CSCI 6515 - Machine Learning for Big Data (Fall 2023)

Assignment No. 1

Mudra Verma
Banner ID: B00932103

Descriptive Analysis :

1. Task 1

The PM2.5 data and the traffic data was obtained through the Nova Scotia Data website [1][2]. A description of labels in the dataset was also examined.

2. Task 2

i) Subtask 2.i

In [1]:
```
#### Filtering the traffic dataset to represent Halifax region ####

import pandas as pd

# Load the dataset
traffic_data = pd.read_csv('Traffic_Volumes_-_Provincial_Highway_System.csv')

# Remove the specified columns from the DataFrame
columns_to_remove = ['HIGHWAY', 'SECTION ID', 'SECTION', 'SECTION LENGTH', 'SECT
traffic_data = traffic_data.drop(columns=columns_to_remove)

# Filter the dataset to include only rows with 'county' value 'HFX' and year >=
traffic_data = traffic_data[(traffic_data['COUNTY'] == 'HFX') & (traffic_data['D
```

In [2]:
```
#### Creating a workable dataset through feature selection ####

# Convert the 'Date' column to datetime format
traffic_data['Date'] = pd.to_datetime(traffic_data['Date'], errors='coerce')

# Sort the dataset by the 'Date' column in ascending order
traffic_data = traffic_data.sort_values(by='Date')

# Remove the specified columns from the DataFrame
columns_to_remove = ['COUNTY']
traffic_data = traffic_data.drop(columns=columns_to_remove)

# Clean and convert numeric columns by removing commas and converting to float
numeric_columns = ['ADT', 'AADT', 'PTRUCKS', '85PCT']
for col in numeric_columns:
    if traffic_data[col].dtype == object:
```

```
        traffic_data[col] = traffic_data[col].str.replace(',', '').astype(float)

# Replace empty rows in the 'DIRECTION' column with 'T'
traffic_data['DIRECTION'].fillna('T', inplace=True)

# Forward fill the 'GROUP' column to fill missing values
traffic_data['GROUP'].fillna(method='ffill', inplace=True)

# Group the data by 'Date' and choose an aggregation function (e.g., 'mean')
traffic_data = traffic_data.groupby('Date').agg({
    'ADT': 'mean',
    'AADT': 'mean',
    'PTRUCKS': 'mean',
    '85PCT': 'mean',
    'DIRECTION': 'first',  # Choose 'first' for the 'DIRECTION' column
    'GROUP': 'first'       # Choose 'first' for the 'GROUP' column
}).reset_index()

# Handle missing values by filling with the mean of each colum
traffic_data['ADT'].fillna(traffic_data['ADT'].mean(), inplace=True)
traffic_data['AADT'].fillna(traffic_data['AADT'].mean(), inplace=True)
traffic_data['PTRUCKS'].fillna(traffic_data['PTRUCKS'].mean(), inplace=True)
traffic_data['85PCT'].fillna(traffic_data['85PCT'].mean(), inplace=True)

# Save the sorted and filtered dataset to a new CSV file
traffic_data.to_csv('traffic_data.csv', index=False)

print(traffic_data)
```

```
           Date           ADT      AADT   PTRUCKS       85PCT DIRECTION GROUP
0    2019-01-01  10073.304419  17902.50  9.000000  114.666667         T     C
1    2019-04-29  26354.000000  25500.00  8.345707  102.702778         S     A
2    2019-05-02   5705.000000   5568.75  4.500000  102.702778         W     B
3    2019-05-09   1103.625000   1138.75  6.500000  102.702778         T     B
4    2019-05-16   3495.142857   3380.00  7.500000  102.702778         W     A
..          ...           ...       ...       ...         ...       ...   ...
133  2022-10-25  19488.000000  19500.00  8.345707  102.702778         T     A
134  2022-11-02   5794.000000   6020.00  8.345707  102.702778         T     A
135  2022-11-06  27875.500000  28185.00  8.345707  102.702778         N    AA
136  2022-11-17  23241.000000  23060.00  8.345707  102.702778         N    AA
137  2022-11-23   6287.500000   7210.00  8.345707  102.702778         E     C

[138 rows x 7 columns]
```

Step by Step Description of Traffic Dataset Preparation

1. Convert Date to DateTime Format: Converting 'Date' column to datetime format.
2. Sort Data by Date: Sorting the dataset by the 'Date' column in ascending order.
3. Remove Unwanted Columns: Eliminating unwanted columns in the dataset through feature selection.
4. Clean Numeric Columns: Cleaning up columns and converting them to float data type.
5. Handle Missing Values in 'DIRECTION': As per the traffic data description missing values in the 'DIRECTION' column indicates a two way direction. So filling them with 'T', indicating two way traffic.
6. Forward Fill 'GROUP' Column: Using forward filling for the 'GROUP' column, which means filling missing values with the most recent non-missing value in the column

[3].

7. Group Data by Date: Calculating mean for numerical data to aggregate it. Keeping the first 'DIRECTION' and 'GROUP' values for each date since they shouldn't change within a day [4].
8. Handle Remaining Missing Values: Filling missing values with the mean value of respective columns [5].
9. Save the Processed Data: Saving cleaned, sorted, and aggregated dataset to a new CSV file called 'traffic_data.csv'.

ii) Subtask 2.ii

In [3]:
```
##### Computing daily averages for PM2.5 data to serve as labels ####
import pandas as pd
from sklearn.preprocessing import MinMaxScaler

# Load PM2.5 dataset
pm25_data = pd.read_csv('Nova_Scotia_Provincial_Ambient_Fine_Particulate_Matter_

# Convert the 'Date & time' column to datetime format
pm25_data['Date & time'] = pd.to_datetime(pm25_data['Date & time'], format='%Y/%

# Extract the date from the datetime column
pm25_data['Date'] = pm25_data['Date & time'].dt.date

# Filter the data starting from 2019
pm25_data = pm25_data[pm25_data['Date & time'].dt.year >= 2019]

# Compute daily average PM2.5 levels
daily_pm25 = pm25_data.groupby('Date')['Average'].mean().reset_index()
daily_pm25.columns = ['Date', 'daily_average_pm25']

print(daily_pm25)
```

```
            Date  daily_average_pm25
0     2019-01-01            3.083333
1     2019-01-02            2.625000
2     2019-01-03            5.625000
3     2019-01-04            5.136364
4     2019-01-05            8.208333
...          ...                 ...
1091  2021-12-27            3.470833
1092  2021-12-28            4.025000
1093  2021-12-29            4.191667
1094  2021-12-30            4.875000
1095  2021-12-31            5.700000

[1096 rows x 2 columns]
```

iii) Subtask 2.iii

In [4]:
```
##### Normalizing the PM2.5 levels and discretizing them using a threshold of 0.
##### Labeling the data so that 0 represents Low and 1 represents High ####

# Initialize the Min-Max scaler
scaler = MinMaxScaler()
```

```
# Normalize the 'daily_average_pm25' column
daily_pm25['normalized_pm25'] = scaler.fit_transform(daily_pm25[['daily_average_

# Discretize PM2.5 levels based on the threshold
daily_pm25['pm25_category'] = daily_pm25['normalized_pm25'].apply(lambda x: 'Hig

# Map 'Low' to 0 and 'High' to 1 in the 'pm25_category' column
daily_pm25['pm25_category'] = daily_pm25['pm25_category'].map({'Low': 0, 'High':

# Save only the 'date' and 'pm25_category' columns to a new CSV file
daily_pm25[['Date','pm25_category']].to_csv('pm_dataset.csv', index=False)

print(daily_pm25)
```

```
            Date  daily_average_pm25  normalized_pm25  pm25_category
0     2019-01-01            3.083333         0.153734              0
1     2019-01-02            2.625000         0.126891              0
2     2019-01-03            5.625000         0.302587              0
3     2019-01-04            5.136364         0.273970              0
4     2019-01-05            8.208333         0.453880              0
...          ...                 ...              ...            ...
1091  2021-12-27            3.470833         0.176428              0
1092  2021-12-28            4.025000         0.208882              0
1093  2021-12-29            4.191667         0.218643              0
1094  2021-12-30            4.875000         0.258663              0
1095  2021-12-31            5.700000         0.306979              0

[1096 rows x 4 columns]
```

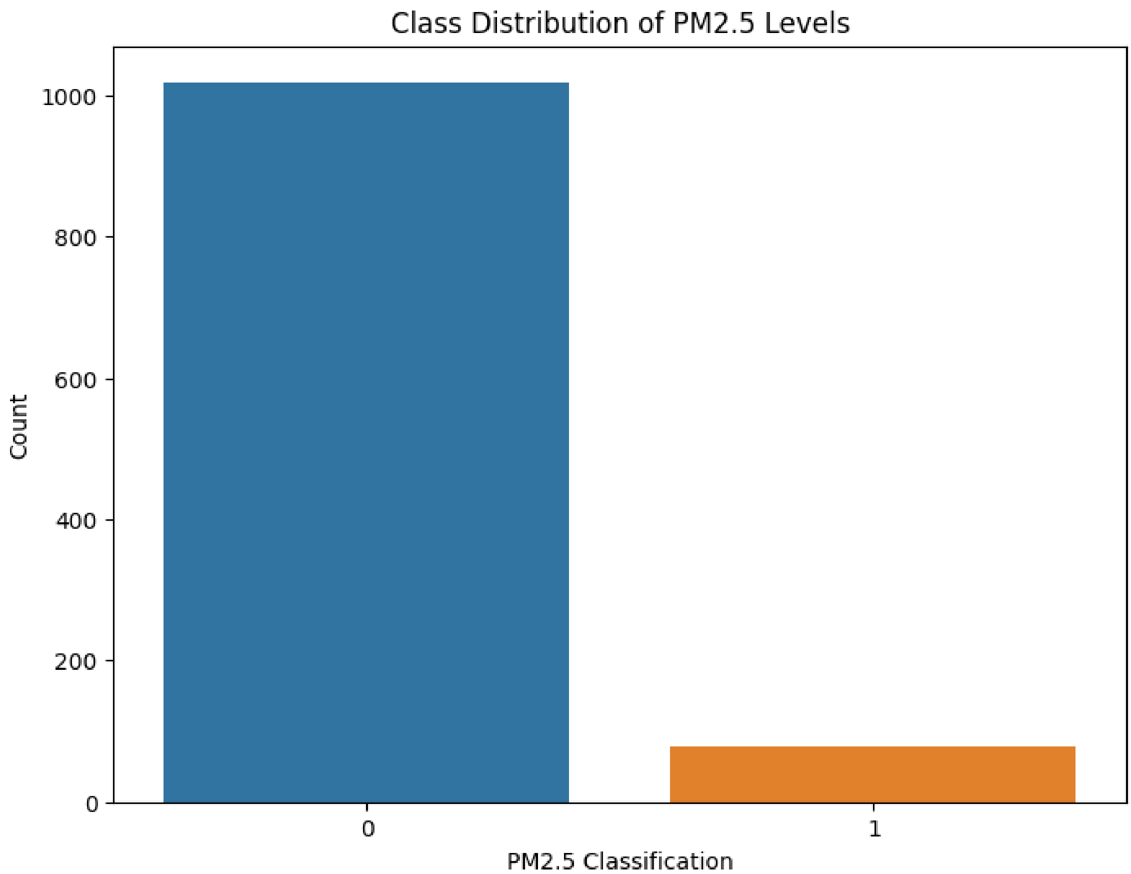
In [5]:
```
####  Visualization of Data using Class Distribution [8] ####

import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Load PM2.5 dataset
daily_pm25 = pd.read_csv('pm_dataset.csv')

# Calculate class distribution
class_distribution = daily_pm25['pm25_category'].value_counts()

# Plot class distribution (0 vs. 1)
plt.figure(figsize=(8, 6))
sns.barplot(x=class_distribution.index, y=class_distribution.values)
plt.title('Class Distribution of PM2.5 Levels')
plt.xlabel('PM2.5 Classification')
plt.ylabel('Count')
plt.show()
```



iv) Subtask 2.iv

In [6]:

```
#### Merging the PM2.5 dataset and Traffic dataset ####
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Load traffic dataset
traffic_data = pd.read_csv('traffic_data.csv')

# Load the CSV file containing the target variable
target_data = pd.read_csv('pm_dataset.csv')

# Assuming there's a common column 'ID' between the datasets
merged_data = pd.merge(traffic_data, target_data, on='Date')

print(merged_data)

# Save the sorted and filtered dataset to a new CSV file
merged_data.to_csv('merged_data.csv', index=False)
```

```
          Date           ADT          AADT    PTRUCKS       85PCT DIRECTION  \
0   2019-01-01  10073.304419  17902.500000   9.000000  114.666667         T   
1   2019-04-29  26354.000000  25500.000000   8.345707  102.702778         S   
2   2019-05-02   5705.000000   5568.750000   4.500000  102.702778         W   
3   2019-05-09   1103.625000   1138.750000   6.500000  102.702778         T   
4   2019-05-16   3495.142857   3380.000000   7.500000  102.702778         W   
..         ...           ...           ...        ...         ...       ...   
92  2021-11-01  33425.600000  34720.000000  13.500000  102.702778         N   
93  2021-11-04   1104.777778   1173.333333   6.750000  102.702778         T   
94  2021-11-17  10944.333333  10852.111111   8.345707  102.702778         E   
95  2021-11-22  11742.000000  11600.000000   8.345707  102.702778         E   
96  2021-11-25  14323.400000  14416.000000   8.345707  102.702778         W   

   GROUP  pm25_category  
0      C              0  
1      A              0  
2      B              0  
3      B              0  
4      A              0  
..   ...            ...  
92     A              0  
93     C              0  
94     A              0  
95    AA              1  
96    AA              0  

[97 rows x 8 columns]
```

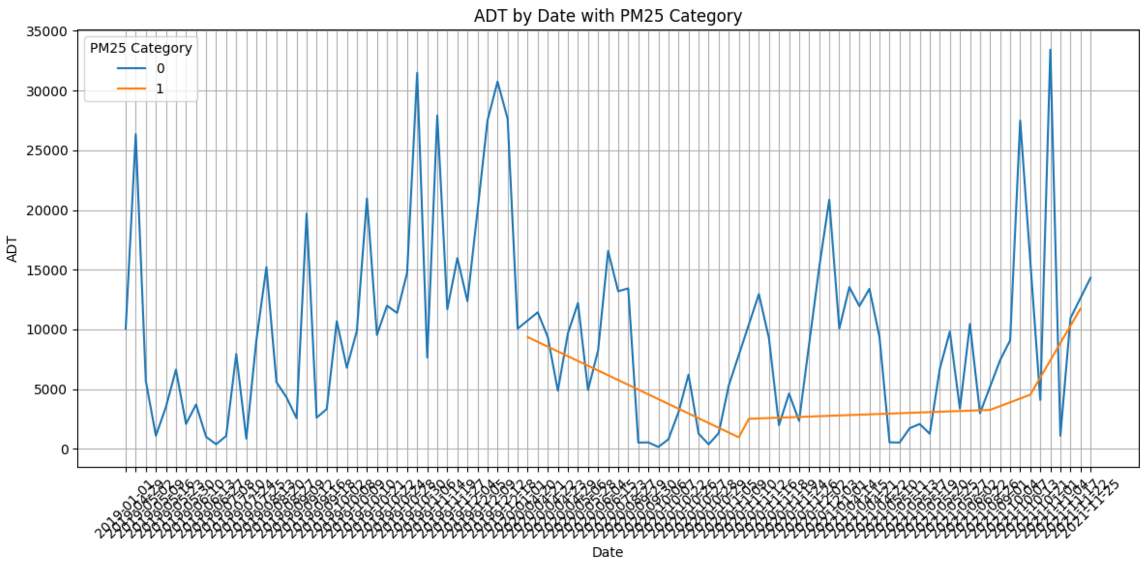
In [7]:

```python
####  Visualization of Data using Line Graph ####

import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

# Load dataset
dataset = pd.read_csv('merged_data.csv')

# Create a line plot for ADT by date with different colors for each 'pm25_catego
plt.figure(figsize=(12, 6))
sns.lineplot(x='Date', y='ADT', hue='pm25_category', data=dataset)
plt.xlabel('Date')
plt.ylabel('ADT')
plt.title('ADT by Date with PM25 Category')
plt.xticks(rotation=45)
plt.grid(True)
plt.legend(title='PM25 Category')
plt.tight_layout()
plt.show()
```



3. Task 3

i) Subtask 3.i

In [8]:
```
#### Preprocessing the data by performing one-hot encoding on 'DIRECTION' column

import pandas as pd
from category_encoders import BinaryEncoder

# Load dataset
data = pd.read_csv('merged_data.csv')

# Perform one-hot encoding for the 'Direction' column
data = pd.get_dummies(data, columns=['DIRECTION'], prefix=['DIRECTION'])

# Perform binary encoding for the 'Group' column using BinaryEncoder
binary_encoder = BinaryEncoder(cols=['GROUP'])
data = binary_encoder.fit_transform(data)

# Save the modified dataset back to 'merged_data.csv'
data.to_csv('merged_data.csv', index=False)

print(data)
```

```
          Date           ADT          AADT    PTRUCKS       85PCT  GROUP_0  \
0   2019-01-01  10073.304419  17902.500000   9.000000  114.666667        0
1   2019-04-29  26354.000000  25500.000000   8.345707  102.702778        0
2   2019-05-02   5705.000000   5568.750000   4.500000  102.702778        0
3   2019-05-09   1103.625000   1138.750000   6.500000  102.702778        0
4   2019-05-16   3495.142857   3380.000000   7.500000  102.702778        0
..         ...           ...           ...        ...         ...      ...
92  2021-11-01  33425.600000  34720.000000  13.500000  102.702778        0
93  2021-11-04   1104.777778   1173.333333   6.750000  102.702778        0
94  2021-11-17  10944.333333  10852.111111   8.345707  102.702778        0
95  2021-11-22  11742.000000  11600.000000   8.345707  102.702778        1
96  2021-11-25  14323.400000  14416.000000   8.345707  102.702778        1

    GROUP_1  GROUP_2  pm25_category  DIRECTION_E  DIRECTION_N  DIRECTION_S  \
0         0        1              0            0            0            0
1         1        0              0            0            0            1
2         1        1              0            0            0            0
3         1        1              0            0            0            0
4         1        0              0            0            0            0
..      ...      ...            ...          ...          ...          ...
92        1        0              0            0            1            0
93        0        1              0            0            0            0
94        1        0              0            1            0            0
95        0        1              1            1            0            0
96        0        1              0            0            0            0

    DIRECTION_T  DIRECTION_W
0             1            0
1             0            0
2             0            1
3             1            0
4             0            1
..          ...          ...
92            0            0
93            1            0
94            0            0
95            0            0
96            0            1

[97 rows x 14 columns]
```

Description:

One-Hot Encoding was performed on the 'DIRECTION' column to convert the categorical directions into a numerical format. The 'DIRECTION' column contains categorical data, such as 'N', 'S', 'E', 'W', and 'T'. This allows machine learning models to use this information effectively for training and prediction [9].

Binary Encoding was performed on the 'GROUP' column to convert the ordinal categorical data into a numerical format that retains the ordinal relationship between different groups. This is useful since the order of categories matters in predictions [10].

In [9]:
```
#### Using Information Gain as decision criterion to select attribute to split c

import pandas as pd
import numpy as np

# Load dataset
```

```
data = pd.read_csv('merged_data.csv')

# Define the target variable
target_variable = 'pm25_category'

# Calculate the initial entropy of the target variable
def entropy(data, target_variable):
    unique_classes = data[target_variable].unique()
    entropy_value = 0
    total_samples = len(data)

    for c in unique_classes:
        p = len(data[data[target_variable] == c]) / total_samples
        entropy_value -= p * np.log2(p)

    return entropy_value

initial_entropy = entropy(data, target_variable)

# Calculate entropy for each feature and Information Gain
features_to_exclude = ['Date', target_variable]
feature_entropy = {}

for feature in data.columns:
    if feature not in features_to_exclude:
        weighted_entropy = 0
        unique_values = data[feature].unique()

        for value in unique_values:
            subset = data[data[feature] == value]
            weighted_entropy += len(subset) / len(data) * entropy(subset, target

        feature_entropy[feature] = weighted_entropy

# Calculate Information Gain for each feature
information_gain = {feature: initial_entropy - entropy for feature, entropy in f

# Sort features by Information Gain in descending order
sorted_features = sorted(information_gain.items(), key=lambda x: x[1], reverse=T

# Get the Information Gain for the best feature (first item in sorted_features)
best_feature, best_ig = sorted_features[0]

# Print the Information Gain for the best feature
print(f"Best Feature for Splitting based on Information Gain: {best_feature}")
```

```
Best Feature for Splitting based on Information Gain: ADT
```

Calculations of Information Gain for column 'Direction' :

Entropy of dataset

Here 0 = Low + 1 = High PM 2.5 Values.

$H(S) = -p(0) * \log_2(p(0)) - p(1) \times \log_2(p(1))$

$= -\frac{91}{97} \times \log_2\left(\frac{91}{97}\right) - \frac{6}{97} \times \log_2\left(\frac{6}{97}\right)$

$= -(-0.026) - (-0.075)$

$= 0.101.$

$\boxed{\therefore H(S) = 0.101}$

IG for Direction.

Categorical values = N, S, E, W, T (For two-way traffic)

H (Direction = N)

$= -\left(\frac{12}{12}\right) \times \log\left(\frac{12}{12}\right) - 0 = 0.$

H (Direction = S)

$= -\left(\frac{12}{13}\right) \times \log\left(\frac{12}{13}\right) - \left(\frac{1}{13}\right) \times \log\left(\frac{1}{13}\right)$

$= -(-0.032) - (-0.086)$

$= 0.118.$

H (Direction = E) =
$= -\left(\frac{14}{15}\right) \times \log\left(\frac{14}{15}\right) - \left(\frac{1}{15}\right) \times \log\left(\frac{1}{15}\right)$
$= -(-0.028) - (-0.078)$
$= 0.106.$

H (Direction = W) =
$= -\left(\frac{15}{15}\right) \times \log\left(\frac{15}{15}\right) - 0$
$= 0$

H (Direction = T) =
$= -\left(\frac{38}{42}\right) \times \log\left(\frac{38}{42}\right) - \left(\frac{4}{42}\right) \times \log\left(\frac{4}{42}\right)$
$= -(-0.039) - (-0.097)$
$= 0.136.$

Average Entropy Information for Direction -

I (Direction) = $P(N) \times H$ (Direction = N) +
$P(S) \times H$ (Direction = S) +
$P(E) \times H$ (Direction = E) +
$P(W) \times H$ (Direction = W) +
$P(T) \times H$ (Direction = T)

$= \frac{12}{97} \times 0 + \frac{13}{97} \times 0.113 + \frac{15}{97} \times 0.106 + \frac{15}{97} \times 0 +$
$\left(\frac{42}{97}\right) \times 0.136$

$= 0.091$

Information Gain $= H(S) - I(\text{Direction})$

$= 0.101 - 0.091$

$= 0.01$

ii) Subtask 3.ii

In [10]:
```
#### Using Gini Index as decision criterion to select attribute to split on ####

import pandas as pd
from sklearn.tree import DecisionTreeClassifier

# Load dataset
data = pd.read_csv('merged_data.csv')

# Define the target variable
target_variable = 'pm25_category'

# Exclude 'Date' column from the feature set
```

```
X = data.drop(['Date', target_variable], axis=1)
y = data[target_variable]

# Initialize the DecisionTreeClassifier
clf = DecisionTreeClassifier(random_state=0)

# Create an empty dictionary to store Gini importances
gini_importance = {}

# Loop through each feature and fit the classifier
for feature in X.columns:
    X_feature = X[[feature]]
    clf.fit(X_feature, y)
    gini_importance[feature] = clf.tree_.impurity[0]

# Sort features by Gini impurity in descending order
sorted_features = sorted(gini_importance.items(), key=lambda x: x[1], reverse=Tr

# Get the best feature for splitting
best_feature, best_gini = sorted_features[0]

print(f"Best Feature for Splitting based on Gini Index:", best_feature)
```

```
Best Feature for Splitting based on Gini Index: ADT
```

Calculations of Gini Index for column 'Direction' :

* Gini Index for Direction

$$Gini = 1 - \sum_{i=1}^{n} (p_i)^2$$

$$Gini\ (Direction = N) = 1 - \left(\frac{12}{12}\right)^2 - \left(\frac{\cancel{12}\ 0}{12}\right)^2 = 0.$$

$$Gini\ (Direction = S) = 1 - \left(\frac{12}{13}\right)^2 - \left(\frac{1}{13}\right)^2 = 0.142.$$

$$Gini\ (Direction = E) = 1 - \left(\frac{14}{15}\right)^2 - \left(\frac{1}{15}\right)^2 = 0.124.$$

$$Gini\ (Direction = W) = 1 - \left(\frac{15}{15}\right)^2 - \left(\frac{0}{15}\right)^2 = 0.$$

$$Gini\ (Direction = T) = 1 - \left(\frac{38}{42}\right)^2 - \left(\frac{4}{42}\right)^2 = 0.172.$$

$$\text{Gini Index (Direction)} = P(N) \times \text{Gini}(N) + P(S) \times \text{Gini}(S) + P(E) \times \text{Gini}(E) + P(W) \times \text{Gini}(W) + P(T) \times \text{Gini}(T)$$

$$= \frac{12}{97} \times 0 + \frac{13}{97} \times 0.142 + \frac{15}{97} \times 0.124 + \frac{15}{97} \times 0 + \frac{42}{97} \times 0.172.$$

$$= 0.113.$$

$$\boxed{\therefore \text{Gini index (Direction)} = 0.113.}$$

iii) Subtask 3.iii

In [11]:
```
#### Load and Preprocess Data to be used by both Classifiers i.e. IG and Gini In
import pandas as pd
from sklearn.model_selection import train_test_split

# Load dataset
data = pd.read_csv('merged_data.csv')

# Convert the 'Date' column to datetime format
data['Date'] = pd.to_datetime(data['Date'], errors='coerce')

# Extract year, month, and day from the 'Date' column
data['Year'] = data['Date'].dt.year
```

```
data['Month'] = data['Date'].dt.month
data['Day'] = data['Date'].dt.day

# Drop the original 'Date' column
data = data.drop(columns=['Date'])

# Save the modified dataset back to 'merged_data.csv'
data.to_csv('merged_data.csv', index=False)

# Separate the features (X) and target (y) variable
X = data.drop(columns=['pm25_category'])  # Features
y = data['pm25_category']  # Target variable

# Split the dataset into a training set and a test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_
```

a. Decision tree using IG with default parameters.

In [12]:

```
#### Decision Tree using IG with default parameters ####

from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

# Create a DecisionTreeClassifier with 'entropy' as the criterion
clf_info_gain = DecisionTreeClassifier(criterion='entropy')

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

# Make predictions on the test data
y_pred_info_gain = clf_info_gain.predict(X_test)

# Evaluate the model's accuracy
accuracy_info_gain = accuracy_score(y_test, y_pred_info_gain)
print("Information Gain Accuracy:", accuracy_info_gain)
```

```
Information Gain Accuracy: 0.9
```

b. Decision tree using Gini Index with default parameters.

In [13]:

```
#### Decision Tree using Gini Index with default parameters ####

from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

# Create a DecisionTreeClassifier with 'gini' as the criterion
clf_gini = DecisionTreeClassifier(criterion='gini')

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

# Make predictions on the test data
y_pred_gini = clf_gini.predict(X_test)

# Evaluate the model's accuracy
accuracy_gini = accuracy_score(y_test, y_pred_gini)
print("Gini Index Accuracy:", accuracy_gini)
```

```
Gini Index Accuracy: 0.9
```

c. Confusion Matrix to obtain accuracy, precision, recall, specificity, and f-measure

In [14]:
```
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, r

# Create a confusion matrix for the Information Gain classifier
confusion_matrix_info_gain = confusion_matrix(y_test, y_pred_info_gain)

# Create a confusion matrix for the Gini Index classifier
confusion_matrix_gini = confusion_matrix(y_test, y_pred_gini)

# Define a function to calculate specificity
def specificity_score(confusion_matrix):
    tn, fp, fn, tp = confusion_matrix.ravel()
    specificity = tn / (tn + fp)
    return specificity

# Calculate accuracy, precision, recall, specificity, and f-measure for both cla
accuracy_info_gain = accuracy_score(y_test, y_pred_info_gain)
precision_info_gain = precision_score(y_test, y_pred_info_gain, pos_label=0)
recall_info_gain = recall_score(y_test, y_pred_info_gain, pos_label=0)
specificity_info_gain = specificity_score(confusion_matrix_info_gain)
f1_score_info_gain = f1_score(y_test, y_pred_info_gain, pos_label=0)

accuracy_gini = accuracy_score(y_test, y_pred_gini)
precision_gini = precision_score(y_test, y_pred_gini, pos_label=0)
recall_gini = recall_score(y_test, y_pred_gini, pos_label=0)
specificity_gini = specificity_score(confusion_matrix_gini)
f1_score_gini = f1_score(y_test, y_pred_gini, pos_label=0)

# Display the results
print("Information Gain Classifier:")
print("Accuracy:", accuracy_info_gain)
print("Precision:", precision_info_gain)
print("Recall:", recall_info_gain)
print("Specificity:", specificity_info_gain)
print("F-measure:", f1_score_info_gain)

print("\nGini Index Classifier:")
print("Accuracy:", accuracy_gini)
print("Precision:", precision_gini)
print("Recall:", recall_gini)
print("Specificity:", specificity_gini)
print("F-measure:", f1_score_gini)
```

```
Information Gain Classifier:
Accuracy: 0.9
Precision: 0.9
Recall: 1.0
Specificity: 1.0
F-measure: 0.9473684210526316

Gini Index Classifier:
Accuracy: 0.9
Precision: 0.9
Recall: 1.0
Specificity: 1.0
F-measure: 0.9473684210526316
```

Since both my classifiers are giving same results the reason behind this may be that my dataset is highly balanced and well-separated leading to same decision boundaries. However, after observing the dataset I thing the better splitting criteria would be Information Gain as it measures the reduction in uncertainty, making it suitable for feature selection. It also works well when the dataset is balanced and there is no severe class imbalance. Moreover, since decision tree interpretability is a priority, and Information Gain tends to create more balanced trees [11].

d. Optimal max_depth, min_values_split, or min_values_leaf for model with 5-fold cross validation.

In [15]:
```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import f1_score, make_scorer
from sklearn.model_selection import GridSearchCV

# Load dataset
data = pd.read_csv('merged_data.csv')

# Separate the features (X) and target (y) variable
X = data.drop(columns=['pm25_category'])  # Features
y = data['pm25_category']  # Target variable

# Define the parameter grid to search over
param_grid = {
    'max_depth': [1, 2, 4],  # You can specify different values here
    'min_samples_split': [2, 5, 10],  # You can specify different values here
    'min_samples_leaf': [1, 2, 4]  # You can specify different values here
}

# Create a DecisionTreeClassifier
clf = DecisionTreeClassifier(random_state=42)

# Create a custom scoring function (F1-score)
scorer = make_scorer(f1_score, pos_label=0)

# Perform grid search with 5-fold cross-validation
grid_search = GridSearchCV(estimator=clf, param_grid=param_grid, scoring=scorer,
grid_search.fit(X, y)

# Get the best hyperparameters
best_max_depth = grid_search.best_params_['max_depth']
best_min_samples_split = grid_search.best_params_['min_samples_split']
best_min_samples_leaf = grid_search.best_params_['min_samples_leaf']

print("Best max_depth:", best_max_depth)
print("Best min_samples_split:", best_min_samples_split)
print("Best min_samples_leaf:", best_min_samples_leaf)
```

```
Best max_depth: 1
Best min_samples_split: 2
Best min_samples_leaf: 1
```

iv) Subtask 3.iv

In [16]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
```

```python
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_sc

# Load dataset
data = pd.read_csv('merged_data.csv')

# Separate the features (X) and target (y) variable
X = data.drop(columns=['pm25_category'])  # Features (excluding 'Date' column)
y = data['pm25_category']  # Target variable

# Split the data into training and testing sets (e.g., 80% training, 20% testing
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_

# Create a Random Forest classifier
rf_classifier = RandomForestClassifier(random_state=42)

# Train the Random Forest model
rf_classifier.fit(X_train, y_train)

# Make predictions
y_pred_rf = rf_classifier.predict(X_test)

# Evaluate the Random Forest model
accuracy_rf = accuracy_score(y_test, y_pred_rf)
precision_rf = precision_score(y_test, y_pred_rf, pos_label=0)
recall_rf = recall_score(y_test, y_pred_rf, pos_label=0)
f1_score_rf = f1_score(y_test, y_pred_rf, pos_label=0)
confusion_matrix_rf = confusion_matrix(y_test, y_pred_rf)

print("Random Forest:")
print("Accuracy:", accuracy_rf)
print("Precision:", precision_rf)
print("Recall:", recall_rf)
print("F1-score:", f1_score_rf)
```

```
Random Forest:
Accuracy: 0.9
Precision: 0.9
Recall: 1.0
F1-score: 0.9473684210526316
```

The accuracy of random forest is identical to decision tree because of the dataset being highly balanced and seperated. Though my experience working with dataset has helped me deduce that Random Forest is better for classification problems. Due to its ability to mitigate overfitting and improve model robustness. It works by constructing multiple decision trees and combining their predictions, resulting in improved accuracy and reduced variance. Random Forest also provides a measure of feature importance, aiding in feature selection. Moreover, it can handle large datasets with high dimensionality effectively. These attributes make Random Forest a preferred choice in various real-world applications where predictive accuracy and generalization are paramount [12].

References:

1. NoVA Scotia Provincial Ambient Fine Particulate Matter (PM2.5) Hourly Data Halifax BAM/T640 | Open Data | Nova Scotia. (2022, November 23).

https://data.novascotia.ca/Environment-and-Energy/Nova-Scotia-Provincial-Ambient-Fine-Particulate-Ma/ddk3-mz42

2. Traffic volumes - Provincial Highway System | Open Data | Nova Scotia. (2023, April 5). https://data.novascotia.ca/Roads-Driving-and-Transport/Traffic-Volumes-Provincial-Highway-System/8524-ec3n
3. GeeksforGeeks. (2021). Python Pandas dataframe.ffill. GeeksforGeeks. https://www.geeksforgeeks.org/python-pandas-dataframe-ffill/
4. How to Group Data by Date using Python Pandas and Datetime | Saturn Cloud Blog. (2023, September 8). https://saturncloud.io/blog/how-to-group-data-by-date-using-python-pandas-and-datetime/
5. Eddie. (2023). How to handle missing data in Python? [Explained in 5 easy steps]. Analytics Vidhya. https://www.analyticsvidhya.com/blog/2021/05/dealing-with-missing-values-in-python-a-complete-guide/
6. Verma, J. (2023). How to Normalize Data Using scikit-learn in Python. DigitalOcean. https://www.digitalocean.com/community/tutorials/normalize-data-in-python
7. Sole. (2022, July 4). Data discretization in machine learning. Train in Data Blog. https://www.blog.trainindata.com/data-discretization-in-machine-learning/
8. Visualizing distributions of data — seaborn 0.13.0 documentation. (n.d.). https://seaborn.pydata.org/tutorial/distributions.html
9. Brownlee, J. (2020). Why One-Hot encode data in machine learning? MachineLearningMastery.com. https://machinelearningmastery.com/why-one-hot-encode-data-in-machine-learning/
10. Saxena, S. (2023). What are Categorical Data Encoding Methods | Binary Encoding. Analytics Vidhya. https://www.analyticsvidhya.com/blog/2020/08/types-of-categorical-data-encoding/
11. Dash, S. (2022, November 3). Decision Trees explained — entropy, information gain, Gini index, CCP pruning. Medium. https://towardsdatascience.com/decision-trees-explained-entropy-information-gain-gini-index-ccp-pruning-4d78070db36c
12. Talari, S. (n.d.). Random Forest vs Decision Tree: Key Differences - KDnuggets. KDnuggets. https://www.kdnuggets.com/2022/02/random-forest-decision-tree-key-differences.html#:~:text=The%20critical%20difference%20between%20the,work%20accc