PROJECT REPORT

Course: Programming in Python
Section: B

Submitted By,

Name	ID
Anik Sen	20-42138-1
Tanvir Chowdhury	20-42699-1
Afrida Mahrin Chowdhury	20-42729-1
Sanjida Esha	20-42705-1

Date of Submission: 3rd May, 2023

Table of Contents

SECTION 1

Project Overview:

The goal of this project is to compare the performance of different classification models on a real dataset. The dataset has 9471 instances and requires data preprocessing and exploratory data analysis.

The project will be implemented in Python and will make use of the scikit-learn library for machine learning. The following classifiers will be used in this project:

1. Naive Bayes
2. K-Nearest Neighbors (KNN)
3. Decision Tree
4. Logistic Regression
5. Support Vector Machine (SVM)

The data cleaning process will involve dropping unnecessary columns and handling missing values using forward fill and backward fill techniques.

Exploratory data analysis will be carried out using histogram plots to analyze the distribution of each feature in the dataset. The correlation matrix will also be plotted as a heatmap to show the correlation between the features.

The dataset will be split into training and testing sets using the train_test_split method from scikit-learn. The models will be trained on the training set and tested on the testing set to evaluate their performance.

The performance of the models will be evaluated based on their predictive accuracy, which will be computed using the accuracy_score method from scikit-learn. Classification reports and confusion matrices will also be used to evaluate the performance of the models.

Finally, the results of the model comparison will be summarized and conclusions will be drawn based on the performance of the models. The project will demonstrate the effectiveness of different classification models and their suitability for different types of datasets.

SECTION 2

Data Set overview:

The dataset contains 9358 instances of hourly averaged responses from an array of 5 metal oxide chemical sensors embedded in an Air Quality Chemical Multisensor Device. The device was located on the field in a significantly polluted area, at road level, within an Italian city. Data were recorded from March 2004 to February 2005 (one year)representing the longest freely available recordings of on field deployed air quality chemical sensor devices responses. Ground Truth hourly averaged concentrations for CO, Non Metanic Hydrocarbons, Benzene, Total Nitrogen Oxides (NOx) and

Nitrogen Dioxide (NO2) and were provided by a co-located reference certified analyzer. Evidences of cross-sensitivities as well as both concept and sensor drifts are present as described in De Vito et al., Sens. And Act. B, Vol. 129,2,2008 (citation required) eventually affecting sensors concentration estimation capabilities. Missing values are tagged with -200 value. This dataset can be used exclusively for research purposes. Commercial purposes are fully excluded.

Attribute Information:

0 Date (DD/MM/YYYY)

1 Time (HH.MM.SS)

2 True hourly averaged concentration CO in mg/m^3 (reference analyzer)

3 PT08.S1 (tin oxide) hourly averaged sensor response (nominally CO targeted)

4 True hourly averaged overall Non Metanic HydroCarbons concentration in microg/m^3 (reference analyzer)

5 True hourly averaged Benzene concentration in microg/m^3 (reference analyzer)

6 PT08.S2 (titania) hourly averaged sensor response (nominally NMHC targeted)

7 True hourly averaged NOx concentration in ppb (reference analyzer)

8 PT08.S3 (tungsten oxide) hourly averaged sensor response (nominally NOx targeted)

9 True hourly averaged NO2 concentration in microg/m^3 (reference analyzer)

10 PT08.S4 (tungsten oxide) hourly averaged sensor response (nominally NO2 targeted)

11 PT08.S5 (indium oxide) hourly averaged sensor response (nominally O3 targeted)

12 Temperature in Â°C

13 Relative Humidity (%)

14 AH Absolute Humidity

Dataset reference link:

https://archive.ics.uci.edu/ml/datasets/air+quality?fbclid=IwAR1swsn_wCInfHjjBlr26ETclr0hZ_IFeBIfMiT929tpeYQlsuffFG0HYgk

SECTION 3

Data Preprocessing:

In the code, unnecessary columns are dropped and missing values are handled using forward fill and backward fill methods, which are common techniques in data preprocessing to ensure that the dataset is cleaned and ready for analysis.

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score
```

In [2]:
```
df = pd.read_csv('AirQualityUCI.csv', delimiter=';', decimal=',')
```

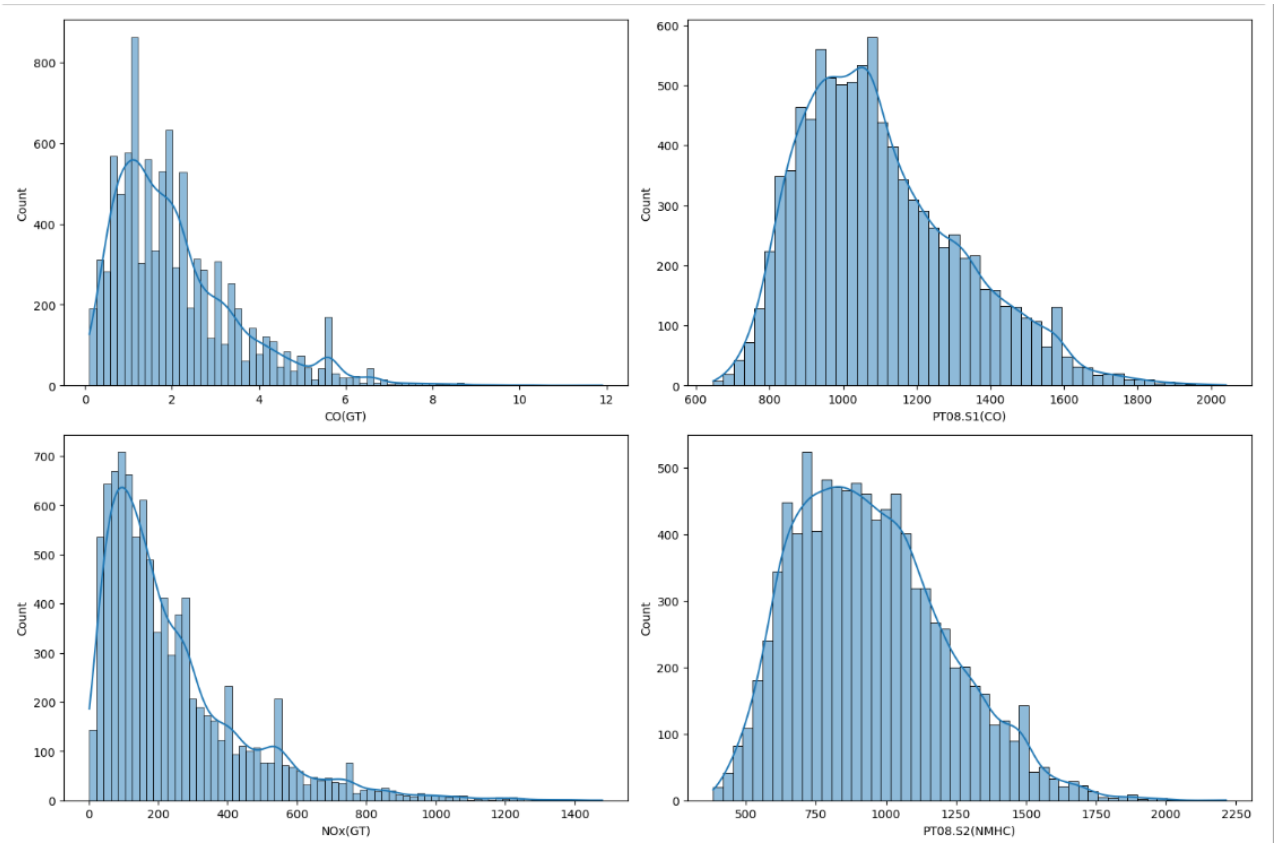
In [3]:
```
# drop unnecessary columns
df.drop(columns=['Date', 'Time', 'Unnamed: 15', 'Unnamed: 16'], inplace=True)
```

In [4]:
```
# handle missing values
df.replace(to_replace=-200, value=np.nan, inplace=True)
df.fillna(method='ffill', inplace=True)
df.fillna(method='bfill', inplace=True)
```

Exploratory Data Analysis for Air Quality Dataset:

The code generates four histograms for four different variables to visualize their distributions in the dataset. These variables are CO(GT), PT08.S1(CO), NOx(GT), and PT08.S2(NMHC). By analyzing these histograms, one can get an idea of the range, skewness, and shape of the distribution of each variable.or Air Quality Dataset

In [5]:
```
# perform exploratory data analysis
fig, axs = plt.subplots(2, 2, figsize=(15, 10))

sns.histplot(data=df, x='CO(GT)', kde=True, ax=axs[0][0])
axs[0][0].set_xlabel('CO(GT)')

sns.histplot(data=df, x='PT08.S1(CO)', kde=True, ax=axs[0][1])
axs[0][1].set_xlabel('PT08.S1(CO)')

sns.histplot(data=df, x='NOx(GT)', kde=True, ax=axs[1][0])
axs[1][0].set_xlabel('NOx(GT)')

sns.histplot(data=df, x='PT08.S2(NMHC)', kde=True, ax=axs[1][1])
axs[1][1].set_xlabel('PT08.S2(NMHC)')

plt.tight_layout()
plt.show()
```



The code performs exploratory data analysis on a dataset using the Seaborn library. It creates a 2x2 grid of subplots using matplotlib, with each subplot containing a histogram plot of a different variable from the dataset. The variables plotted are CO(GT), PT08.S1(CO), NOx(GT), and PT08.S2(NMHC). The histograms have kernel density estimation (KDE) plots overlaid, which show the estimated probability density function of the data. Finally, the plots are displayed using plt.show() and plt.tight_layout() is used to adjust the spacing between the subplots.

SECTION 4:

Model development:

Splitting the dataset into training and testing sets for regression analysis:

In [6]:
```
# split the dataset into training and testing sets
X = df.drop(columns=['C6H6(GT)'])
y = df['C6H6(GT)']  # y is of type object, so sklearn cannot recognize its type
y = y.astype('int')
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42)
```

Training Multiple Classification Models on a Dataset:

The code creates instances of five different classification models (Gaussian Naive Bayes, K-Nearest Neighbors, Decision Tree, Logistic Regression, and Support Vector Machine) and trains them on a given dataset using the fit() method.

In [43]:
```

# fit the models
nb = GaussianNB()
nb.fit(X_train, y_train)

knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)

dt = DecisionTreeClassifier(random_state=42)
dt.fit(X_train, y_train)

lr = LogisticRegression(random_state=42)
lr.fit(X_train, y_train)

svm = SVC(kernel='rbf', random_state=42)
svm.fit(X_train, y_train)
```

In [8]:
```
# predict on the test set
nb_pred = nb.predict(X_test)
knn_pred = knn.predict(X_test)
dt_pred = dt.predict(X_test)
lr_pred = lr.predict(X_test)
svm_pred = svm.predict(X_test)
```

In [25]:
```
# classification reports and accuracy scores
print('Naive Bayes:')
print(classification_report(y_test, nb_pred))
print('Accuracy:', round(accuracy_score(y_test, nb_pred), 2))
print('Confusion Matrix:')
print(confusion_matrix(y_test, nb_pred))
```

```
Naive Bayes:
              precision    recall  f1-score   support

           0       0.83      0.91      0.87        33
           1       0.84      0.87      0.86        78
           2       0.89      0.91      0.90       153
           3       0.85      0.88      0.86       136
           4       0.96      0.80      0.87       155
           5       0.88      0.96      0.92       124
           6       0.96      0.94      0.95       142
           7       0.94      0.94      0.94       110
           8       0.94      0.97      0.96       106
           9       0.93      0.94      0.94        89
          10       0.85      0.91      0.88        75
          11       0.93      0.88      0.90        91
          12       0.82      0.88      0.85        60
          13       0.93      0.85      0.89        74
          14       0.97      0.95      0.96        59
          15       0.88      0.90      0.89        51
          16       0.87      0.91      0.89        43
          17       1.00      0.95      0.98        43
          18       0.97      0.85      0.91        34
          19       0.81      0.88      0.85        25
          20       0.93      0.96      0.95        27
```

In [8]:
```
# predict on the test set
nb_pred = nb.predict(X_test)
knn_pred = knn.predict(X_test)
dt_pred = dt.predict(X_test)
lr_pred = lr.predict(X_test)
svm_pred = svm.predict(X_test)
```

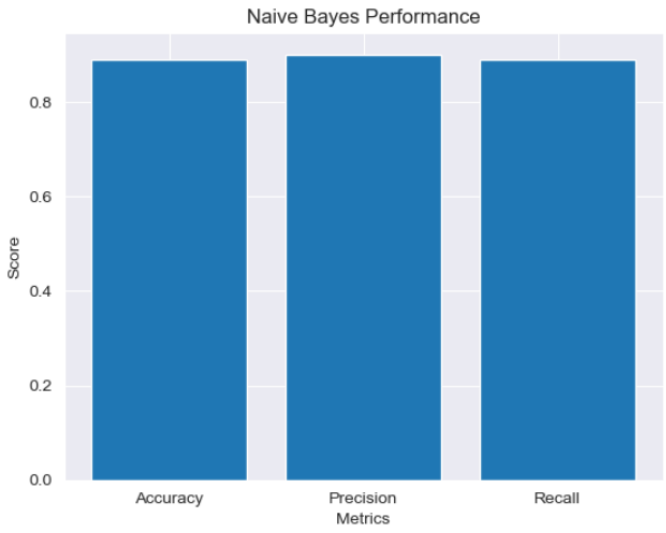
In [25]:
```
# classification reports and accuracy scores
print('Naive Bayes:')
print(classification_report(y_test, nb_pred))
print('Accuracy:', round(accuracy_score(y_test, nb_pred), 2))
print('Confusion Matrix:')
print(confusion_matrix(y_test, nb_pred))
```

```
Naive Bayes:
              precision    recall  f1-score   support

           0       0.83      0.91      0.87        33
           1       0.84      0.87      0.86        78
           2       0.89      0.91      0.90       153
           3       0.85      0.88      0.86       136
           4       0.96      0.80      0.87       155
           5       0.88      0.96      0.92       124
           6       0.96      0.94      0.95       142
           7       0.94      0.94      0.94       110
           8       0.94      0.97      0.96       106
           9       0.93      0.94      0.94        89
          10       0.85      0.91      0.88        75
          11       0.93      0.88      0.90        91
          12       0.82      0.88      0.85        60
          13       0.93      0.85      0.89        74
          14       0.97      0.95      0.96        59
          15       0.88      0.90      0.89        51
          16       0.87      0.91      0.89        43
          17       1.00      0.95      0.98        43
```

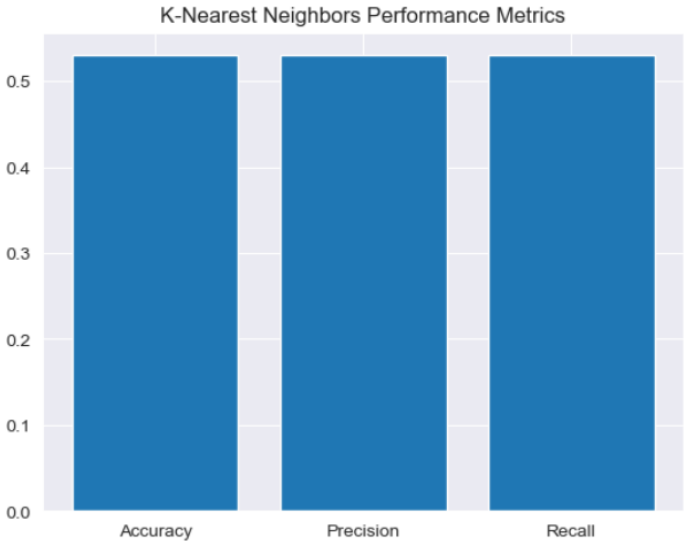
```
              22       0.94      1.00      0.97        17
              23       1.00      1.00      1.00        15
              24       0.90      1.00      0.95        18
              25       1.00      0.80      0.89        10
              26       0.88      0.94      0.91        16
              27       0.91      0.74      0.82        27
              28       0.45      0.83      0.59         6
              29       0.83      0.71      0.77         7
              30       0.50      0.25      0.33         4
              31       0.33      0.20      0.25         5
              32       0.54      1.00      0.70         7
              33       1.00      0.33      0.50         6
              34       0.50      1.00      0.67         3
              35       0.75      0.75      0.75         4
              36       0.50      0.20      0.29         5
              37       0.38      1.00      0.55         3
              38       0.00      0.00      0.00         3
              43       0.00      0.00      0.00         0
              44       1.00      1.00      1.00         1
              45       0.00      0.00      0.00         1
              47       0.00      0.00      0.00         1
              50       0.00      0.00      0.00         1
              63       0.00      0.00      0.00         1

    accuracy                           0.89      1895
   macro avg       0.72      0.73      0.71      1895
weighted avg       0.90      0.89      0.89      1895

Accuracy: 0.89
Confusion Matrix:
[[ 30   3   0 ...   0   0   0]
 [  6  68   4 ...   0   0   0]
 [  0  10 139 ...   0   0   0]
 ...
 [  0   0   0 ...   0   0   0]
 [  0   0   0 ...   0   0   0]
 [  0   0   0 ...   0   0   0]]
```

In [26]:

```
# create a bar chart of accuracy, precision, and recall for Naive Bayes
nb_report = classification_report(y_test, nb_pred, output_dict=True)
nb_scores = {'Accuracy': round(accuracy_score(y_test, nb_pred), 2),
             'Precision': round(nb_report['weighted avg']['precision'], 2),
             'Recall': round(nb_report['weighted avg']['recall'], 2)}
plt.bar(nb_scores.keys(), nb_scores.values())
plt.xlabel('Metrics')
plt.ylabel('Score')
plt.title('Naive Bayes Performance')
plt.show()
```

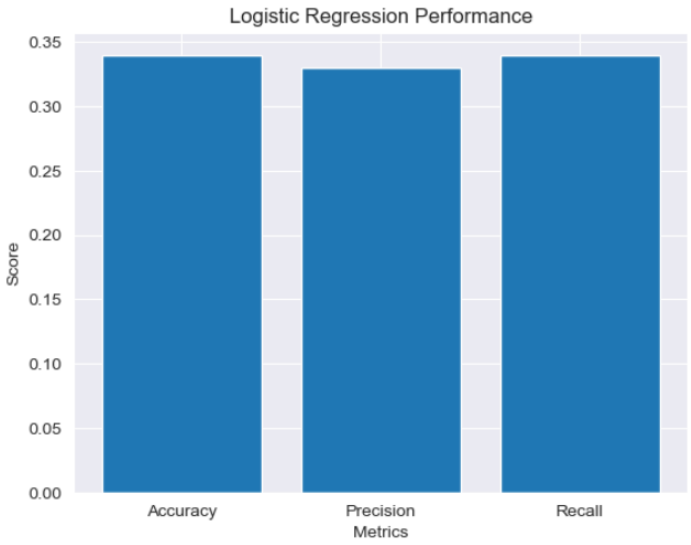


In [45]:
```
# create a bar chart of accuracy, precision, and recall for K-Nearest Neighbors
knn_report = classification_report(y_test, knn_pred, output_dict=True)
knn_scores = {'Accuracy': round(accuracy_score(y_test, knn_pred), 2),
              'Precision': round(knn_report['weighted avg']['precision'], 2),
              'Recall': round(knn_report['weighted avg']['recall'], 2)}
plt.bar(knn_scores.keys(), knn_scores.values())
plt.xlabel('Metrics')
plt.ylabel('Score')
plt.title('K-Nearest Neighbors Performance')
plt.show()
```

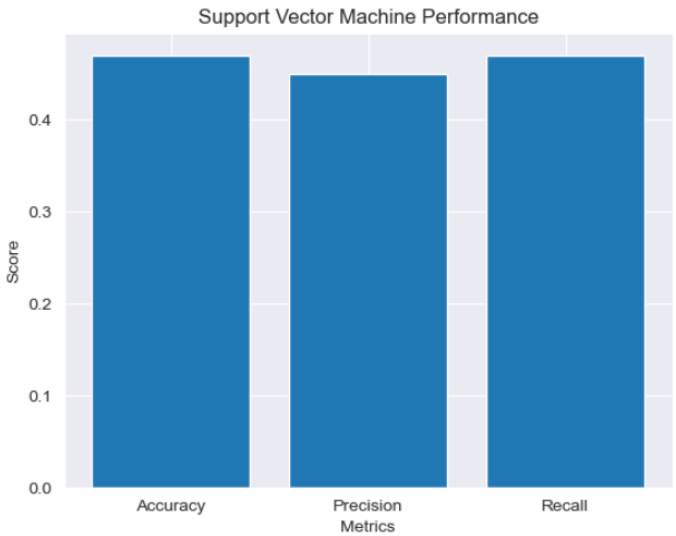


In [47]:
```
# create a bar chart of accuracy, precision, and recall for Decision Tree
dt_report = classification_report(y_test, dt_pred, output_dict=True)
dt_scores = {'Accuracy': round(accuracy_score(y_test, dt_pred), 2),
             'Precision': round(dt_report['weighted avg']['precision'], 2),
             'Recall': round(dt_report['weighted avg']['recall'], 2)}
plt.bar(dt_scores.keys(), dt_scores.values())
plt.xlabel('Metrics')
plt.ylabel('Score')
plt.title('Decision Tree Neighbors Performance')
plt.show()
```

In [48]:
```
# create a bar chart of accuracy, precision, and recall for Logistic Regression
lr_report = classification_report(y_test, lr_pred, output_dict=True)
lr_scores = {'Accuracy': round(accuracy_score(y_test, lr_pred), 2),
             'Precision': round(lr_report['weighted avg']['precision'], 2),
             'Recall': round(lr_report['weighted avg']['recall'], 2)}
plt.bar(lr_scores.keys(), lr_scores.values())
plt.xlabel('Metrics')
plt.ylabel('Score')
plt.title('Logistic Regression Performance')
plt.show()
```



In [49]:
```
# create a bar chart of accuracy, precision, and recall for Support Vector Machine
svm_report = classification_report(y_test, svm_pred, output_dict=True)
svm_scores = {'Accuracy': round(accuracy_score(y_test, svm_pred), 2),
              'Precision': round(svm_report['weighted avg']['precision'], 2),
              'Recall': round(svm_report['weighted avg']['recall'], 2)}
plt.bar(svm_scores.keys(), svm_scores.values())
plt.xlabel('Metrics')
plt.ylabel('Score')
plt.title('Support Vector Machine Performance')
plt.show()
```



In [36]:
```
print('K-Nearest Neighbors:')
print(classification_report(y_test, knn_pred))
print('Accuracy:', round(accuracy_score(y_test, knn_pred), 2))
print('Confusion Matrix:')
print(confusion_matrix(y_test, knn_pred))
```

```
K-Nearest Neighbors:
              precision    recall  f1-score   support

           0       0.83      0.88      0.85        33
           1       0.79      0.77      0.78        78
           2       0.76      0.85      0.80       153
           3       0.67      0.71      0.69       136
           4       0.68      0.59      0.63       155
           5       0.51      0.64      0.56       124
           6       0.58      0.54      0.56       142
           7       0.56      0.51      0.53       110
           8       0.58      0.59      0.59       106
           9       0.49      0.47      0.48        89
          10       0.45      0.56      0.50        75
          11       0.56      0.48      0.52        91
          12       0.35      0.38      0.37        60
          13       0.53      0.47      0.50        74
          14       0.33      0.36      0.34        59
          15       0.32      0.29      0.31        51
          16       0.28      0.35      0.31        43
          17       0.31      0.21      0.25        43
          18       0.35      0.24      0.28        34
          19       0.23      0.28      0.25        25
          20       0.26      0.22      0.24        27
          21       0.25      0.27      0.26        26
          22       0.16      0.29      0.21        17
          23       0.20      0.20      0.20        15
          24       0.14      0.11      0.12        18
          25       0.33      0.20      0.25        10
          26       0.40      0.25      0.31        16
          27       0.74      0.74      0.74        27
          28       0.00      0.00      0.00         6
          29       0.25      0.14      0.18         7
          30       0.14      0.25      0.18         4
          31       0.00      0.00      0.00         5
          32       0.25      0.14      0.18         7

          33       0.00      0.00      0.00         6
          34       0.38      1.00      0.55         3
          35       0.00      0.00      0.00         4
          36       0.00      0.00      0.00         5
          37       0.00      0.00      0.00         3
          38       0.00      0.00      0.00         3
          41       0.00      0.00      0.00         0
          44       0.00      0.00      0.00         1
          45       0.00      0.00      0.00         1
          47       0.00      0.00      0.00         1
          49       0.00      0.00      0.00         0
          50       0.00      0.00      0.00         1
          63       0.00      0.00      0.00         1

    accuracy                           0.53      1895
   macro avg       0.30      0.30      0.29      1895
weighted avg       0.53      0.53      0.52      1895

Accuracy: 0.53
Confusion Matrix:
[[ 29   4   0 ...   0   0   0]
 [  5  60  13 ...   0   0   0]
 [  1  12 130 ...   0   0   0]
 ...
 [  0   0   0 ...   0   0   0]
 [  0   0   0 ...   0   0   0]
 [  0   0   0 ...   0   0   0]]
```

In [16]:
```
print('Decision Tree:')
print(classification_report(y_test, dt_pred))
print('Accuracy:', round(accuracy_score(y_test, dt_pred), 2))
print('Confusion Matrix:')
print(confusion_matrix(y_test, dt_pred))
```

```
Decision Tree:
              precision    recall  f1-score   support

           0       0.97      1.00      0.99        33
           1       1.00      0.99      0.99        78
           2       0.99      1.00      1.00       153
           3       1.00      0.99      1.00       136
           4       1.00      1.00      1.00       155
           5       1.00      1.00      1.00       124
           6       1.00      1.00      1.00       142
           7       1.00      0.98      0.99       110
           8       0.98      1.00      0.99       106
           9       0.99      1.00      0.99        89
          10       0.99      0.99      0.99        75
          11       1.00      0.99      0.99        91
          12       0.97      1.00      0.98        60
          13       1.00      0.97      0.99        74
          14       1.00      1.00      1.00        59
          15       1.00      0.98      0.99        51
          16       0.98      1.00      0.99        43
          17       0.98      1.00      0.99        43
          18       1.00      0.91      0.95        34
          19       0.93      1.00      0.96        25
          20       1.00      1.00      1.00        27
          21       0.93      1.00      0.96        26
          22       1.00      0.88      0.94        17
          23       1.00      1.00      1.00        15
          24       1.00      1.00      1.00        18
          25       1.00      1.00      1.00        10
          26       1.00      1.00      1.00        16
          27       1.00      1.00      1.00        27
          28       0.75      1.00      0.86         6
          29       1.00      0.71      0.83         7
          30       1.00      0.50      0.67         4
          31       0.50      0.40      0.44         5

          33       1.00      0.67      0.80         6
          34       0.75      1.00      0.86         3
          35       1.00      1.00      1.00         4
          36       1.00      1.00      1.00         5
          37       0.75      1.00      0.86         3
          38       1.00      0.33      0.50         3
          39       0.00      0.00      0.00         0
          44       0.00      0.00      0.00         1
          45       0.00      0.00      0.00         1
          46       0.00      0.00      0.00         0
          47       0.00      0.00      0.00         1
          49       0.00      0.00      0.00         0
          50       0.00      0.00      0.00         1
          63       0.00      0.00      0.00         1

    accuracy                           0.98      1895
   macro avg       0.79      0.77      0.77      1895
weighted avg       0.99      0.98      0.98      1895

Accuracy: 0.98
Confusion Matrix:
[[ 33   0   0 ...   0   0   0]
 [  1  77   0 ...   0   0   0]
 [  0   0 153 ...   0   0   0]
 ...
 [  0   0   0 ...   0   0   0]
 [  0   0   0 ...   1   0   0]
 [  0   0   0 ...   1   0   0]]
```

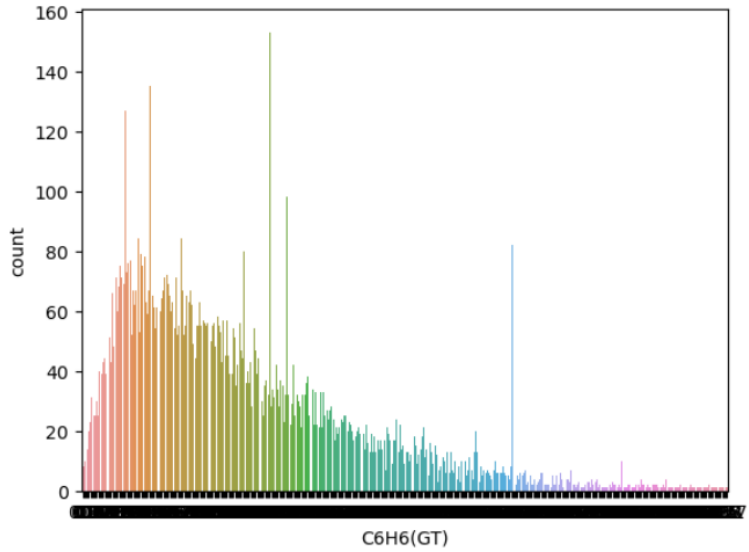
In [17]:
```
print('Logistic Regression:')
print(classification_report(y_test, lr_pred))
print('Accuracy:', round(accuracy_score(y_test, lr_pred), 2))
print('Confusion Matrix:')
print(confusion_matrix(y_test, lr_pred))
```

```
Logistic Regression:
              precision    recall  f1-score   support

           0       0.55      0.64      0.59        33
           1       0.54      0.41      0.47        78
           2       0.61      0.75      0.67       153
           3       0.51      0.57      0.54       136
           4       0.53      0.35      0.43       155
           5       0.28      0.44      0.34       124
           6       0.36      0.41      0.38       142
           7       0.29      0.15      0.19       110
           8       0.35      0.28      0.31       106
           9       0.23      0.18      0.20        89
          10       0.32      0.44      0.37        75
          11       0.27      0.37      0.31        91
          12       0.23      0.08      0.12        60
          13       0.24      0.31      0.27        74
          14       0.13      0.20      0.16        59
          15       0.12      0.10      0.11        51
          16       0.17      0.21      0.19        43
          17       0.25      0.09      0.14        43
          18       0.00      0.00      0.00        34
          19       0.16      0.28      0.21        25
          20       0.18      0.19      0.18        27
          21       0.10      0.15      0.12        26
          22       0.07      0.06      0.06        17
          23       0.00      0.00      0.00        15
          24       0.27      0.17      0.21        18
          25       0.11      0.10      0.11        10
          26       0.12      0.19      0.15        16
          27       0.34      0.74      0.47        27
          28       0.00      0.00      0.00         6
          29       0.00      0.00      0.00         7
          30       0.00      0.00      0.00         4
          31       0.00      0.00      0.00         5
          32       0.00      0.00      0.00         7
          33       0.00      0.00      0.00         6
          34       0.00      0.00      0.00         3
          35       0.00      0.00      0.00         4
          36       0.00      0.00      0.00         5
          37       0.00      0.00      0.00         3
          38       0.00      0.00      0.00         3
          44       0.00      0.00      0.00         1
          45       0.00      0.00      0.00         1
          47       0.00      0.00      0.00         1
          50       0.00      0.00      0.00         1
          63       0.00      0.00      0.00         1

    accuracy                           0.34      1895
   macro avg       0.17      0.18      0.17      1895
weighted avg       0.33      0.34      0.32      1895

Accuracy: 0.34
Confusion Matrix:
[[ 21  11   1 ...   0   0   0]
 [ 13  32  33 ...   0   0   0]
 [  3  11 114 ...   0   0   0]
 ...
 [  0   0   0 ...   0   0   0]
 [  0   0   0 ...   0   0   0]
 [  0   0   0 ...   0   0   0]]
```

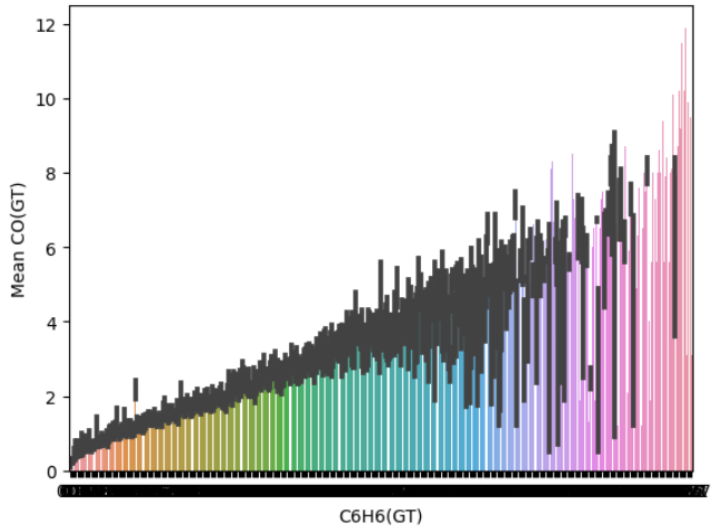
In [13]:
```
print('Support Vector Machine:')
print(classification_report(y_test, svm_pred))
print('Accuracy:', round(accuracy_score(y_test, svm_pred), 2))
print('Confusion Matrix:')
print(confusion_matrix(y_test, svm_pred))

import warnings
warnings.filterwarnings("ignore", message="Precision and F-score are ill-defined")
```

```
Support Vector Machine:
              precision    recall  f1-score   support

           0       0.77      0.61      0.68        33
           1       0.68      0.69      0.68        78
           2       0.74      0.82      0.78       153
           3       0.70      0.70      0.70       136
           4       0.71      0.58      0.64       155
           5       0.46      0.62      0.53       124
           6       0.55      0.59      0.57       142
           7       0.47      0.40      0.43       110
           8       0.47      0.45      0.46       106
           9       0.45      0.21      0.29        89
          10       0.39      0.69      0.50        75
          11       0.48      0.46      0.47        91
          12       0.27      0.27      0.27        60
          13       0.45      0.39      0.42        74
          14       0.31      0.46      0.37        59
          15       0.20      0.20      0.20        51
          16       0.21      0.28      0.24        43
          17       0.40      0.05      0.08        43
          18       0.00      0.00      0.00        34
          19       0.14      0.64      0.23        25
          20       0.00      0.00      0.00        27
          21       0.09      0.12      0.10        26
          22       0.00      0.00      0.00        17
          23       0.00      0.00      0.00        15
          24       0.00      0.00      0.00        18
          25       0.00      0.00      0.00        10
          26       0.00      0.00      0.00        16
          27       0.25      0.93      0.39        27
          28       0.00      0.00      0.00         6
          29       0.00      0.00      0.00         7
          30       0.00      0.00      0.00         4
          31       0.00      0.00      0.00         5
          32       0.00      0.00      0.00         7
          33       0.00      0.00      0.00         6
          34       0.00      0.00      0.00         3
          35       0.00      0.00      0.00         4
          36       0.00      0.00      0.00         5
          37       0.00      0.00      0.00         3
          38       0.00      0.00      0.00         3
          44       0.00      0.00      0.00         1
          45       0.00      0.00      0.00         1
          47       0.00      0.00      0.00         1
          50       0.00      0.00      0.00         1
          63       0.00      0.00      0.00         1

    accuracy                           0.47      1895
   macro avg       0.21      0.23      0.20      1895
weighted avg       0.45      0.47      0.45      1895

Accuracy: 0.47
Confusion Matrix:
[[ 20  13   0 ...   0   0   0]
 [  4  54  20 ...   0   0   0]
 [  2  11 125 ...   0   0   0]
 ...
 [  0   0   0 ...   0   0   0]
 [  0   0   0 ...   0   0   0]
 [  0   0   0 ...   0   0   0]]
```

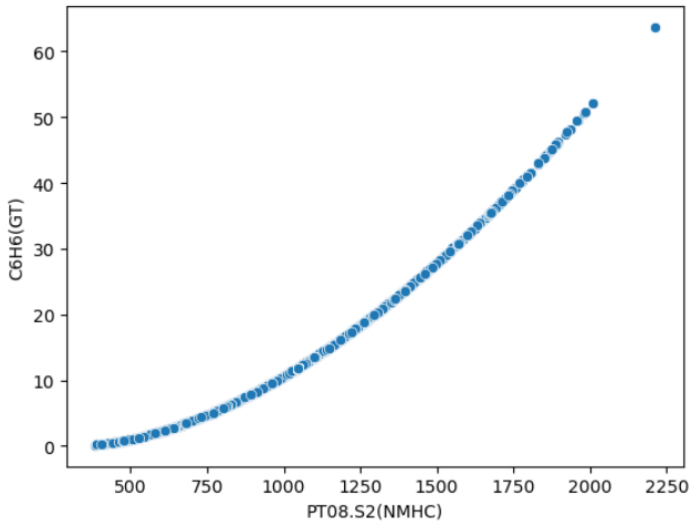
In [18]:
```
sns.countplot(data=df, x='C6H6(GT)')
plt.xlabel('C6H6(GT)')
plt.show()
```



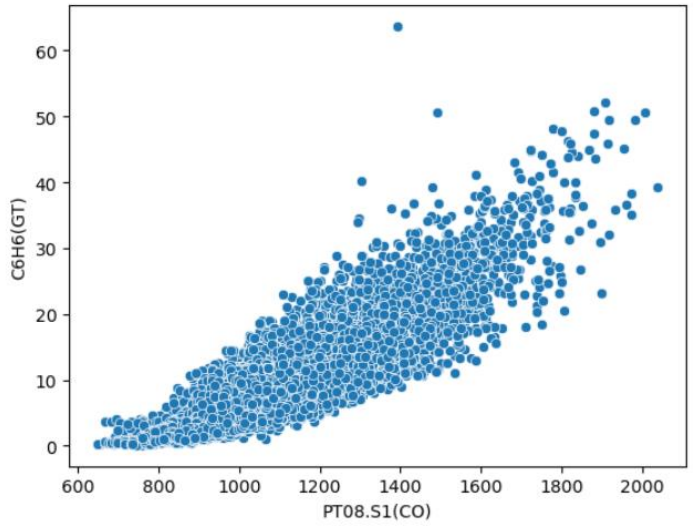
In [22]:
```
sns.barplot(data=df, x='C6H6(GT)', y='CO(GT)')
plt.xlabel('C6H6(GT)')
plt.ylabel('Mean CO(GT)')
plt.show()
```



In [19]:
```
sns.scatterplot(data=df, x='PT08.S2(NMHC)', y='C6H6(GT)')
plt.xlabel('PT08.S2(NMHC)')
plt.ylabel('C6H6(GT)')
plt.show()
```



In [20]:
```
sns.scatterplot(data=df, x='PT08.S1(CO)', y='C6H6(GT)')
plt.xlabel('PT08.S1(CO)')
plt.ylabel('C6H6(GT)')
plt.show()
```

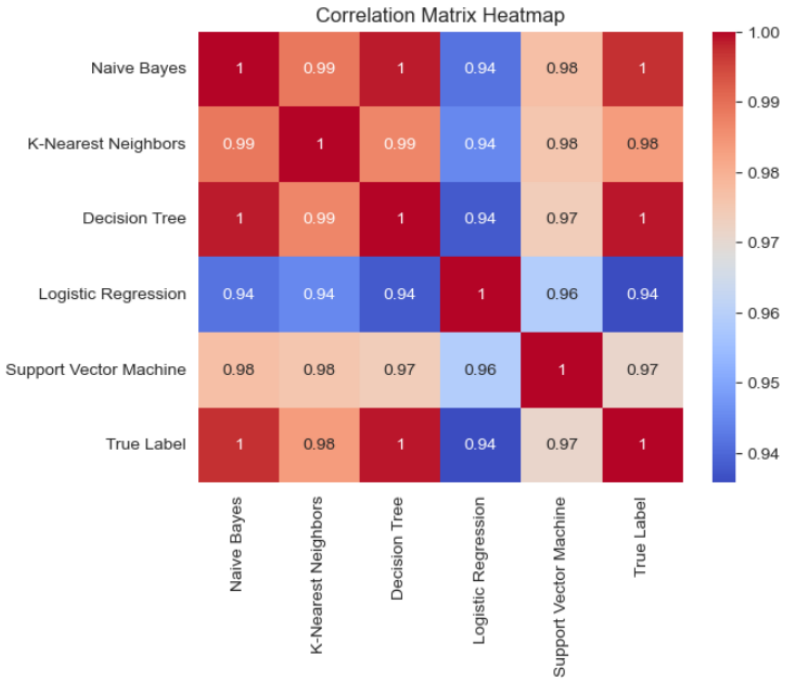


SECTION 5:

Comparison of Predicted Labels from Multiple Machine Learning Models using Correlation Matrix Heatmap :

This code takes the predicted labels from Naive Bayes, K-Nearest Neighbors, Decision Tree, Logistic Regression, and Support Vector Machine models and creates a correlation matrix heatmap to visualize the correlation between these predicted labels and the true labels. The heatmap provides an overview of how the models are performing and how they are correlated with each other.

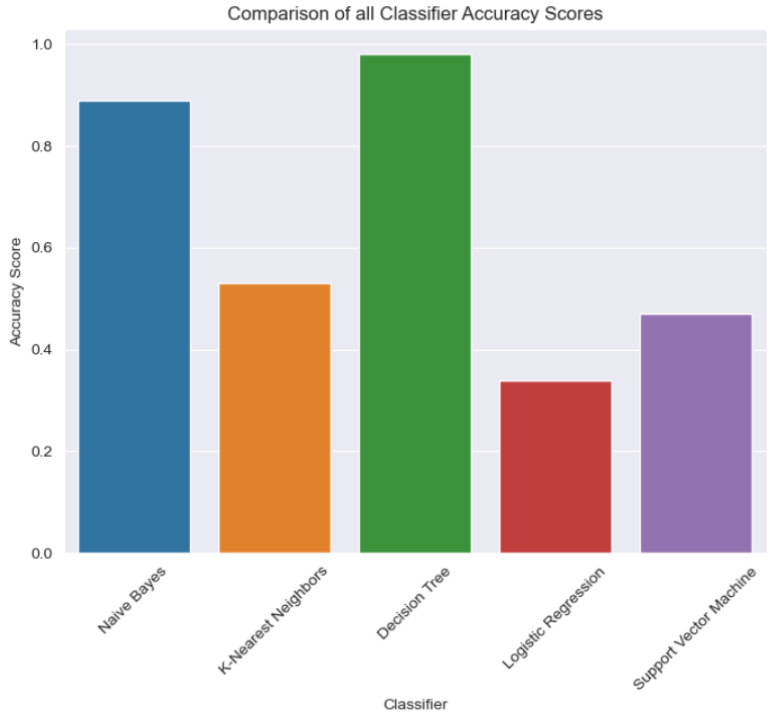
In [41]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Assume X_test is the test data and y_test is the true labels
nb_pred = nb.predict(X_test)
knn_pred = knn.predict(X_test)
dt_pred = dt.predict(X_test)
lr_pred = lr.predict(X_test)
svm_pred = svm.predict(X_test)

# Create a DataFrame with the predicted labels
pred_df = pd.DataFrame({'Naive Bayes': nb_pred,
                        'K-Nearest Neighbors': knn_pred,
                        'Decision Tree': dt_pred,
                        'Logistic Regression': lr_pred,
                        'Support Vector Machine': svm_pred,
                        'True Label': y_test})

# Compute the correlation matrix
correlation_df = pred_df.corr()

# Plot the heatmap
sns.heatmap(correlation_df, cmap='coolwarm', annot=True)
plt.title('Correlation Matrix Heatmap')
plt.show()
```



Comparison of all Classifier Accuracy Scores

This following code generates a bar chart showing the accuracy scores of various classification models including Naive Bayes, K-Nearest Neighbors, Decision Tree, Logistic Regression, and Support Vector Machine. The chart helps to visualize and compare the performance of these models based on their accuracy scores.

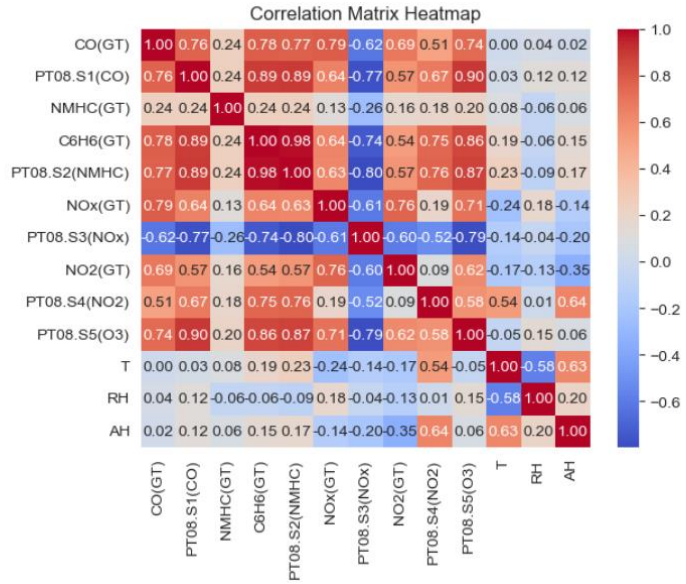
In [21]:
```
# classification reports and accuracy scores
accuracy_scores = {
    'Naive Bayes': round(accuracy_score(y_test, nb_pred), 2),
    'K-Nearest Neighbors': round(accuracy_score(y_test, knn_pred), 2),
    'Decision Tree': round(accuracy_score(y_test, dt_pred), 2),
    'Logistic Regression': round(accuracy_score(y_test, lr_pred), 2),
    'Support Vector Machine': round(accuracy_score(y_test, svm_pred), 2)
}



sns.set_style('darkgrid')
plt.figure(figsize=(8, 6))
sns.barplot(x=list(accuracy_scores.keys()), y=list(accuracy_scores.values()))
plt.title('Comparison of all Classifier Accuracy Scores')
plt.xlabel('Classifier')
plt.ylabel('Accuracy Score')
plt.xticks(rotation=45)
plt.show()
```



Correlation Matrix Heatmap

In [22]:

```
# plot heatmap
sns.heatmap(corr, cmap='coolwarm', annot=True, fmt='.2f')
plt.title('Correlation Matrix Heatmap')
plt.show()
```



In conclusion, this project had aimed to explore the performance of different classification models on a real dataset using Python. The project involved data preprocessing, exploratory data analysis, and the implementation of five different classification models including Naive Bayes, KNN, Decision Tree, Logistic Regression, and SVM.

The project had focused on evaluating the predictive accuracy of the models on the testing set, and further analysis had been done using classification reports and confusion matrices. Through this project, we had aimed to demonstrate the suitability of different classification models for different types of datasets and provide a comparison of their performance.

By carrying out this project, we had hoped to provide valuable insights into the effectiveness of different classification models and their potential applications.
