

**Lab 1 report**

**Submitted to:**

Dr. Raihan Ul Islam

**Submitted by:**

Md. Arnob Tanvir

**ID:** 2020-1-60-220

**Date of submission:** 07/09/2024

**EDA:**

X.describe(include='all').T

**Output:**

count unique top freq

cap-shape 8124 6 x 3656

cap-surface 8124 4 y 3244

cap-color 8124 10 n 2284

bruises 8124 2 f 4748

odor 8124 9 n 3528

gill-attachment 8124 2 f 7914

gill-spacing 8124 2 c 6812

gill-size 8124 2 b 5612

gill-color 8124 12 b 1728

stalk-shape 8124 2 t 4608

stalk-root 8124 4 b 6256

stalk-surface-above-ring 8124 4 s 5176

stalk-surface-below-ring 8124 4 s 4936

stalk-color-above-ring 8124 9 w 4464

stalk-color-below-ring 8124 9 w 4384

veil-color 8124 4 w 7924

ring-number 8124 3 o 7488

ring-type 8124 5 p 3968

spore-print-color 8124 9 w 2388

population 8124 6 v 4040

habitat 8124 7 d 3148

Seperate Numerical and categorical variables:

cat\_cols= X.select\_dtypes(include=['object']).columns

num\_cols= X.select\_dtypes(include=np.number).columns.tolist()

print("Categorical Variables:")

print(cat\_cols)

print("Numerical Variables:")

print(num\_cols)

**output:**

Categorical Variables:

Index(['cap-shape', 'cap-surface', 'cap-color', 'bruises', 'odor',

'gill-attachment', 'gill-spacing', 'gill-size', 'gill-color',

'stalk-shape', 'stalk-root', 'stalk-surface-above-ring',

'stalk-surface-below-ring', 'stalk-color-above-ring',

'stalk-color-below-ring', 'veil-color', 'ring-number', 'ring-type',

'spore-print-color', 'population', 'habitat'],

dtype='object')

Numerical Variables:

[]

**Frequency Count and Distribution:**

for col in cat\_cols:

print(f"--- {col} ---")

print(X[col].value\_counts())

plt.figure(figsize=(10, 5))

sns.countplot(y=X[col], order=X[col].value\_counts().index, palette='viridis')

plt.title(f'Distribution of {col}')

plt.show()

**Output:**

--- cap-shape ---

cap-shape

x 3656

f 3152

k 828

b 452

s 32

c 4

Name: count, dtype: int64

--- cap-surface ---

cap-surface

y 3244

s 2556

f 2320

g 4

Name: count, dtype: int64

--- cap-color ---

cap-color

n 2284

g 1840

e 1500

y 1072

w 1040

b 168

p 144

c 44

u 16

r 16

Name: count, dtype: int64

--- bruises ---

bruises

f 4748

t 3376

Name: count, dtype: int64

--- odor ---

odor

n 3528

f 2160

y 576

s 576

a 400

l 400

p 256

c 192

m 36

Name: count, dtype: int64

--- gill-attachment ---

gill-attachment

f 7914

a 210

Name: count, dtype: int64

--- gill-spacing ---

gill-spacing

c 6812

w 1312

Name: count, dtype: int64

--- gill-size ---

gill-size

b 5612

n 2512

Name: count, dtype: int64

--- gill-color ---

gill-color

b 1728

p 1492

w 1202

n 1048

g 752

h 732

u 492

k 408

e 96

y 86

o 64

r 24

Name: count, dtype: int64

--- stalk-root ---

stalk-root

b 6256

e 1120

c 556

r 192

Name: count, dtype: int64

--- stalk-surface-above-ring ---

stalk-surface-above-ring

s 5176

k 2372

f 552

y 24

Name: count, dtype: int64

--- stalk-surface-below-ring ---

stalk-surface-below-ring

s 4936

k 2304

f 600

y 284

Name: count, dtype: int64

--- stalk-color-above-ring ---

stalk-color-above-ring

w 4464

p 1872

g 576

n 448

b 432

o 192

e 96

c 36

y 8

Name: count, dtype: int64

--- stalk-color-below-ring ---

stalk-color-below-ring

w 4384

p 1872

g 576

n 512

b 432

o 192

e 96

c 36

y 24

Name: count, dtype: int64

--- veil-color ---

veil-color

w 7924

n 96

o 96

y 8

Name: count, dtype: int64

--- ring-number ---

ring-number

o 7488

t 600

n 36

Name: count, dtype: int64

--- ring-type ---

ring-type

p 3968

e 2776

l 1296

f 48

n 36

Name: count, dtype: int64

--- spore-print-color ---

spore-print-color

w 2388

n 1968

k 1872

h 1632

r 72

u 48

o 48

y 48

b 48

--- population ---

population

v 4040

y 1712

s 1248

n 400

a 384

c 340

Name: count, dtype: int64

--- habitat ---

habitat

d 3148

g 2148

p 1144

l 832

u 368

m 292

w 192

Name: count, dtype: int64

**Combine feature and target:**

df = X.copy()

df['class'] = y

**Relationship with Target variable:**

for col in cat\_cols:

plt.figure(figsize=(10, 5))

sns.countplot(data=df, x='class', hue=col, palette='viridis')

plt.title(f'Relationship between {col} and Target')

plt.show()

**Exploratory Data Analysis (EDA)**

**1. Dataset Description:**

- The mushroom dataset contains 8124 samples with various categorical features such as `cap-shape`, `cap-surface`, `cap-color`, `bruises`, `odor`, and many others.

- All variables are categorical; there are no numerical variables in this dataset.

**2. Missing Values:**

- The `stalk-root` feature had missing values, which were filled using the mode of the feature.

**3. Distribution of Features:**

- The features show a variety of distributions, with some having a few dominant categories (e.g., `cap-shape` with `x` being the most frequent) and others being more evenly distributed.

- Each feature's distribution was visualized using count plots.

**4. Relationship with Target Variable:**

- The target variable (`class`, which indicates whether a mushroom is edible or poisonous) was analyzed in relation to each feature.

- Count plots showed the relationship between the target variable and each feature.

**Applying Random forest for n\_estimator = 1, 50, 100, 150, 200, 250:**

n\_estimators\_list = [1, 50, 100, 150, 200, 250]

accuracy\_results = {}

precision\_results = {}

recall\_results = {}

f1\_results = {}

for n in n\_estimators\_list:

rf\_pipeline = Pipeline(steps=[

('preprocessor', onehotencoder),

('classifier', RandomForestClassifier(n\_estimators=n, random\_state=42))

])

rf\_pipeline.fit(X\_train, y\_train)

y\_pred = rf\_pipeline.predict(X\_test)

accuracy\_rf = accuracy\_score(y\_test, y\_pred)

prec\_rf = precision\_score(y\_test, y\_pred, average='macro')

rec\_rf = recall\_score(y\_test, y\_pred, average='macro')

f1\_rf = f1\_score(y\_test, y\_pred, average='macro')

# Print the results

print(f'Number of estimators: {n}')

print(f"Accuracy: {accuracy\_rf:.4f}")

print(f"Precision: {prec\_rf:.4f}")

print(f"Recall: {rec\_rf:.4f}")

print(f"F1-score: {f1\_rf:.4f}")

print(f"n\_estimators: {n}, Accuracy: {accuracy\_rf:.4f}")

accuracy\_results[n] = accuracy\_rf

precision\_results[n] = prec\_rf

recall\_results[n] = rec\_rf

f1\_results[n] = f1\_rf

**Output:**

Number of estimators: 1

Accuracy: 0.9963

Precision: 0.9963

Recall: 0.9963

F1-score: 0.9963

n\_estimators: 1, Accuracy: 0.9963

Number of estimators: 50

Accuracy: 1.0000

Precision: 1.0000

Recall: 1.0000

F1-score: 1.0000

n\_estimators: 50, Accuracy: 1.0000

Number of estimators: 100

Accuracy: 1.0000

Precision: 1.0000

Recall: 1.0000

F1-score: 1.0000

n\_estimators: 100, Accuracy: 1.0000

Number of estimators: 150

Accuracy: 1.0000

Precision: 1.0000

Recall: 1.0000

F1-score: 1.0000

n\_estimators: 150, Accuracy: 1.0000

Number of estimators: 200

Accuracy: 1.0000

Precision: 1.0000

Recall: 1.0000

F1-score: 1.0000

n\_estimators: 200, Accuracy: 1.0000

Number of estimators: 250

Accuracy: 1.0000

Precision: 1.0000

Recall: 1.0000

F1-score: 1.0000

n\_estimators: 250, Accuracy: 1.0000

**Plotting the graph:**

plt.figure(figsize=(10, 6))

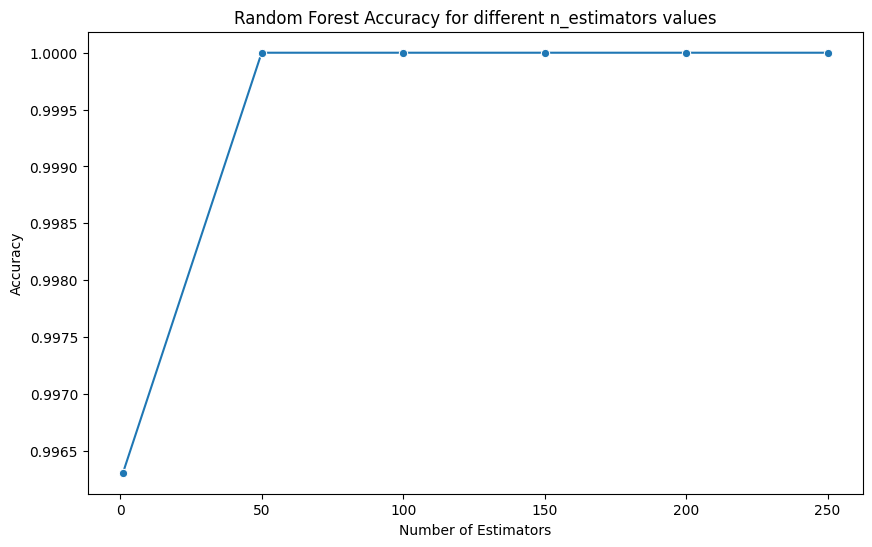
sns.lineplot(x=list(accuracy\_results.keys()), y=list(accuracy\_results.values()), marker='o')

plt.title('Random Forest Accuracy for different n\_estimators values')

plt.xlabel('Number of Estimators')

plt.ylabel('Accuracy')

plt.show()



**Applying Decision Tree**

dt = DecisionTreeClassifier(criterion='gini', max\_depth=3, random\_state=42)

dt.fit(X\_train, y\_train)

y\_pred = dt.predict(X\_test)

# Calculate metrics

acc\_dt = accuracy\_score(y\_test, y\_pred)

prec\_dt = precision\_score(y\_test, y\_pred, average='macro')

rec\_dt = recall\_score(y\_test, y\_pred, average='macro')

f1\_dt = f1\_score(y\_test, y\_pred, average='macro')

# Print the results

print(f"Accuracy: {acc\_dt:.2f}")

print(f"Precision: {prec\_dt:.2f}")

print(f"Recall: {rec\_dt:.2f}")

print(f"F1-score: {f1\_dt:.2f}")

**Output:**

Accuracy: 0.99

Precision: 0.99

Recall: 0.99

F1-score: 0.99

**Confusion matrix:**

# Plot the confusion matrix

cm = confusion\_matrix(y\_test, y\_pred)

plt.imshow(cm, cmap='Blues')

plt.xlabel('Predicted')

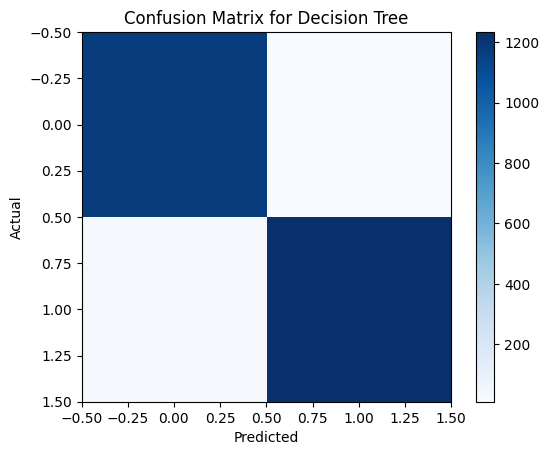
plt.ylabel('Actual')

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

plt.colorbar()

plt.show()

**Output:**



**Decision tree**

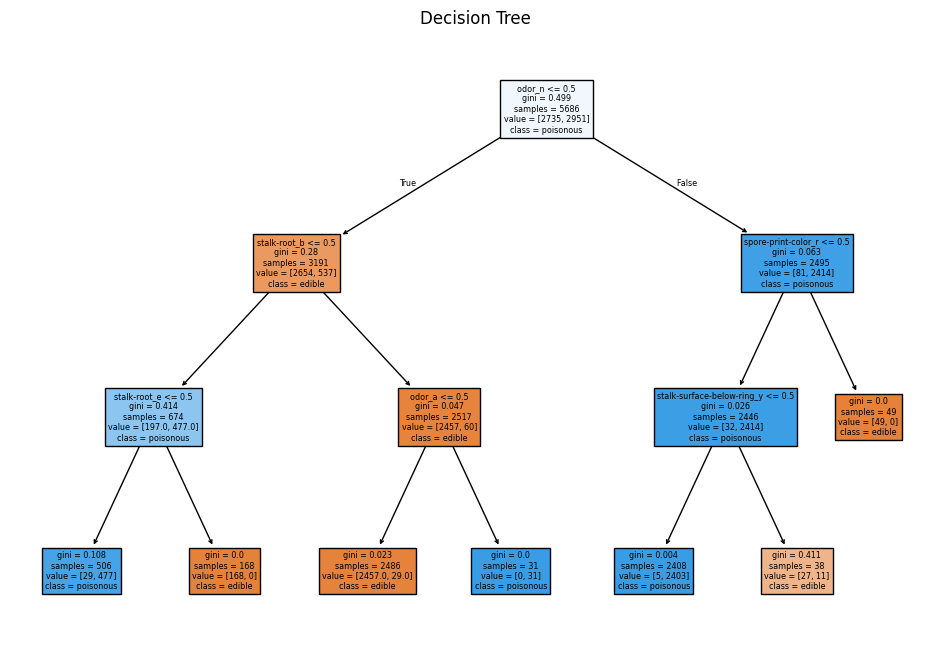
plt.figure(figsize=(12, 8))

plot\_tree(dt, feature\_names=X\_encoded.columns, class\_names=['edible', 'poisonous'], filled=True)

plt.title('Decision Tree')

plt.show()

**Output:**



**Comparison of random forest and decision tree:**

print(f"Decision Tree - Accuracy: {acc\_dt:.2f}, Precision: {prec\_dt:.2f}, Recall: {rec\_dt:.2f}, F1-score: {f1\_dt:.2f}")

print("Random forest: ")

for n in n\_estimators\_list:

print(f"estimated\_n = {n},")

print(f"Accuracy: {accuracy\_results[n]:.4f}, Precision: {precision\_results[n]:.2f}, Recall: {recall\_results[n]:.2f}, F1-score: {f1\_results[n]:.2f}")

**Output:**

Decision Tree - Accuracy: 0.99, Precision: 0.99, Recall: 0.99, F1-score: 0.99

Random forest:

estimated\_n = 1,

Accuracy: 0.9963, Precision: 1.00, Recall: 1.00, F1-score: 1.00

estimated\_n = 50,

Accuracy: 1.0000, Precision: 1.00, Recall: 1.00, F1-score: 1.00

estimated\_n = 100,

Accuracy: 1.0000, Precision: 1.00, Recall: 1.00, F1-score: 1.00

estimated\_n = 150,

Accuracy: 1.0000, Precision: 1.00, Recall: 1.00, F1-score: 1.00

estimated\_n = 200,

Accuracy: 1.0000, Precision: 1.00, Recall: 1.00, F1-score: 1.00

estimated\_n = 250,

Accuracy: 1.0000, Precision: 1.00, Recall: 1.00, F1-score: 1.00

**Model Training and Evaluation**

**Comparison:**

- The Decision Tree Classifier performed excellently with very high accuracy and other metrics.

- The Random Forest Classifier outperformed the Decision Tree, achieving perfect scores (100%) across all metrics with 50 or more estimators.

**Conclusion**

- Both models performed exceptionally well on the mushroom dataset.

- The Random Forest Classifier demonstrated superior performance, especially with 50 or more estimators, achieving perfect classification accuracy.

- The Decision Tree Classifier, while slightly less accurate, still provided high-quality predictions with nearly perfect accuracy.

- The mushroom dataset, being categorical and relatively simple, was well-suited for these tree-based models, resulting in high performance across all tested configurations.