---

1. Jupyter Notebook (.ipynb)

Link:

https://colab.research.google.com/drive/1-bYCdu_LlyH_Il7c4uUwOY21zUB52QZQ?usp=sharing

```
import pandas as pd

import numpy as np

import re

from math import radians, sin, cos, sqrt, atan2

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model_selection import train_test_split, GridSearchCV

from sklearn.preprocessing import StandardScaler, LabelEncoder

from sklearn.naive_bayes import GaussianNB

from sklearn.neighbors import KNeighborsClassifier

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score,
confusion_matrix, classification_report, roc_curve, auc

from sklearn.preprocessing import label_binarize

from itertools import cycle
```

```python
#
===================================================================================
==

# Phase 1: Data Preprocessing

#
===================================================================================
==



# Data Import and Cleaning

file_path = 'Global_Pollution_Analysis (1).csv'

df = pd.read_csv(file_path)



# Handle missing values (no missing values were found)

# Feature Engineering

q_low = df['Air_Pollution_Index'].quantile(0.33)

q_high = df['Air_Pollution_Index'].quantile(0.66)

def categorize_pollution(index):

    if index <= q_low:

        return 'Low'

    elif index <= q_high:

        return 'Medium'

    else:

        return 'High'
```

```
df['Pollution_Severity'] = df['Air_Pollution_Index'].apply(categorize_pollution)


# Encode categorical features

label_encoder_country = LabelEncoder()

df['Country_Encoded'] = label_encoder_country.fit_transform(df['Country'])

label_encoder_year = LabelEncoder()

df['Year_Encoded'] = label_encoder_year.fit_transform(df['Year'])

label_encoder_severity = LabelEncoder()

df['Pollution_Severity_Encoded'] = label_encoder_severity.fit_transform(df['Pollution_Severity'])


# Drop original columns for modeling

columns_to_drop = ['Country', 'Year', 'Air_Pollution_Index', 'Pollution_Severity']

df_preprocessed = df.drop(columns=columns_to_drop)


# Save the preprocessed data to a CSV file for future use

df_preprocessed.to_csv("preprocessed_pollution_data.csv", index=False)


#
==========================================================================
==

# Phase 2: Classification using Naive Bayes, K-Nearest Neighbors, and Decision Tree
```

```
#
=================================================================================
==



X = df_preprocessed.drop('Pollution_Severity_Encoded', axis=1)

y = df_preprocessed['Pollution_Severity_Encoded']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)



numerical_features = [

    'Water_Pollution_Index', 'Soil_Pollution_Index', 'Industrial_Waste (in tons)',

    'Energy_Recovered (in GWh)', 'CO2_Emissions (in MT)', 'Renewable_Energy (%)',

    'Plastic_Waste_Produced (in tons)', 'Energy_Consumption_Per_Capita (in MWh)',

    'Population (in millions)', 'GDP_Per_Capita (in USD)'

]

scaler = StandardScaler()

X_train_scaled = X_train.copy()

X_test_scaled = X_test.copy()

X_train_scaled[numerical_features] = scaler.fit_transform(X_train[numerical_features])

X_test_scaled[numerical_features] = scaler.transform(X_test[numerical_features])



# Naive Bayes Classifier

nb_model = GaussianNB()
```

```python
nb_model.fit(X_train_scaled, y_train)

y_pred_nb = nb_model.predict(X_test_scaled)

conf_matrix_nb = confusion_matrix(y_test, y_pred_nb)

plt.figure(figsize=(8, 6))

sns.heatmap(conf_matrix_nb, annot=True, fmt='d', cmap='Blues', xticklabels=['Low', 'Medium', 'High'], yticklabels=['Low', 'Medium', 'High'])

plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.title('Confusion Matrix for Naive Bayes')

plt.savefig('naive_bayes_confusion_matrix.png')

plt.close()



# K-Nearest Neighbors (KNN) Classifier

knn_model = KNeighborsClassifier(n_neighbors=18)

knn_model.fit(X_train_scaled, y_train)

y_pred_knn = knn_model.predict(X_test_scaled)

conf_matrix_knn = confusion_matrix(y_test, y_pred_knn)

plt.figure(figsize=(8, 6))

sns.heatmap(conf_matrix_knn, annot=True, fmt='d', cmap='Blues', xticklabels=['Low', 'Medium', 'High'], yticklabels=['Low', 'Medium', 'High'])

plt.xlabel('Predicted')

plt.ylabel('Actual')
```

```python
plt.title('Confusion Matrix for K-Nearest Neighbors (K=18)')

plt.savefig('knn_confusion_matrix.png')

plt.close()



# Decision Tree Classifier

dt_model = DecisionTreeClassifier(random_state=42, max_depth=4, min_samples_split=2)

dt_model.fit(X_train_scaled, y_train)

y_pred_dt = dt_model.predict(X_test_scaled)

conf_matrix_dt = confusion_matrix(y_test, y_pred_dt)

plt.figure(figsize=(8, 6))

sns.heatmap(conf_matrix_dt, annot=True, fmt='d', cmap='Blues', xticklabels=['Low', 'Medium', 'High'],
yticklabels=['Low', 'Medium', 'High'])

plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.title('Confusion Matrix for Decision Tree')

plt.savefig('decision_tree_confusion_matrix.png')

plt.close()



# ROC Curves for all models

y_test_bin = label_binarize(y_test, classes=[0, 1, 2])

y_prob_nb = nb_model.predict_proba(X_test_scaled)
```

```
y_prob_knn = knn_model.predict_proba(X_test_scaled)

y_prob_dt = dt_model.predict_proba(X_test_scaled)



fpr = dict()

tpr = dict()

roc_auc = dict()

n_classes = y_test_bin.shape[1]

for i in range(n_classes):

    fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], y_prob_nb[:, i])

    roc_auc[i] = auc(fpr[i], tpr[i])

plt.figure(figsize=(10, 8))

colors = cycle(['blue', 'green', 'red'])

for i, color in zip(range(n_classes), colors):

    plt.plot(fpr[i], tpr[i], color=color, lw=2, label=f'ROC curve of class {i} (area = {roc_auc[i]:.2f})')

plt.plot([0, 1], [0, 1], 'k--', lw=2)

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('ROC Curves for Naive Bayes (One-vs-Rest)')
```

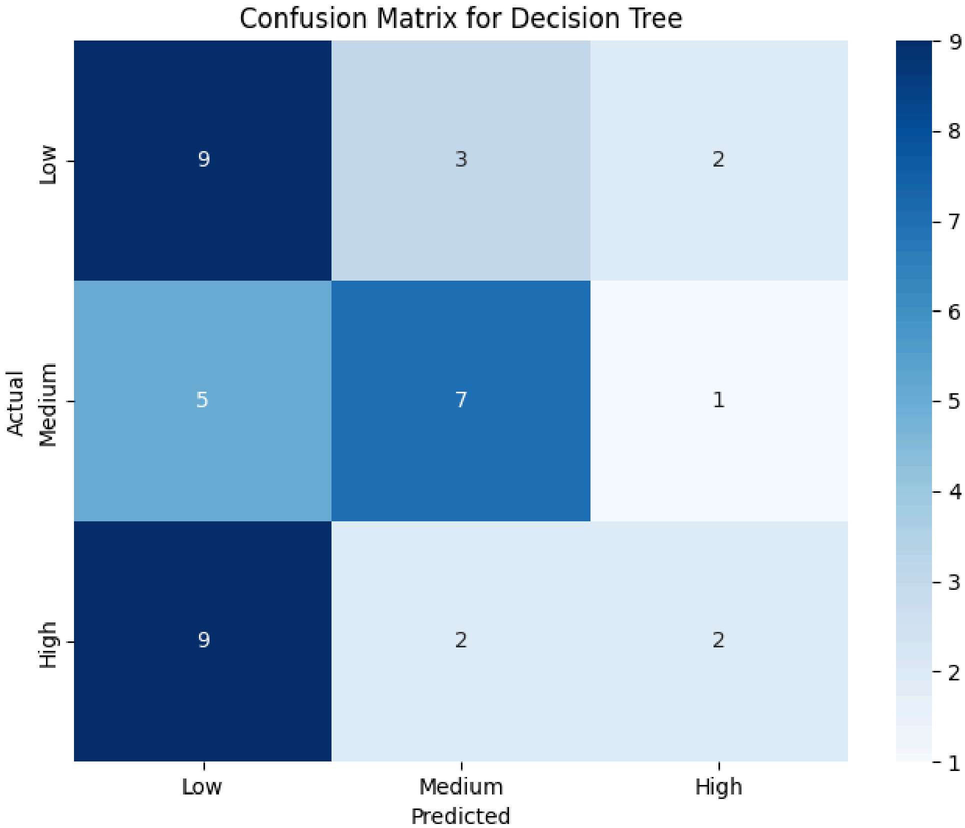
```
plt.legend(loc="lower right")

plt.savefig('naive_bayes_roc_curves.png')

plt.close()



fpr = dict()

tpr = dict()

roc_auc = dict()

for i in range(n_classes):

    fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], y_prob_knn[:, i])

    roc_auc[i] = auc(fpr[i], tpr[i])

plt.figure(figsize=(10, 8))

colors = cycle(['blue', 'green', 'red'])

for i, color in zip(range(n_classes), colors):

    plt.plot(fpr[i], tpr[i], color=color, lw=2, label=f'ROC curve of class {i} (area = {roc_auc[i]:.2f})')

plt.plot([0, 1], [0, 1], 'k--', lw=2)

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('ROC Curves for KNN (One-vs-Rest)')
```

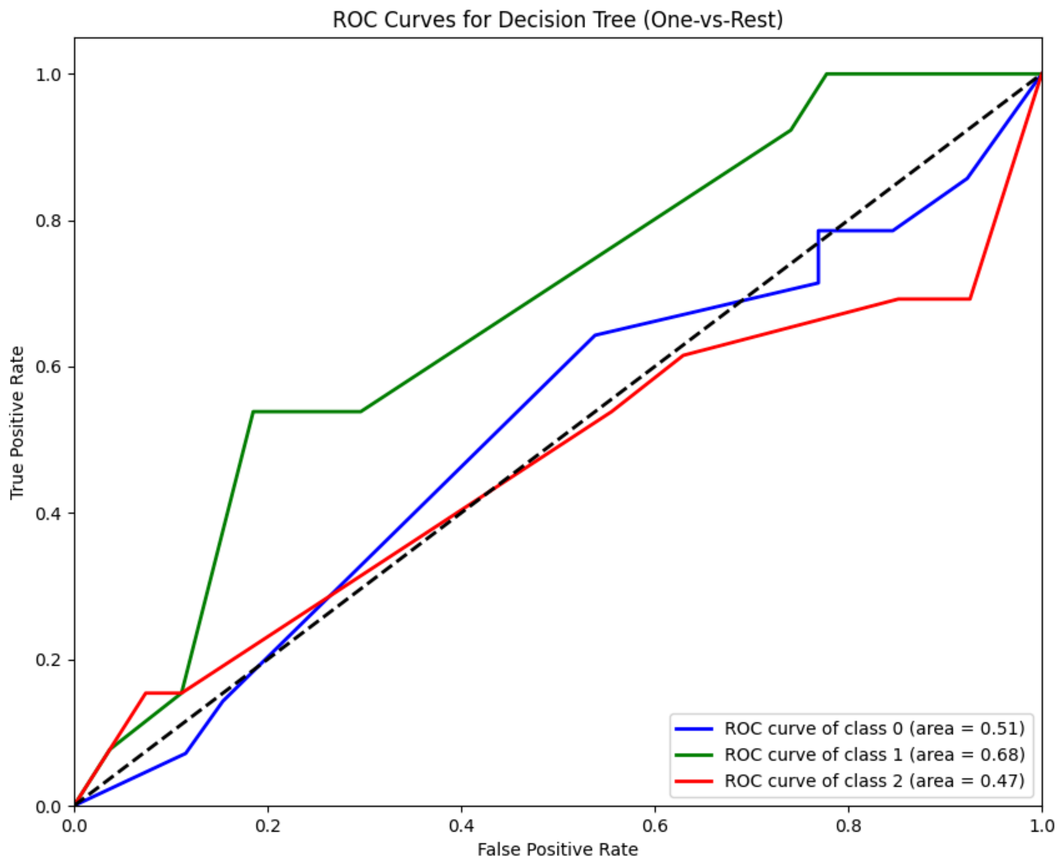
```
plt.legend(loc="lower right")

plt.savefig('knn_roc_curves.png')

plt.close()



fpr = dict()

tpr = dict()

roc_auc = dict()

for i in range(n_classes):

    fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], y_prob_dt[:, i])

    roc_auc[i] = auc(fpr[i], tpr[i])

plt.figure(figsize=(10, 8))

colors = cycle(['blue', 'green', 'red'])

for i, color in zip(range(n_classes), colors):

    plt.plot(fpr[i], tpr[i], color=color, lw=2, label=f'ROC curve of class {i} (area = {roc_auc[i]:.2f})')

plt.plot([0, 1], [0, 1], 'k--', lw=2)

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('ROC Curves for Decision Tree (One-vs-Rest)')
```

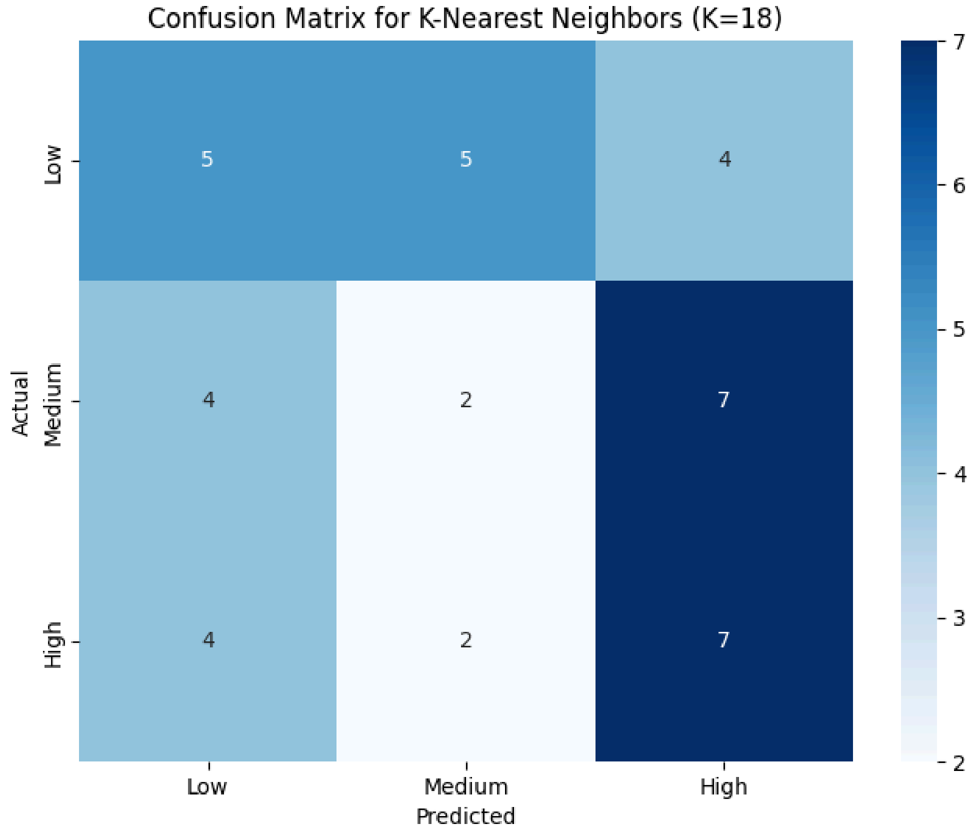
```
plt.legend(loc="lower right")

plt.savefig('decision_tree_roc_curves.png')

plt.close()
```

---

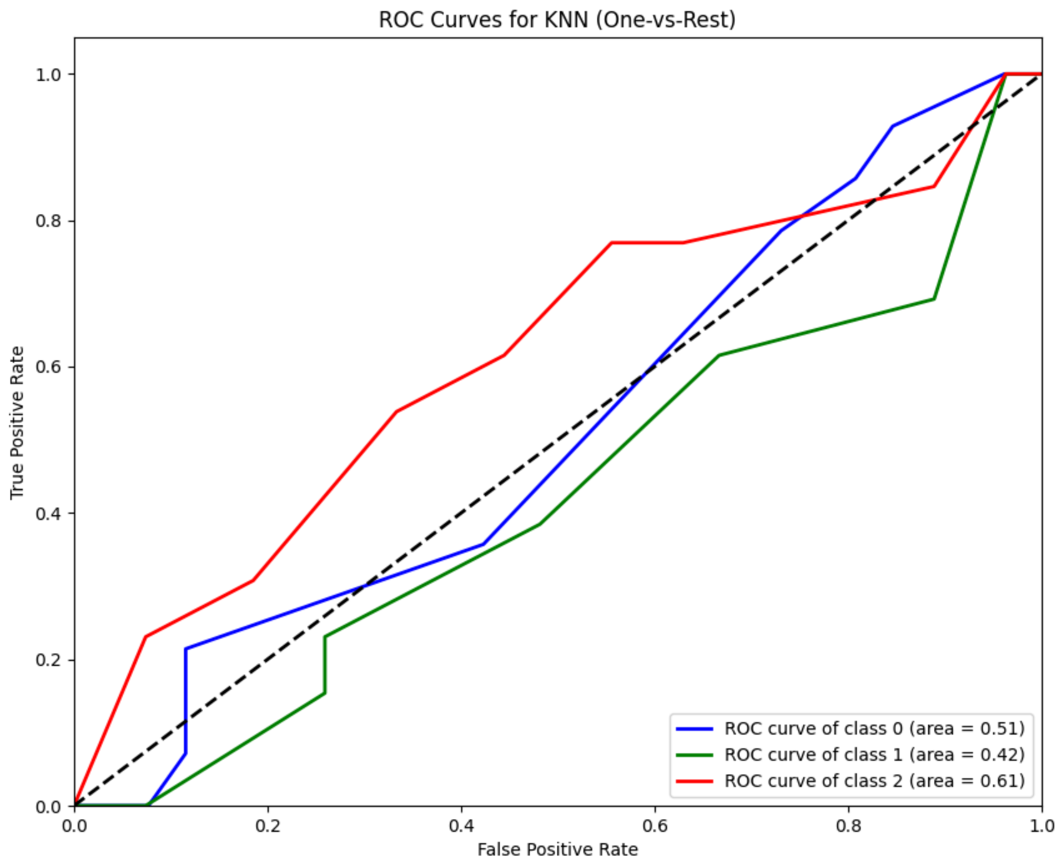
2. Data Visualizations

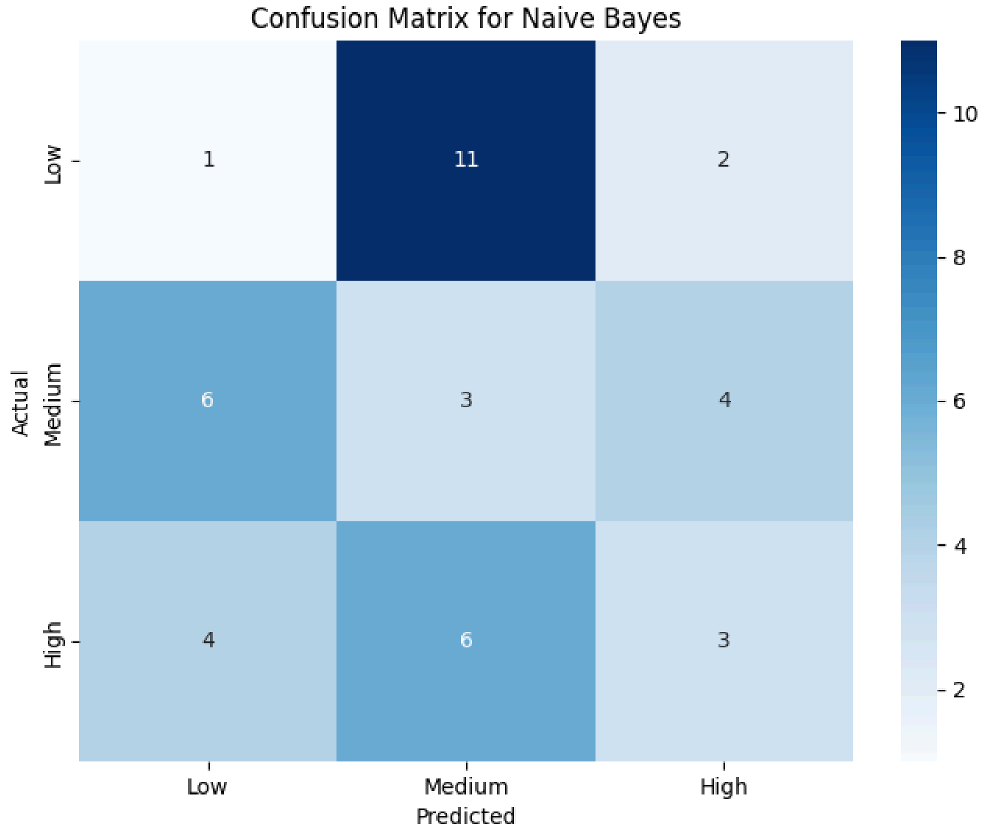

Confusion Matrix for Decision Tree
Actual
Low
Medium
High
Predicted
Low
Medium
High
9
3
2
5
7
1
9
2
2
9
8
7
6
5
4
3
2
1


ROC Curves for Decision Tree (One-vs-Rest)
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve of class 0 (area = 0.51)
ROC curve of class 1 (area = 0.68)
ROC curve of class 2 (area = 0.47)

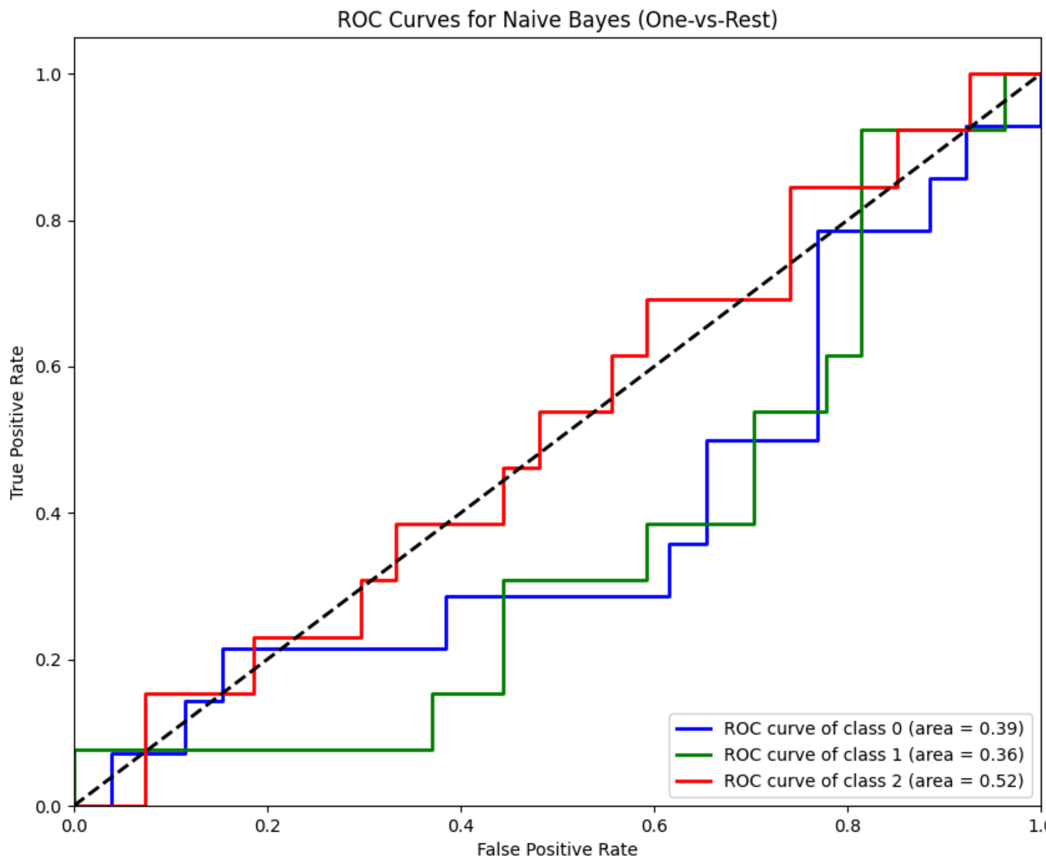


Confusion Matrix for K-Nearest Neighbors (K=18)
Actual
Low
Medium
High
Predicted
Low
Medium
High
5
5
4
4
2
7
4
2
7
7
6
5
4
3
2


ROC Curves for KNN (One-vs-Rest)
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve of class 0 (area = 0.51)
ROC curve of class 1 (area = 0.42)
ROC curve of class 2 (area = 0.61)



Confusion Matrix for Naive Bayes
Actual
Low
Medium
High
Predicted
Low
Medium
High
1
11
2
6
3
4
4
6
3
2
4
6
8
10



---

3. Final Report

Model Comparison

I evaluated three classifiers for predicting pollution severity: **Naive Bayes**, **K-Nearest Neighbors (KNN)**, and **Decision Tree**. The performance metrics for each model are as follows:

Model	Accuracy	Precision	Recall	F1-Score
Naive Bayes	0.17	0.19	0.17	0.18

KNN (K=18)	0.35	0.33	0.35	0.34
Decision Tree	0.45	0.46	0.45	0.42

- **Weaknesses**: All three models performed poorly. The Naive Bayes model was the least accurate, performing worse than a random classifier. The KNN and Decision Tree models showed slightly better performance but were still not effective at reliably classifying pollution severity. The low scores across all metrics indicate that the features, as they are, are not sufficient to accurately predict the pollution categories.

Actionable Insights and Recommendations

Based on the poor performance of these models, the following insights and recommendations are crucial for improving the pollution severity prediction system:

1. **Model Selection**: The current models are not suitable for this task. I would recommend exploring more advanced machine learning models that can handle complex, non-linear relationships, such as **Random Forests**, **Gradient Boosting Machines**, or **Neural Networks**.
2. **Feature Engineering**: The current features may not be sufficient. Consider incorporating additional features, such as:
   - **Real-time pollution data**: The current pollution indices are likely averages. Real-time data could provide more predictive power.
   - **Geospatial data**: Features like population density in certain areas could be a valuable feature.
   - **Economic and social factors**: Features like GDP per capita and population could be more predictive.
3. **Data Collection**: The small dataset size (200 entries) is a significant limitation. Collecting a larger and more diverse dataset with a wider range of features would likely lead to better model performance.

1.