rowID	hpwren_timestamp	air_pressu	air_temp	avg_wind_	avg_wind_	max_wind	max_wind	min_wind_	min_wind_	rain_accur	rain_durat	relative_hu

Apply hierarchical clustering algorithm for the dataset Minute_weather (Attached).

 Draw the Dendrogram for varying number of clusters

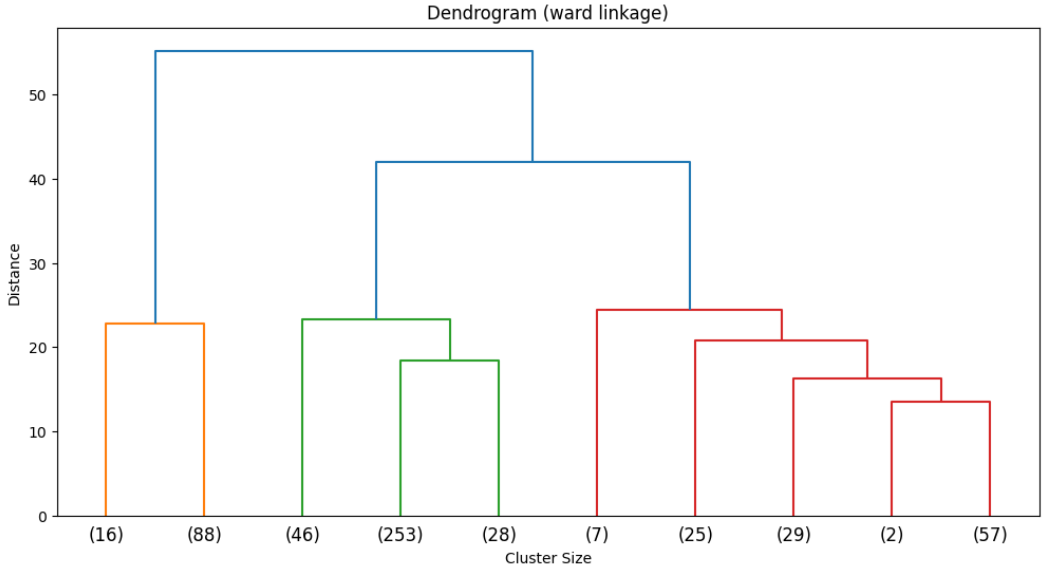
 Apply different linkage methods and compare the results

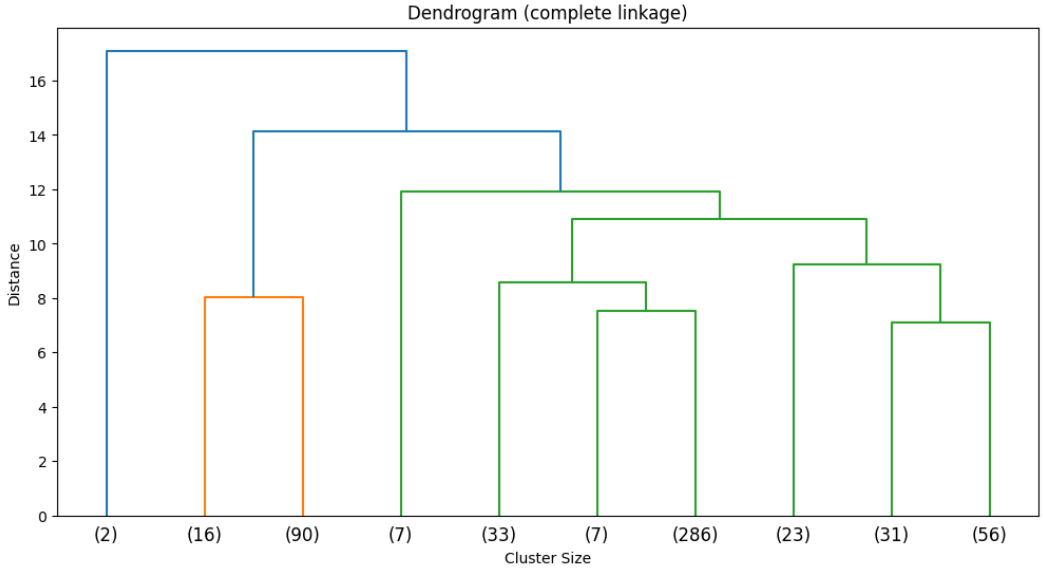
 Evaluate the performance of hierarchical clustering algorithm

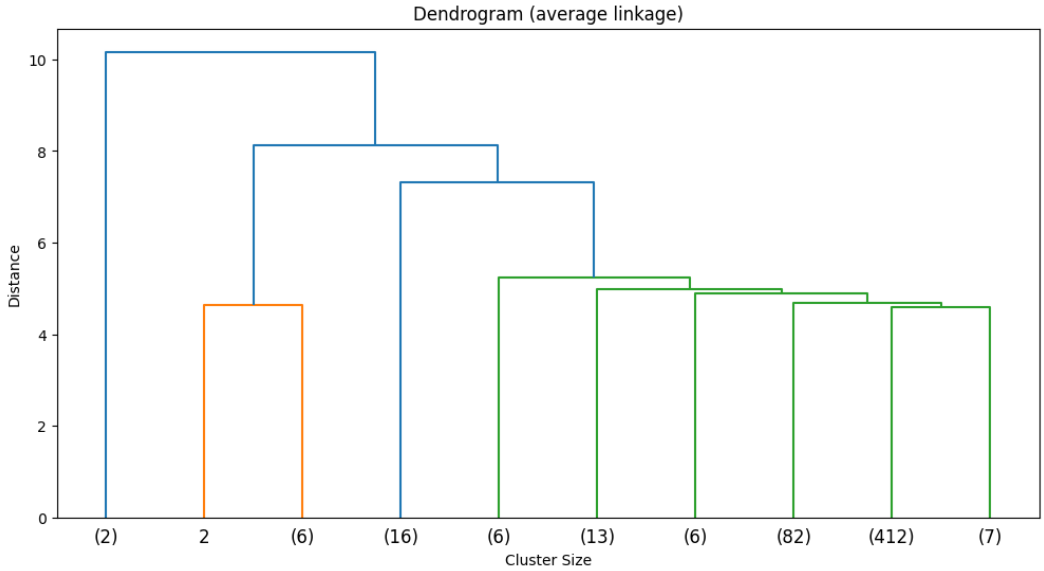
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.cluster.hierarchy import dendrogram, linkage, fcluster
from sklearn.preprocessing import StandardScaler

# Load the Minute_weather dataset
data = pd.read_csv('/content/minute_weather_HC.csv')


# Select relevant columns for clustering
selected_columns = [
    'air_pressure',
    'air_temp',
    'avg_wind_direction',
    'avg_wind_speed',
    'max_wind_direction',
    'max_wind_speed',
    'min_wind_direction',
    'min_wind_speed',
    'rain_accumulation',
    'rain_duration',
    'relative_humidity'
]


X = data[selected_columns]

# Standardize the data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)



X = data[selected_columns]

# Handle missing values (NaN) in the data
imputer = SimpleImputer(strategy='mean')
X = imputer.fit_transform(X)


# Handle infinite values in the data
X[np.isinf(X)] = np.nan
X = imputer.fit_transform(X)

# Standardize the data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

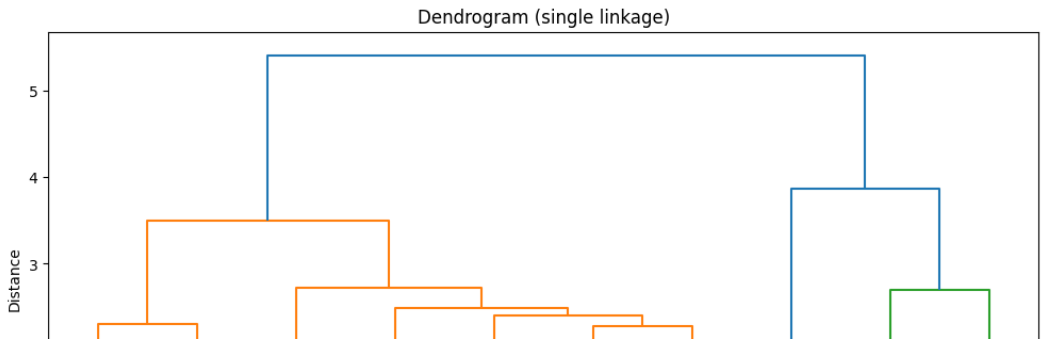
Draw the Dendrogram for varying number of clusters

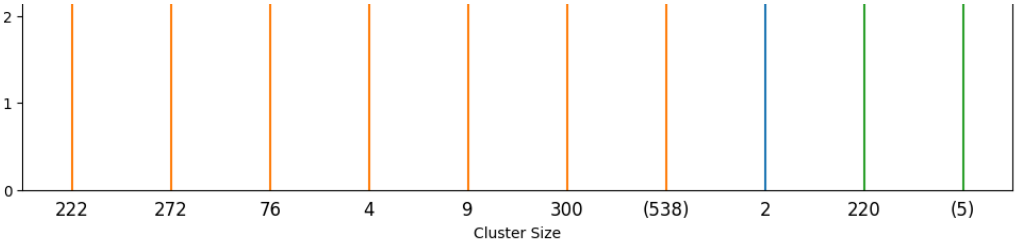
```
# Hierarchical clustering with different linkage methods
linkage_methods = ['ward', 'complete', 'average', 'single']

for method in linkage_methods:
    # Perform hierarchical clustering
    Z = linkage(X_scaled, method=method)

    # Plot dendrogram for varying numbers of clusters
    plt.figure(figsize=(12, 6))
    plt.title(f'Dendrogram ({method} linkage)')
    dendrogram(Z, truncate_mode='lastp', p=10)
    plt.xlabel('Cluster Size')
    plt.ylabel('Distance')
    plt.show()
```


Dendrogram (ward linkage)
Distance
Cluster Size
(16)
(88)
(46)
(253)
(28)
(7)
(25)
(29)
(2)
(57)


Dendrogram (complete linkage)
Distance
Cluster Size
(2)
(16)
(90)
(7)
(33)
(7)
(286)
(23)
(31)
(56)


Dendrogram (average linkage)
Distance
Cluster Size
(2)
2
(6)
(16)
(6)
(13)
(6)
(82)
(412)
(7)


Dendrogram (single linkage)
Distance



Evaluate the performance of hierarchical clustering

```
# Choose the number of clusters based on the dendrogram and linkage method
selected_linkage = 'ward'  # Choose the linkage method
num_clusters = 3  # Choose the number of clusters
```

Perform clustering with the selected parameters

```
clusters = fcluster(Z, num_clusters, criterion='maxclust')

# Add the cluster labels to the original dataset
data['cluster'] = clusters
```

Evaluate the performance of clustering, e.g., by examining cluster statistics or visualizations

```
# For example, you can print the mean values for each cluster
cluster_means = data.groupby('cluster')[selected_columns].mean()
print(cluster_means)
```

```
             air_pressure   air_temp  avg_wind_direction  avg_wind_speed  \
cluster
1          911.661905  64.946735          109.098639        1.565646
2          911.400000  66.320000           52.000000        0.350000
3          912.300000  64.220000           77.000000        0.700000

         max_wind_direction  max_wind_speed  min_wind_direction  \
cluster
1                128.346939        1.960204           89.248299
2                112.500000        0.566667          336.000000
3                143.000000        1.200000          324.000000
```

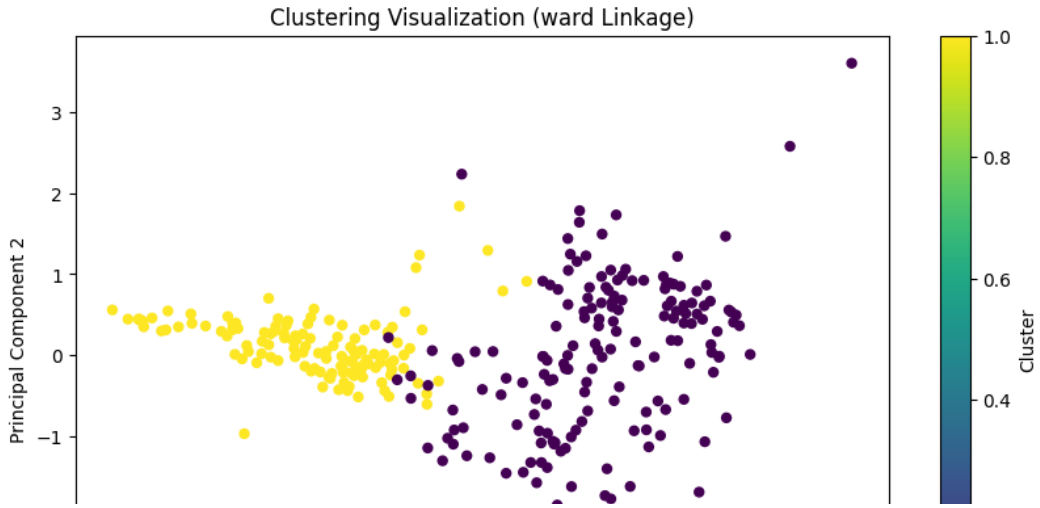
```
         min_wind_speed  rain_accumulation  rain_duration  relative_humidity
cluster
1              1.177211                0.0            0.0          50.479252
2              0.116667                0.0            0.0          41.100000
3              0.300000                0.0            0.0          43.000000
```

Evaluate performance using silhouette score

```
# Select relevant columns for clustering
X = data[["air_pressure", "air_temp", "avg_wind_direction", "avg_wind_speed", "relative_humidity"]]

# Drop rows with missing values
X.dropna(inplace=True)

# Standardize the data
scaler = StandardScaler()
X_std = scaler.fit_transform(X)


best_score = -1
best_method = ""
linkage_methods = ["single", "complete", "average", "ward"]

for method in linkage_methods:
    cluster_model = AgglomerativeClustering(n_clusters=2, linkage=method)
    cluster_labels = cluster_model.fit_predict(X_std)
    score = silhouette_score(X_std, cluster_labels)
    print(f"Silhouette Score ({method} Linkage): {score}")
    if score > best_score:
        best_score = score
        best_method = method

print(f"The best linkage method is '{best_method}' with a silhouette score of {best_score}")


# Calculate silhouette score for the visualization
silhouette_avg = silhouette_score(X_std, cluster_labels)
print(f"Silhouette Score ({best_method} Linkage): {silhouette_avg}")
```

```
Silhouette Score (single Linkage): 0.15791042948656703
Silhouette Score (complete Linkage): 0.42348146860994645
Silhouette Score (average Linkage): 0.42348146860994645
Silhouette Score (ward Linkage): 0.4437383213679116
The best linkage method is 'ward' with a silhouette score of 0.4437383213679116
Silhouette Score (ward Linkage): 0.4437383213679116
<ipython-input-33-44721a8e4089>:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus
  X.dropna(inplace=True)
```

```
# Apply hierarchical clustering with the best linkage method
cluster_model = AgglomerativeClustering(n_clusters=2, linkage=best_method)
cluster_labels = cluster_model.fit_predict(X_std)

# Reduce dimensionality using PCA for visualization
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_std)


# Create a scatter plot to visualize the clusters
plt.figure(figsize=(10, 6))
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=cluster_labels, cmap='viridis', marker='o', s=25)
plt.title(f"Clustering Visualization ({best_method} Linkage)")
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.colorbar(label="Cluster")
plt.show()
```




```
# Calculate silhouette score for the visualization
silhouette_avg = silhouette_score(X_std, cluster_labels)
print(f"Silhouette Score ({best_method} Linkage): {silhouette_avg}")
```

```
Silhouette Score (ward Linkage): 0.4437383213679116
```

Question-2

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report
from sklearn.impute import SimpleImputer

# Load the dataset
data = pd.read_csv('/content/customer_data_NN.csv')
```

Exploratory Data Analysis

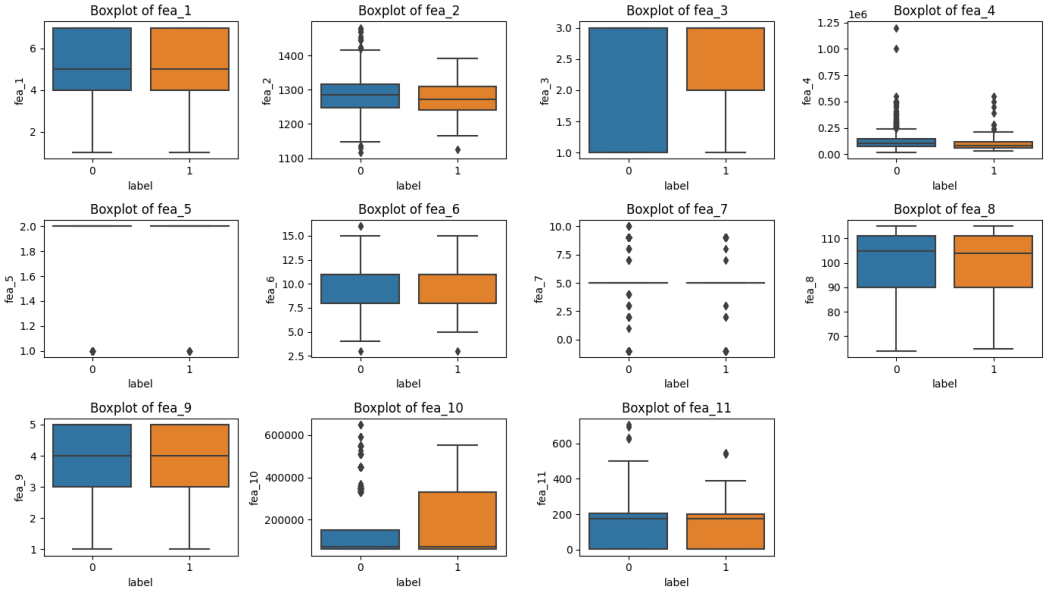
```
# Check for missing values
print("Missing Values:")
print(data.isnull().sum())
```

```
Missing Values:
label       0
id          0
fea_1       0
fea_2     149
fea_3       0
fea_4       0
fea_5       0
fea_6       0
fea_7       0
fea_8       0
fea_9       0
fea_10      0
fea_11      0
dtype: int64
```

```
# Handle missing values
imputer = SimpleImputer(strategy='mean')
data_filled = pd.DataFrame(imputer.fit_transform(data), columns=data.columns)
```

```
# Split data into features (X) and labels (y)
X = data_filled.drop('label', axis=1)
y = data_filled['label']
```

```
# Boxplots to visualize feature distributions by label
feature_columns = data.columns[2:]  # Exclude 'id' and 'label'
plt.figure(figsize=(14, 8))
for i, feature in enumerate(feature_columns, 1):
    plt.subplot(3, 4, i)
    sns.boxplot(x='label', y=feature, data=data)
    plt.title(f'Boxplot of {feature}')
plt.tight_layout()
plt.show()
```

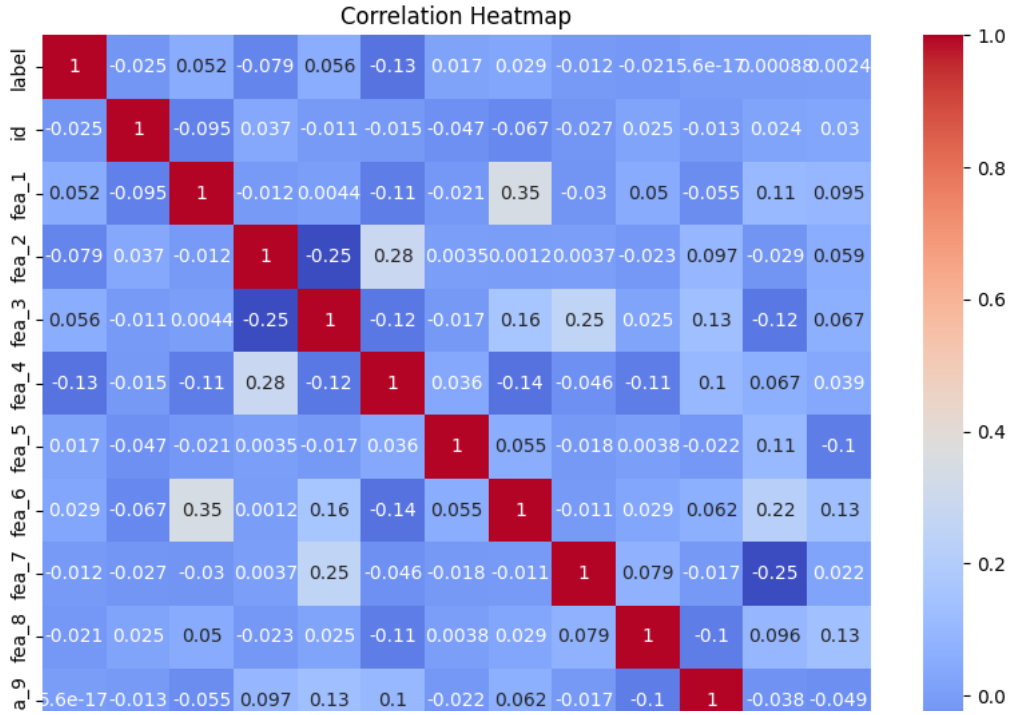



```
# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

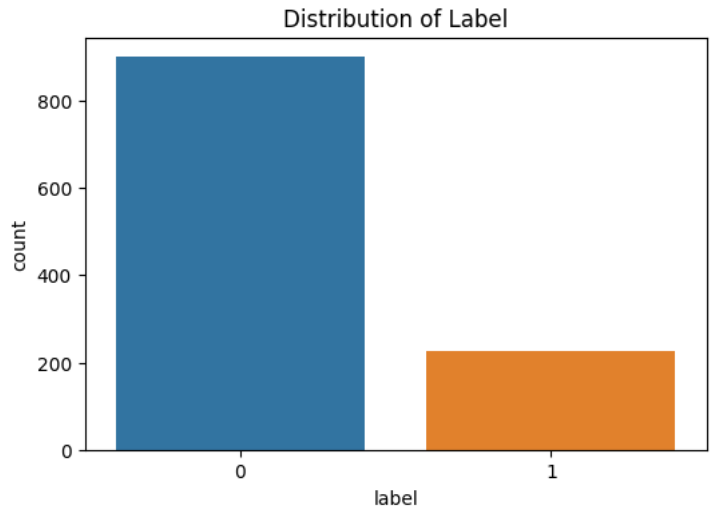
```
# Create and fit Decision Tree Classifier
clf = DecisionTreeClassifier(random_state=42)
clf.fit(X_train, y_train)
```

```
DecisionTreeClassifier
DecisionTreeClassifier(random_state=42)
```

```
# Correlation heatmap
corr_matrix = data.corr()
plt.figure(figsize=(10, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm')
plt.title('Correlation Heatmap')
plt.show()
```



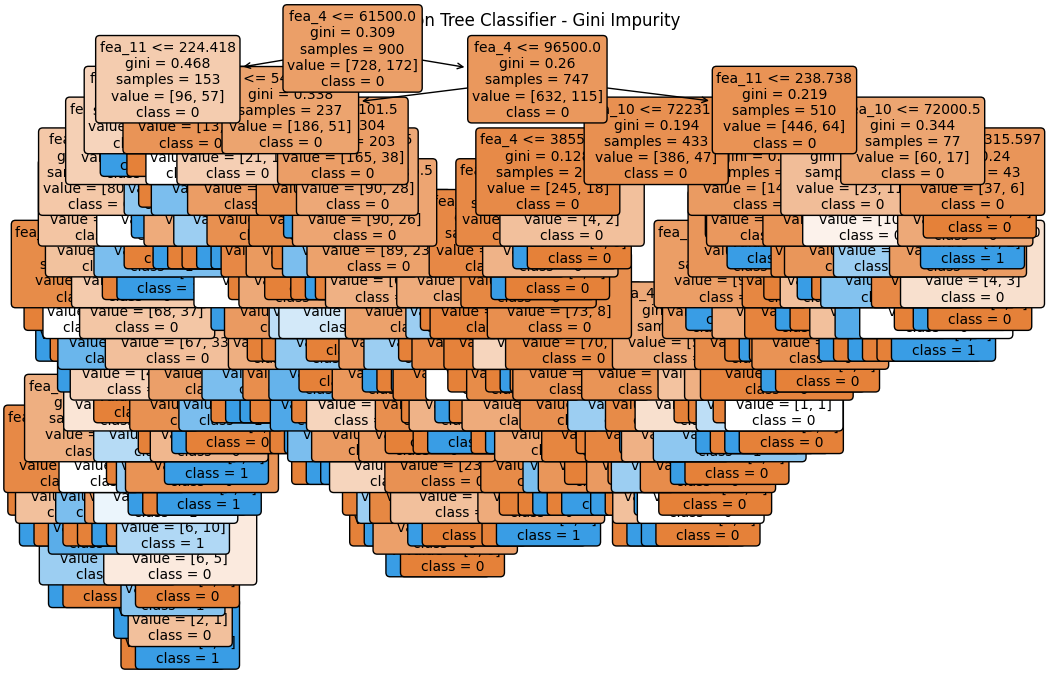

```
# Distribution of the target variable 'label'
plt.figure(figsize=(6, 4))
sns.countplot(x='label', data=data)
plt.title('Distribution of Label')
plt.show()
```




```
# Evaluate the model with various attribute selection measures
def evaluate_classifier(clf, X_test, y_test):
    y_pred = clf.predict(X_test)
    print("Classification Report:")
    print(classification_report(y_test, y_pred))

print("Decision Tree Classifier - Gini Impurity:")
evaluate_classifier(clf, X_test, y_test)
```

```
Decision Tree Classifier - Gini Impurity:
Classification Report:
              precision    recall  f1-score   support

         0.0       0.78      0.78      0.78       172
         1.0       0.29      0.28      0.29        53

    accuracy                           0.67       225
   macro avg       0.53      0.53      0.53       225
weighted avg       0.66      0.67      0.67       225
```

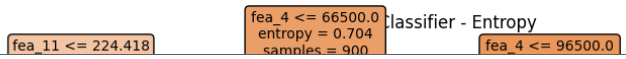
```
from sklearn.tree import plot_tree
plt.figure(figsize=(12, 8))
plot_tree(clf, filled=True, feature_names=X.columns, class_names=['0', '1'], rounded=True, fontsize=10)
plt.title('Decision Tree Classifier - Gini Impurity')
plt.show()
```




```
# You can also evaluate using other attribute selection measures like 'entropy' or 'max_depth'.
clf_entropy = DecisionTreeClassifier(criterion='entropy', random_state=42)
clf_entropy.fit(X_train, y_train)

print("Decision Tree Classifier - Entropy:")
evaluate_classifier(clf_entropy, X_test, y_test)
```

```
Decision Tree Classifier - Entropy:
Classification Report:
              precision    recall  f1-score   support

         0.0       0.79      0.81      0.80       172
         1.0       0.31      0.28      0.30        53

    accuracy                           0.68       225
   macro avg       0.55      0.55      0.55       225
weighted avg       0.67      0.68      0.68       225
```

```
import matplotlib.pyplot as plt
from sklearn.tree import plot_tree

# Visualize Decision Tree Classifier - Entropy
plt.figure(figsize=(12, 8))
plot_tree(clf_entropy, filled=True, feature_names=X.columns, class_names=['0', '1'], rounded=True, fontsize=10)
plt.title('Decision Tree Classifier - Entropy')
plt.show()
```



✓ 9s completed at 12:10 PM