BRAC
UNIVERSITY

Inspiring Excellence

CSE422: Artificial intelligence

Project Name: Air Quality Classification

Group: 10

No.	Name	ID
1	Md. Ridwanur Rahman	22301532
2	Anita Haque Sushma	22301050

Submitted to

[HJB] Hasnat Jamil Bhuiyan
[MZW] Maazin Munawar

Submission Date:
6th January, 2025

Table of Contents

Introduction:

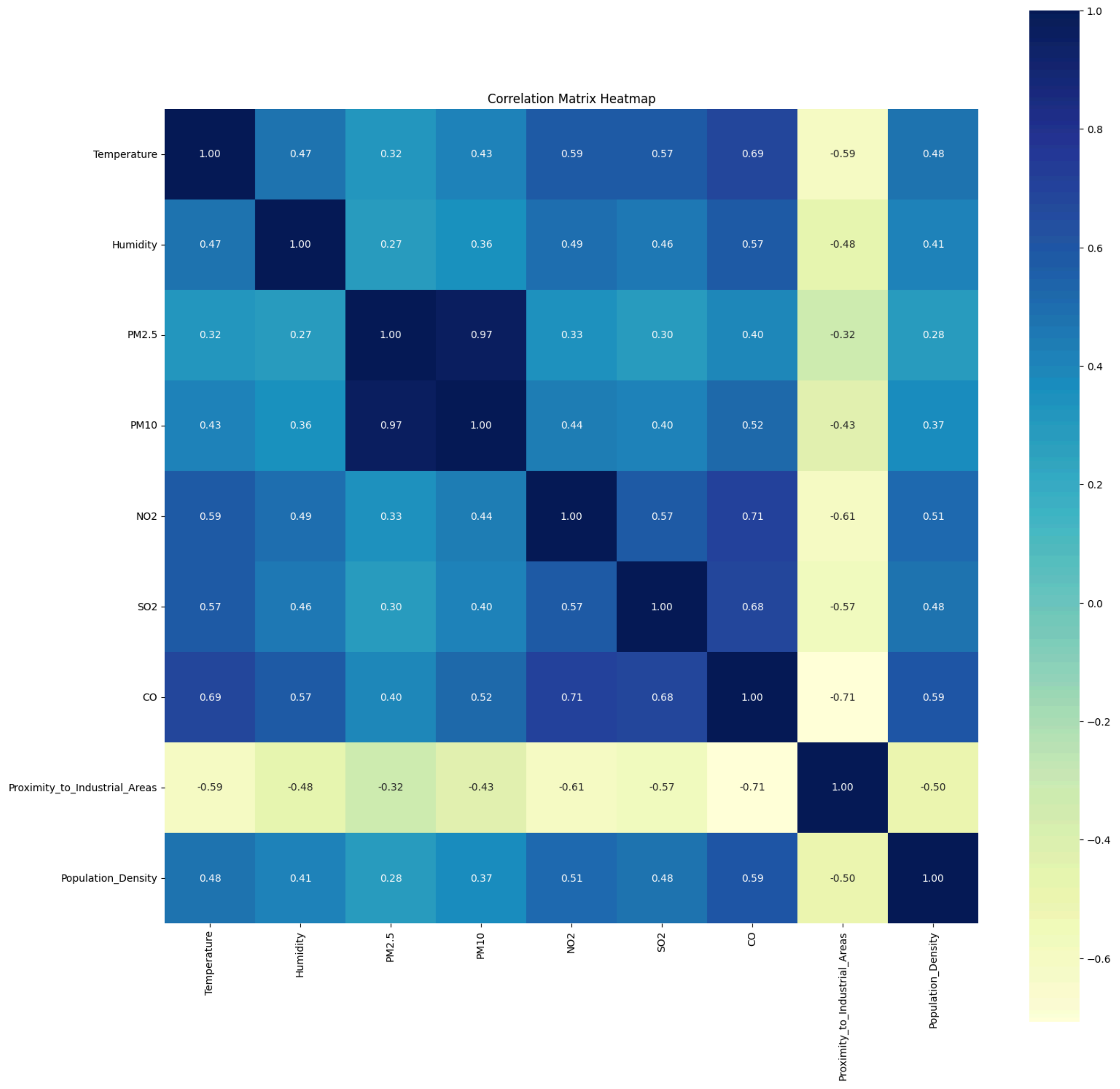
We have executed an Air Quality Prediction utilizing Machine Learning Calculations. The objective of this project is to analyze and predict air quality based on various environmental factors. We will examine the quality extend within the dataset. It's a classification problem. We have prepared an air quality classification utilizing three ML calculations. This shows the quality based on key factors like temperature, humidity, NO2, SO2, etc. We collected data in various indicators associated with air quality, pre-processed and engineered the feature, and trained and evaluated the machine learning model for accuracy. By using these techniques, we hope to increase awareness and warn the public about the urgent issue of air pollution, promoting preventive steps.

Dataset Description:

We used the dataset from Kaggle named Air Quality and Pollution Assessment. This dataset focuses on air quality assessment across various regions. The dataset contains 5000 samples and captures critical environmental and demographic factors that influence pollution levels.
This dataset has a total 10 Features where 9 of them are quantitative and 1 of them is Categorical. We are trying to solve the classification problem, as the target variable, Air Quality contains discrete categories. The numerical features are measured on different scales and provide crucial input data for identifying patterns and trends in pollution.
The target variable, Air Quality, consists of four distinct classes, representing categories like Good, Moderate, Poor, and Hazardous. These classes are discrete and continuous, making this a classification problem. The goal is to predict the air quality category based on various input features, Using both environmental metrics and demographic factors.

In this dataset, there are a total of 5000 rows, indicating 5000 data points. By preparing the data, we aim to create a reliable model that can accurately identify air quality levels.

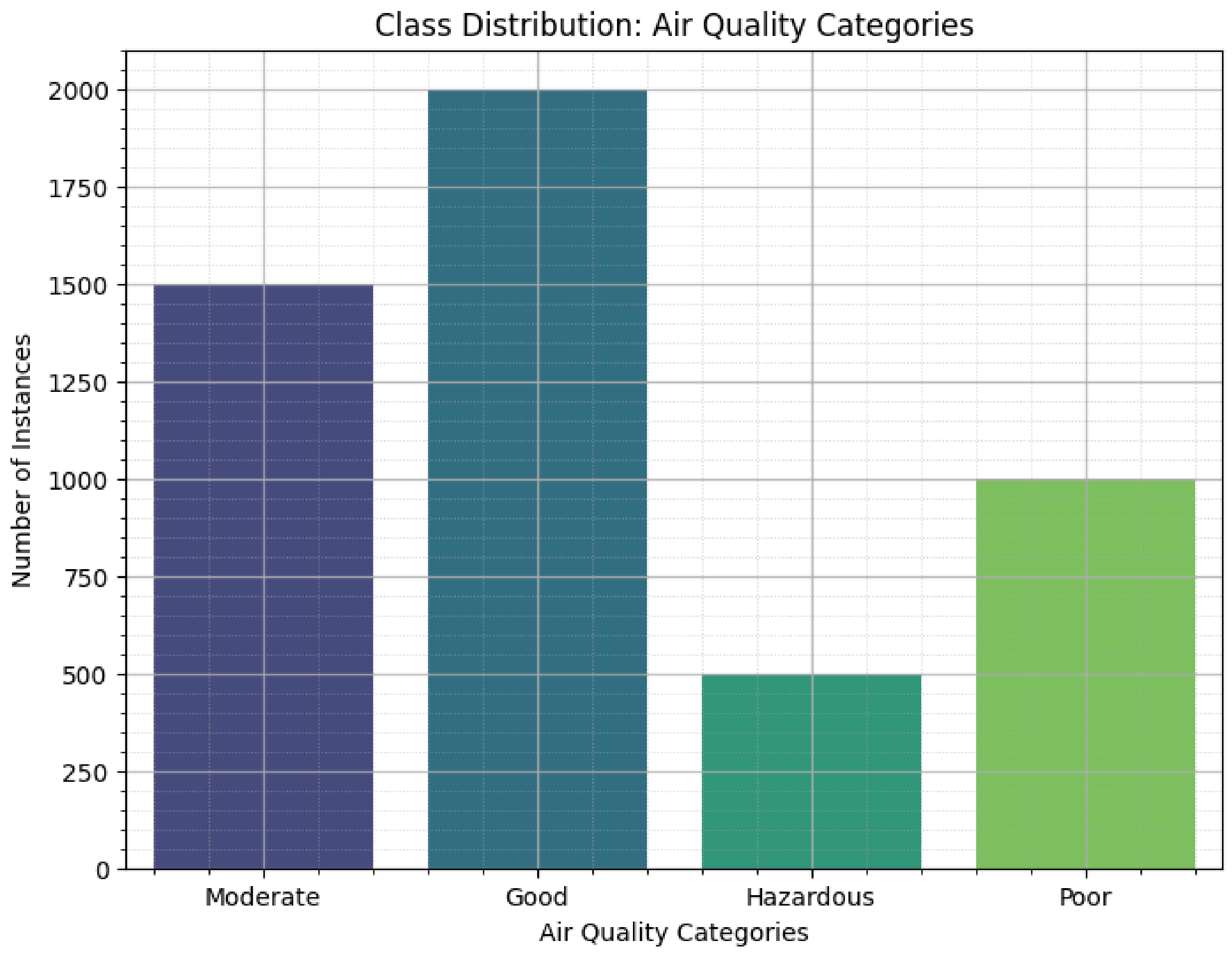
After this we implement the correlation analysis by applying heatmap using seaborn library.



	Temperature	Humidity	PM2.5	PM10	NO2	SO2	CO	Proximity_to_Industrial_Areas	Population_Density
Temperature	1.00	0.47	0.32	0.43	0.59	0.57	0.69	-0.59	0.48
Humidity	0.47	1.00	0.27	0.36	0.49	0.46	0.57	-0.48	0.41
PM2.5	0.32	0.27	1.00	0.97	0.33	0.30	0.40	-0.32	0.28
PM10	0.43	0.36	0.97	1.00	0.44	0.40	0.52	-0.43	0.37
NO2	0.59	0.49	0.33	0.44	1.00	0.57	0.71	-0.61	0.51
SO2	0.57	0.46	0.30	0.40	0.57	1.00	0.68	-0.57	0.48
CO	0.69	0.57	0.40	0.52	0.71	0.68	1.00	-0.71	0.59
Proximity_to_Industrial_Areas	-0.59	-0.48	-0.32	-0.43	-0.61	-0.57	-0.71	1.00	-0.50
Population_Density	0.48	0.41	0.28	0.37	0.51	0.48	0.59	-0.50	1.00

From the above heatmap visualization we can see a high correlation between PM2.5 and PM10. Now when you will learn this data to a model both will give the same output for PM2.5 and PM10. Including both features in the dataset can cause redundancy, as they provide similar information. To avoid this, we removed the PM2.5 column, simplifying the dataset and preventing potential overfitting. This step makes the model simpler and easier to understand while still maintaining its accuracy.

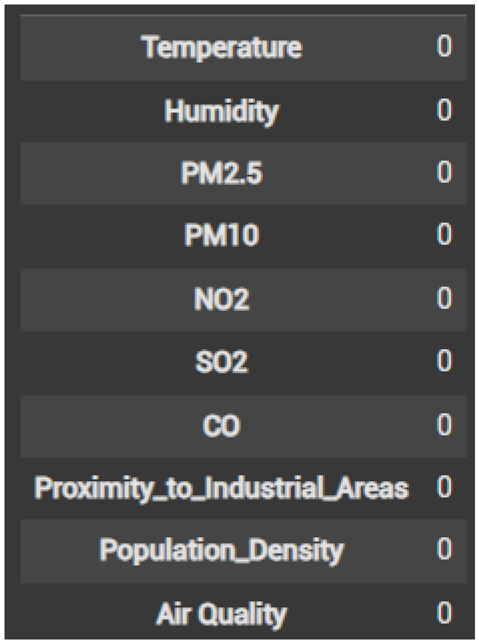
Now the For the output feature the class distribution is imbalanced.
A bar chart is shown below



The bar chart highlights the class distribution of the target variable Air Quality. It is clear that the dataset is imbalanced with the Good category having the highest number of instances followed by the Moderate and Poor. Hazardous has the fewest instances.

Dataset Preprocessing

Here no null was found.

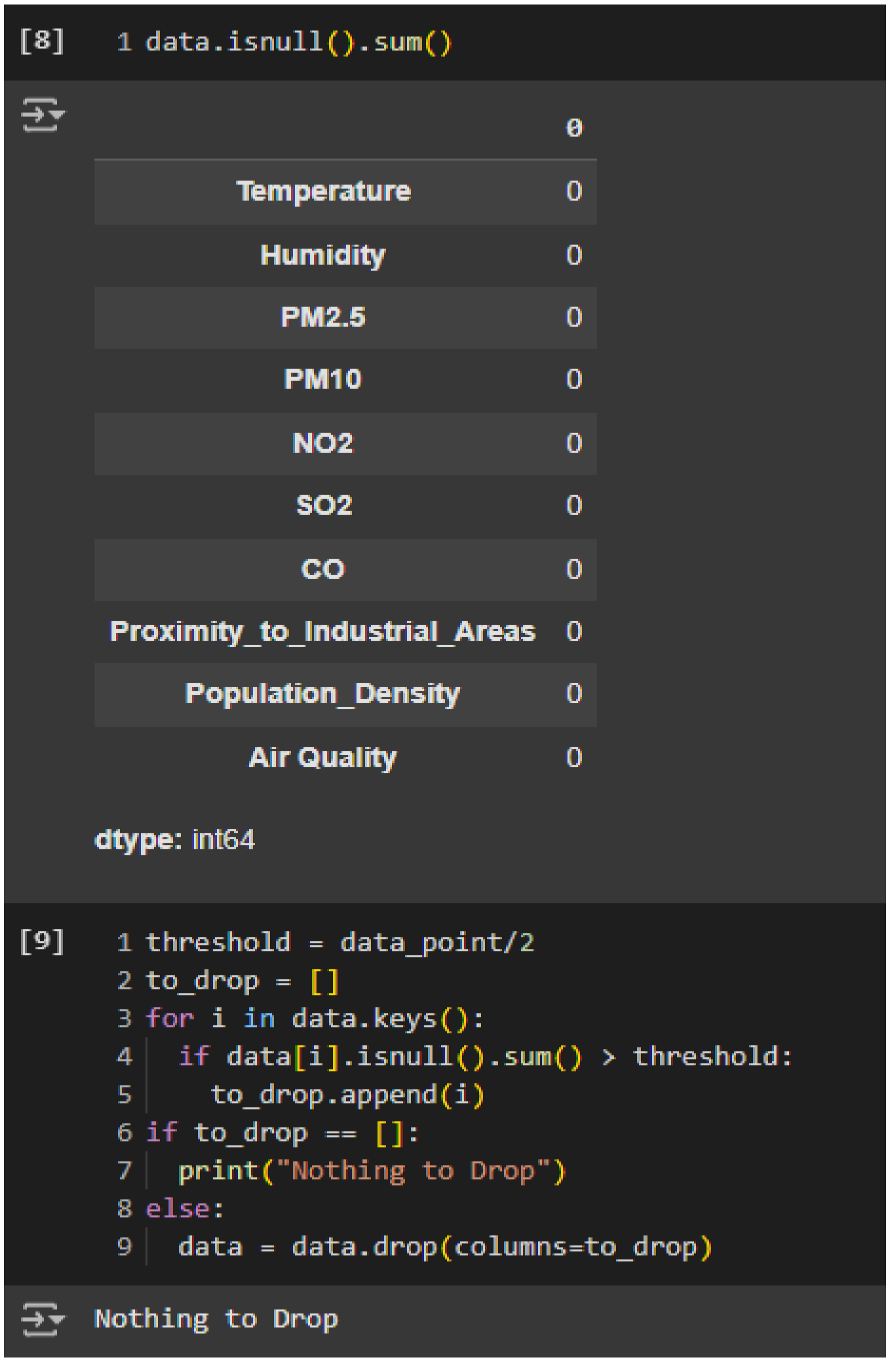


Temperature	0
Humidity	0
PM2.5	0
PM10	0
NO2	0
SO2	0
CO	0
Proximity_to_Industrial_Areas	0
Population_Density	0
Air Quality	0

If null value was found in the data, we handle them using following methods:

1. **Imputation:** Replace null values with statistical measures like the mean, median, or mode, depending on the data type and distribution. In our dataset, no null values were found so we don't need imputation.

2. **Dropping Rows/Columns:** If the null values are few, drop the rows or columns with missing data. However, this is not ideal if you lose too much information.



[8]
```
data.isnull().sum()
```

	0
Temperature	0
Humidity	0
PM2.5	0
PM10	0
NO2	0
SO2	0
CO	0
Proximity_to_Industrial_Areas	0
Population_Density	0
Air Quality	0

dtype: int64

[9]
```
threshold = data_point/2
to_drop = []
for i in data.keys():
  if data[i].isnull().sum() > threshold:
    to_drop.append(i)
if to_drop == []:
  print("Nothing to Drop")
else:
  data = data.drop(columns=to_drop)
```

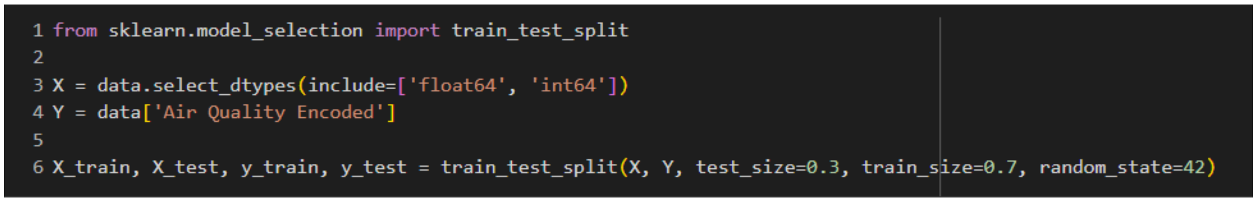
```
Nothing to Drop
```

The dataset has no missing values, as confirmed by the data.isnull().sum() output, which shows zero missing entries across all columns. The code further checks for columns with a high percentage of missing values to determine if any should be dropped. The output validates "Nothing to Drop," suggesting a clean dataset prepared for analysis, because there are no such columns.

3. **Categorical Values:** Classification algorithms such as Logistic Regression, k-Nearest Neighbors, and Naive Bayes treat the target classes as distinct categories when trained on categorical labels. In such cases, encoding is not necessary, as the algorithms optimize for class separation directly. Some algorithms, like decision trees or gradient boosting models, can handle categorical data directly without encoding. For this we didn't use label encoding.

Dataset Splitting

To train the model, data splitting was done at 70% for the training and 30% of the testing. We took the random state = 42 which means whenever we use 42 as a random state, it'll return a shuffled dataset.



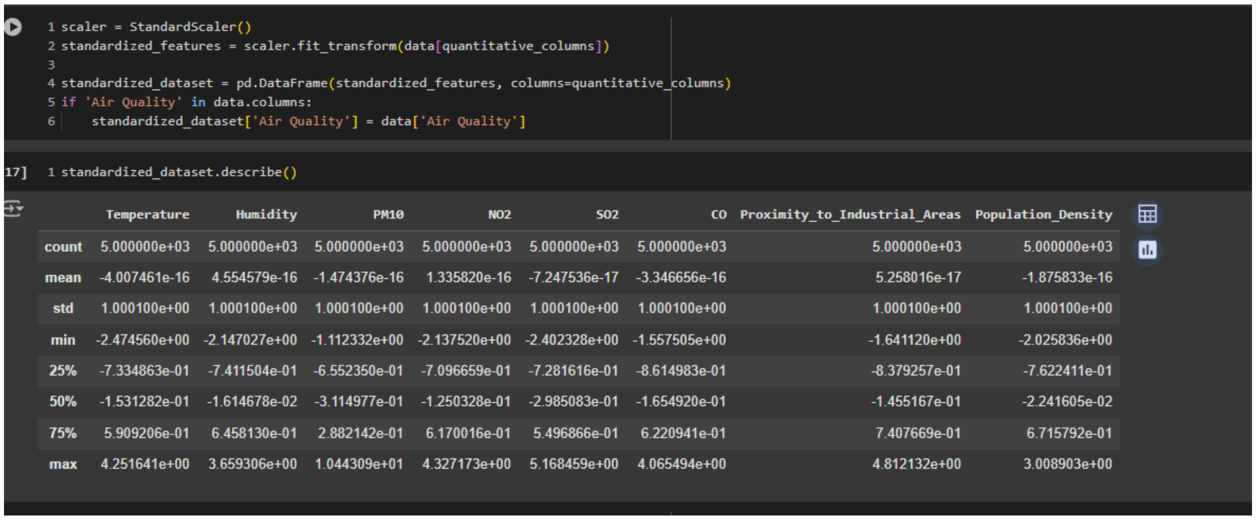
```
from sklearn.model_selection import train_test_split

X = data.select_dtypes(include=['float64', 'int64'])
Y = data['Air Quality Encoded']

X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.3, train_size=0.7, random_state=42)
```

Feature Scaling

We used a Standard scaler for our project. It makes sure that features with large ranges don't dominate features with smaller ranges and ensures all the features contribute equally.

This was necessary for KNN, which relies on distance metrics, as unscaled features with larger ranges could dominate the distance calculation. For Naive Bayes (Gaussian), scaling helps meet the assumption of normal distribution for better performance.

Although Decision Tree models are not sensitive to feature scales, standardization does not harm their performance, so applying it uniformly across all models simplifies preprocessing.



```python
scaler = StandardScaler()
standardized_features = scaler.fit_transform(data[quantitative_columns])

standardized_dataset = pd.DataFrame(standardized_features, columns=quantitative_columns)
if 'Air Quality' in data.columns:
    standardized_dataset['Air Quality'] = data['Air Quality']
```

```python
standardized_dataset.describe()
```

	Temperature	Humidity	PM10	NO2	SO2	CO	Proximity_to_Industrial_Areas	Population_Density
count	5.000000e+03	5.000000e+03	5.000000e+03	5.000000e+03	5.000000e+03	5.000000e+03	5.000000e+03	5.000000e+03
mean	-4.007461e-16	4.554579e-16	-1.474376e-16	1.335820e-16	-7.247536e-17	-3.346656e-16	5.258016e-17	-1.875833e-16
std	1.000100e+00	1.000100e+00	1.000100e+00	1.000100e+00	1.000100e+00	1.000100e+00	1.000100e+00	1.000100e+00
min	-2.474560e+00	-2.147027e+00	-1.112332e+00	-2.137520e+00	-2.402328e+00	-1.557505e+00	-1.641120e+00	-2.025836e+00
25%	-7.334863e-01	-7.411504e-01	-6.552350e-01	-7.096659e-01	-7.281616e-01	-8.614983e-01	-8.379257e-01	-7.622411e-01
50%	-1.531282e-01	-1.614678e-02	-3.114977e-01	-1.250328e-01	-2.985083e-01	-1.654920e-01	-1.455167e-01	-2.241605e-02
75%	5.909206e-01	6.458130e-01	2.882142e-01	6.170016e-01	5.496866e-01	6.220941e-01	7.407669e-01	6.715792e-01
max	4.251641e+00	3.659306e+00	1.044309e+01	4.327173e+00	5.168459e+00	4.065494e+00	4.812132e+00	3.008903e+00

Model Training and Testing

We select three models and they are KNN, Decision Tree, and Gasussian Naive Bayes

KNN

Knn uses a feature set to classify data points based on the majority class of their k nearest neighbors. It is ideal for instances involving multiple classes and where a decision boundary is necessary.

Advantages:

- Easy to understand and use.
- Good for small datasets and complex boundaries.
- No training needed, so it works instantly.

Disadvantages:

- Slow when there are many data points.
- Affected by irrelevant features and needs data scaling.
- Results depend a lot on the choice of "k" and distance method.

Decision Tree

A decision tree uses measures like information gain to determine which splits are optimal after dividing a dataset into subsets according to feature values. It is effective at measuring the significance of features. It handles both categorical and numerical data.

Advantages:

- Simple to visualize and explain.
- Works with both numbers and categories.
- Doesn’t assume anything about the data.

Disadvantages:

- Can overfit if the tree is too detailed.
- Small changes in data can change the tree a lot.
- Needs pruning or combining with other methods to improve results.

Naive Bayes

The Bayes theorem based probabilistic classifier Naive Bayes assumes feature independence and performs quickly and effectively on large datasets. It functions well with categorical and conditionally independent features.

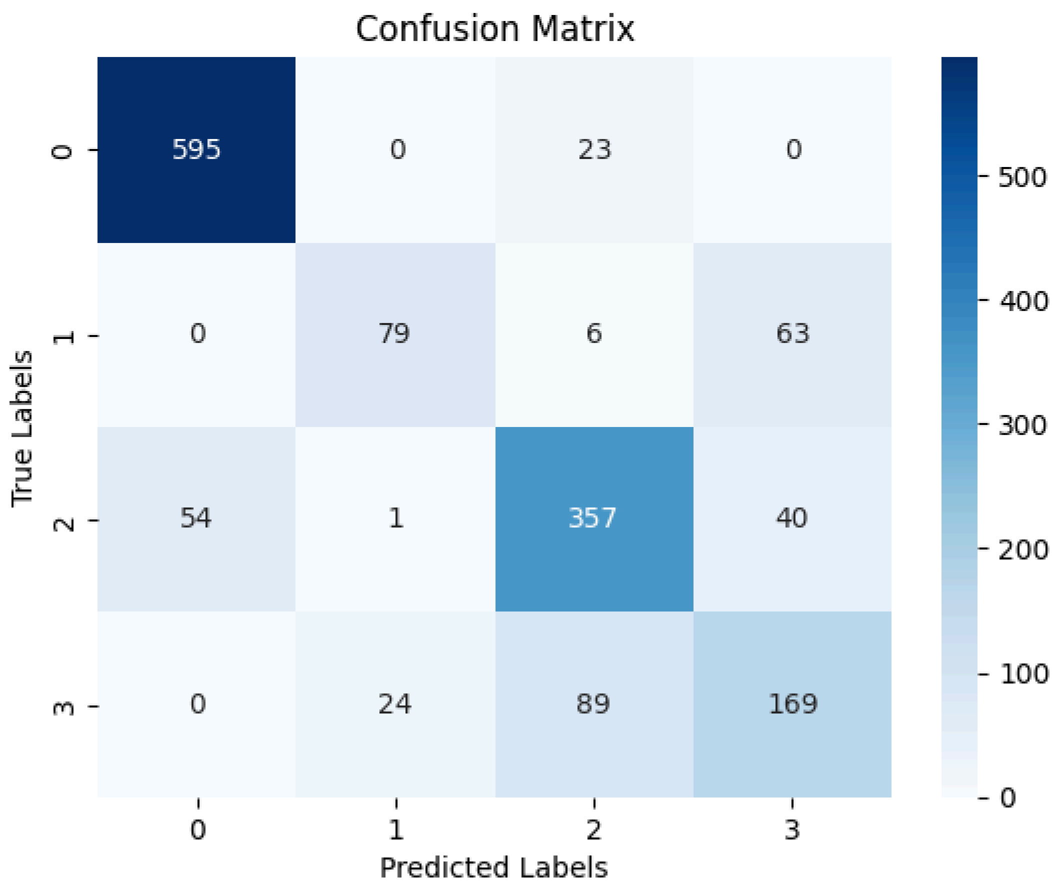
Advantages:

- Very fast, even with lots of data.
- Great for text data and classification tasks.
- Handles irrelevant features well.

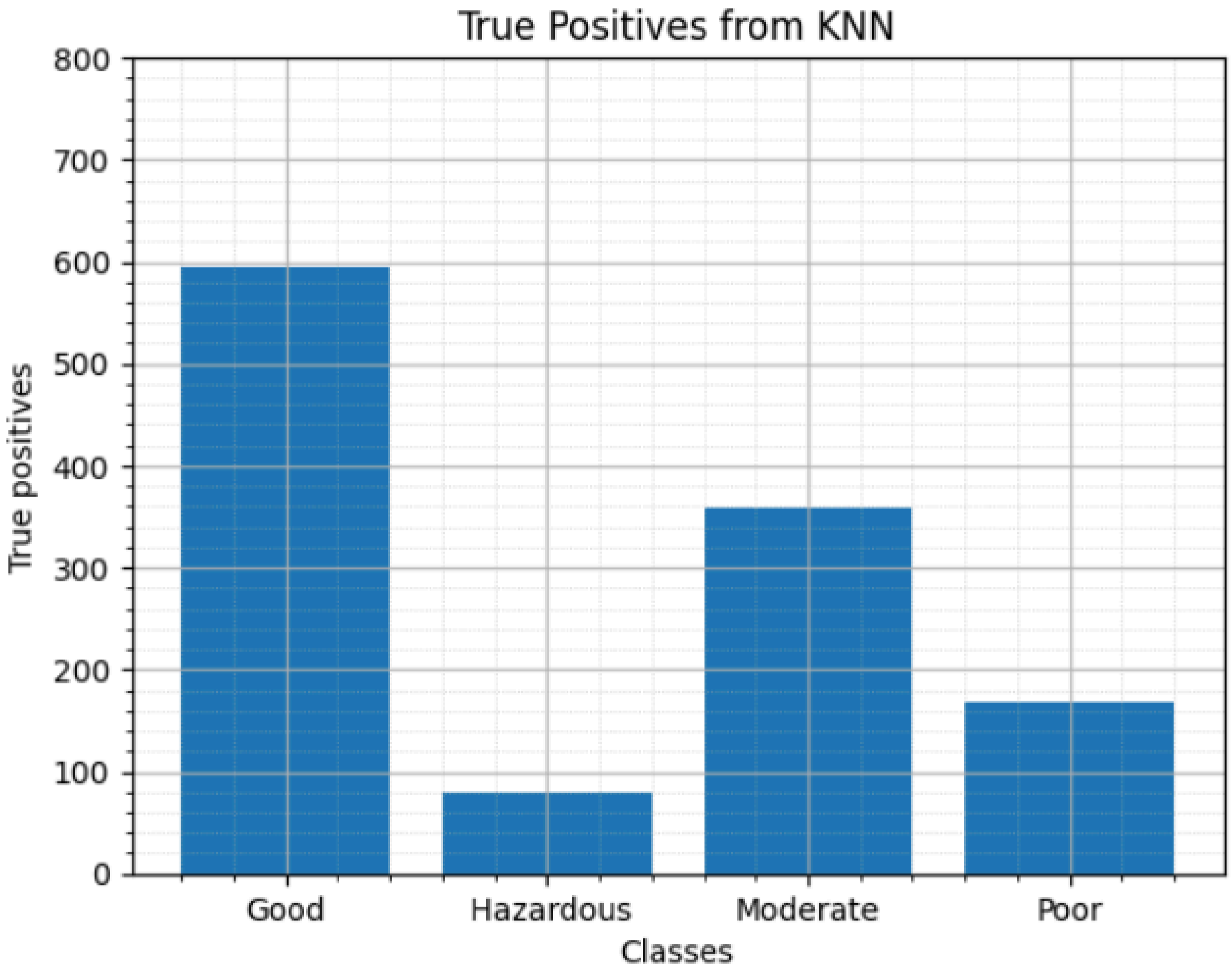
Disadvantages:

- Assumes features don’t depend on each other, which isn’t always true.
- Doesn’t work well when features are strongly related.
- Can give bad results with small data or rare events.

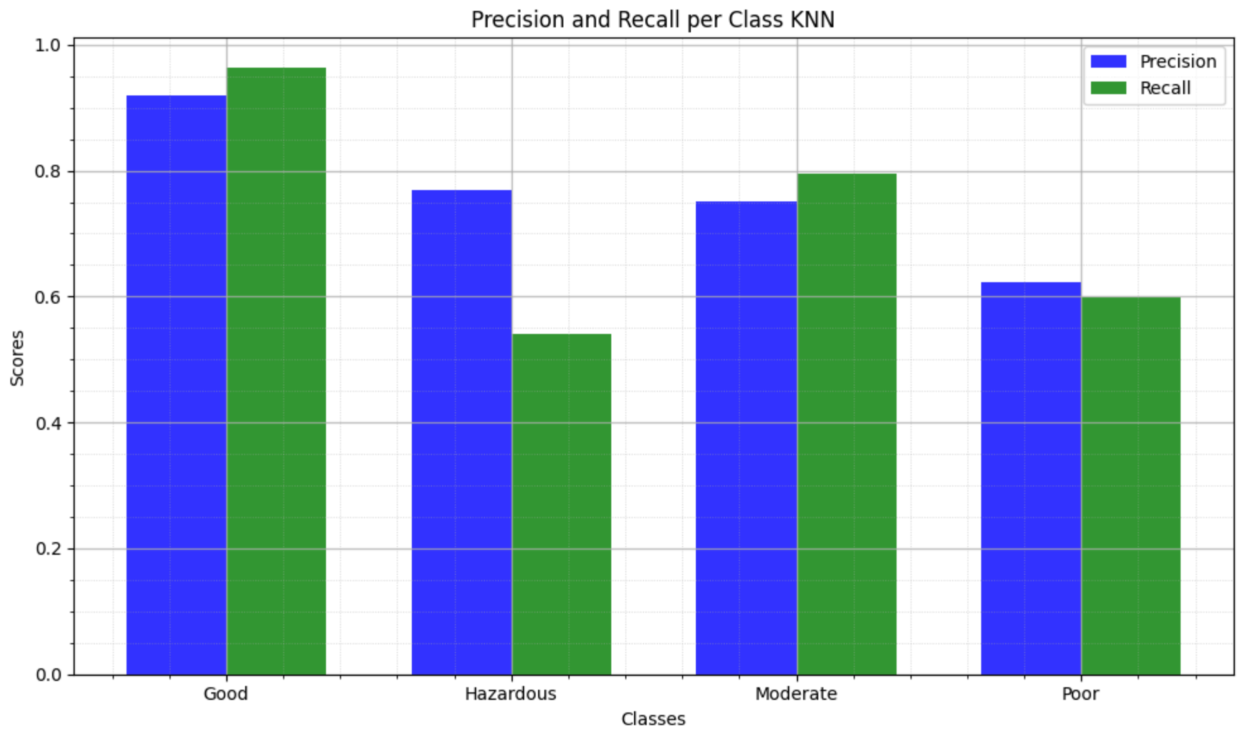
For **KNN** we get



The confusion matrix shows how the model predicts each class. For the Good class, the model performs very well, correctly predicting 595 instances and misclassifying only 23 as Moderate. For the Hazardous class, it correctly identifies 79 instances but misclassifies 63 as Poor and 6 as Moderate. The Moderate class has 357 correct predictions, with some misclassified 54 as Good, 40 as Poor, and 1 as Hazardous. Lastly, for the Poor class, the model correctly predicts 169 instances but misclassifies 24 as Hazardous and 89 as Moderate. These results suggest strong performance for the Good class but challenges in distinguishing between the Hazardous, Moderate, and Poor classes, due to imbalanced data.

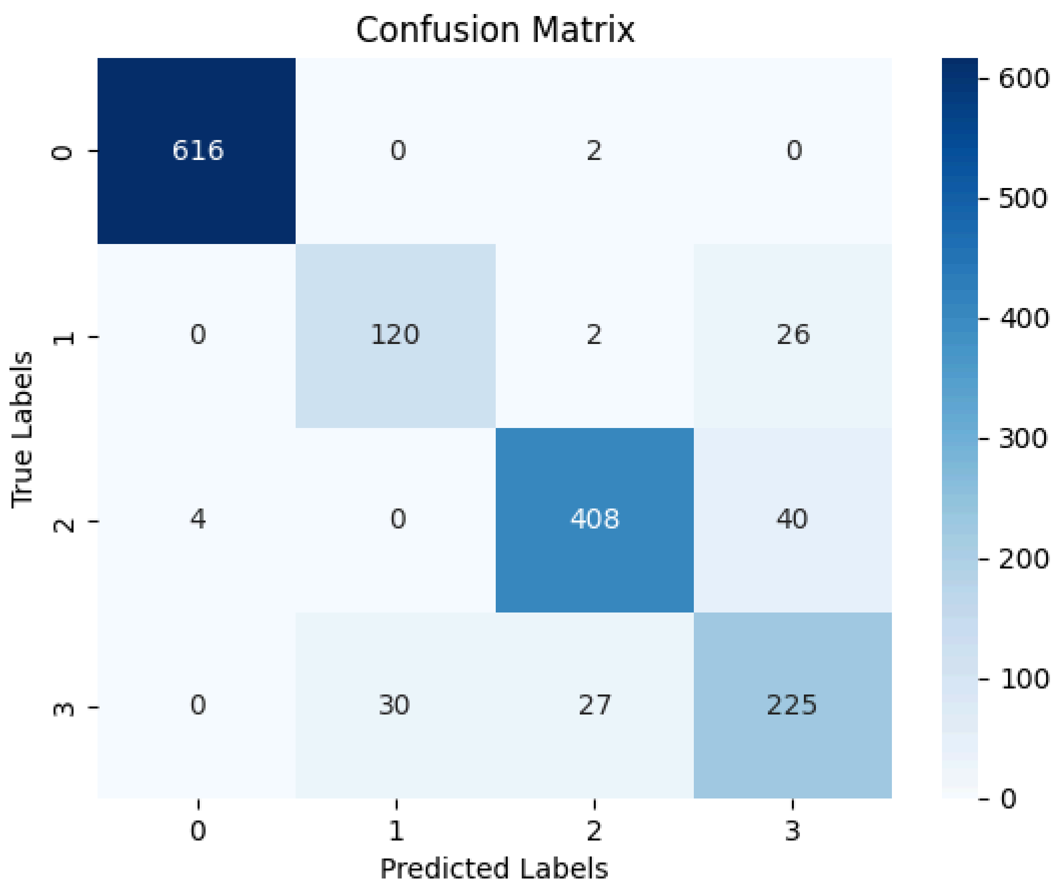


From the above Bar chart we can see that the true positive of the Good class is the highest and the true positive of Hazardous is lowest. This means that if the model is given good it will predict good 595 times and if we give hazardous in the model it will predict hazardous 79 times. This indicates that the model is more effective in identifying instances of Good, possibly due to a higher representation in the dataset or distinct features making it easier to classify. In contrast, the lower true positives for Hazardous may be due to fewer samples in the dataset, leading to less effective training for this class.

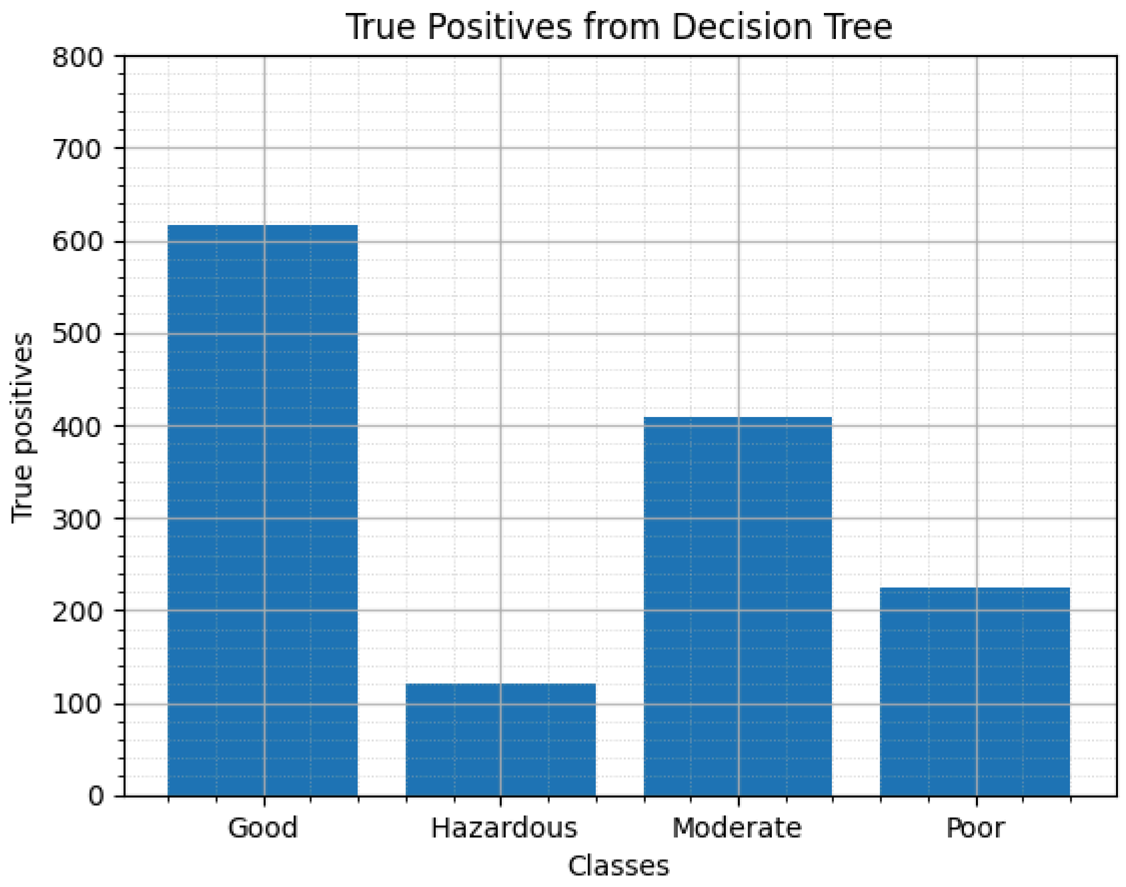


The precision and recall of the KNN classifier for each class are displayed in the bar chart. With a precision of 0.92 and recall of 0.96 for the Good class, the model demonstrates remarkable performance in predicting and retrieving actual Good instances. With a precision of 0.75 and recall of 0.79, the Moderate class performs well overall. But for the Hazardous class, where recall is just 0.54 and precision is 0.77, the performance drastically falls, indicating that many instances in reality of this class are being overlooked. The model performs worst in the Poor class, as seen by its lowest precision 0.62 and recall 0.60. These variations in performance suggest potential challenges with class imbalance, particularly for the Hazardous and Poor classes.

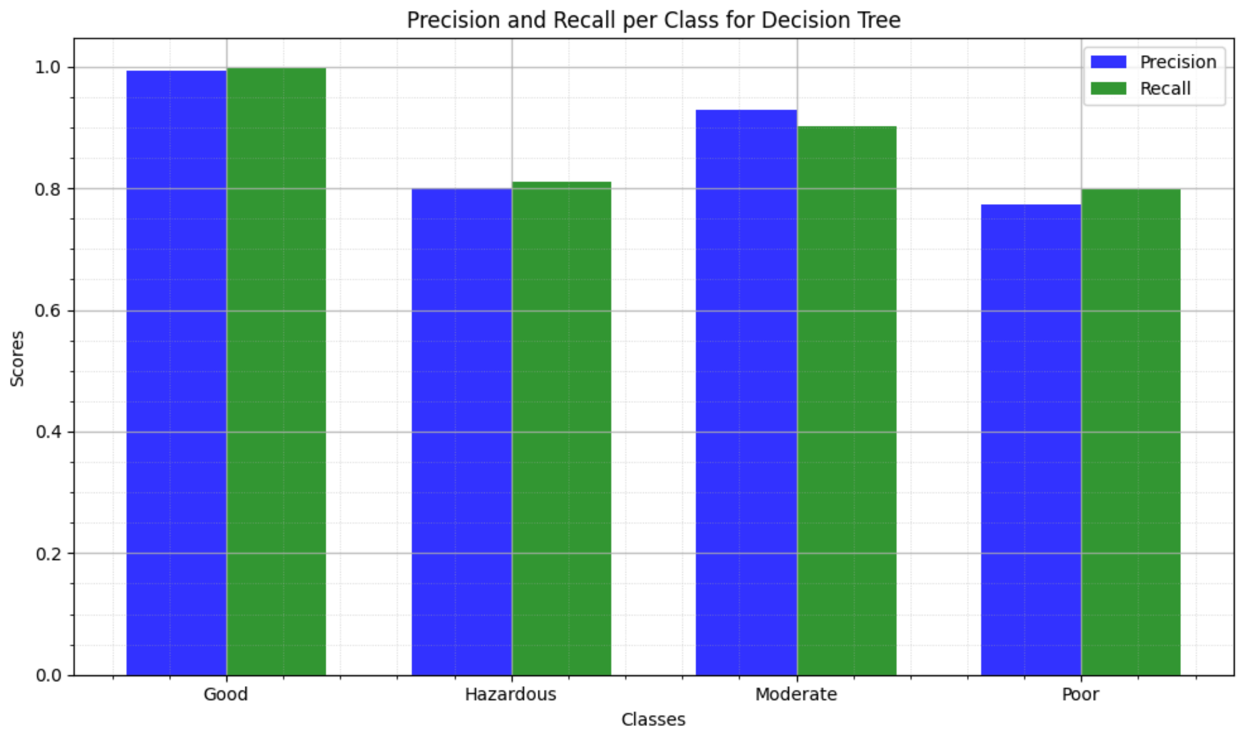
For **Decision Tree** we get



Here, in the confusion matrix, we observe the true positives for each of the 4 classes along the diagonal of the matrix. For instance, the model correctly predicted 616 samples of class Good, but misclassified 2 as Moderate. Similarly, for class Hazardous, the model correctly predicted 120 instances but misclassified 26 as Poor and 2 as Moderate. For the Moderate class, the model correctly predicted 408 samples, and misclassified 40 as Poor. Lastly, for the Poor class, the model predicted 225 correctly but misclassified 30 as Hazardous and 27 as Moderate. These misclassifications may arise from insufficient differentiation in the training data.

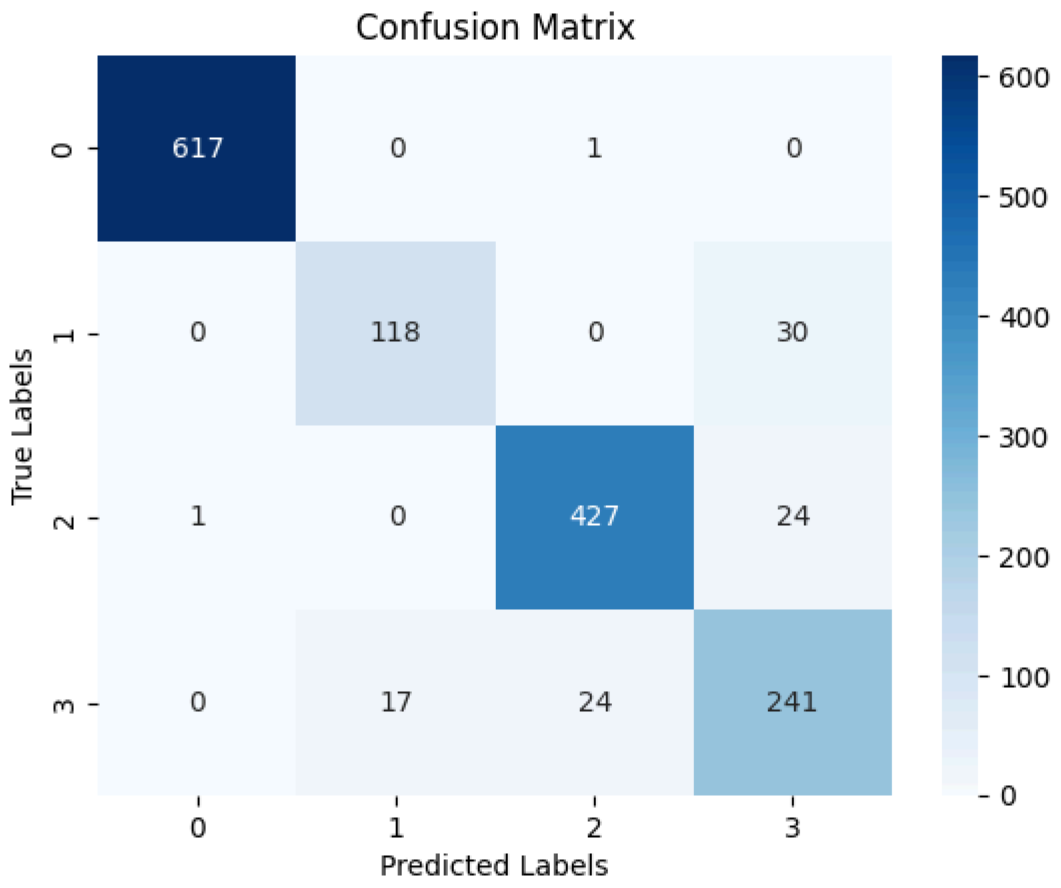


From the above bar chart, we can see that the true positive count for the Good class is the highest, while the Hazardous class has the lowest. This indicates that the model predicts Good correctly 616 times and Hazardous 120 times when provided with respective instances. This suggests that the model is more effective at identifying instances of Good, likely due to distinct features or a higher representation of this class in the dataset. On the other hand, the lower true positives for Hazardous may be attributed to fewer samples or overlapping features with other classes, leading to challenges in accurately identifying this class during training.

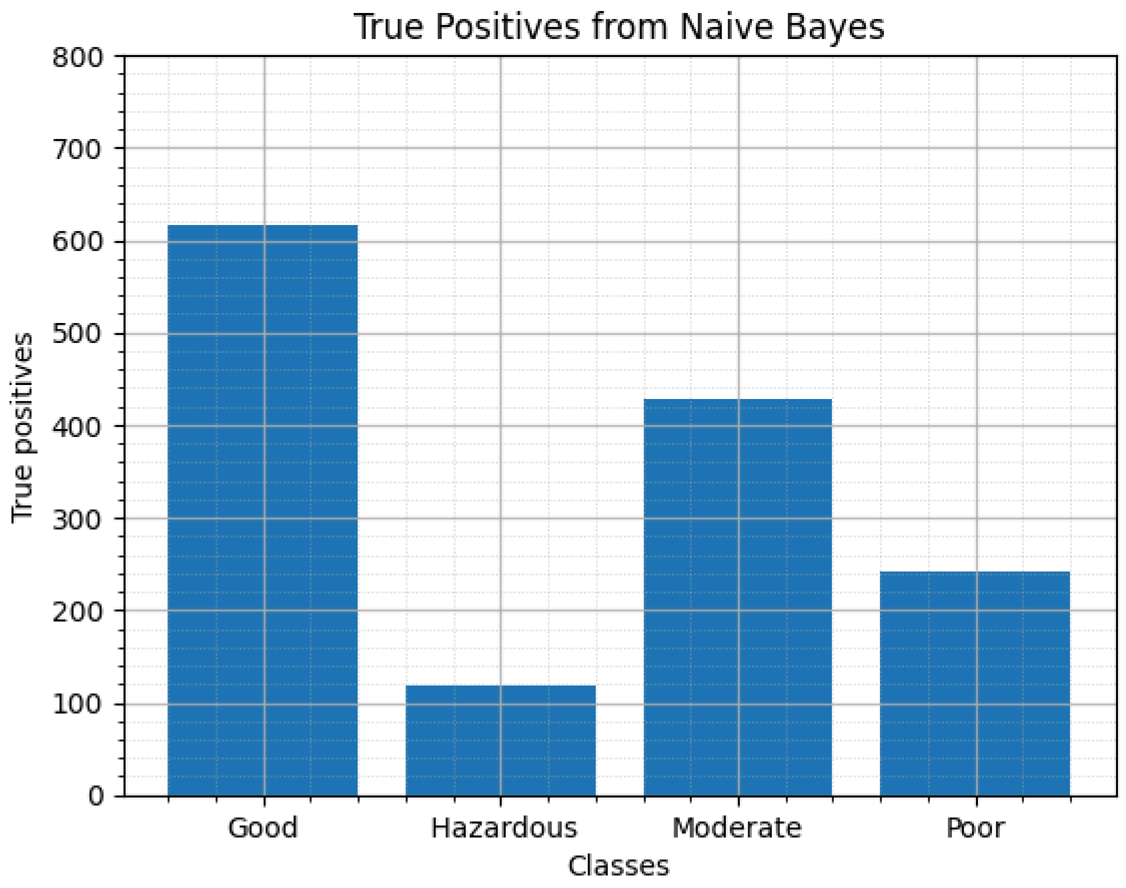


The precision and recall for the Decision Tree classifier for each class are displayed in the bar chart. The model performs exceptionally well for the Good class, with a precision 1.0 and recall of 0.98, demonstrating its strong capability to correctly predict and retrieve Good instances. The Moderate class also shows good performance, with a precision of approximately 0.91 and recall of 0.89. However, the performance drops for the Hazardous class, where the precision is 0.81 and recall is 0.78, indicating the model misses some true instances of this class. The lowest performance is observed in the Poor class, with a precision of 0.82 and recall of 0.77. These results highlight that while the Decision Tree model performs well for dominant and distinguishable classes like Good and Moderate, it faces challenges in accurately identifying Hazardous and Poor classes, due to class imbalance in the dataset. Addressing these issues could further improve the model's overall effectiveness.

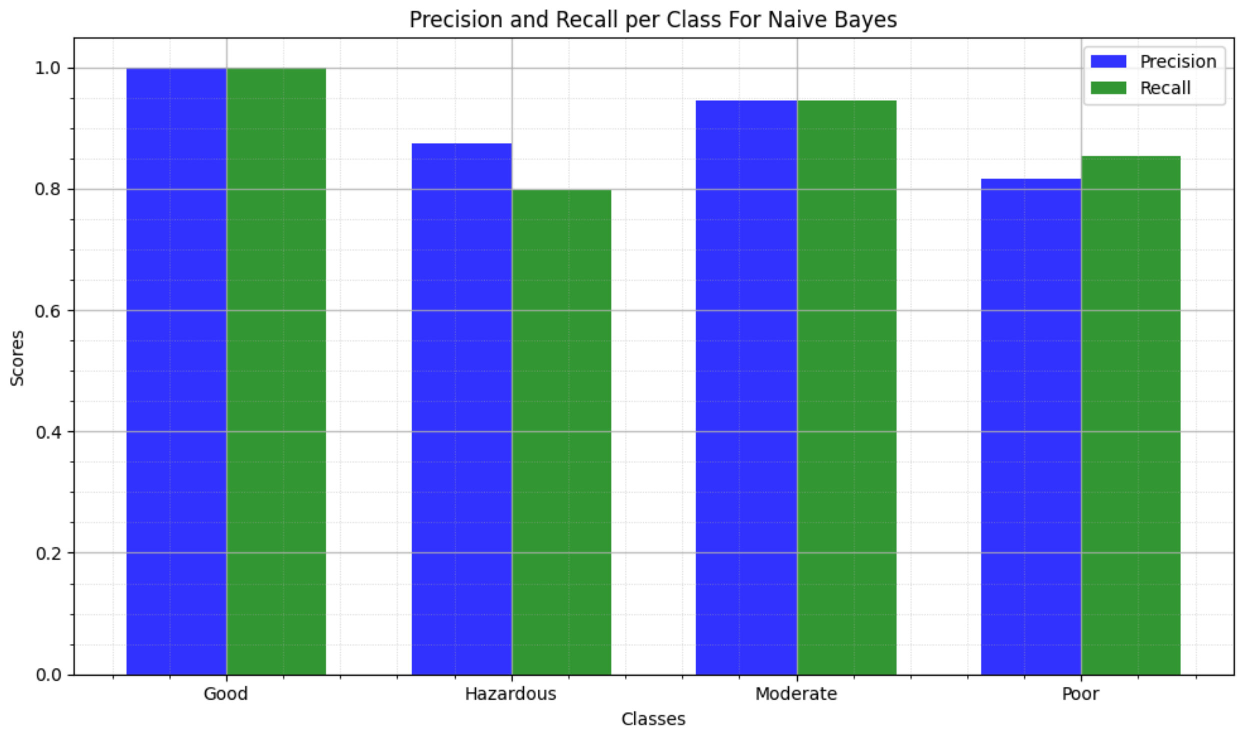
For **Naive Bayes** we get



The confusion matrix shows the model's predictions for each of the four classes, with the true positives represented along the diagonal. For example, the model correctly identified 617 samples of the Good class, but it misclassified 1 as Moderate. Similarly, it correctly predicted 118 instances of the Hazardous class but misclassified 30 as Poor. For the Moderate class, 427 samples were predicted correctly, while 24 were misclassified as Poor and 1 as Good. Lastly, for the Poor class, the model correctly predicted 241 samples but mistakenly classified 17 as Hazardous and 24 as Moderate. These errors suggest that while the Naive Bayes model performs well with the Good class, it struggles to clearly differentiate between Hazardous, Moderate, and Poor, due to an imbalance in the dataset.



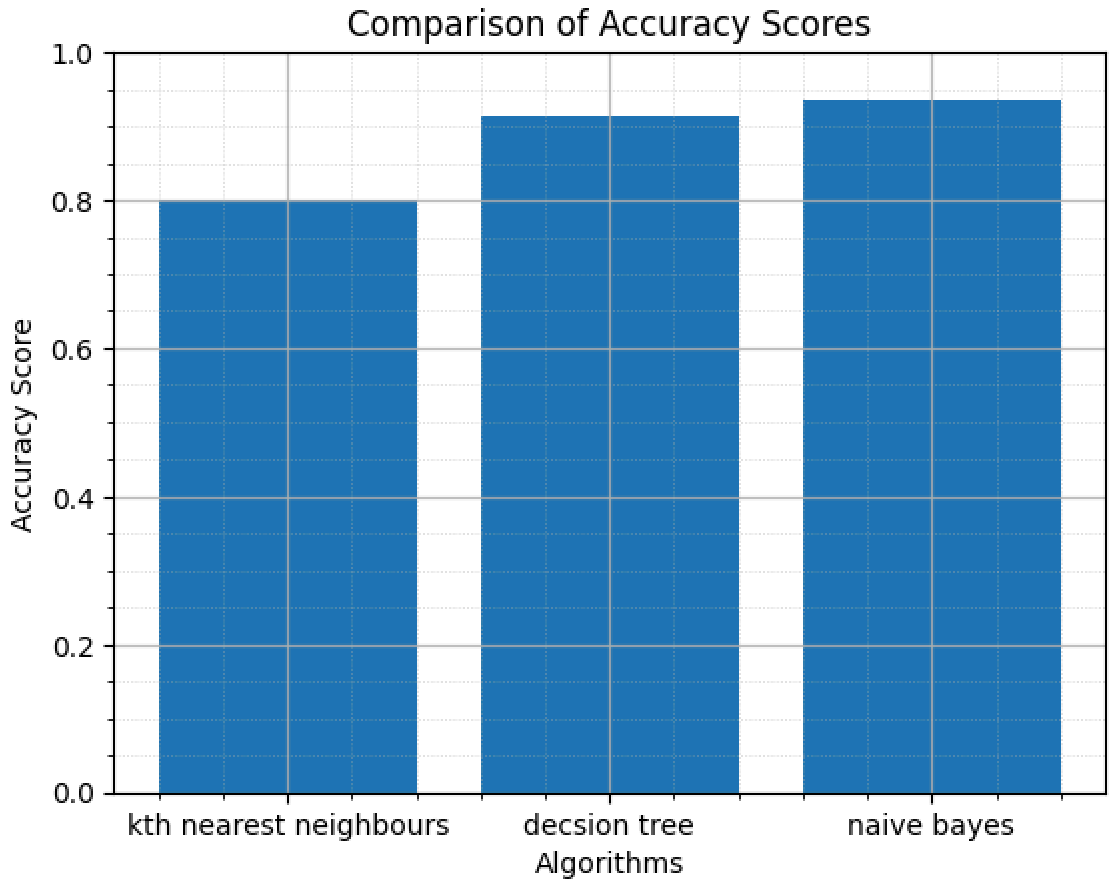
The bar chart shows the true positives for each class predicted by the Naive Bayes model. The model performs best for the Good class, correctly identifying over 600 instances. For the Moderate class, it predicts around 400 instances accurately, which is also a strong performance. However, the Hazardous class has the lowest true positives, with fewer than 150 correctly predicted instances, indicating the model struggles the most with this category. The Poor class has a moderate performance, with over 200 true positives. These results suggest that the model is highly effective for the 'Good' class but faces challenges in accurately identifying the Hazardous and Poor classes, due to imbalances in the dataset.



The bar chart shows the Naive Bayes model performs best for the Good and Moderate classes, with high precision and recall 1.0 for Good and 0.94 and 0.94 for Moderate . The Hazardous class performs decently, with both scores above 0.8, while the Poor class shows slightly lower precision and recall around 0.8. Overall, the model is highly accurate for Good and Moderate but struggles a bit with Hazardous and Poor, likely due to dataset imbalances.

The results are quite ok but we could have done better.

Comparison Analysis



The K-nearest neighbor (KNN) model displays a comparatively lower accuracy of 80% in contrast to the Decision tree 91% and Naive Bayes 94% model.

Conclusion

The Naive Bayes model, with an accuracy of 94%, was the best-performing model in this project. However, there is still a chance that it may overfit the data. The good performance was achieved because of preprocessing techniques like feature scaling and handling class imbalance, which improved the model's predictions. This project shows that machine learning can be a useful tool for predicting air quality, which is important for public health and raising awareness about pollution. With further improvements, these models can become even more accurate and reliable in helping society. Future work can focus on better data preprocessing methods, like creating synthetic data to handle class imbalance, and adding more features to make predictions more detailed. Using larger datasets and testing advanced models like ensemble methods or deep learning could also improve the results.

Overall, while the current results are promising, more work is needed to make this system more effective for air quality prediction and creating environmental awareness.

References

https://www.kaggle.com/datasets/mujtabamatin/air-quality-and-pollution-assessment