

# AIR QUALITY AND ANALYSIS AND PREDICTION IN TAMILNADU

## FEATURE ENGINEERING:

Feature Engineering in data science is the process of creating new features from existing ones or transforming the data to improve the performance of machine learning models. It involves selecting, modifying, or combining features to make them more informative and relevant for the task at hand. Effective feature engineering can enhance model accuracy, reduce overfitting, and uncover hidden patterns in the data. It requires domain knowledge and creativity to extract meaningful information, such as using one-hot encoding for categorical variables, generating interaction terms, or scaling numerical features.

## Import Libraries and Load Data:

Import the necessary libraries and load the data from the dataset. We will employ the pandas library to load our dataset. Pandas is a powerful data manipulation library in python that provides an efficient and flexible way to work with structured data.

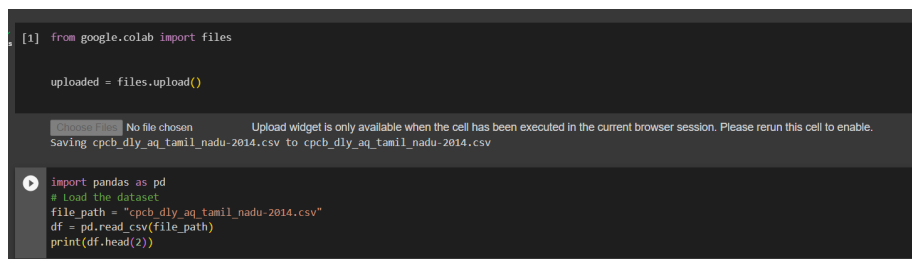
## Dataset Link:

<https://tn.data.gov.in/resource/location-wise-daily-ambient-air-quality-tamil-nadu-year-2014>

## INPUT:

```
import pandas as pd
# Load the dataset
file_path = "cpcb_dly_aq_tamil_nadu-2014.csv"
df = pd.read_csv(file_path)
```

## OUTPUT:



```
[1] from google.colab import files

uploaded = files.upload()

Choose File No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving cpcb_dly_aq_tamil_nadu-2014.csv to cpcb_dly_aq_tamil_nadu-2014.csv

import pandas as pd
# Load the dataset
file_path = "cpcb_dly_aq_tamil_nadu-2014.csv"
df = pd.read_csv(file_path)
print(df.head(2))
```

## DATA EXPLORATION:

Before dive into preprocessing, it's a crucial to explore the dataset.

## INPUT:

```
print(df.head(2))
```

## OUTPUT:

```
Stn Code Sampling Date      State City/Town/Village/Area \
0      38      01-02-14    Tamil Nadu      Chennai
1      38      01-07-14    Tamil Nadu      Chennai

Location of Monitoring Station \
0 Kathivakkam, Municipal Kalyana Mandapam, Chennai
1 Kathivakkam, Municipal Kalyana Mandapam, Chennai

Agency Type of location SO2 NO2 \
0 Tamilnadu State Pollution Control Board Industrial Area 11.0 17.0
1 Tamilnadu State Pollution Control Board Industrial Area 13.0 17.0

RSPM/PM10 PM 2.5
0      55.0    NaN
1      45.0    NaN
```

## DATE AND TIME HANDLING:

Extracting 'Year' and 'Month' helps the model capture potential seasonal or temporal variations in air quality, which can be crucial for forecasting. Convert the 'Sampling Date' column into separate 'Year' and 'Month' features. This allows the model to capture potential seasonal or yearly patterns in air quality.

## INPUT:

```
#Date and Time Handling
df['Sampling Date'] = pd.to_datetime(df['Sampling Date'],
format='%d-%m-%y')
df['Year'] = df['Sampling Date'].dt.year
df['Month'] = df['Sampling Date'].dt.month

# Display the DataFrame after date handling
print("DataFrame after Date and Time Handling:")
print(df[['Sampling Date', 'Year', 'Month']])
```

## OUTPUT:

```
DataFrame after Date and Time Handling:
Sampling Date Year Month
0      2014-02-01 2014     2
1      2014-07-01 2014     7
2      2014-01-21 2014     1
3      2014-01-23 2014     1
4      2014-01-28 2014     1
...
2874    2014-03-12 2014     3
2875    2014-10-12 2014    10
2876    2014-12-17 2014    12
2877    2014-12-24 2014    12
2878    2014-12-31 2014    12

[2879 rows x 3 columns]
```

## COMBINING POLLUTANTS:

Creating the 'Pollution Index' combines multiple pollutant measurements into a single feature, providing a more informative representation of air quality. Create a new feature, 'Pollution Index,' by combining the 'SO2' and 'NO2' values. This feature provides a holistic view of air pollution.

### INPUT:

```
#Combining Pollutants
df['Pollution Index'] = df['SO2'] + df['NO2']

# Display the DataFrame after combining pollutants
print("DataFrame after Combining Pollutants:")
print(df[['SO2', 'NO2', 'Pollution Index']])
```

### OUTPUT:

```
DataFrame after Combining Pollutants:
   SO2  NO2  Pollution Index
0  11.0  17.0             28.0
1  13.0  17.0             30.0
2  12.0  18.0             30.0
3  15.0  16.0             31.0
4  13.0  14.0             27.0
...   ...   ...           ...
2874  15.0  18.0             33.0
2875  12.0  14.0             26.0
2876  19.0  22.0             41.0
2877  15.0  17.0             32.0
2878  14.0  16.0             30.0

[2879 rows x 3 columns]
```

## LOCATION ENCODING:

Encode categorical variables like 'Type of Location' and 'Agency' using one-hot encoding to convert them into numerical features.

### INPUT:

```
# Feature 3: Location Encoding
categorical_features = ['Type of Location', 'Agency']
df = pd.get_dummies(df, columns=categorical_features,
prefix=categorical_features)

# Display the DataFrame after location encoding
print("DataFrame after Location Encoding:")
print(df.head(2))
```

## OUTPUT:

```
DataFrame after Location Encoding:
  Stn Code Sampling Date      State City/Town/Village/Area \
0      38      01-02-14  Tamil Nadu      Chennai
1      38      01-07-14  Tamil Nadu      Chennai

  Location of Monitoring Station  SO2  NO2  RSPM/PM10 \
0  Kathivakkam, Municipal Kalyana Mandapam, Chennai  11.0  17.0    55.0
1  Kathivakkam, Municipal Kalyana Mandapam, Chennai  13.0  17.0    45.0

  PM 2.5  Type of Location_Industrial Area \
0      NaN                               1
1      NaN                               1

  Type of Location_Residential, Rural and other Areas \
0
1

  Agency_National Environmental Engineering Research Institute \
0
1

  Agency_Tamilnadu State Pollution Control Board
0      1
1      1
```

## MODEL TRAINING AND EVALUATION:

Model training is the process of feeding a machine learning model with labeled data to enable it to learn patterns and make predictions. During training, the model optimizes its parameters to minimize the difference between its predictions and the actual labels. This is typically achieved using an algorithm and involves techniques like gradient descent.

Model evaluation is the stage where the model's performance is assessed on unseen data. This is done by measuring various metrics, such as accuracy, precision, recall, or F1-score, to determine how well the model generalizes and whether it's suitable for its intended task. The model may also be compared to other models to select the best performer. Evaluation is crucial for ensuring the model's reliability and effectiveness in real-world applications.

## LINEAR REGRESSION:

LinearRegression is a machine learning algorithm used for predicting numerical values based on input features. It models a linear relationship between the features and the target variable. The algorithm finds the best-fitting line or hyperplane by minimizing the sum of squared differences between predicted and actual values. It's suitable for regression tasks and provides coefficients that indicate the impact of each feature on the target. LinearRegression is widely used in various fields, including finance, economics, and science, for making predictions and understanding data relationships.

## INPUT:

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
```

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.impute import SimpleImputer

# Load the dataset
data = pd.read_csv('cpcb_dly_aq_tamil_nadu-2014.csv')

# Choose your target variable and features
target_variable = 'RSPM/PM10'
features = ['SO2', 'NO2']

# Split the data into features (X) and target (y)
X = data[features]
y = data[target_variable]

# Impute missing values in both features and the target variable
imputer = SimpleImputer(strategy='mean')
X = imputer.fit_transform(X)
y = imputer.fit_transform(y.values.reshape(-1, 1))

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=0)

# Create a linear regression model
model = LinearRegression()

# Fit the model to the training data
model.fit(X_train, y_train)

# Make predictions on the test data
y_pred = model.predict(X_test)

# Calculate model performance metrics
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

# Print the model coefficients and metrics
print("Model Coefficients: ", model.coef_)
print("Mean Squared Error: ", mse)
print("R-squared (R2) Score: ", r2)
```

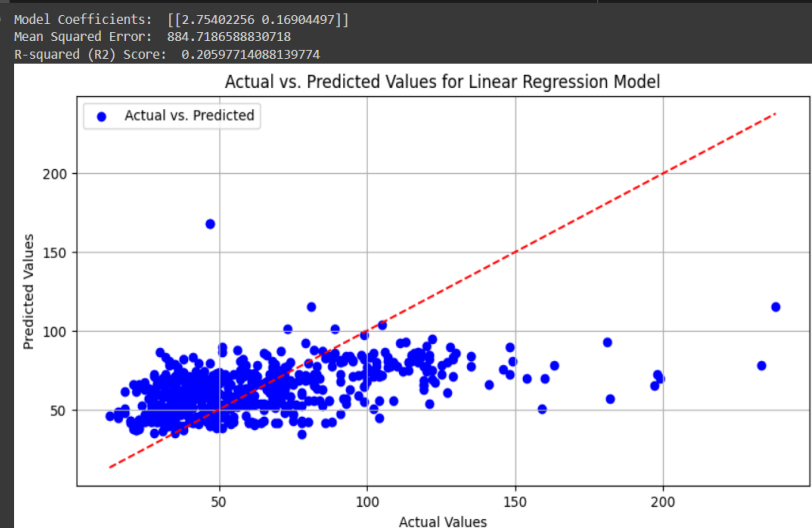
```

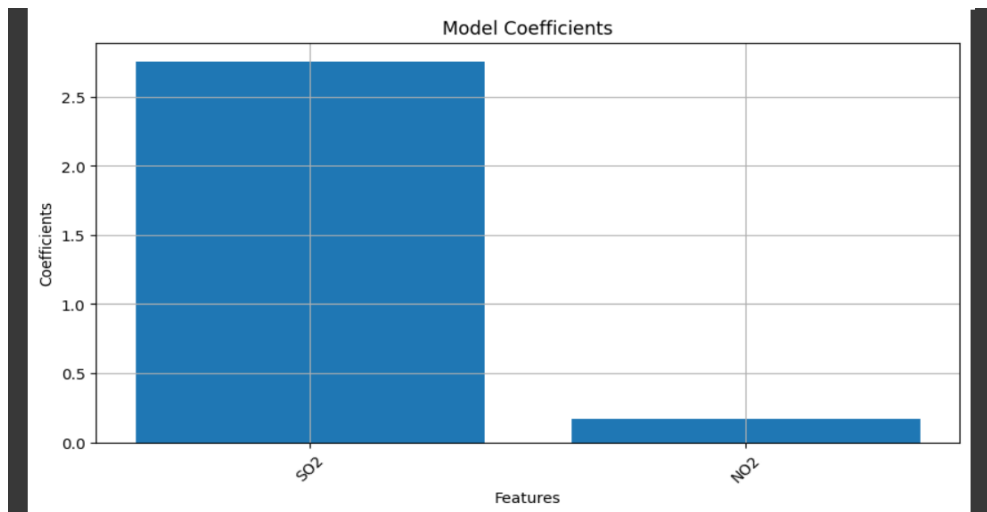
# Visualize the actual vs. predicted values
plt.figure(figsize=(10, 5))
plt.scatter(y_test, y_pred, c='blue', label='Actual vs. Predicted')
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)],
color='red', linestyle='--')
plt.xlabel("Actual Values")
plt.ylabel("Predicted Values")
plt.title("Actual vs. Predicted Values for Linear Regression Model")
plt.legend(loc='upper left')
plt.grid(True)
plt.show()

# Visualize the model coefficients
coefficients = model.coef_[0]
plt.figure(figsize=(10, 5))
plt.bar(features, coefficients)
plt.xlabel("Features")
plt.ylabel("Coefficients")
plt.title("Model Coefficients")
plt.grid(True)
plt.xticks(rotation=45)
plt.show()

```

## OUTPUT:





### RANDOM FOREST REGRESSION:

Random Forest Regression is an ensemble machine learning technique used for regression tasks. It combines multiple decision trees to make accurate predictions. Each tree in the forest independently predicts the target variable, and the final prediction is an average or weighted combination of these individual predictions. This approach reduces overfitting and enhances predictive accuracy. Random Forest Regression handles complex relationships between features and the target, making it suitable for a wide range of applications, such as predicting stock prices, house values, and more. It also provides feature importance scores, helping identify the most influential variables in the prediction.

### INPUT:

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.impute import SimpleImputer

# Load the dataset
data = pd.read_csv('cpcb_dly_aq_tamil_nadu-2014.csv')

# Select features and target
features = ['SO2', 'NO2']
target = 'RSPM/PM10'

# Handle missing values in features
feature_imputer = SimpleImputer(strategy='mean')
data[features] = feature_imputer.fit_transform(data[features])
```

```

# Handle missing values in the target variable
target_imputer = SimpleImputer(strategy='mean')
data[target] = target_imputer.fit_transform(data[[target]])

# Split the data into training and testing sets
X = data[features]
y = data[target]
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=0)

# Create a Random Forest regression model
model = RandomForestRegressor(n_estimators=100, random_state=0) # You
can adjust the number of estimators (trees)

# Fit the model to the training data
model.fit(X_train, y_train)

# Make predictions on the test data
y_pred = model.predict(X_test)

# Calculate model performance metrics
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

# Print the model's performance
print("Mean Squared Error: ", mse)
print("R-squared (R2) Score: ", r2)

# Visualize the actual vs. predicted values
plt.figure(figsize=(10, 5))
plt.scatter(y_test, y_pred, c='blue', label='Actual vs. Predicted')
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)],
color='red', linestyle='--')
plt.xlabel("Actual Values")
plt.ylabel("Predicted Values")
plt.title("Actual vs. Predicted Values for Random Forest Regression
Model")
plt.legend(loc='upper left')
plt.grid(True)
plt.show()

```



```
# Visualize feature importances
feature_importances = model.feature_importances_
plt.figure(figsize=(10, 5))
sns.barplot(x=features, y=feature_importances)
plt.xlabel("Features")
plt.ylabel("Feature Importance")
plt.title("Feature Importances for Random Forest Regression Model")
plt.xticks(rotation=45)
plt.grid(True)
plt.show()
```

OUTPUT:

