg,X_test,y_test)

# 7. Decision Tree Regression
dt_reg = DecisionTreeRegressor(random_state=12)
dt_reg.fit(X_train, y_train)
dt_reg_metrics=evaluate_model(dt_reg,X_test,y_test)

# 8. Random Forest Regression
rf_reg = RandomForestRegressor(random_state=12)
rf_reg.fit(X_train, y_train)
rf_reg_metrics=evaluate_model(rf_reg,X_test,y_test)

# Evaluate all models
models = ['Simple LR', 'Multiple LR', 'Polynomial LR', 'Ridge', 'Lasso', 'SVR', 'Decision
metrics = [simple_linear_reg_metrics, multiple_linear_reg_metrics, poly_reg_metrics, ridg
results = pd.DataFrame(metrics, columns=['MAE', 'MSE', 'RMSE', 'R2'], index=models)
```

```
print(results)
```

```
                     MAE          MSE       RMSE        R2
Simple LR      20.943821   846.704924  29.098195  0.728816
Multiple LR    19.117010   735.675199  27.123333  0.764377
Polynomial LR  15.648078   584.202988  24.170291  0.812891
Ridge          19.116987   735.672527  27.123284  0.764378
Lasso          19.210513   740.873741  27.218996  0.762712
SVR            18.695228   776.086387  27.858327  0.751434
Decision Tree  20.987106  1139.842247  33.761550  0.634930
Random Forest  14.250897   543.056822  23.303580  0.826069
```

In [19]:

```
# Tune the best perfoming model in this case Random Forest regression

# Define the parameter grid for hyperparameter tuning
param_grid_rf = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

# Create the Random Forest Regression model
rf_reg = RandomForestRegressor(random_state=12)

# Perform GridSearchCV for hyperparameter tuning
grid_search_rf = GridSearchCV(estimator=rf_reg, param_grid=param_grid_rf, cv=5, n_jobs=-1
grid_search_rf.fit(X_train, y_train)

# Get the best model and its evaluation metrics on the test set
best_rf_reg = grid_search_rf.best_estimator_
best_rf_reg_metrics = evaluate_model(best_rf_reg, X_test, y_test)


print("Best hyperparameters for Random Forest:", grid_search_rf.best_params_)
print("Best score for Random Forest:", -grid_search_rf.best_score_)
```

```
Best hyperparameters for Random Forest: {'max_depth': None, 'min_samples_l
eaf': 4, 'min_samples_split': 2, 'n_estimators': 300}
Best score for Random Forest: -0.8215618059503266
```

In [ ]:

```
#Tune model using Randomized search
# Define the hyperparameter distribution
param_dist = {
    'n_estimators': randint(10, 200),
    'max_depth': randint(1, 30),
    'min_samples_split': randint(2, 10)
}

# Create a random forest regressor
rforest = RandomForestRegressor(random_state=12)

# Instantiate the randomized search with cross-validation
kfold = KFold(n_splits=5, shuffle=True, random_state=12)
random_search = RandomizedSearchCV(estimator=rforest, param_distributions=param_dist, n_i

# Fit the randomized search to the data
random_search.fit(X_train, y_train)

# Print the best combination of hyperparameters and their score
print("Best hyperparameters:", random_search.best_params_)
print("Best score:", -random_search.best_score_)
```

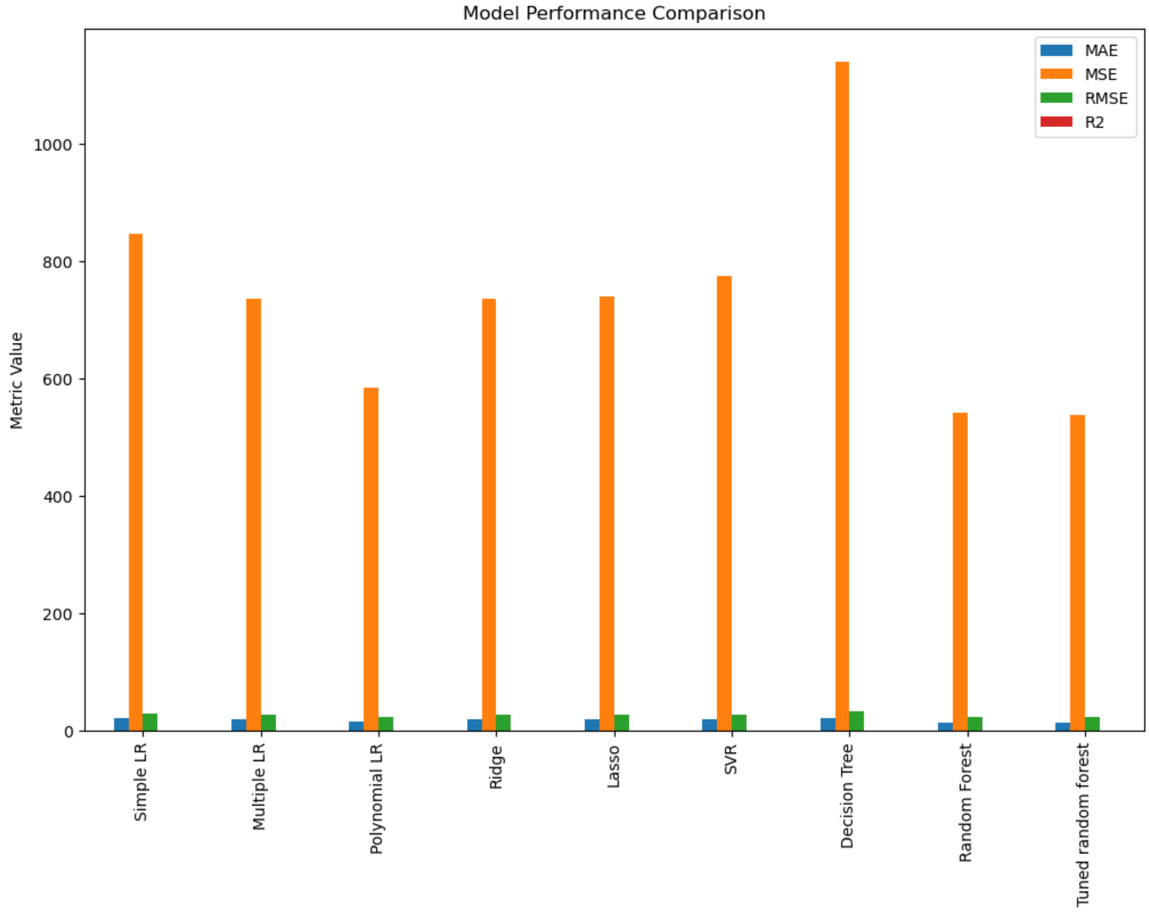
In [21]:

```
# Evaluate all models
models = ['Simple LR', 'Multiple LR', 'Polynomial LR', 'Ridge', 'Lasso', 'SVR', 'Decision
metrics = [simple_linear_reg_metrics, multiple_linear_reg_metrics, poly_reg_metrics, ridg
results = pd.DataFrame(metrics, columns=['MAE', 'MSE', 'RMSE', 'R2'], index=models)
print(results)
```

```
                          MAE          MSE       RMSE        R2
Simple LR           20.943821   846.704924  29.098195  0.728816
Multiple LR         19.117010   735.675199  27.123333  0.764377
Polynomial LR       15.648078   584.202988  24.170291  0.812891
Ridge               19.116987   735.672527  27.123284  0.764378
Lasso               19.210513   740.873741  27.218996  0.762712
SVR                 18.695228   776.086387  27.858327  0.751434
Decision Tree       20.987106  1139.842247  33.761550  0.634930
Random Forest       14.250897   543.056822  23.303580  0.826069
Tuned random forest 14.042782   537.855046  23.191702  0.827735
```

In [22]:

```
# Visualize the performance of each model using a bar plot
import matplotlib.pyplot as plt
results.plot(kind='bar', figsize=(12, 8), ylabel='Metric Value', title='Model Performance
plt.show()
```

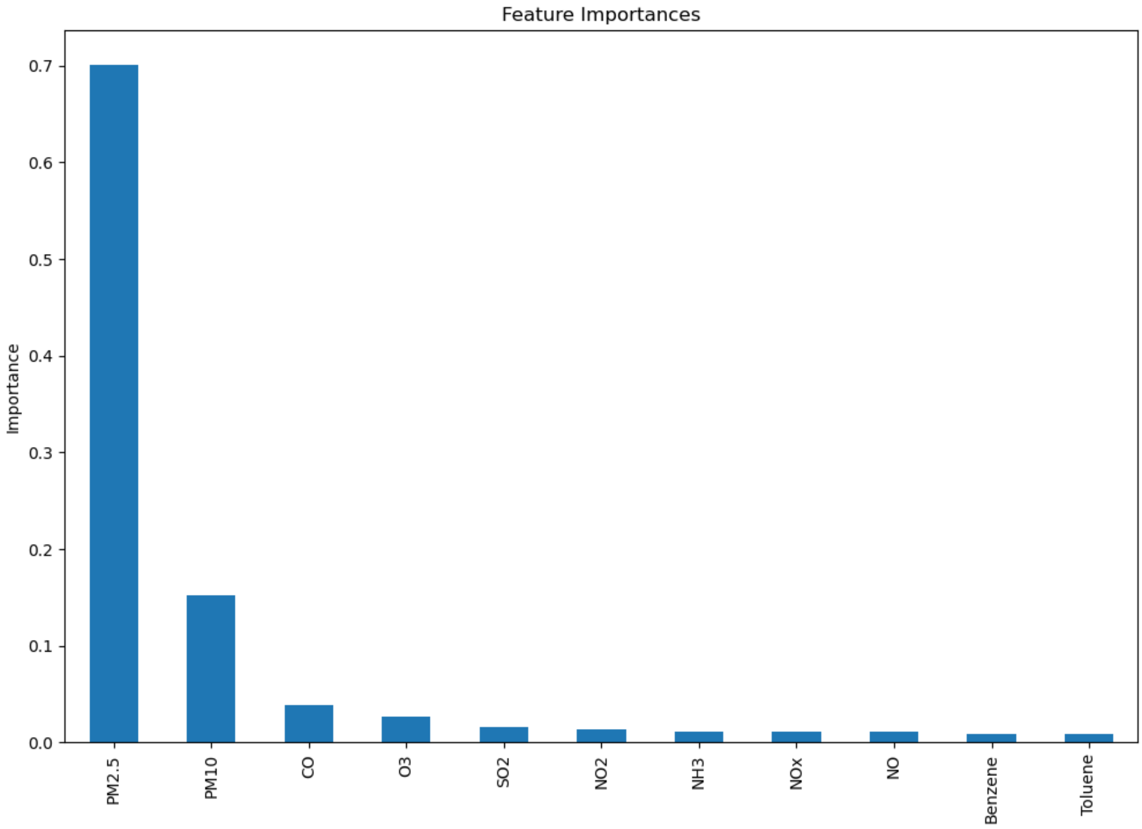


In [23]:

```
# Select the best model (e.g., tuned random forest)
best_model =best_rf_reg
```

In [25]:

```
# Analyze feature importances for the Random Forest model
importances = best_model.feature_importances_
feature_importances = pd.Series(importances, index=X.columns)
feature_importances.sort_values(ascending=False).plot(kind='bar', figsize=(12, 8), ylabel
plt.show()
```



In [26]:

```
#save the best model
import joblib
joblib.dump(best_model,'regression_model.pkl')
```

Out[26]:

```
['regression_model.pkl']
```

In [ ]:

In [ ]: