Name : Shreyash Kamat **Div : D15C** **Roll No : 22**

Experiment No. - 6 :

Problem Statement: Classification modelling

a. Choose a classifier for a classification problem.
b. Evaluate the performance of the classifier.
Perform Classification using the below 4 classifiers on the same dataset which you have used for

- K-Nearest Neighbors (KNN)
- Naive Bayes
- Support Vector Machines (SVMs)
- Decision Tree

Introduction :

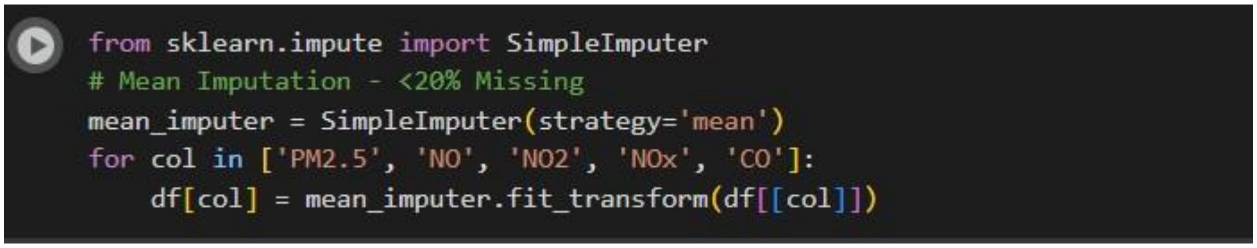
Classification algorithms are crucial in predicting categorical outcomes and analyzing labeled data. In this experiment, we applied two supervised learning techniques — Naive Bayes and K-Nearest Neighbors (KNN) — to classify air quality based on the Air Quality Index (AQI) dataset.

Naive Bayes is a probabilistic classifier based on Bayes' theorem, known for its simplicity, efficiency, and effectiveness, especially with small datasets or when feature independence is assumed. On the other hand, KNN is a distance-based classifier that assigns a class label based on the majority of the nearest neighbors, making it intuitive yet sensitive to feature scaling and noise.

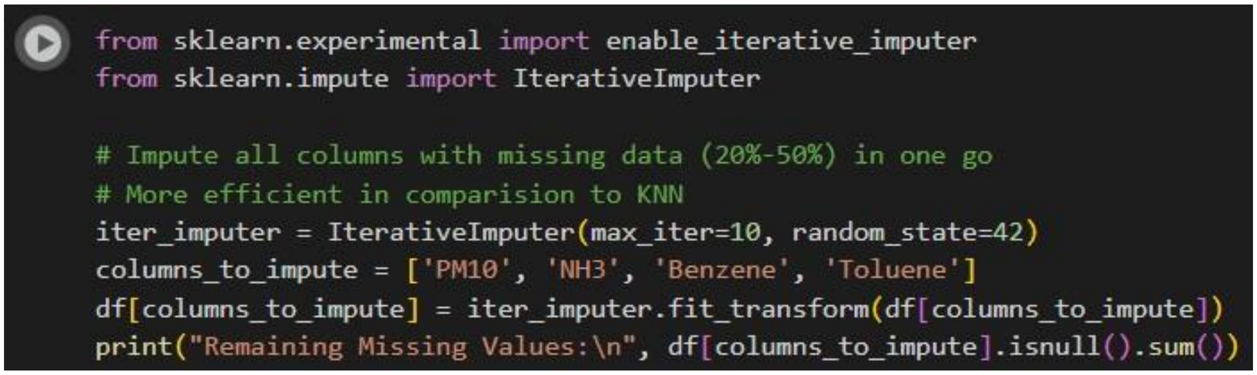
The primary objective was to classify AQI levels as either "Good" or "Bad" and evaluate the effectiveness of these algorithms. By comparing their accuracy, precision, recall, and F1-score, we gained insights into the suitability of these methods for air quality classification.

Implementation Process :

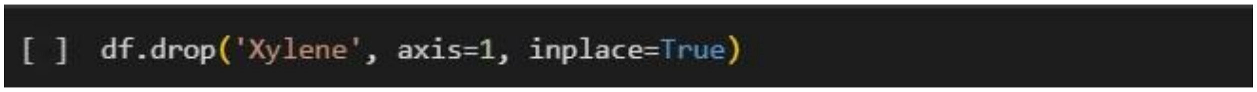
Step 1 : We perform a series of data pre-processing operations that involve calculating the missing percentage for each column ,using mean imputation for columns that have less than 20 % missing values,using iterative imputation methods for columns between 20 - 50% missing values and dropping the xylene column that had around 60 % missing values.



```python
from sklearn.impute import SimpleImputer
# Mean Imputation - <20% Missing
mean_imputer = SimpleImputer(strategy='mean')
for col in ['PM2.5', 'NO', 'NO2', 'NOx', 'CO']:
    df[col] = mean_imputer.fit_transform(df[[col]])
```

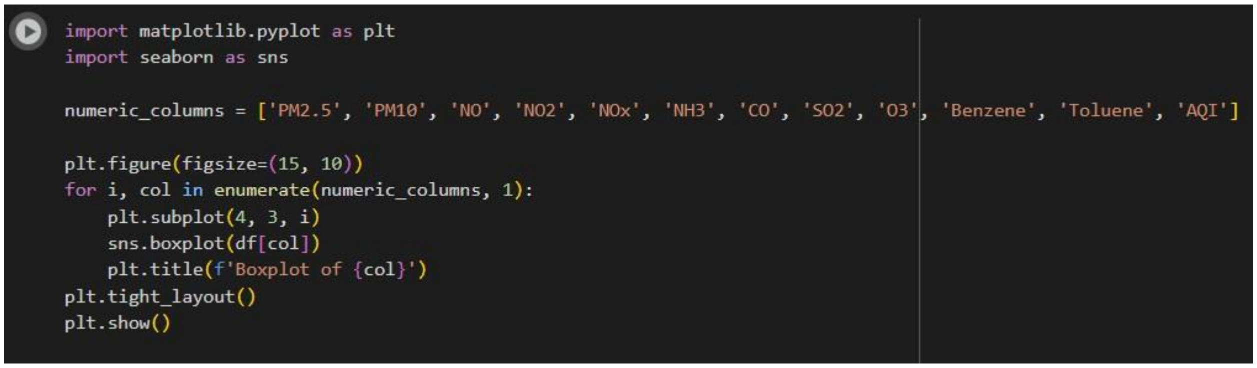


```
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer

# Impute all columns with missing data (20%-50%) in one go
# More efficient in comparision to KNN
iter_imputer = IterativeImputer(max_iter=10, random_state=42)
columns_to_impute = ['PM10', 'NH3', 'Benzene', 'Toluene']
df[columns_to_impute] = iter_imputer.fit_transform(df[columns_to_impute])
print("Remaining Missing Values:\n", df[columns_to_impute].isnull().sum())
```

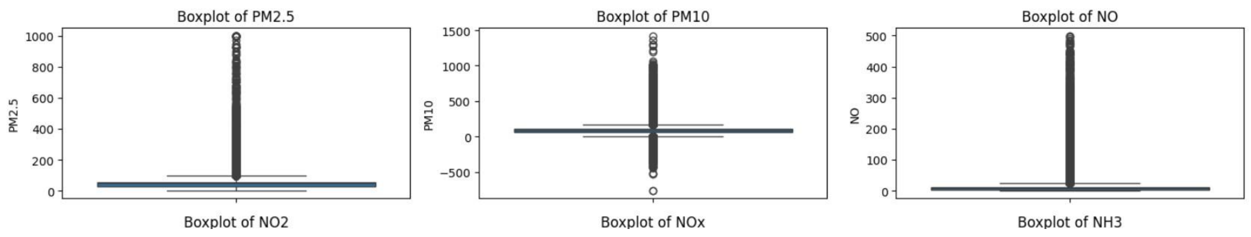


```
[ ]  df.drop('Xylene', axis=1, inplace=True)
```

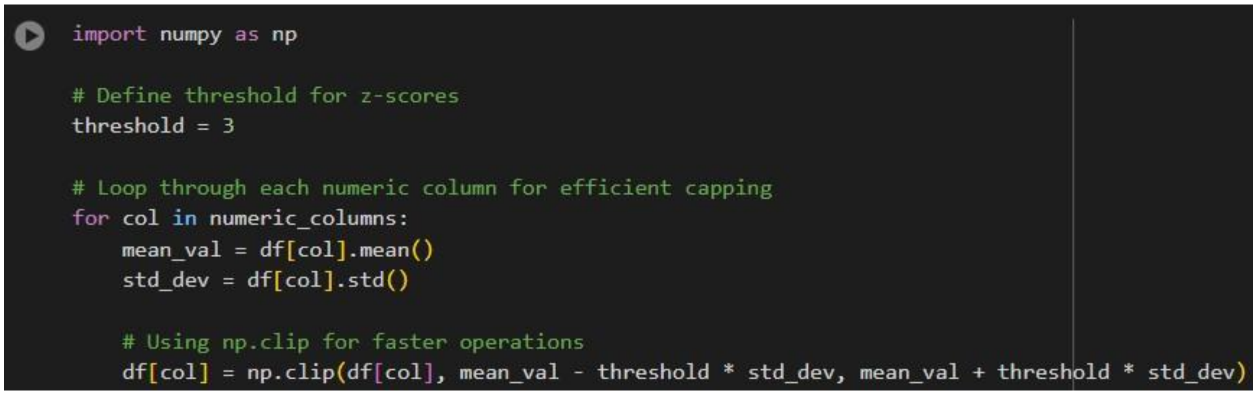
Step 2 : We verify that all of the missing values have been removed and then move to identifying and eliminating the outliers using boxplot analysis.



```
import matplotlib.pyplot as plt
import seaborn as sns

numeric_columns = ['PM2.5', 'PM10', 'NO', 'NO2', 'NOx', 'NH3', 'CO', 'SO2', 'O3', 'Benzene', 'Toluene', 'AQI']

plt.figure(figsize=(15, 10))
for i, col in enumerate(numeric_columns, 1):
    plt.subplot(4, 3, i)
    sns.boxplot(df[col])
    plt.title(f'Boxplot of {col}')
plt.tight_layout()
plt.show()
```



Step 3 : Apply standardizing operations using the z-score method.

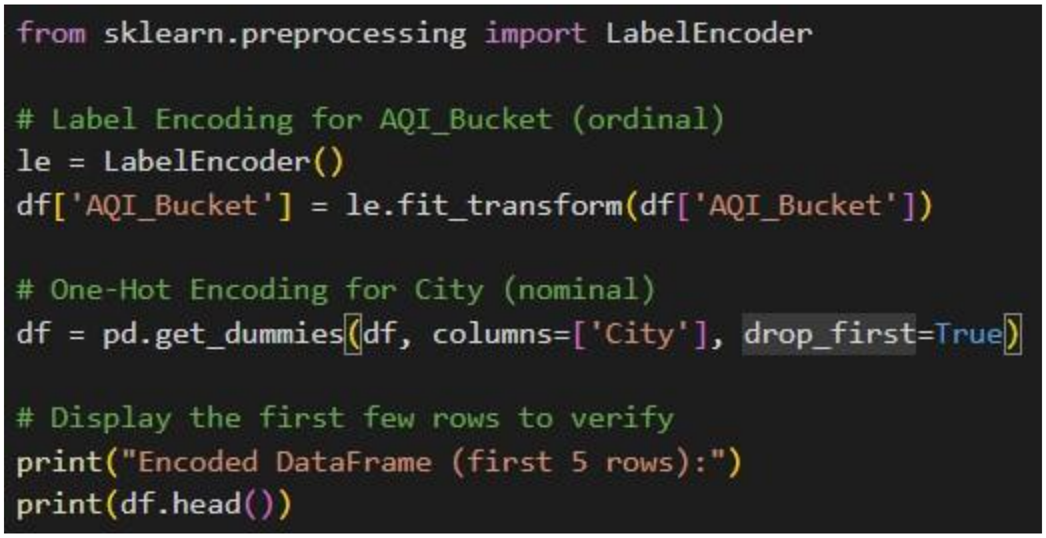


```python
import numpy as np

# Define threshold for z-scores
threshold = 3

# Loop through each numeric column for efficient capping
for col in numeric_columns:
    mean_val = df[col].mean()
    std_dev = df[col].std()

    # Using np.clip for faster operations
    df[col] = np.clip(df[col], mean_val - threshold * std_dev, mean_val + threshold * std_dev)
```

Step 4 : We then encode categorical features in the dataset to prepare it for machine learning algorithms, as our model requires numerical data.

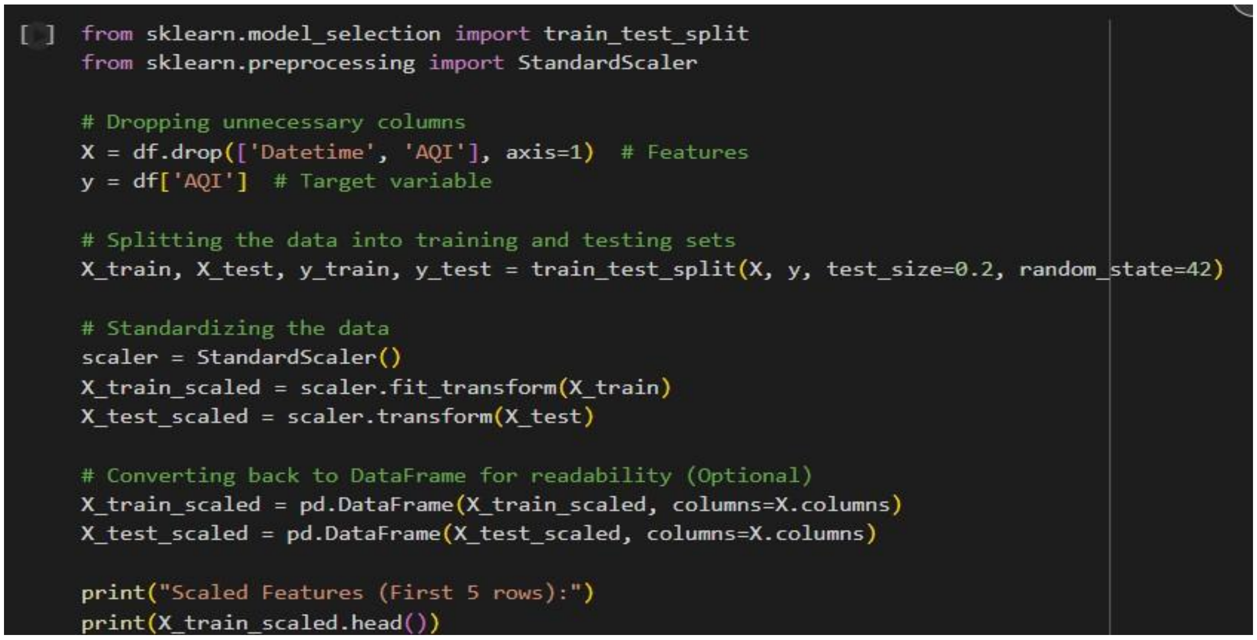
The LabelEncoder converts the categorical values of the AQI_Bucket column (like "Good", "Satisfactory", "Poor", etc.) into integer labels (0, 1, 2, etc.).

One-Hot Encoding creates binary columns for each unique value in the City column (like Delhi, Mumbai, etc.).



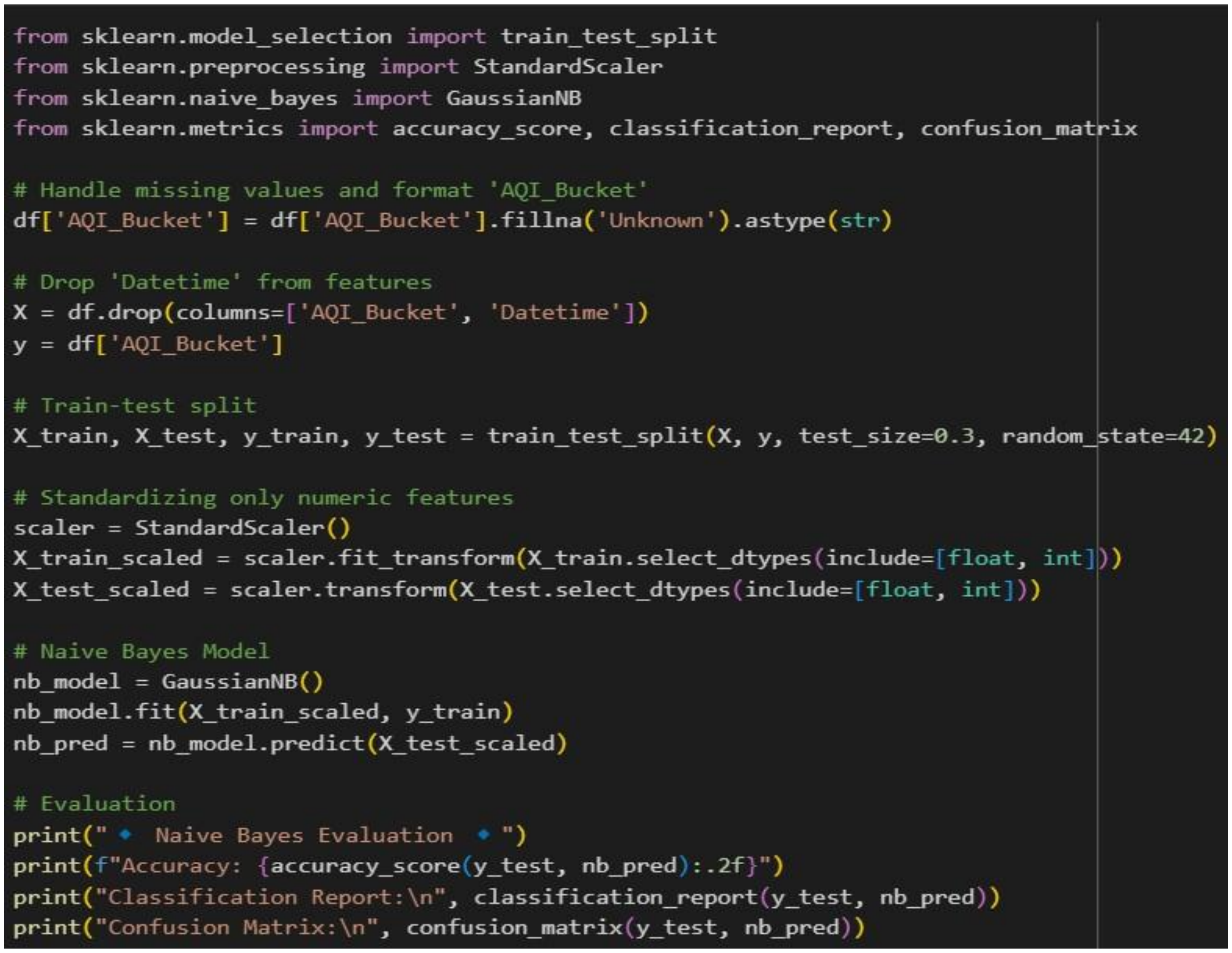
```python
from sklearn.preprocessing import LabelEncoder

# Label Encoding for AQI_Bucket (ordinal)
le = LabelEncoder()
df['AQI_Bucket'] = le.fit_transform(df['AQI_Bucket'])

# One-Hot Encoding for City (nominal)
df = pd.get_dummies(df, columns=['City'], drop_first=True)

# Display the first few rows to verify
print("Encoded DataFrame (first 5 rows):")
print(df.head())
```

Step 5 : We then perform splitting of the dataset into a 70 : 30 split to proceed with the preparation and evaluation of the models.



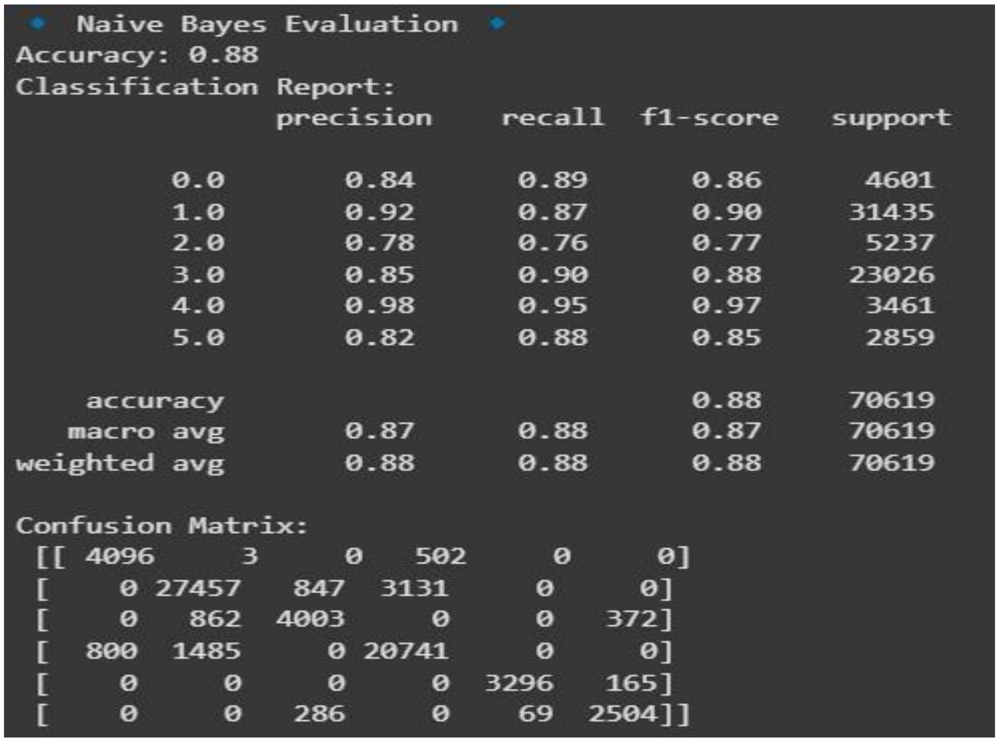
```
[ ]  from sklearn.model_selection import train_test_split
     from sklearn.preprocessing import StandardScaler

     # Dropping unnecessary columns
     X = df.drop(['Datetime', 'AQI'], axis=1)  # Features
     y = df['AQI']  # Target variable

     # Splitting 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=42)

     # Standardizing the data
     scaler = StandardScaler()
     X_train_scaled = scaler.fit_transform(X_train)
     X_test_scaled = scaler.transform(X_test)

     # Converting back to DataFrame for readability (Optional)
     X_train_scaled = pd.DataFrame(X_train_scaled, columns=X.columns)
     X_test_scaled = pd.DataFrame(X_test_scaled, columns=X.columns)

     print("Scaled Features (First 5 rows):")
     print(X_train_scaled.head())
```

Step 6 : The preparation of the Naive Bayes Classifier for application on to the dataset.



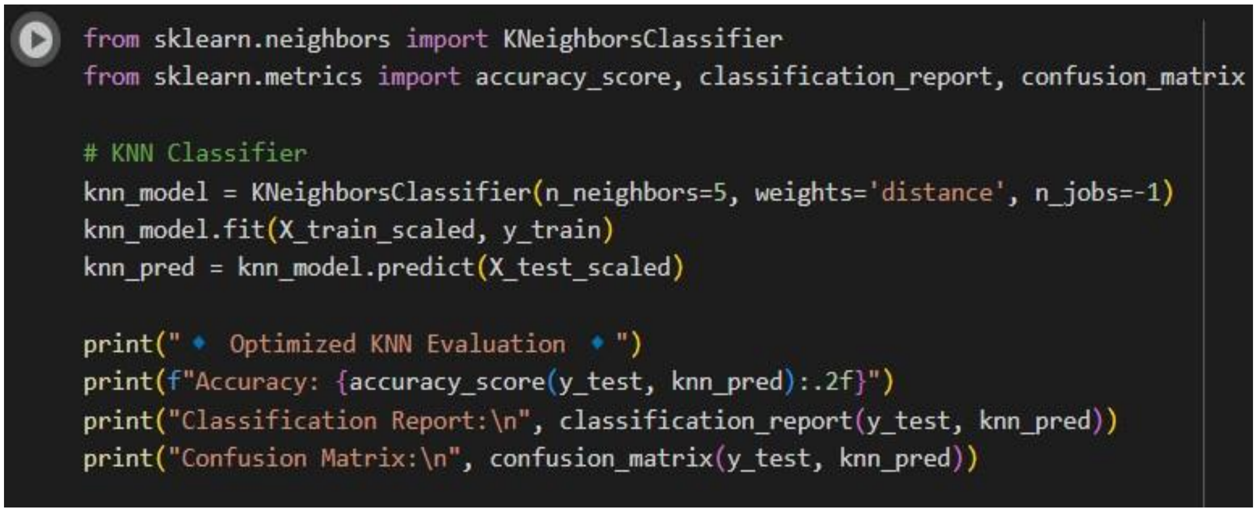
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# Handle missing values and format 'AQI_Bucket'
df['AQI_Bucket'] = df['AQI_Bucket'].fillna('Unknown').astype(str)

# Drop 'Datetime' from features
X = df.drop(columns=['AQI_Bucket', 'Datetime'])
y = df['AQI_Bucket']

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Standardizing only numeric features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train.select_dtypes(include=[float, int]))
X_test_scaled = scaler.transform(X_test.select_dtypes(include=[float, int]))

# Naive Bayes Model
nb_model = GaussianNB()
nb_model.fit(X_train_scaled, y_train)
nb_pred = nb_model.predict(X_test_scaled)

# Evaluation
print(" 🔹 Naive Bayes Evaluation 🔹 ")
print(f"Accuracy: {accuracy_score(y_test, nb_pred):.2f}")
print("Classification Report:\n", classification_report(y_test, nb_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, nb_pred))
```

This classifier is then evaluated for accuracy and a classification report and a confusion matrix is prepared for the same.



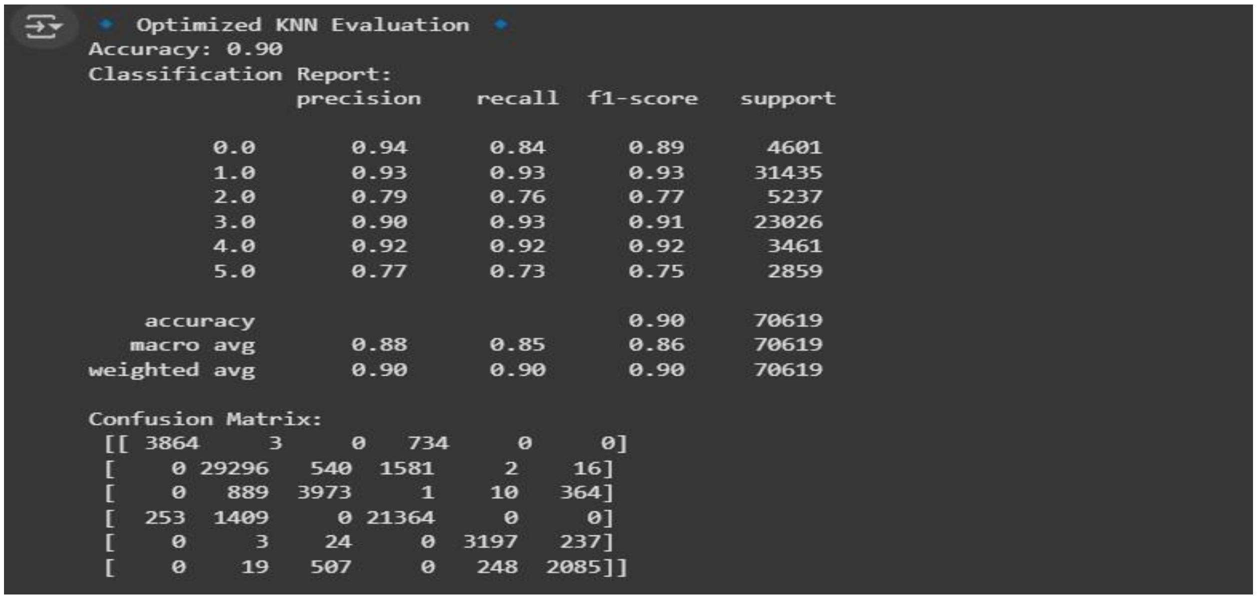
```
 ◆  Naive Bayes Evaluation  ◆
Accuracy: 0.88
Classification Report:
               precision    recall  f1-score   support

         0.0       0.84      0.89      0.86      4601
         1.0       0.92      0.87      0.90     31435
         2.0       0.78      0.76      0.77      5237
         3.0       0.85      0.90      0.88     23026
         4.0       0.98      0.95      0.97      3461
         5.0       0.82      0.88      0.85      2859

    accuracy                           0.88     70619
   macro avg       0.87      0.88      0.87     70619
weighted avg       0.88      0.88      0.88     70619

Confusion Matrix:
 [[ 4096     3     0   502     0     0]
 [    0 27457   847  3131     0     0]
 [    0   862  4003     0     0   372]
 [  800  1485     0 20741     0     0]
 [    0     0     0     0  3296   165]
 [    0     0   286     0    69  2504]]
```

Step 7 : The preparation of the KNN (K-Nearest Neighbors) for application on to the dataset.



```python
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# KNN Classifier
knn_model = KNeighborsClassifier(n_neighbors=5, weights='distance', n_jobs=-1)
knn_model.fit(X_train_scaled, y_train)
knn_pred = knn_model.predict(X_test_scaled)

print(" ◆  Optimized KNN Evaluation  ◆ ")
print(f"Accuracy: {accuracy_score(y_test, knn_pred):.2f}")
print("Classification Report:\n", classification_report(y_test, knn_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, knn_pred))
```

This classifier is then evaluated for accuracy and a classification report and a confusion matrix is prepared for the same.



```
🔹 Optimized KNN Evaluation 🔹
Accuracy: 0.90
Classification Report:
              precision    recall  f1-score   support

         0.0       0.94      0.84      0.89      4601
         1.0       0.93      0.93      0.93     31435
         2.0       0.79      0.76      0.77      5237
         3.0       0.90      0.93      0.91     23026
         4.0       0.92      0.92      0.92      3461
         5.0       0.77      0.73      0.75      2859

    accuracy                           0.90     70619
   macro avg       0.88      0.85      0.86     70619
weighted avg       0.90      0.90      0.90     70619

Confusion Matrix:
 [[ 3864     3     0   734     0     0]
 [    0 29296   540  1581     2    16]
 [    0   889  3973     1    10   364]
 [  253  1409     0 21364     0     0]
 [    0     3    24     0  3197   237]
 [    0    19   507     0   248  2085]]
```

Conclusion :

Based on the classification results obtained from both Naive Bayes and the optimized K-Nearest Neighbors (KNN) algorithms, the following conclusions can be drawn:

1. Model Performance: The optimized KNN model achieved a higher accuracy of 90% compared to 88% for Naive Bayes. This indicates that KNN is slightly better at correctly predicting the AQI categories for this dataset.

2. Evaluation Metrics: The KNN model consistently outperforms Naive Bayes across key metrics like precision, recall, and F1-score. Particularly, the recall scores indicate that KNN is better at correctly identifying instances for most classes.

3. Confusion Matrix Analysis: The confusion matrix for KNN shows fewer misclassifications across most categories, highlighting its robustness in handling complex decision boundaries for this dataset.

4. Model Suitability: Given the high dimensionality of the dataset, the performance of Naive Bayes is still noteworthy, as it is computationally efficient. However, KNN's higher overall metrics suggest better suitability for this dataset when classification accuracy is prioritized.

Both models show strong performance, but KNN, with optimized parameters, proves to be more effective for the classification task of predicting AQI categories. Depending on the use case—whether quick, interpretable results (Naive Bayes) or higher accuracy (KNN) are preferred—either model could be selected.