2348441-lab-11

April 26, 2024

Lab Exercise 11 – – Ensemble Learning

Created by : Nileem Kaveramma C C | 2348441 Created DATE:26-04-2024 Edited Date: 26-04-2024

Aim:

This experiment aims to evaluate the efficacy of ensemble learning techniques in enhancing predictive performance compared to individual models. Through comparative analysis of various ensemble methods across different datasets, the study seeks to identify optimal approaches for diverse predictive modeling tasks. Key objectives include assessing the impact of ensemble size, base model diversity, and hyperparameter tuning on predictive accuracy. Ultimately, this research aims to provide actionable insights for leveraging ensemble learning to improve predictive modeling outcomes across various domains

DATA DESCRIPTION:

The dataset comprises patient data related to heart health, including attributes such as age, gender, chest pain type, resting blood pressure, serum cholesterol level, fasting blood sugar, resting electrocardiographic results, maximum heart rate achieved, exercise-induced angina presence, ST depression induced by exercise, peak exercise ST segment slope, number of major vessels colored by fluoroscopy, thalassemia type, and the presence of heart disease. With features covering a range of physiological indicators, this dataset is likely intended for heart disease diagnosis or prediction tasks, with the target variable indicating the presence or absence of heart disease.

[ ]:
```
from google.colab import drive
drive.mount('/content/drive')
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

```
Mounted at /content/drive
```

[ ]:
```
df=pd.read_csv('/content/archive (12).zip',encoding='latin-1')
df
```

[ ]:
```
   age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak  \
0   52    1   0       125   212    0        1      168      0      1.0
1   53    1   0       140   203    1        0      155      1      3.1
2   70    1   0       145   174    0        1      125      1      2.6
3   61    1   0       148   203    0        1      161      0      0.0
4   62    0   0       138   294    1        1      106      0      1.9
```

```
…     …    …   ..       …     …   …       …        …      …       …
1020  59   1   1       140   221  0        1       164     1       0.0
1021  60   1   0       125   258  0        0       141     1       2.8
1022  47   1   0       110   275  0        0       118     1       1.0
1023  50   0   0       110   254  0        0       159     0       0.0
1024  54   1   0       120   188  0        1       113     0       1.4

      slope  ca  thal  target
0         2   2     3       0
1         0   0     3       0
2         0   0     3       0
3         2   1     3       0
4         1   3     2       0
…       …  ..    …     …
1020      2   0     2       1
1021      1   1     3       0
1022      1   1     2       0
1023      2   0     2       1
1024      1   1     3       0

[1025 rows x 14 columns]
```

df.shape - attribute is used to get the dimensions of the DataFrame

[ ]:
```
df.shape
```

[ ]:
```
(1025, 14)
```

df.head() method is used to display the first few rows of a DataFrame

[ ]:
```
df.head()
```

[ ]:
```
   age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak  slope  \
0   52    1   0       125   212    0        1      168      0      1.0      2
1   53    1   0       140   203    1        0      155      1      3.1      0
2   70    1   0       145   174    0        1      125      1      2.6      0
3   61    1   0       148   203    0        1      161      0      0.0      2
4   62    0   0       138   294    1        1      106      0      1.9      1

   ca  thal  target
0   2     3       0
1   0     3       0
2   0     3       0
3   1     3       0
4   3     2       0
```

df.tail() method is used to display the last few rows of a DataFrame.

[ ]:
```
df.tail()
```

[ ]:
```
      age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak  \
1020   59    1   1       140   221    0        1      164      1      0.0
1021   60    1   0       125   258    0        0      141      1      2.8
1022   47    1   0       110   275    0        0      118      1      1.0
1023   50    0   0       110   254    0        0      159      0      0.0
1024   54    1   0       120   188    0        1      113      0      1.4

      slope  ca  thal  target
1020      2   0     2       1
1021      1   1     3       0
1022      1   1     2       0
1023      2   0     2       1
1024      1   1     3       0
```

df.columns attribute is used to retrieve the column labels or names of the DataFrame.

[ ]:
```
df.columns
```

[ ]:
```
Index(['age', 'sex', 'cp', 'trestbps', 'chol', 'fbs', 'restecg', 'thalach',
       'exang', 'oldpeak', 'slope', 'ca', 'thal', 'target'],
      dtype='object')
```

df.dtypes attribute is used to retrieve the data types of each column in a DataFrame

[ ]:
```
df.dtypes
```

[ ]:
```
age           int64
sex           int64
cp            int64
trestbps      int64
chol          int64
fbs           int64
restecg       int64
thalach       int64
exang         int64
oldpeak     float64
slope         int64
ca            int64
thal          int64
target        int64
dtype: object
```

the code df.isnull().count() in Pandas is used to count the total number of rows for each column in a DataFrame, including both missing (null or NaN) and non-missing values.

[ ]:
```
df.isnull().count()
```

[ ]:
```
age         1025
sex         1025
cp          1025
trestbps    1025
chol        1025
fbs         1025
restecg     1025
thalach     1025
exang       1025
oldpeak     1025
slope       1025
ca          1025
thal        1025
target      1025
dtype: int64
```

df.info() method in Pandas provides a concise summary of a DataFrame, including information about the data types, non-null values, and memory usage

[ ]:
```
df.info()
```

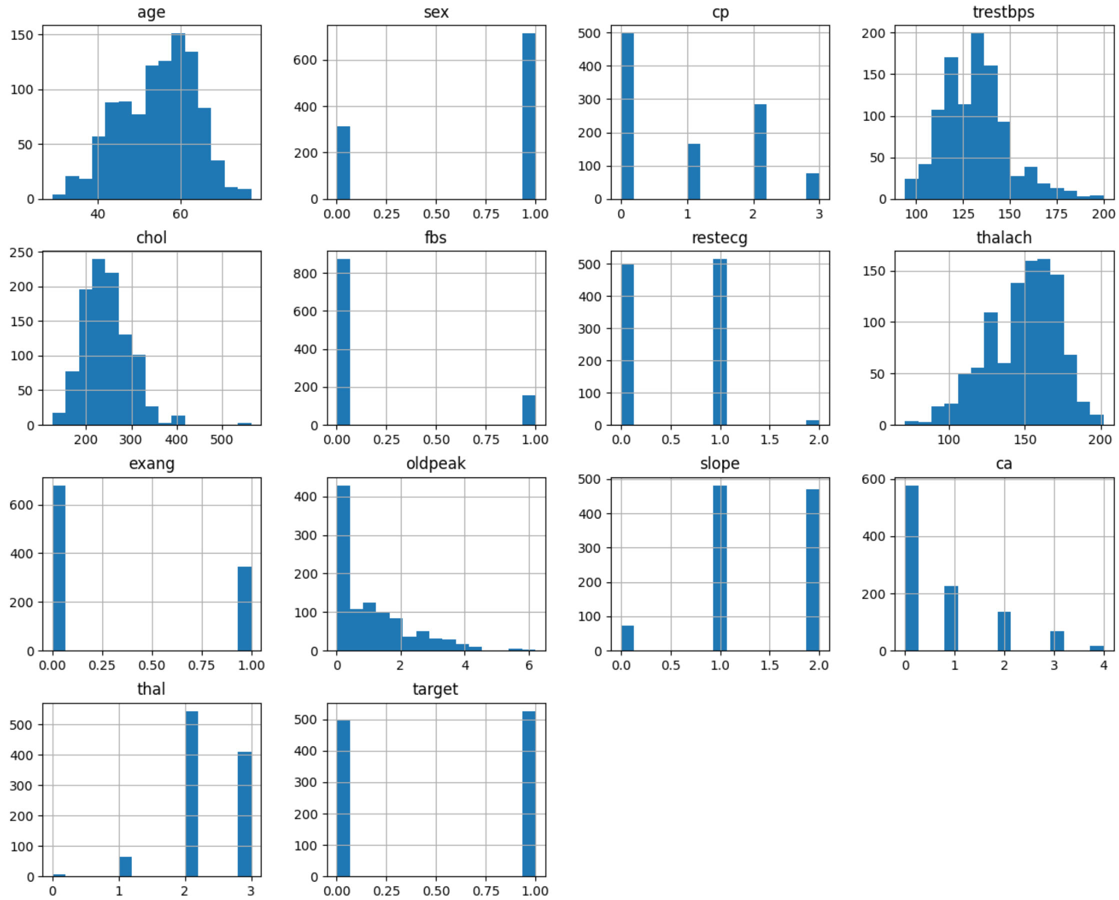
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1025 entries, 0 to 1024
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   age       1025 non-null   int64
 1   sex       1025 non-null   int64
 2   cp        1025 non-null   int64
 3   trestbps  1025 non-null   int64
 4   chol      1025 non-null   int64
 5   fbs       1025 non-null   int64
 6   restecg   1025 non-null   int64
 7   thalach   1025 non-null   int64
 8   exang     1025 non-null   int64
 9   oldpeak   1025 non-null   float64
 10  slope     1025 non-null   int64
 11  ca        1025 non-null   int64
 12  thal      1025 non-null   int64
 13  target    1025 non-null   int64
dtypes: float64(1), int64(13)
memory usage: 112.2 KB
```

The df.describe() method in Pandas is used to generate descriptive statistics that summarize the central tendency, dispersion, and shape of a dataset's distribution

[ ]:
```
df.describe()
```

[ ]:
```
              age          sex           cp     trestbps        chol  \
count  1025.000000  1025.000000  1025.000000  1025.000000  1025.00000
mean     54.434146     0.695610     0.942439   131.611707   246.00000
std       9.072290     0.460373     1.029641    17.516718    51.59251
min      29.000000     0.000000     0.000000    94.000000   126.00000
25%      48.000000     0.000000     0.000000   120.000000   211.00000
50%      56.000000     1.000000     1.000000   130.000000   240.00000
75%      61.000000     1.000000     2.000000   140.000000   275.00000
max      77.000000     1.000000     3.000000   200.000000   564.00000

              fbs      restecg      thalach        exang      oldpeak  \
count  1025.000000  1025.000000  1025.000000  1025.000000  1025.000000
mean      0.149268     0.529756   149.114146     0.336585     1.071512
std       0.356527     0.527878    23.005724     0.472772     1.175053
min       0.000000     0.000000    71.000000     0.000000     0.000000
25%       0.000000     0.000000   132.000000     0.000000     0.000000
50%       0.000000     1.000000   152.000000     0.000000     0.800000
75%       0.000000     1.000000   166.000000     1.000000     1.800000
max       1.000000     2.000000   202.000000     1.000000     6.200000

            slope           ca         thal       target
count  1025.000000  1025.000000  1025.000000  1025.000000
mean      1.385366     0.754146     2.323902     0.513171
std       0.617755     1.030798     0.620660     0.500070
min       0.000000     0.000000     0.000000     0.000000
25%       1.000000     0.000000     2.000000     0.000000
50%       1.000000     0.000000     2.000000     1.000000
75%       2.000000     1.000000     3.000000     1.000000
max       2.000000     4.000000     3.000000     1.000000
```

[ ]:
```python
df.hist(figsize=(15,12),bins = 15)
plt.title("Features Distribution")
plt.show()
```



[ ]:
```
df.isnull().sum()
```

[ ]:
```
age         0
sex         0
cp          0
trestbps    0
chol        0
fbs         0
restecg     0
thalach     0
exang       0
oldpeak     0
slope       0
ca          0
thal        0
target      0
dtype: int64
```

[ ]:
```
df.median()
```

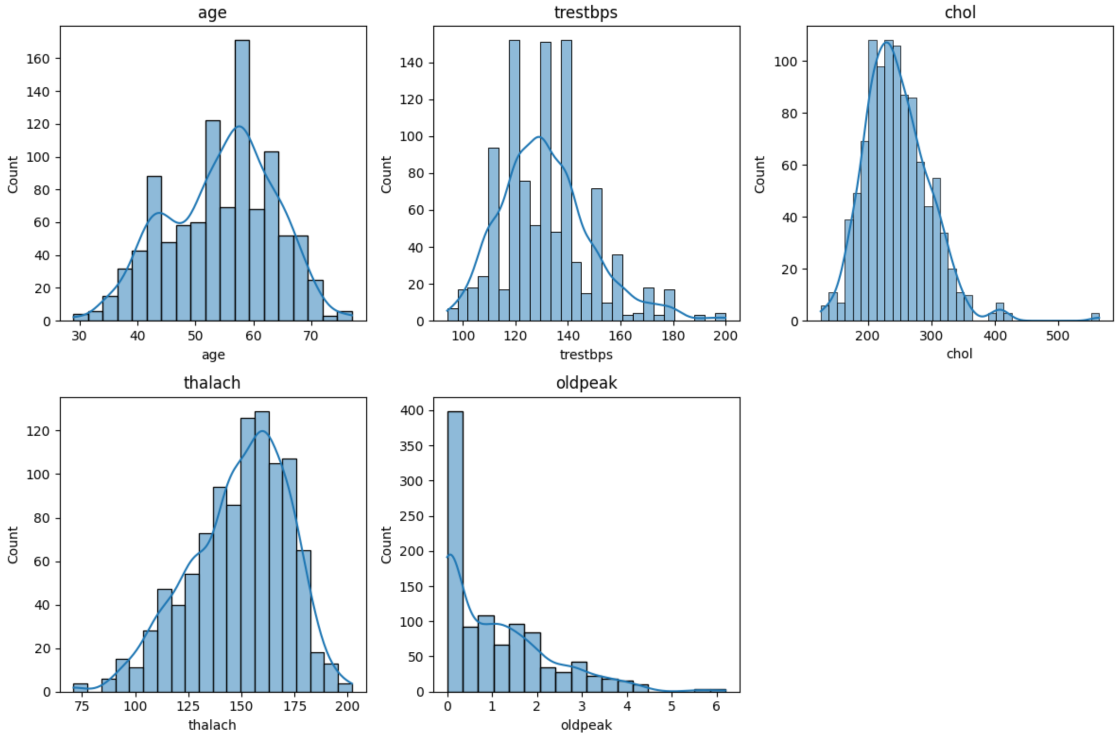
```
[ ]: age          56.0
     sex           1.0
     cp            1.0
     trestbps    130.0
     chol        240.0
     fbs           0.0
     restecg       1.0
     thalach     152.0
     exang         0.0
     oldpeak       0.8
     slope         1.0
     ca            0.0
     thal          2.0
     target        1.0
     dtype: float64
```

```
[ ]: df.mode()
```

```
[ ]:     age  sex   cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak  \
     0  58.0  1.0  0.0     120.0   204  0.0      1.0    162.0    0.0      0.0
     1   NaN  NaN  NaN       NaN   234  NaN      NaN      NaN    NaN      NaN

        slope   ca  thal  target
     0    1.0  0.0   2.0     1.0
     1    NaN  NaN   NaN     NaN
```

```
[ ]: # Selecting the numerical columns
     num_columns=['age','trestbps','chol','thalach','oldpeak']
```

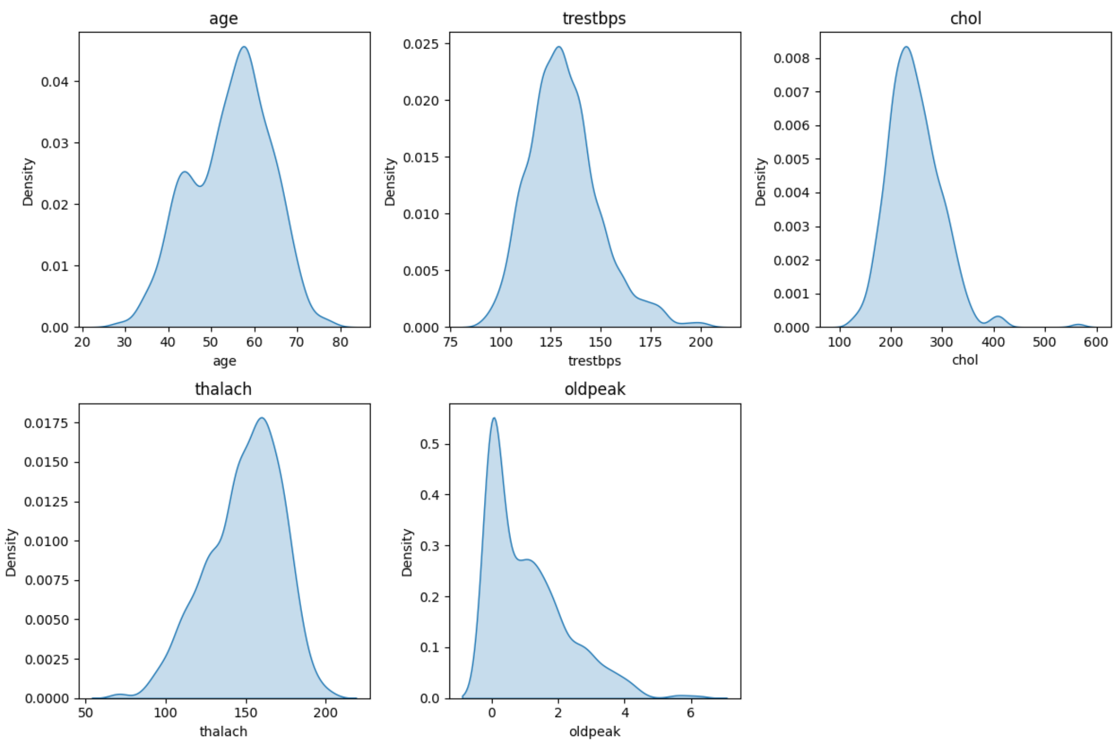
```
[ ]: import seaborn as sns
     # Plotting histograms for numerical columns
     plt.figure(figsize=(12, 8))
     for i, col in enumerate(num_columns, 1):
         plt.subplot(2, 3, i)
         sns.histplot(df[col], kde=True)
         plt.title(col)
     plt.tight_layout()
     plt.show()
```



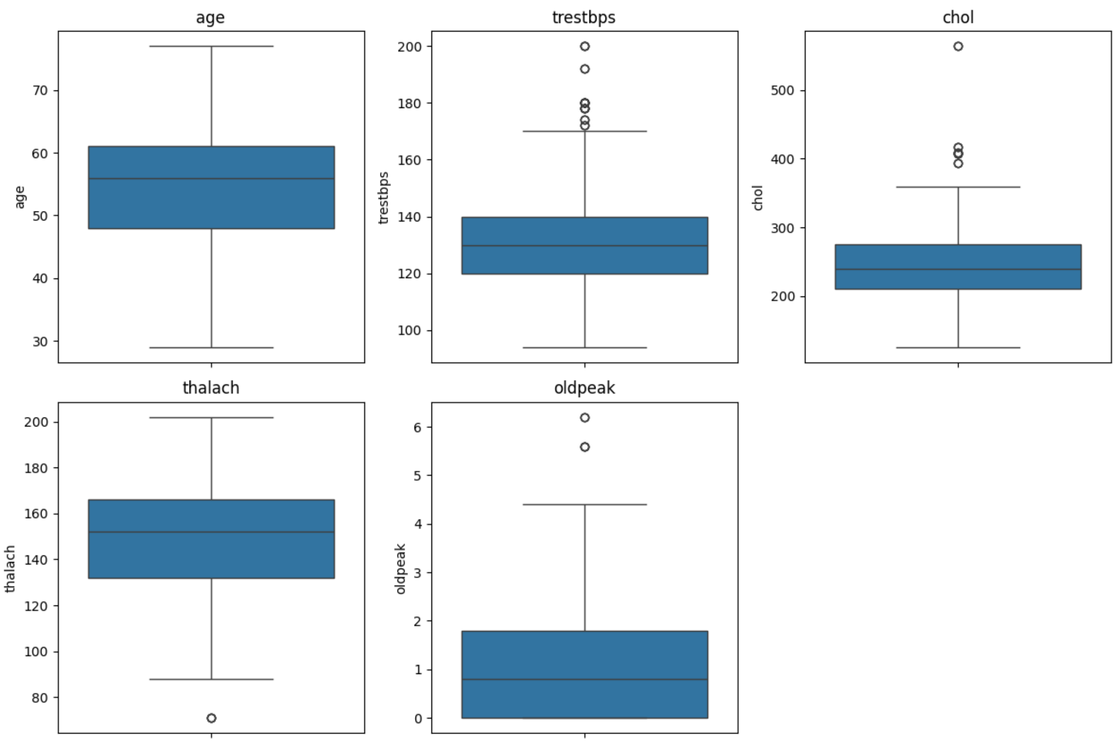
[ ]:
```
# Plotting kernel density plots for numerical columns
plt.figure(figsize=(12, 8))
for i, col in enumerate(num_columns, 1):
    plt.subplot(2, 3, i)
    sns.kdeplot(df[col], shade=True)
    plt.title(col)
plt.tight_layout()
plt.show()
```

```
<ipython-input-18-e394b15a5257>:5: FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

  sns.kdeplot(df[col], shade=True)
<ipython-input-18-e394b15a5257>:5: FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

  sns.kdeplot(df[col], shade=True)
<ipython-input-18-e394b15a5257>:5: FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
```

```
This will become an error in seaborn v0.14.0; please update your code.

  sns.kdeplot(df[col], shade=True)
<ipython-input-18-e394b15a5257>:5: FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

  sns.kdeplot(df[col], shade=True)
<ipython-input-18-e394b15a5257>:5: FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

  sns.kdeplot(df[col], shade=True)
```



[ ]:
```
# Plotting box plots for numerical columns
plt.figure(figsize=(12, 8))
for i, col in enumerate(num_columns, 1):
    plt.subplot(2, 3, i)
    sns.boxplot(y=df[col])
    plt.title(col)
plt.tight_layout()
plt.show()
```




```
[ ]: # Selecting the categorical columns
categorical_columns = ['sex', 'cp', 'fbs', 'restecg', 'exang', 'slope', 'ca',␣
 ↪'thal', 'target']
```

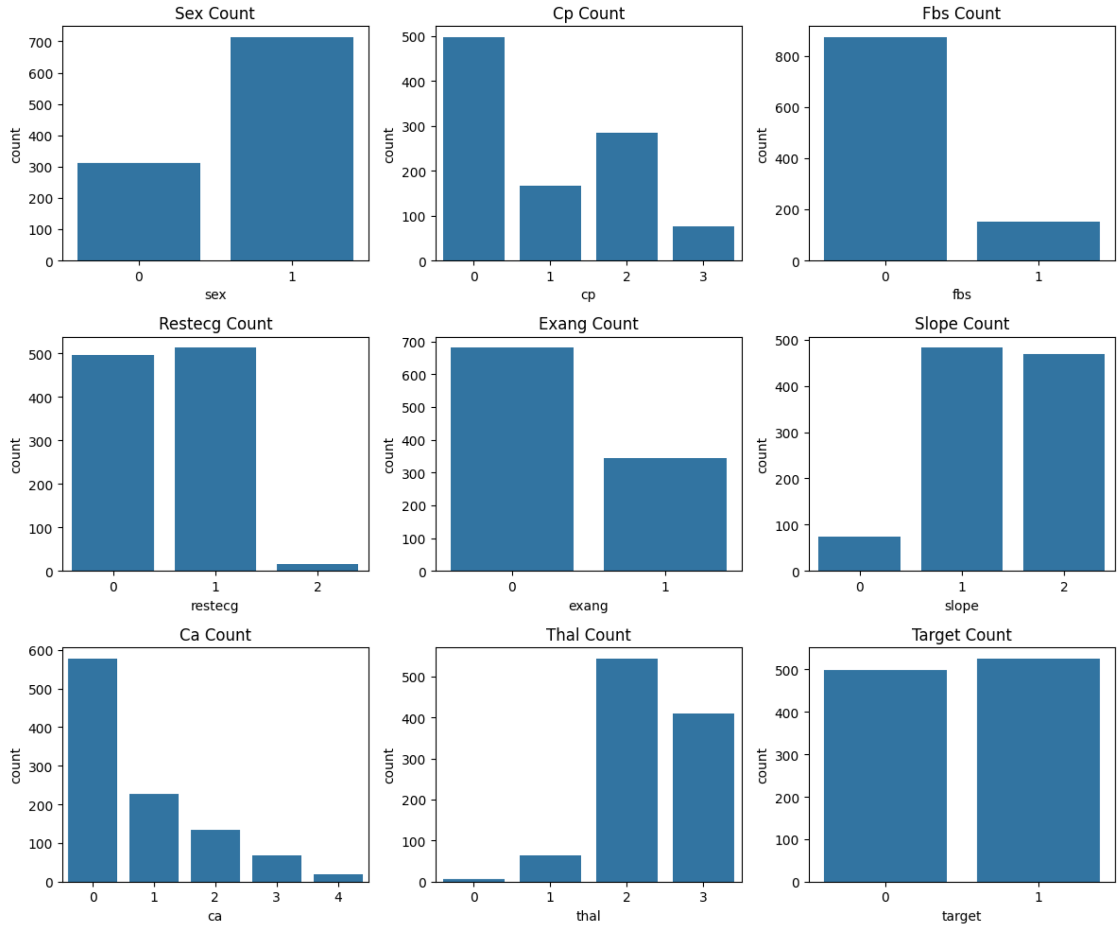
```
[ ]: # Display frequency tables showing counts and percentages
for col in categorical_columns:
    print(f"\n{col.capitalize()} Frequency Table:")
    frequency_table = df[col].value_counts().reset_index()
    frequency_table.columns = [col, 'Count']
    frequency_table['Percentage'] = (frequency_table['Count'] / len(df)) * 100
    print(frequency_table)
```

```
Sex Frequency Table:
   sex  Count  Percentage
0    1    713   69.560976
1    0    312   30.439024

Cp Frequency Table:
   cp  Count  Percentage
0   0    497   48.487805
1   2    284   27.707317
2   1    167   16.292683
3   3     77    7.512195
```

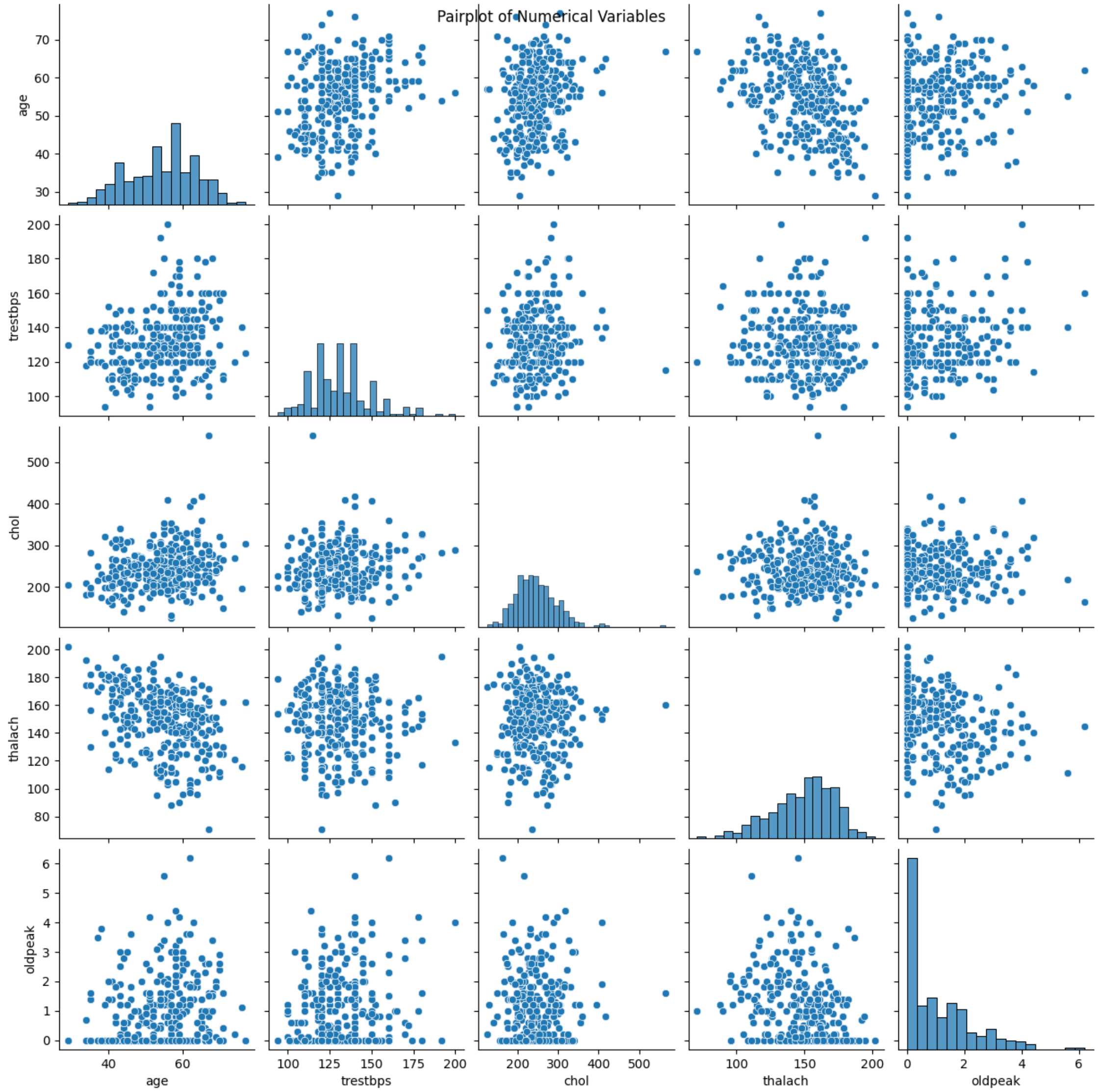
```
Fbs Frequency Table:
   fbs  Count  Percentage
0    0    872   85.073171
1    1    153   14.926829

Restecg Frequency Table:
   restecg  Count  Percentage
0        1    513   50.048780
1        0    497   48.487805
2        2     15    1.463415

Exang Frequency Table:
   exang  Count  Percentage
0      0    680   66.341463
1      1    345   33.658537

Slope Frequency Table:
   slope  Count  Percentage
0      1    482   47.024390
1      2    469   45.756098
2      0     74    7.219512

Ca Frequency Table:
   ca  Count  Percentage
0   0    578   56.390244
1   1    226   22.048780
2   2    134   13.073171
3   3     69    6.731707
4   4     18    1.756098

Thal Frequency Table:
   thal  Count  Percentage
0     2    544   53.073171
1     3    410   40.000000
2     1     64    6.243902
3     0      7    0.682927

Target Frequency Table:
   target  Count  Percentage
0       1    526   51.317073
1       0    499   48.682927
```

[ ]:
```
# Visualize using bar plots
plt.figure(figsize=(12, 10))
for i, col in enumerate(categorical_columns, 1):
    plt.subplot(3, 3, i)
```

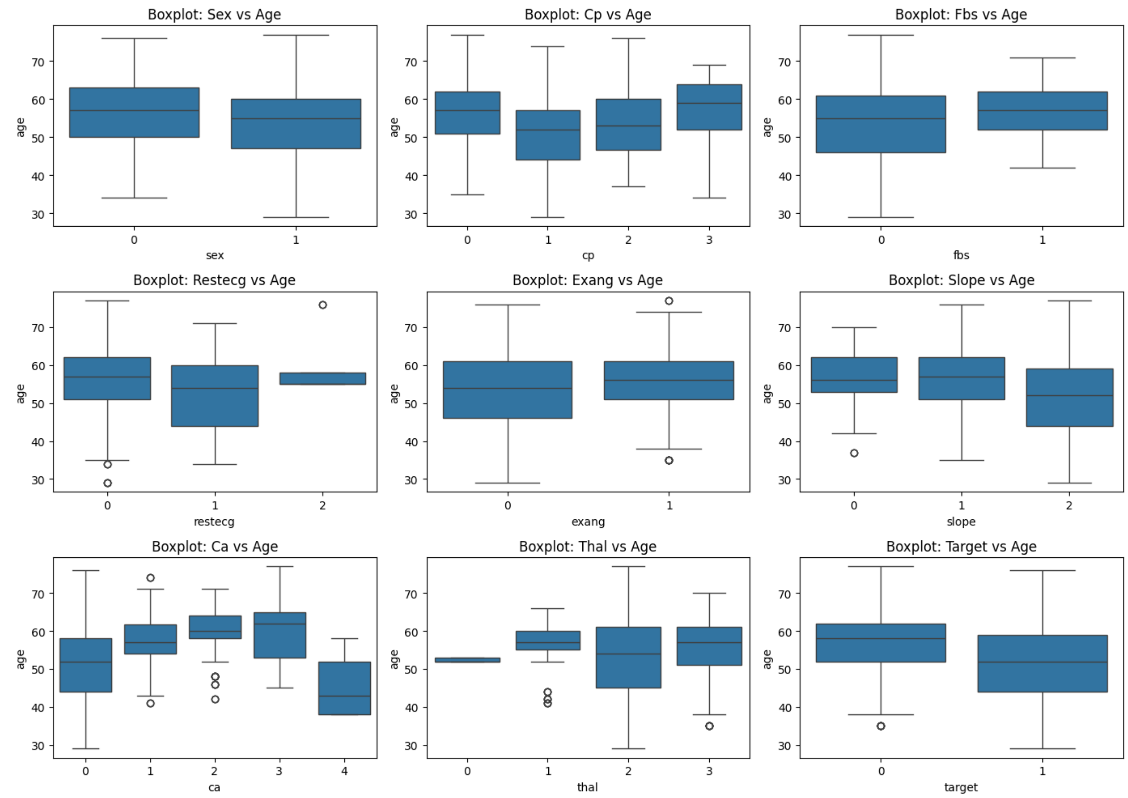
```
    sns.countplot(x=col, data=df)
    plt.title(col.capitalize() + ' Count')
plt.tight_layout()
plt.show()
```



[ ]:
```
# Scatter plots or pair plots for numerical variables
sns.pairplot(df[num_columns])
plt.suptitle('Pairplot of Numerical Variables')
plt.show()
```



[ ]:
```
# Box plots or violin plots for numerical vs categorical variables
plt.figure(figsize=(14, 10))
for i, col in enumerate(categorical_columns, 1):
    plt.subplot(3, 3, i)
    sns.boxplot(x=col, y='age', data=df)
    plt.title(f'Boxplot: {col.capitalize()} vs Age')
plt.tight_layout()
plt.show()
```



[ ]:
```
# Calculate correlation coefficients between numerical variables
correlation_matrix = df[num_columns].corr()
print("\nCorrelation Matrix:")
print(correlation_matrix)
```

```
Correlation Matrix:
               age  trestbps      chol   thalach   oldpeak
age       1.000000  0.271121  0.219823 -0.390227  0.208137
trestbps  0.271121  1.000000  0.127977 -0.039264  0.187434
chol      0.219823  0.127977  1.000000 -0.021772  0.064880
thalach  -0.390227 -0.039264 -0.021772  1.000000 -0.349796
oldpeak   0.208137  0.187434  0.064880 -0.349796  1.000000
```

[ ]:
```
from scipy.stats import pearsonr
# Calculate correlation coefficients between specific pairs of numerical␣
↪variables
for var1 in num_columns:
    for var2 in num_columns:
        if var1 != var2:
            corr_coef, p_value = pearsonr(df[var1], df[var2])
```

```
            print(f"\nCorrelation coefficient between {var1} and {var2}:␣
↪{corr_coef:.3f} (p-value: {p_value:.3f})")
```

```
Correlation coefficient between age and trestbps: 0.271 (p-value: 0.000)

Correlation coefficient between age and chol: 0.220 (p-value: 0.000)

Correlation coefficient between age and thalach: -0.390 (p-value: 0.000)

Correlation coefficient between age and oldpeak: 0.208 (p-value: 0.000)

Correlation coefficient between trestbps and age: 0.271 (p-value: 0.000)

Correlation coefficient between trestbps and chol: 0.128 (p-value: 0.000)

Correlation coefficient between trestbps and thalach: -0.039 (p-value: 0.209)

Correlation coefficient between trestbps and oldpeak: 0.187 (p-value: 0.000)

Correlation coefficient between chol and age: 0.220 (p-value: 0.000)

Correlation coefficient between chol and trestbps: 0.128 (p-value: 0.000)

Correlation coefficient between chol and thalach: -0.022 (p-value: 0.486)

Correlation coefficient between chol and oldpeak: 0.065 (p-value: 0.038)

Correlation coefficient between thalach and age: -0.390 (p-value: 0.000)

Correlation coefficient between thalach and trestbps: -0.039 (p-value: 0.209)

Correlation coefficient between thalach and chol: -0.022 (p-value: 0.486)

Correlation coefficient between thalach and oldpeak: -0.350 (p-value: 0.000)

Correlation coefficient between oldpeak and age: 0.208 (p-value: 0.000)

Correlation coefficient between oldpeak and trestbps: 0.187 (p-value: 0.000)

Correlation coefficient between oldpeak and chol: 0.065 (p-value: 0.038)

Correlation coefficient between oldpeak and thalach: -0.350 (p-value: 0.000)
```

[ ]:
```
X = df.drop('target', axis=1)
y = df['target']
```

[ ]:
```python
print("Features (X) shape:", X.shape)
print("Target variable (y) shape:", y.shape)
```

```
Features (X) shape: (1025, 13)
Target variable (y) shape: (1025,)
```

[ ]:
```python
from sklearn.preprocessing import StandardScaler, MinMaxScaler
```

[ ]:
```python
# Perform standardization on the features
scaler = StandardScaler()
X_standardized = scaler.fit_transform(X)
```

[ ]:
```python
# Perform normalization on the features
scaler = MinMaxScaler()
X_normalized = scaler.fit_transform(X)
```

[ ]:
```python
# Display the shape of X_standardized and X_normalized to verify
print("Shape of X_standardized:", X_standardized.shape)
print("Shape of X_normalized:", X_normalized.shape)
```

```
Shape of X_standardized: (1025, 13)
Shape of X_normalized: (1025, 13)
```

[ ]:
```python
from sklearn.model_selection import train_test_split
```

Splitting the data into training and testing data

[ ]:
```python
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)
```

[ ]:
```python
# Display the shapes of the training and testing sets to verify
print("X_train shape:", X_train.shape)
print("X_test shape:", X_test.shape)
print("y_train shape:", y_train.shape)
print("y_test shape:", y_test.shape)
```

```
X_train shape: (820, 13)
X_test shape: (205, 13)
y_train shape: (820,)
y_test shape: (205,)
```

Modelling Individual Classifiers

- KNN
- MLP
- SVM
- LR

GridSearchCV

- The model's hyperparameters are tuned using **GridSearchCV**, which performs an exhaustive search over the specified parameter values for the estimator. The best performing model is selected based on cross-validation.

[ ]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsClassifier
#create new a knn model
knn = KNeighborsClassifier()
#create a dictionary of all values we want to test for n_neighbors
params_knn = {'n_neighbors': np.arange(1, 26)}
#use gridsearch to test all values for n_neighbors
knn_gs = GridSearchCV(knn, params_knn, cv=5)
#fit model to training data
knn_gs.fit(X_train, y_train)
```

[ ]:
```
GridSearchCV(cv=5, estimator=KNeighborsClassifier(),
             param_grid={'n_neighbors': array([ 1,  2,  3,  4,  5,  6,  7,  8,
9, 10, 11, 12, 13, 14, 15, 16, 17,
       18, 19, 20, 21, 22, 23, 24, 25])})
```

[38]:
```
#save best model
knn_best = knn_gs.best_estimator_
#check best n_neigbors value
print(knn_gs.best_params_)
```

```
{'n_neighbors': 1}
```

[39]:
```
print('KNN_Best: {}'.format(round(knn_best.score(X_test, y_test)*100,2))+'%')
```

```
KNN_Best: 98.54%
```

[40]:
```
from sklearn.neural_network import MLPClassifier
mlp_clf = MLPClassifier(activation='relu', solver='lbfgs',␣
 ↪learning_rate='constant', alpha=0.0001, hidden_layer_sizes=(5, 2),␣
 ↪random_state=1)
mlp_clf.fit(X_train, y_train)
```

[40]:
```
MLPClassifier(hidden_layer_sizes=(5, 2), random_state=1, solver='lbfgs')
```

[41]:
```
from sklearn.metrics import accuracy_score
res = mlp_clf.predict(X_test)
res_acc = accuracy_score(y_test, res)
print ('MLP Accuracy : ',format(res_acc*100,'.2f')+'%')
```

```
MLP Accuracy :  50.24%
```

[42]:
```
from sklearn import svm
SVC_rbf = svm.SVC(kernel='rbf', probability=True).fit(X_train, y_train)
```

```
print("Accuracy Radial Basis Kernel:", round(SVC_rbf.score(X_test,␣
↪y_test)*100,2),"%")
```

```
Accuracy Radial Basis Kernel: 68.29 %
```

Voting Classifier

[46]:
```
#test the three models with the test data and print their accuracy scores
print('kNN: {}'.format(round(knn_best.score(X_test, y_test)*100,2))+'%')
print ('MLP : ',format(res_acc*100,'.2f')+'%')
print('SVM: {}'.format(round(SVC_rbf.score(X_test, y_test)*100,2))+'%')
print('log_reg: {}'.format(round(log_reg.score(X_test, y_test)*100,2))+'%')
```

```
kNN: 98.54%
MLP :  50.24%
SVM: 68.29%
log_reg: 78.54%
```

Voting Classifier supports two types of votings:

Hard Voting: In hard voting, the predicted output class is a class with the highest majority of votes i.e the class which had the highest probability of being predicted by each of the classifiers. Suppose three classifiers predicted the output class(A, A, B), so here the majority predicted A as output. Hence A will be the final prediction.

Soft Voting: In soft voting, the output class is the prediction based on the average of probability given to that class. Suppose given some input to three models, the prediction probability for class A = (0.30, 0.47, 0.53) and B = (0.20, 0.32, 0.40). So the average for class A is 0.4333 and B is 0.3067, the winner is clearly class A because it had the highest probability averaged by each classifier

[47]:
```
#Hard / Soft Voting
ensemble.fit_transform(X_train, y_train)
ensemble.transform(X_train)
#print(ensemble.predict(X_train))
score=ensemble.score(X_test, y_test)*100
print("Accuracy :"+format(score,".2f")+"%")
```

```
[Voting] … (1 of 4) Processing knn, total=   0.0s
[Voting] … (2 of 4) Processing SVC, total=   0.1s
[Voting] … (3 of 4) Processing log_reg, total=   0.0s
[Voting] … (4 of 4) Processing MLP, total=   0.0s
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
```

```
regression
  n_iter_i = _check_optimize_result(
```

```
Accuracy :93.66%
```

Training and Testing Data : Bagging Classifier - Decision Trees

[48]:
```
from sklearn.model_selection import cross_val_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import BaggingClassifier
```

[49]:
```
scores = cross_val_score(DecisionTreeClassifier(),X,y,cv=20)
scores
```

[49]:
```
array([1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1.])
```

[50]:
```
scores.mean()
```

[50]:
```
1.0
```

[51]:
```
bag_model = BaggingClassifier(
estimator=DecisionTreeClassifier(),
n_estimators=100,
max_samples=0.8,
oob_score=True
)
```

[52]:
```
bag_model.fit(X_train,y_train)
```

[52]:
```
BaggingClassifier(estimator=DecisionTreeClassifier(), max_samples=0.8,
                  n_estimators=100, oob_score=True)
```

[53]:
```
bag_model.oob_score_
```

[53]:
```
0.9926829268292683
```

[54]:
```
bag_model.score(X_test,y_test)
```

[54]:
```
0.9853658536585366
```

[55]:
```
scores = cross_val_score(bag_model,X,y,cv=20)
scores
```

[55]:
```
array([1.        , 1.        , 1.        , 1.        , 1.        ,
       1.        , 1.        , 1.        , 1.        , 1.        ,
       1.        , 1.        , 1.        , 1.        , 1.        ,
       1.        , 0.94117647, 1.        , 1.        , 1.        ])
```

[56]:
```
format(scores.mean()*100,".2f")+"%"
```

[56]: '99.71%'

Training and Testing Data : Bagging Classifier - Random Forest

[57]:
```
from sklearn.metrics import confusion_matrix, accuracy_score,␣
↪classification_report


def evaluate(model, X_train, X_test, y_train, y_test):
    y_test_pred = model.predict(X_test)
    y_train_pred = model.predict(X_train)

    print("TRAINIG RESULTS: \n===============================")
    clf_report = pd.DataFrame(classification_report(y_train, y_train_pred,␣
↪output_dict=True))
    print(f"CONFUSION MATRIX:\n{confusion_matrix(y_train, y_train_pred)}")
    print(f"ACCURACY SCORE:\n{accuracy_score(y_train, y_train_pred):.4f}")
    print(f"CLASSIFICATION REPORT:\n{clf_report}")

    print("TESTING RESULTS: \n===============================")
    clf_report = pd.DataFrame(classification_report(y_test, y_test_pred,␣
↪output_dict=True))
    print(f"CONFUSION MATRIX:\n{confusion_matrix(y_test, y_test_pred)}")
    print(f"ACCURACY SCORE:\n{accuracy_score(y_test, y_test_pred)*100:.2f}"+"%")
    print(f"CLASSIFICATION REPORT:\n{clf_report}")
```

[58]:
```
from sklearn.ensemble import RandomForestClassifier

rf_clf = RandomForestClassifier(random_state=42, n_estimators=1000)
rf_clf.fit(X_train, y_train)
evaluate(rf_clf, X_train, X_test, y_train, y_test)
```

```
TRAINIG RESULTS:
===============================
CONFUSION MATRIX:
[[397   0]
 [  0 423]]
ACCURACY SCORE:
1.0000
CLASSIFICATION REPORT:
               0      1  accuracy  macro avg  weighted avg
precision    1.0    1.0       1.0        1.0           1.0
recall       1.0    1.0       1.0        1.0           1.0
f1-score     1.0    1.0       1.0        1.0           1.0
support    397.0  423.0       1.0      820.0         820.0
TESTING RESULTS:
```

```
================================
CONFUSION MATRIX:
[[102   0]
 [  3 100]]
ACCURACY SCORE:
98.54%
CLASSIFICATION REPORT:
                    0           1  accuracy   macro avg  weighted avg
precision    0.971429    1.000000  0.985366    0.985714      0.985784
recall       1.000000    0.970874  0.985366    0.985437      0.985366
f1-score     0.985507    0.985222  0.985366    0.985364      0.985364
support    102.000000  103.000000  0.985366  205.000000    205.000000
```

[59]:

```
from sklearn.ensemble import AdaBoostClassifier

ada_boost_clf = AdaBoostClassifier(n_estimators=30)
ada_boost_clf.fit(X_train, y_train)
evaluate(ada_boost_clf, X_train, X_test, y_train, y_test)
```

```
TRAINIG RESULTS:
================================
CONFUSION MATRIX:
[[372  25]
 [ 27 396]]
ACCURACY SCORE:
0.9366
CLASSIFICATION REPORT:
                    0           1  accuracy   macro avg  weighted avg
precision    0.932331    0.940618  0.936585    0.936474      0.936606
recall       0.937028    0.936170  0.936585    0.936599      0.936585
f1-score     0.934673    0.938389  0.936585    0.936531      0.936590
support    397.000000  423.000000  0.936585  820.000000    820.000000
TESTING RESULTS:
================================
CONFUSION MATRIX:
[[82 20]
 [12 91]]
ACCURACY SCORE:
84.39%
CLASSIFICATION REPORT:
                    0           1  accuracy   macro avg  weighted avg
precision    0.872340    0.819820  0.843902    0.846080      0.845952
recall       0.803922    0.883495  0.843902    0.843708      0.843902
f1-score     0.836735    0.850467  0.843902    0.843601      0.843634
support    102.000000  103.000000  0.843902  205.000000    205.000000
```

[ ]: