Cardio Vascular Diseases Prediction

Introduction

Cardiovascular diseases are the leading cause of death globally. It is therefore necessary to identify the causes and develop a system to predict heart attacks in an effective manner. The data below has the information about the factors that might have an impact on cardiovascular health.

Variable Description

Age - Age in years

Sex - 1 = male; 0 = female

cp - Chest pain type

trestbps- Resting blood pressure (in mm Hg on admission to the hospital)

chol - Serum cholesterol in mg/dl

fbs - Fasting blood sugar > 120 mg/dl (1 = true; 0 = false)

restecg - Resting electrocardiographic results

thalach - Maximum heart rate achieved

exang - Exercise induced angina (1 = yes; 0 = no)

oldpeak - ST depression induced by exercise relative to rest

slope - Slope of the peak exercise ST segment

ca - Number of major vessels (0-3) colored by fluoroscopy

thal - 3 = normal; 6 = fixed defect; 7 = reversible defect

Target - 1 or 0

Import all the liabraries -

Pandas - Used to analyse data. It has function for analysing,cleaning,exploring and manipulating data.

Numpy - Mostly work on numerical values for making Arithmatic Operations.

Matplotlib - Comprehensive library for creating static,animated and intractive visualization.

Seaborn - Seaborn is a python data visualization library based on matplotlib. It provides a high-level interface for drawing intractive and informative statastical graphics.

Warnings - warnings are provided to warn the developer of situation that are not necessarily exceptions and ignore them.

By read_csv() function we are reading the dataset present in " Cardio_vascular .csv" file.

Took dataset from Kaggle

Will be making predictions models using Machine Learning Algorithams.

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

import warnings
warnings.filterwarnings("ignore")
```

In [2]:
```
df=pd.read_csv("Cardio_vascular.csv")
```

In [3]:
```
df
```

Out[3]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	51	1	0	125	212	0	1	168	0	1.0	2	2	3	0
1	54	1	0	140	203	1	0	155	1	3.1	0	0	3	0
2	71	1	0	145	174	0	1	125	1	2.6	0	0	3	0
3	60	1	0	148	203	0	1	161	0	0.0	2	1	3	0
4	61	0	0	138	294	1	1	106	0	1.9	1	3	2	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1020	59	1	1	173	221	0	1	164	1	1.2	2	0	2	1
1021	60	1	0	174	258	0	0	141	1	1.4	1	1	3	0
1022	47	1	0	173	275	0	0	118	1	0.0	1	1	2	0
1023	50	0	0	165	254	0	0	159	0	3.8	2	0	2	1
1024	54	1	0	168	188	0	1	113	0	0.4	1	1	3	0

1025 rows × 14 columns

Find information about Data by using df.info()

This Cardio_vascular .csv dataset contains the information of patients cardiovascular health condition through their parameters with 1025 rows and 14 columns.

In [4]:
```
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
```

It provides information about the DataFrame, including the data types of each column, the number of non-null values, and memory usage.

This dataset contain 1025 rows and 14 columns out of there are 14 numerical columns.

How many people have chances of having heart attack?

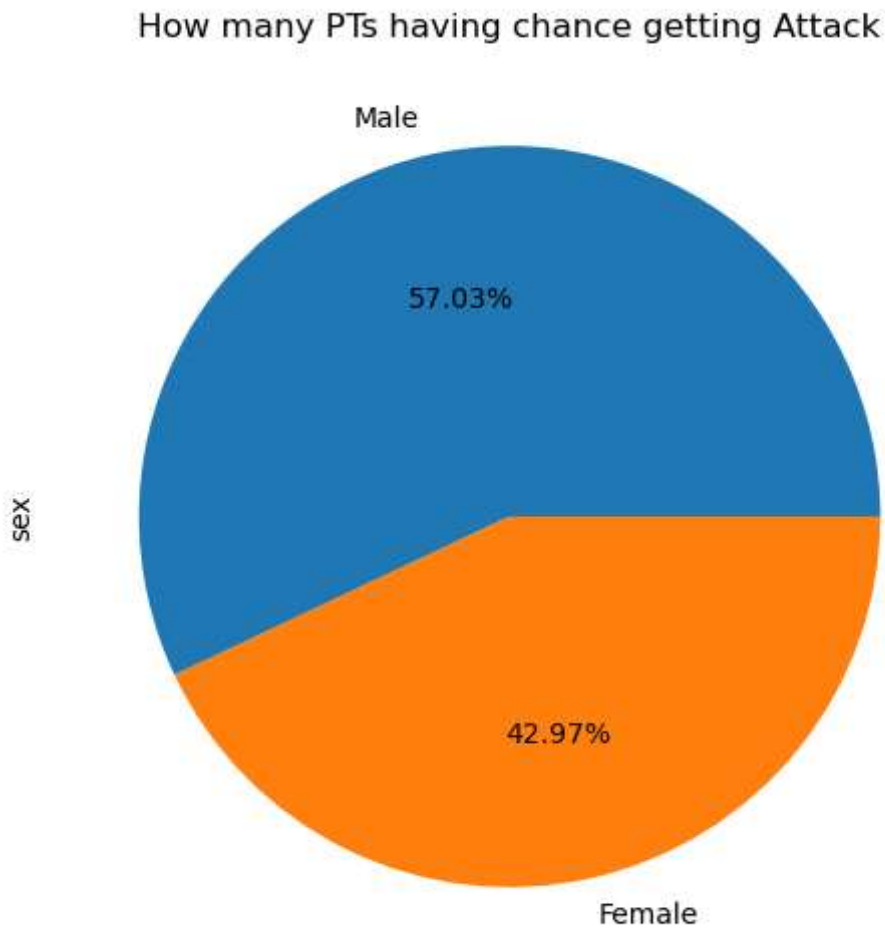
In [5]:
```
df["target"].value_counts()
```

Out[5]:
```
1    526
0    499
Name: target, dtype: int64
```

Here is 526 people have chances of getting attack and 499 people have no chances of getting attack.

Out of all the pts how many are getting chances of attack,what was the precentage of male and female.

In [6]:
```
plt.figure(figsize=(6,6))
df[df["target"]==1]["sex"].value_counts().plot.pie(autopct="%1.2f%%",labels=["Male","F
plt.title("How many PTs having chance getting Attack")
plt.show()
```



In [7]:
```
df["chol"].max()
```

Out[7]:
```
564
```

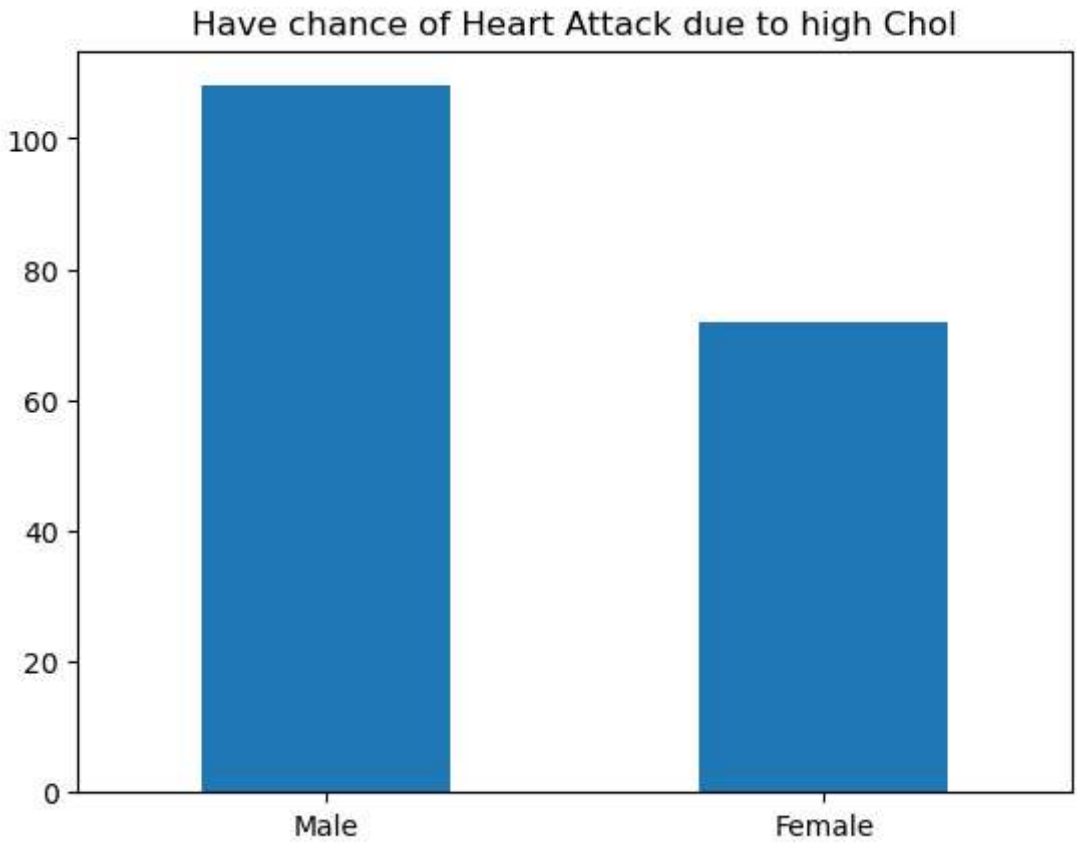
In [8]:
```
df["chol"].min()
```

Out[8]:
```
126
```

The pie chart shows the count in percent of male and female have chance of attack in that Male count is 57.03 % and Female count is 42.97 %.

How many people may having chance of attack on the basis of cholestrol level.

In [9]:
```
df[(df["chol"]>250)&(df["target"]==1)]["sex"].value_counts().plot.bar("chol","Target")
plt.title("Have chance of Heart Attack due to high Chol")
plt.xticks(rotation=360,ticks=[0,1],labels=["Male","Female"])
plt.show()
```

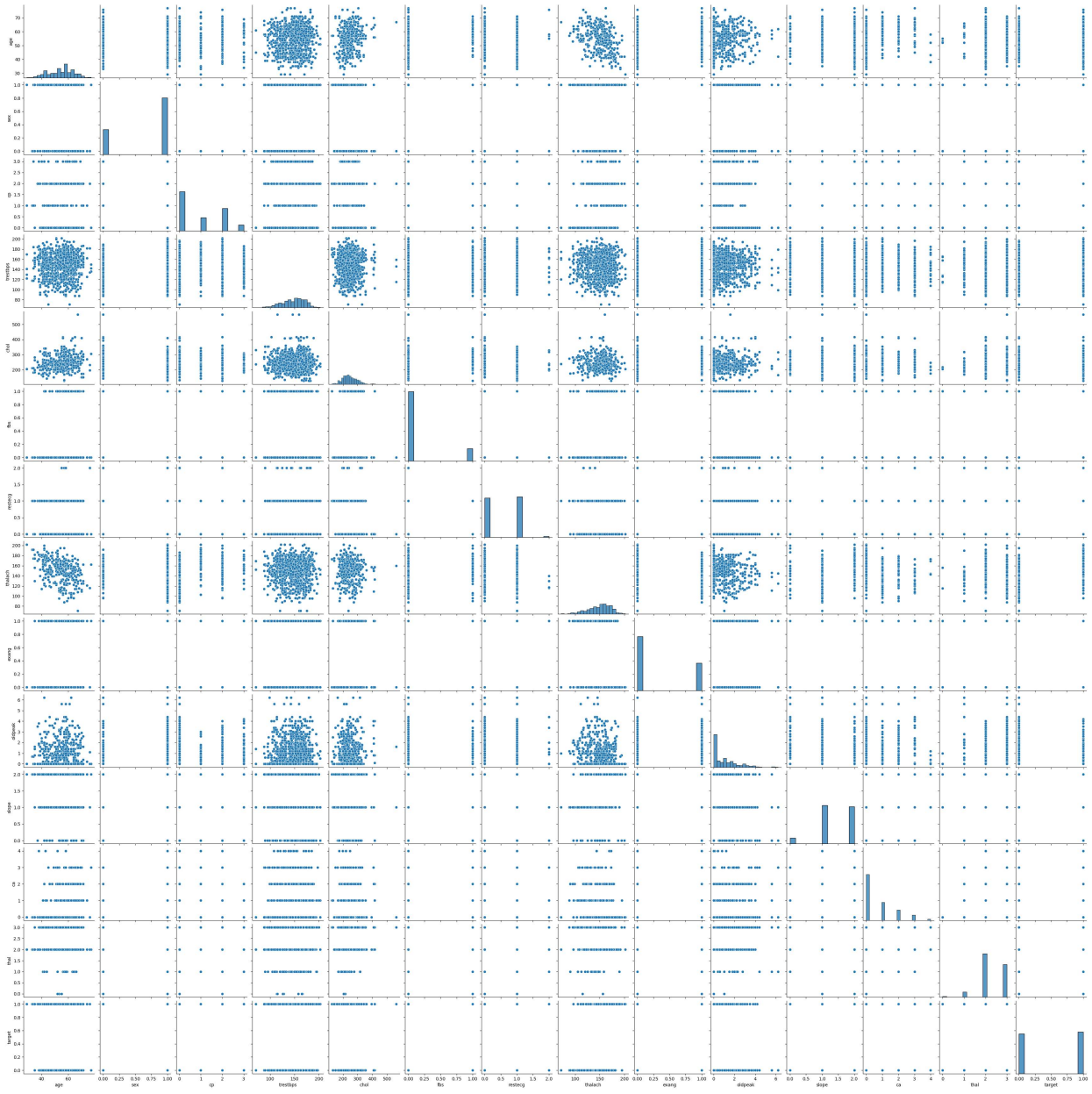


The Bar graph shows there is 110 Male and 70 Females are having chance of heart attack.

Show relationship between each other

In [10]:
```
sns.pairplot(df)
```

Out[10]:
```
<seaborn.axisgrid.PairGrid at 0x24c56d891e0>
```



It help us to visualize the relationships between different variables.

Splitting Data Into Features and Target

In [11]:
```
x=df.iloc[:,:-1]
x
```

Out[11]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal
0	51	1	0	125	212	0	1	168	0	1.0	2	2	3
1	54	1	0	140	203	1	0	155	1	3.1	0	0	3
2	71	1	0	145	174	0	1	125	1	2.6	0	0	3
3	60	1	0	148	203	0	1	161	0	0.0	2	1	3
4	61	0	0	138	294	1	1	106	0	1.9	1	3	2
...	...	...	...	...	...	...	...	...	...	...	...	...	...
1020	59	1	1	173	221	0	1	164	1	1.2	2	0	2
1021	60	1	0	174	258	0	0	141	1	1.4	1	1	3
1022	47	1	0	173	275	0	0	118	1	0.0	1	1	2
1023	50	0	0	165	254	0	0	159	0	3.8	2	0	2
1024	54	1	0	168	188	0	1	113	0	0.4	1	1	3

1025 rows × 13 columns

We are splitting overall data except target column that is "target" in x variable

In [12]:
```
y=df["target"]
y
```

Out[12]:
```
0       0
1       0
2       0
3       0
4       0
       ..
1020    1
1021    0
1022    0
1023    1
1024    0
Name: target, Length: 1025, dtype: int64
```

In that separate "target" column as a target in y variable.

Apply Standard Scaler to Scale the data at one level

In [13]:
```
from sklearn.preprocessing import StandardScaler
```

Standardization is the process of rescaling the features so that they have the properties of a standard normal distribution with a mean of 0 and a standard deviation of 1

In [14]:
```
from sklearn.model_selection import train_test_split
xtrain,xtest,ytrain,ytest=train_test_split(x,y,test_size=0.2,random_state=1)
```

train_test_split function from scikit-learn to split your dataset into training and testing sets. This is a common in machine learning to evaluate the performance of a model.

In that testing data is 20% and random state is 1.

In [15]:
```
sc=StandardScaler()
xtrain=sc.fit_transform(xtrain)
xtest=sc.fit_transform(xtest)
```

We make object of standard scaler. In that we have to pass xtrain and xtest for transform into standard scaler.

In [16]:
```
xtrain
```

Out[16]:
```
array([[ 0.27803227,  0.68138514, -0.91755375, ...,  0.99674422,
        -0.72634942,  1.11342872],
       [ 0.49959346,  0.68138514,  2.05633589, ...,  0.99674422,
         1.23611029, -0.5227927 ],
       [ 0.83193525, -1.46759877, -0.91755375, ..., -0.61534702,
        -0.72634942, -0.5227927 ],
       ...,
       [ 0.16725168,  0.68138514, -0.91755375, ..., -0.61534702,
         0.25488044, -0.5227927 ],
       [ 0.83193525,  0.68138514, -0.91755375, ..., -0.61534702,
         1.23611029,  1.11342872],
       [ 1.05349644,  0.68138514,  2.05633589, ..., -0.61534702,
        -0.72634942,  1.11342872]])
```

This is our standardize xtrain data on which we performed Standard Scaling

In [17]:
```
xtest
```

Out[17]:
```
array([[ 0.61998548,  0.58298309, -0.91440872, ...,  0.99064776,
         1.11030741,  1.00455772],
       [-0.36441487, -1.71531562,  0.89673899, ...,  0.99064776,
        -0.75537307, -0.52088178],
       [-1.34881522,  0.58298309, -0.00883487, ...,  0.99064776,
        -0.75537307, -0.52088178],
       ...,
       [-0.03628142,  0.58298309, -0.91440872, ..., -0.66043184,
         0.17746717,  1.00455772],
       [-1.78632649,  0.58298309,  0.89673899, ...,  0.99064776,
         2.97598789, -0.52088178],
       [ 0.07309639,  0.58298309, -0.91440872, ..., -0.66043184,
         0.17746717,  1.00455772]])
```

This is our standardize xtest data on which we performed Standard Scaling

Train_Test_Split for separating data into training and testing phase

In [18]:
```
from sklearn.model_selection import train_test_split
xtrain,xtest,ytrain,ytest=train_test_split(x,y,test_size=0.2,random_state=1)
```

train_test_split function from scikit-learn to split your dataset into training and testing sets. This is a common in machine learning to evaluate the performance of a model.

In that testing data is 20% and random state is 1.

Build a Model by using LogisticRegression Algoritham

In [19]:
```
from sklearn.linear_model import LogisticRegression
```

Logistic regression is a commonly used algorithm for binary classification problems, where the target variable has two possible classes.

In [20]:
```
lr=LogisticRegression()
lr.fit(xtrain,ytrain)
ypred_train=lr.predict(xtrain)
ypred_test=lr.predict(xtest)
```

Training the data on logistic reg and then making predictions on both the training and testing sets. This is a standard procedure in machine learning.

Classification Report of model trained on Logistic Regression

In [21]:
```
from sklearn.metrics import classification_report
```

In [22]:
```
print("Train Data")
print(classification_report(ytrain,ypred_train))
print("Test Data")
print(classification_report(ytest,ypred_test))
```

```
Train Data
              precision    recall  f1-score   support

           0       0.88      0.80      0.84       390
           1       0.83      0.90      0.87       430

    accuracy                           0.85       820
   macro avg       0.86      0.85      0.85       820
weighted avg       0.86      0.85      0.85       820

Test Data
              precision    recall  f1-score   support

           0       0.89      0.75      0.82       109
           1       0.76      0.90      0.82        96

    accuracy                           0.82       205
   macro avg       0.83      0.82      0.82       205
weighted avg       0.83      0.82      0.82       205
```

Classification_report function from scikit-learn to print the classification report for both the training and testing sets. The classification report provides a summary of various classification metrics, such as precision, recall, and F1-score, which can help us understand the performance of your model.

According to classification Report accuracy of training data is 85% and testing data is 82%. This is best fitted model.

By using Hyperparameter or Hypertunners

In [23]:
```
lr=LogisticRegression(solver="liblinear")
lr.fit(xtrain,ytrain)
ypred_train=lr.predict(xtrain)
ypred_test=lr.predict(xtest)
```

In [24]:
```
print("Train Data")
print(classification_report(ytrain,ypred_train))
print("Test Data")
print(classification_report(ytest,ypred_test))
```

```
Train Data
              precision    recall  f1-score   support

           0       0.87      0.80      0.83       390
           1       0.83      0.89      0.86       430

    accuracy                           0.85       820
   macro avg       0.85      0.85      0.85       820
weighted avg       0.85      0.85      0.85       820

Test Data
              precision    recall  f1-score   support

           0       0.88      0.74      0.81       109
           1       0.75      0.89      0.81        96

    accuracy                           0.81       205
   macro avg       0.82      0.81      0.81       205
weighted avg       0.82      0.81      0.81       205
```

In [25]:
```
lr=LogisticRegression(solver="sag")
lr.fit(xtrain,ytrain)
ypred_train=lr.predict(xtrain)
ypred_test=lr.predict(xtest)
```

In [26]:
```
print("Train Data")
print(classification_report(ytrain,ypred_train))
print("Test Data")
print(classification_report(ytest,ypred_test))
```

```
Train Data
              precision    recall  f1-score   support

           0       0.73      0.68      0.71       390
           1       0.73      0.77      0.75       430

    accuracy                           0.73       820
   macro avg       0.73      0.73      0.73       820
weighted avg       0.73      0.73      0.73       820

Test Data
              precision    recall  f1-score   support

           0       0.77      0.66      0.71       109
           1       0.67      0.78      0.72        96

    accuracy                           0.72       205
   macro avg       0.72      0.72      0.72       205
weighted avg       0.73      0.72      0.72       205
```

In [27]:
```
lr=LogisticRegression(solver="saga")
lr.fit(xtrain,ytrain)
ypred_train=lr.predict(xtrain)
ypred_test=lr.predict(xtest)
```

In [28]:
```
print("Train Data")
print(classification_report(ytrain,ypred_train))
print("Test Data")
print(classification_report(ytest,ypred_test))
```

```
Train Data
              precision    recall  f1-score   support

           0       0.71      0.67      0.69       390
           1       0.71      0.75      0.73       430

    accuracy                           0.71       820
   macro avg       0.71      0.71      0.71       820
weighted avg       0.71      0.71      0.71       820

Test Data
              precision    recall  f1-score   support

           0       0.74      0.63      0.68       109
           1       0.64      0.75      0.69        96

    accuracy                           0.69       205
   macro avg       0.69      0.69      0.69       205
weighted avg       0.70      0.69      0.69       205
```

The solver parameter in scikit-learn's LogisticRegression specifies the algorithm to use in the optimization problem, and liblinear,sag and saga is one of the options.

Build a model by using KNN Algoritham

In [29]:
```
from sklearn.neighbors import KNeighborsClassifier
```

KNeighborsClassifier from scikit-learn, which is a classifier based on k-nearest neighbors. K-nearest neighbors is a type of instance-based learning or lazy learning

In [30]:
```
knn=KNeighborsClassifier(n_neighbors=5)
knn.fit(xtrain,ytrain)
ypred_train=knn.predict(xtrain)
ypred_test=knn.predict(xtest)
```

KNeighborsClassifier with n_neighbors=5, fitted it on your training data, and then made predictions on both the training and testing sets. This is a common procedure when working with the k-nearest neighbors algorithm.

Evaluate the model by using Classification_report

In [31]:
```
from sklearn.metrics import classification_report
```

In [32]:
```
print("Train Data")
print(classification_report(ytrain,ypred_train))
print("Test Data")
print(classification_report(ytest,ypred_test))
```

```
Train Data
              precision    recall  f1-score   support

           0       0.81      0.77      0.79       390
           1       0.80      0.83      0.82       430

    accuracy                           0.80       820
   macro avg       0.80      0.80      0.80       820
weighted avg       0.80      0.80      0.80       820

Test Data
              precision    recall  f1-score   support

           0       0.73      0.61      0.66       109
           1       0.62      0.74      0.68        96

    accuracy                           0.67       205
   macro avg       0.67      0.67      0.67       205
weighted avg       0.68      0.67      0.67       205
```

According to classification Report accuracy of training data is 80% and testing data is 67%. This is best fitted model.

Build a model on SVM Algoritham

In [33]:
```
from sklearn.svm import SVC
```

SVC (Support Vector Classification) class from scikit-learn. The SVC is a powerful algorithm commonly used for classification tasks and it's based on the concept of support vectors within a high-dimensional space.

In [34]:
```
svm=SVC()

def mymodel(model):
    model.fit(xtrain,ytrain)
    ypred=model.predict(xtest)
    print(classification_report(ytest,ypred))
    return model
```

Defined a function mymodel that takes a model as an argument, fits the model on the training data, makes predictions on the testing data, and prints the classification report.

In [35]:
```
from sklearn.pipeline import Pipeline
```

We using pipeline beacause data should be process smoothly.

In [36]:
```
pipe=Pipeline(steps=[('scaler',StandardScaler()),('svm',SVC())])
```

In pipe we write steps so that steps will be executed one by one

In [37]:
```
pipe.fit(xtrain,ytrain)
ypred=pipe.predict(xtest)
```

Fitting the entire pipeline (pipe) on the training data (xtrain, ytrain) and then making predictions on the testing data (xtest).

In [38]:
```
mymodel(svm)
```

```
              precision    recall  f1-score   support

           0       0.75      0.57      0.65       109
           1       0.61      0.78      0.69        96

    accuracy                           0.67       205
   macro avg       0.68      0.68      0.67       205
weighted avg       0.69      0.67      0.67       205
```

Out[38]:
```
SVC()
```

By using Hyperparameter or Hypertunners

In [39]:
```
svm=SVC(kernel='linear')
mymodel(svm)
```

```
              precision    recall  f1-score   support

           0       0.91      0.74      0.82       109
           1       0.76      0.92      0.83        96

    accuracy                           0.82       205
   macro avg       0.83      0.83      0.82       205
weighted avg       0.84      0.82      0.82       205
```

Out[39]:
```
SVC(kernel='linear')
```

By using linear kernel i.e hypertunning parameter we are having accuracy 82%.

In [40]:
```
svm=SVC(kernel='sigmoid')
mymodel(svm)
```

```
              precision    recall  f1-score   support

           0       0.58      0.45      0.51       109
           1       0.50      0.62      0.56        96

    accuracy                           0.53       205
   macro avg       0.54      0.54      0.53       205
weighted avg       0.54      0.53      0.53       205
```

Out[40]:
```
SVC(kernel='sigmoid')
```

By using sigmoid kernel i.e hypertunning parameter we are having accuracy 53%. Its very low accuracy.

In [41]:
```
svm=SVC(kernel='poly')
mymodel(svm)
```

```
              precision    recall  f1-score   support

           0       0.75      0.59      0.66       109
           1       0.62      0.78      0.69        96

    accuracy                           0.68       205
   macro avg       0.69      0.68      0.68       205
weighted avg       0.69      0.68      0.68       205
```

Out[41]:
```
SVC(kernel='poly')
```

By using poly kernel i.e hypertunning parameter we are having accuracy 68%.

Build a model by using Decision Tree Classifier

In [42]:
```
from sklearn.tree import DecisionTreeClassifier
```

We are importing the DecisionTreeClassifier from scikit-learn.

In [43]:
```
dt=DecisionTreeClassifier()
dt.fit(xtrain,ytrain)
ytrain_pred=dt.predict(xtrain)
ytest_pred=dt.predict(xtest)
```

We are created a DecisionTreeClassifier, fitted it on your training data, and then made predictions on both the training and testing sets.

Evaluate the model by using Classification_report

In [44]:
```
from sklearn.metrics import classification_report,accuracy_score
```

In [45]:
```
print("Train Data")
print(classification_report(ytrain,ytrain_pred))
print("Test Data")
print(classification_report(ytest,ytest_pred))
```

```
Train Data
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       390
           1       1.00      1.00      1.00       430

    accuracy                           1.00       820
   macro avg       1.00      1.00      1.00       820
weighted avg       1.00      1.00      1.00       820

Test Data
              precision    recall  f1-score   support

           0       0.93      0.94      0.93       109
           1       0.93      0.92      0.92        96

    accuracy                           0.93       205
   macro avg       0.93      0.93      0.93       205
weighted avg       0.93      0.93      0.93       205
```

According to classification Report accuracy of training data is 100% its high bias and testing data is 95% its variance is high. Beacause of this scenario it is overfitted model.

Performing Hypertunning on model

Hyper parameters with impurity checking gini

In [46]:
```
for i in range(1,31):
    dt1=DecisionTreeClassifier(max_depth=i)
    dt1.fit(xtrain,ytrain)
    ypred=dt1.predict(xtest)
    ac=accuracy_score(ytest,ypred)
    print(f"Max Depth:{i} accuracy): {ac}")
```

```
Max Depth:1 accuracy): 0.7609756097560976
Max Depth:2 accuracy): 0.7609756097560976
Max Depth:3 accuracy): 0.824390243902439
Max Depth:4 accuracy): 0.8292682926829268
Max Depth:5 accuracy): 0.8634146341463415
Max Depth:6 accuracy): 0.8878048780487805
Max Depth:7 accuracy): 0.9121951219512195
Max Depth:8 accuracy): 0.9317073170731708
Max Depth:9 accuracy): 0.9170731707317074
Max Depth:10 accuracy): 0.9463414634146341
Max Depth:11 accuracy): 0.9463414634146341
Max Depth:12 accuracy): 0.9414634146341463
Max Depth:13 accuracy): 0.9365853658536586
Max Depth:14 accuracy): 0.9414634146341463
Max Depth:15 accuracy): 0.9463414634146341
Max Depth:16 accuracy): 0.9463414634146341
Max Depth:17 accuracy): 0.9414634146341463
Max Depth:18 accuracy): 0.9463414634146341
Max Depth:19 accuracy): 0.9414634146341463
Max Depth:20 accuracy): 0.9512195121951219
Max Depth:21 accuracy): 0.9463414634146341
Max Depth:22 accuracy): 0.9414634146341463
Max Depth:23 accuracy): 0.9414634146341463
Max Depth:24 accuracy): 0.9317073170731708
Max Depth:25 accuracy): 0.9317073170731708
Max Depth:26 accuracy): 0.9317073170731708
Max Depth:27 accuracy): 0.9463414634146341
Max Depth:28 accuracy): 0.9463414634146341
Max Depth:29 accuracy): 0.9414634146341463
Max Depth:30 accuracy): 0.9365853658536586
```

Using a loop to train Decision Tree models with max_depth and evaluate their accuracy on the testing set. This can be useful for finding the optimal value for the max_depth hyperparameter, which controls the maximum depth of the decision tree.

Building model with final value of max depth as 8

In [47]:
```
dt1=DecisionTreeClassifier(max_depth=9)
mymodel(dt1)
```

```
              precision    recall  f1-score   support

           0       0.93      0.94      0.93       109
           1       0.93      0.92      0.92        96

    accuracy                           0.93       205
   macro avg       0.93      0.93      0.93       205
weighted avg       0.93      0.93      0.93       205
```

Out[47]:
```
DecisionTreeClassifier
DecisionTreeClassifier(max_depth=9)
```

We are created a DecisionTreeClassifier with a specified maximum depth of 8 and then used your mymodel function to fit the model on the training data and evaluate its performance on the testing data.

Lets checking overfitting scenario

In [48]:
```
dt1.score(xtrain,ytrain)
```

Out[48]:
```
0.998780487804878
```

To check for overfitting, we can compare the accuracy of our model on the training set (xtrain, ytrain) with its accuracy on the testing set (xtest, ytest).

Here the accuracy of the trainning data is high as compaired to testing data, this model is overfitting.

Hyper parameters with impurity checking min_sample_split

In [49]:
```
for i in range(5,50):
    dt2=DecisionTreeClassifier(min_samples_split=i)
    dt2.fit(xtrain,ytrain)
    ypred=dt2.predict(xtest)
    ac=accuracy_score(ytest,ypred)
    print(f"min sample split:{i} accuracy): {ac}")
```

```
min sample split:5 accuracy): 0.926829268292683
min sample split:6 accuracy): 0.9024390243902439
min sample split:7 accuracy): 0.9121951219512195
min sample split:8 accuracy): 0.9121951219512195
min sample split:9 accuracy): 0.9121951219512195
min sample split:10 accuracy): 0.9121951219512195
min sample split:11 accuracy): 0.9024390243902439
min sample split:12 accuracy): 0.9073170731707317
min sample split:13 accuracy): 0.9073170731707317
min sample split:14 accuracy): 0.9073170731707317
min sample split:15 accuracy): 0.9073170731707317
min sample split:16 accuracy): 0.9073170731707317
min sample split:17 accuracy): 0.9073170731707317
min sample split:18 accuracy): 0.9121951219512195
min sample split:19 accuracy): 0.9073170731707317
min sample split:20 accuracy): 0.8829268292682927
min sample split:21 accuracy): 0.8829268292682927
min sample split:22 accuracy): 0.8829268292682927
min sample split:23 accuracy): 0.8682926829268293
min sample split:24 accuracy): 0.8682926829268293
min sample split:25 accuracy): 0.8682926829268293
min sample split:26 accuracy): 0.8682926829268293
min sample split:27 accuracy): 0.8682926829268293
min sample split:28 accuracy): 0.8682926829268293
min sample split:29 accuracy): 0.8634146341463415
min sample split:30 accuracy): 0.8682926829268293
min sample split:31 accuracy): 0.8682926829268293
min sample split:32 accuracy): 0.8634146341463415
min sample split:33 accuracy): 0.8634146341463415
min sample split:34 accuracy): 0.8634146341463415
min sample split:35 accuracy): 0.8634146341463415
min sample split:36 accuracy): 0.8634146341463415
min sample split:37 accuracy): 0.8682926829268293
min sample split:38 accuracy): 0.8682926829268293
min sample split:39 accuracy): 0.8682926829268293
min sample split:40 accuracy): 0.8634146341463415
min sample split:41 accuracy): 0.8634146341463415
min sample split:42 accuracy): 0.8634146341463415
min sample split:43 accuracy): 0.8634146341463415
min sample split:44 accuracy): 0.8634146341463415
min sample split:45 accuracy): 0.8634146341463415
min sample split:46 accuracy): 0.8634146341463415
min sample split:47 accuracy): 0.8634146341463415
min sample split:48 accuracy): 0.8634146341463415
min sample split:49 accuracy): 0.8634146341463415
```

Our code is iterating over different values for the min_samples_split parameter of the DecisionTreeClassifier and printing the accuracy on the testing set for each configuration.

Building model with final value of min_samples_split as 5

In [50]:
```
dt2=DecisionTreeClassifier(min_samples_split=5)
mymodel(dt2)
```

```
              precision    recall  f1-score   support

           0       0.92      0.94      0.93       109
           1       0.93      0.91      0.92        96

    accuracy                           0.92       205
   macro avg       0.92      0.92      0.92       205
weighted avg       0.92      0.92      0.92       205
```

Out[50]:

```
DecisionTreeClassifier(min_samples_split=5)
```

We are created a DecisionTreeClassifier with a specified value for the min_samples_split parameter in this case, min_samples_split=5, and then we are used our mymodel function to fit the model on the training data and evaluate its performance on the testing data.

Lets checking overfitting scenario

In [51]:
```
dt2.score(xtrain,ytrain)
```

Out[51]:
```
0.9939024390243902
```

To check for overfitting, we can compare the accuracy of our model on the training set (xtrain, ytrain) with its accuracy on the testing set (xtest, ytest).

Here the accuracy of the trainning data is high as compaired to testing data, this model is overfitting.

Hyper parameters with impurity checking min_sample_leaf

In [52]:
```
for i in range(5,51):
    dt3=DecisionTreeClassifier(min_samples_leaf=i)
    dt3.fit(xtrain,ytrain)
    ypred=dt3.predict(xtest)
    ac=accuracy_score(ytest,ypred)
    print(f"min sample leaf:{i} accuracy): {ac}")
```

```
min sample leaf:5 accuracy): 0.8731707317073171
min sample leaf:6 accuracy): 0.8926829268292683
min sample leaf:7 accuracy): 0.8829268292682927
min sample leaf:8 accuracy): 0.8780487804878049
min sample leaf:9 accuracy): 0.8682926829268293
min sample leaf:10 accuracy): 0.8829268292682927
min sample leaf:11 accuracy): 0.8731707317073171
min sample leaf:12 accuracy): 0.8682926829268293
min sample leaf:13 accuracy): 0.8682926829268293
min sample leaf:14 accuracy): 0.8682926829268293
min sample leaf:15 accuracy): 0.8682926829268293
min sample leaf:16 accuracy): 0.8682926829268293
min sample leaf:17 accuracy): 0.8682926829268293
min sample leaf:18 accuracy): 0.8780487804878049
min sample leaf:19 accuracy): 0.8878048780487805
min sample leaf:20 accuracy): 0.8780487804878049
min sample leaf:21 accuracy): 0.8780487804878049
min sample leaf:22 accuracy): 0.8780487804878049
min sample leaf:23 accuracy): 0.8780487804878049
min sample leaf:24 accuracy): 0.8780487804878049
min sample leaf:25 accuracy): 0.8780487804878049
min sample leaf:26 accuracy): 0.8780487804878049
min sample leaf:27 accuracy): 0.8780487804878049
min sample leaf:28 accuracy): 0.8731707317073171
min sample leaf:29 accuracy): 0.8731707317073171
min sample leaf:30 accuracy): 0.8536585365853658
min sample leaf:31 accuracy): 0.8536585365853658
min sample leaf:32 accuracy): 0.8536585365853658
min sample leaf:33 accuracy): 0.8536585365853658
min sample leaf:34 accuracy): 0.8536585365853658
min sample leaf:35 accuracy): 0.8536585365853658
min sample leaf:36 accuracy): 0.8536585365853658
min sample leaf:37 accuracy): 0.8536585365853658
min sample leaf:38 accuracy): 0.8536585365853658
min sample leaf:39 accuracy): 0.8536585365853658
min sample leaf:40 accuracy): 0.8341463414634146
min sample leaf:41 accuracy): 0.8341463414634146
min sample leaf:42 accuracy): 0.8341463414634146
min sample leaf:43 accuracy): 0.8146341463414634
min sample leaf:44 accuracy): 0.8146341463414634
min sample leaf:45 accuracy): 0.8146341463414634
min sample leaf:46 accuracy): 0.8146341463414634
min sample leaf:47 accuracy): 0.8146341463414634
min sample leaf:48 accuracy): 0.8146341463414634
min sample leaf:49 accuracy): 0.8146341463414634
min sample leaf:50 accuracy): 0.8146341463414634
```

Our code iterates over different values for the min_samples_leaf parameter of the DecisionTreeClassifier and prints the accuracy on the testing set for each configuration.

Building model with final value of max depth as 6

In [53]:
```
dt3=DecisionTreeClassifier(min_samples_split=6)
mymodel(dt3)
```

```
              precision    recall  f1-score   support

           0       0.91      0.92      0.91       109
           1       0.91      0.90      0.90        96

    accuracy                           0.91       205
   macro avg       0.91      0.91      0.91       205
weighted avg       0.91      0.91      0.91       205
```

Out[53]:

```
DecisionTreeClassifier(min_samples_split=6)
```

Lets checking overfitting scenario

In [54]:
```
dt3.score(xtrain,ytrain)
```

Out[54]:
```
0.9841463414634146
```

To check for overfitting, we can compare the accuracy of our model on the training set (xtrain, ytrain) with its accuracy on the testing set (xtest, ytest).

Here the accuracy of the trainning data is high as compaired to testing data, this model is overfitting.

Voting

In [55]:
```
from sklearn.ensemble import VotingClassifier
```

The VotingClassifier in scikit-learn is a meta-estimator that combines the predictions of multiple machine learning models. It allows to combine different classifiers and use a majority vote or weighted voting to make predictions.

In [58]:
```
model=[]
accuracy=[]
model.append(("Logistic Regression",LogisticRegression()))
model.append(("Decision Tree",DecisionTreeClassifier()))
```

We are creating a list named model and appending tuples to it, where each tuple contains a model's name and an instance of that model.

In [59]:
```
model
```

Out[59]:
```
[('Logistic Regression', LogisticRegression()),
 ('Decision Tree', DecisionTreeClassifier())]
```

In [60]:
```
vc=VotingClassifier(estimators=model)
vc.fit(xtrain,ytrain)
ypred_train=vc.predict(xtrain)
ypred_test=vc.predict(xtest)
```

We are created a VotingClassifier using the list of models we defined earlier and trained it on our training data (xtrain, ytrain). We are also made predictions on both the training set (ypred_train) and the test set (ypred_test).

Classification Report

In [61]:
```
print("Train Data")
print(classification_report(ytrain,ypred_train))
print("Test Data")
print(classification_report(ytest,ypred_test))
```

```
Train Data
              precision    recall  f1-score   support

           0       0.90      1.00      0.95       390
           1       1.00      0.90      0.95       430

    accuracy                           0.95       820
   macro avg       0.95      0.95      0.95       820
weighted avg       0.95      0.95      0.95       820

Test Data
              precision    recall  f1-score   support

           0       0.89      0.98      0.93       109
           1       0.98      0.86      0.92        96

    accuracy                           0.93       205
   macro avg       0.93      0.92      0.93       205
weighted avg       0.93      0.93      0.93       205
```

The Best Prediction was done by model trained on Logistic Regression Algoritham with accuracy of 85% on Train Data and 82% on Test Data.

In [ ]: