AMERICAN INTERNATIONAL UNIVERSITY-BANGLADESH

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

FACULTY OF SCIENCE AND TECHNOLOGY (FST)

FALL 2023-2024

HEART FAILURE PREDICTION

PROGRAMMING IN PYTHON

SECTION: B

PROJECT SUBMITTED TO:

DR. ABDUS SALAM

ASSISTANT PROFESSOR,
FACULTY OF SCIENCE AND TECHNOLOGY (FST)

PROJECT SUBMITTED BY:

SL NO	STUDENT NAME	STUDENT ID
01	SABIHA KHARI OHI	20-41905-1
02	RIJOAN FARDOUS	20-41943-1

DATE OF SUBMISSION: **DEC 25, 2023.**

Dataset Name: Heart Failure Prediction.
Dataset Link: https://www.kaggle.com/datasets/fedesoriano/heart-failure-prediction.

Description: Cardiovascular diseases (CVDs) is a global cause of mortality, claiming 17.9 million lives each year, and contributing to 31% of all deaths worldwide. Heart failure, often a consequence of CVDs, requires early identification. This dataset aims to empower predictive modelling for heart disease.

The dataset has a total of **918 entries** with **11 features**, including both **numerical** and **categorical features**, providing diverse insights into cardiovascular health. The features include: Age, Sex, Chest Pain Type, Resting Blood Pressure, Serum Cholesterol, Fasting Blood Sugar, Resting ECG, Max Heart Rate, Exercise Angina, Old peak, ST Segment Slope.

Task 01: Read/Load the dataset file in your program using pandas library.

In the code ‘`np.random.seed(123)`’ was used for reproducing the same result every time. Then the CSV file was read using ‘`pd.read_csv()`’ from Google Drive. The Dataset was randomised using the ‘`shuffle()`’ function from Scikit-Learn.

```
np.random.seed(123)
df=pd.read_csv('/content/drive/MyDrive/python_final_project_group_01/heart_failure_
dataset.csv')
df = shuffle(df)
print(df)
```

Task 02: Apply appropriate data cleaning techniques to the dataset using pandas library.

To check if there were any duplicated entries in the dataset ‘`df.duplicated().any()`’ was used. This returns true if there are duplicated entries. ‘`df.isnull().sum().sum()`’ was used to check if there were any columns containing empty cells. This returns the total number of empty cells. And was used to check whether any columns had wrong data types. The dataset didn’t contain any duplicated entries, empty cells or wrong format.

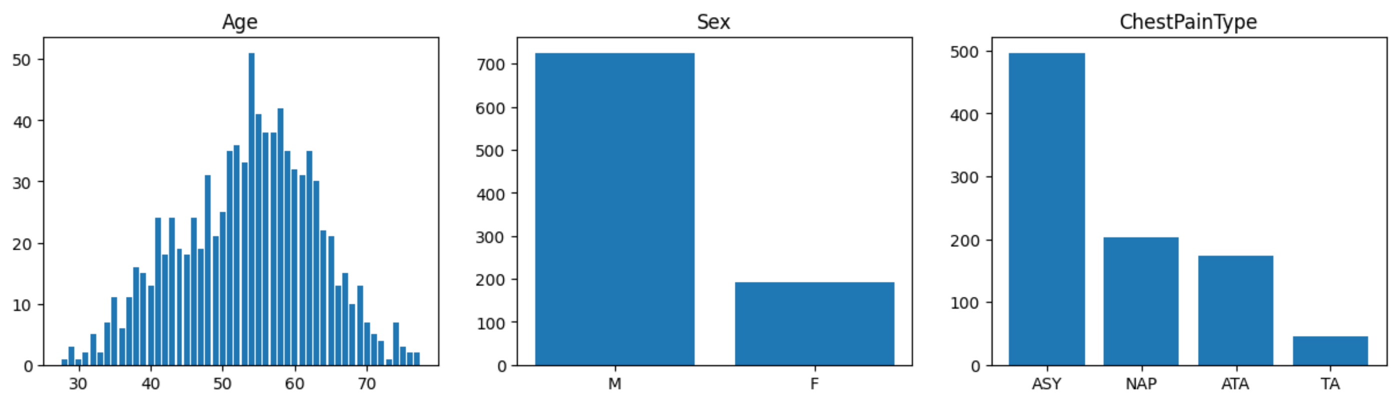
But some of the cells of RestingBP and Cholesterol contained wrong values like 0. First 0s were replaced by np.nan ‘`df['RestingBP'].replace(0, np.nan)`’ and then those nan values were replaced by median ‘`df['RestingBP'].fillna(median_bp)`’. And some of the cholesterol values contained outliers and they were replaced by 450.

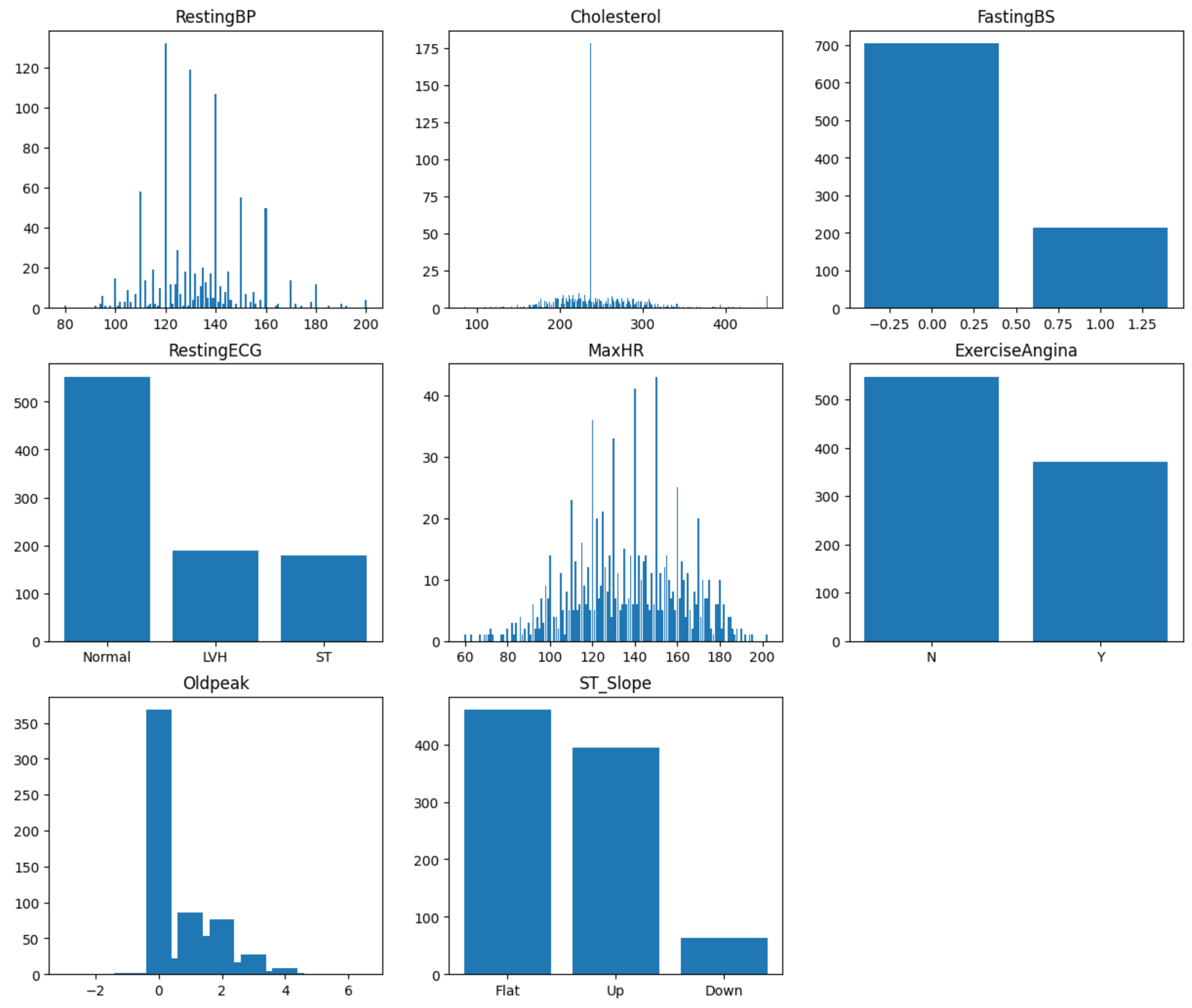
```
duplicated_rows = df.duplicated().any()
print("Have Duplicated Rows: ", duplicated_rows)
empty_cells = df.isnull().sum().sum()
print("Number Of Empty Cells: ", empty_cells)
print("\nData Types:")
print(df.dtypes)

df['RestingBP'] = df['RestingBP'].replace(0, np.nan)
median_bp = df['RestingBP'].median()
df['RestingBP'] = df['RestingBP'].fillna(median_bp)
df['Cholesterol'] = df['Cholesterol'].replace(0, np.nan)
median_cholesterol = df['Cholesterol'].median()
df['Cholesterol'] = df['Cholesterol'].fillna(median_cholesterol)

for x in df.index:
    if df.loc[x, 'Cholesterol'] > 450:
        df.loc[x, 'Cholesterol'] = 450
```

Task 03: Draw graphs to analyse the frequency distributions of the features using Matplotlib.

First width and height were set for the diagram using ‘`plt.figure(figsize=(15, 17))`’ since we had to use ‘`plt.subplot()`’ to draw all the graphs in one diagram. Graphs were arranged in 4 columns and 3 rows. To exclude the last columns (target column) ‘`df.columns[:-1]`’ was used. A ‘for in’ loop was used to iterate through all the features. For each feature, ‘`value_counts()`’ gives us the frequency. Then ‘`plt.bar(frequency.index, frequency.values)`’ was used to draw the graphs.

```
plt.figure(figsize=(15, 17))
for i, column in enumerate(df.columns[:-1]):
  frequency = df[column].value_counts()
  plt.subplot(4, 3, i+1)
  plt.bar(frequency.index, frequency.values)
  plt.title(column)

plt.suptitle("Frequency Distribution of the Features (Frequency in Y axis)")
plt.show()
```

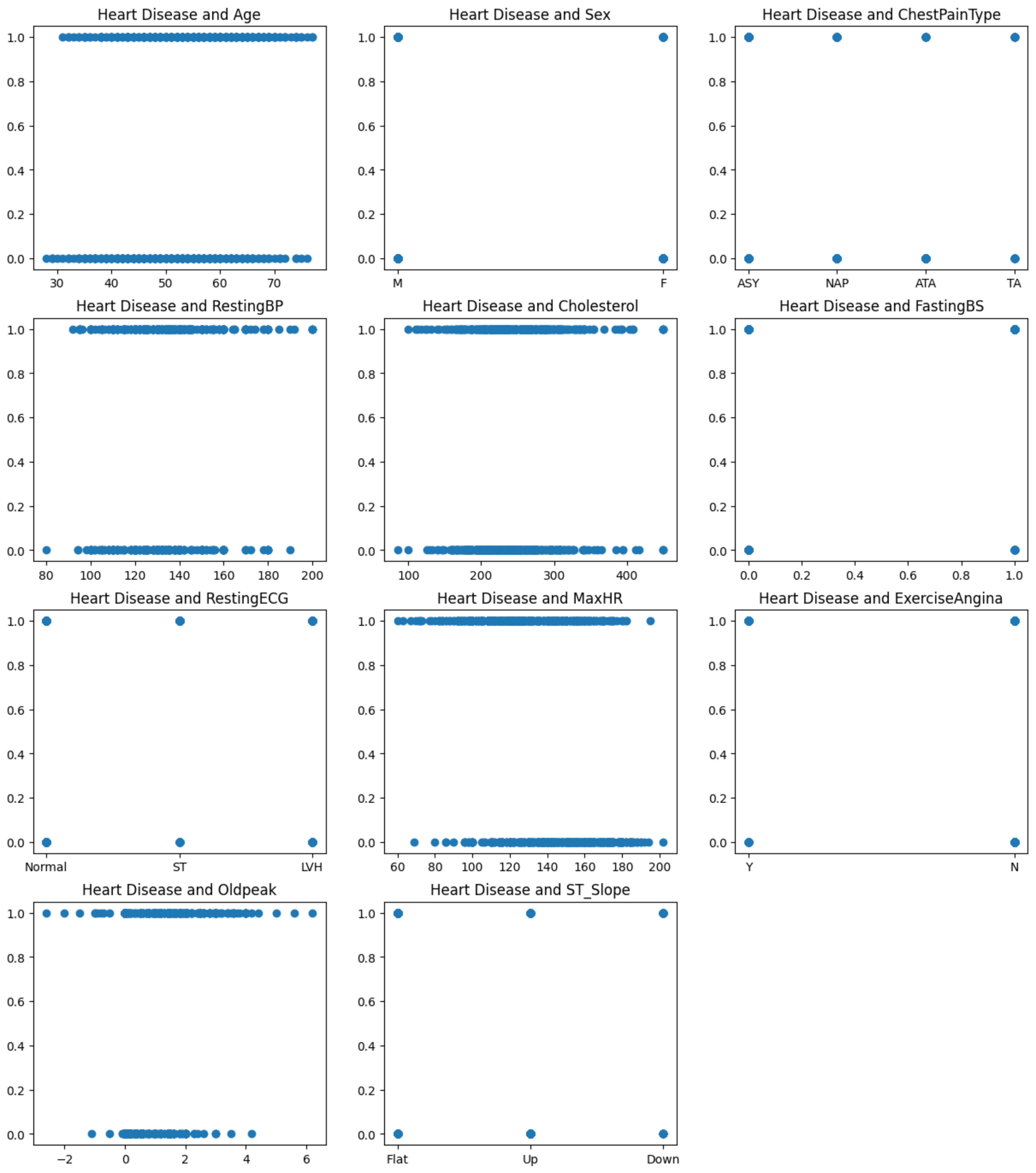




<u>Task 04</u>: Draw graphs to illustrate if there is any relationship between the target column to any other columns of the dataset using Matplotlib.

Task 4 was implemented like task 3 but here scatter diagrams are drawn to illustrate relationships. The target column ‘`HeartDisease`’ was kept in a variable since it was used frequently. ‘`plt.scatter(df[column], df[target_column])`’ was used to draw scatter diagrams. Here the target column ‘`HeartDisease`’ is at the Y-axis [1: Present, 0: Absent].

```
plt.figure(figsize=(15, 17))
target_column = 'HeartDisease'
for i, column in enumerate(df.columns[:11]):
  plt.subplot(4, 3, i+1)
  plt.scatter(df[column], df[target_column])
  plt.title(f'Heart Disease and {column}')
plt.suptitle("Y axis - Heart Disease (1: Present, 0: Absent) and X axis -
Features")
plt.show()
```



Task 05: Perform scaling to the features of the dataset

Before performing scaling to the features, categorical values need to be converted to numerical values. They were converted to numerical values using ‘`LabelEncoder()`’ from Scikit-Learn. Then all of the features were copied into X and target column into y. Then ‘`StandardScaler()`’ from Scikit-Learn was used to perform scaling to the X.

```
categorical_columns = ['Sex', 'ChestPainType', 'RestingECG', 'ExerciseAngina',
'ST_Slope']
label_encoder = LabelEncoder()
for column in categorical_columns:
  df[column] = label_encoder.fit_transform(df[column])

print("\nData Types:")
print(df.dtypes)
all_features = df.columns
features = all_features[:-1]
X = df[features].copy()
y = df['HeartDisease']

scale = StandardScaler()
X[features] = scale.fit_transform(X)
print("\nFeatures After Scaling:")
print(X)
```

Task 06: Split your data into two parts: Training dataset and Testing dataset.

The dataset was split into two parts, training and testing, using the ‘`train_test_split(X, y, test_size=0.2, random_state=123)`’ function. ‘`random_state=123`’ was used for reproducing the same result. ‘`test_size=0.2`’ was used for splitting the dataset into 80:20 ratio. 80% training and 20% testing.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=123)
print("Training Data Shape:", X_train.shape, y_train.shape)
print("Testing Data Shape:", X_test.shape, y_test.shape)
```

Task 07: Apply Naïve Bayes Classifier to the dataset. Build (train) your prediction model in this step.

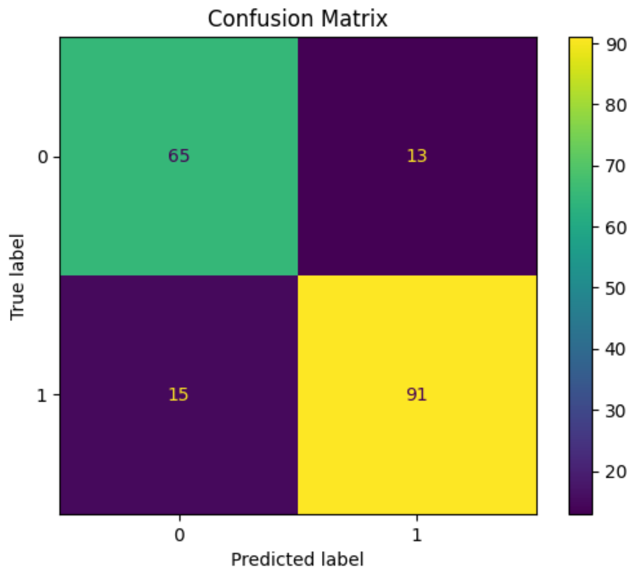
Gaussian Naive Bayes Classifier was applied to the training dataset for building the prediction model. First Gaussian Naive Bayes was initialised by the ‘`GaussianNB()`’ function and then fits it to the training data (X_train and y_train), and then uses the trained model to make predictions on the testing data (X_test). The predicted and actual values are stored in the variables predicted and actual, respectively.

```
naive_bayes_classifier = GaussianNB()
naive_bayes_classifier.fit(X_train, y_train)
predicted = naive_bayes_classifier.predict(X_test)
actual = y_test
```

Task 08: Calculate the confusion matrix for your model.

This task involves calculating and interpreting the confusion matrix for the Naive Bayes Classifier model. This uses Scikit-Learn's ‘`confusion_matrix()`’ function and ConfusionMatrixDisplay to visualise the results.

```
confusion_matrix = metrics.confusion_matrix(actual, predicted)
cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = confusion_matrix,
                              display_labels = naive_bayes_classifier.classes_)
cm_display.plot()
plt.title('Confusion Matrix')
plt.show()
```



The right diagonal represents the instances that were correctly classified, while the left diagonal represents the instances that were incorrectly classified. The model correctly predicted 65 instances of the positive class and 91 instances of the negative class. The model incorrectly predicted 15 instances of the positive class and 13 instances of the negative class.

Task 09: Calculate the accuracy, precision, recall and f-1 score of your model.

In this task, accuracy, precision, recall and f-1 score of the model was calculated using functions such as ‘`accuracy_score(actual, predicted)`’, ‘`precision_score(actual, predicted)`’, ‘`recall_score(actual, predicted)`’, ‘`f1_score(actual, predicted)`’ from Scikit-Learn.

```
Accuracy = metrics.accuracy_score(actual, predicted)
Precision = metrics.precision_score(actual, predicted)
Recall = metrics.recall_score(actual, predicted)
F1_score = metrics.f1_score(actual, predicted)

print("Accuracy:", Accuracy)
print("Precision:", Precision)
print("Recall:", Recall)
print("F1 Score:", F1_score)
```

```
Accuracy: 0.8478260869565217
Precision: 0.875
Recall: 0.8584905660377359
F1 Score: 0.8666666666666667
```

Accuracy 0.8478: This indicates the overall correctness of the model's predictions. In this case, approximately 84.78% of predictions were correct.

Precision 0.875: It means that 87.5% of the instances predicted as positive by the model were actually positive.

Recall (Sensitivity) 0.8585: This means that approximately 85.85% of the actual positive instances in the dataset were successfully captured by the model.

F1 Score 0.8667: The F1 score is approximately 86.67%, indicating a balanced performance in precision and recall on the positive class.

<u>Task 10</u>: Show how 10-fold cross validation can be used to build a naïve bayes classifier.
10-fold cross validation was done using the '`cross_val_score()`' function and calculating the mean value of them.

```
k_folds = KFold(n_splits = 10)
cross_val_scores = cross_val_score(naive_bayes_classifier, X, y, cv=k_folds)
print("Eacn Fold Accuracy:", cross_val_scores)
mean_accuracy = cross_val_scores.mean()
print("\nMean Accuracy:", mean_accuracy)
```

```
Eacn Fold Accuracy: [0.84782609 0.76086957 0.85869565 0.89130435 0.81521739
0.85869565 0.82608696 0.85869565 0.86813187 0.82417582]
Mean Accuracy: 0.8409698996655519
```

The "Each Fold Accuracy" values represent the accuracy of the model when it's trained and tested on different subsets of the dataset. These values range from approximately 76.09% to 89.13% for each fold. The "Mean Accuracy" value, which is about 84.10%, gives us an average accuracy across all folds.