Heart Disease Prediction Using Machine Learning

Group Members:

Aflah Sedhique (AM.EN.U4CSE20105)

Akshay G Sree (AM.EN.U4CSE20107)

C P Ghanshyam (AM.EN.U4CSE20118)

Gourinath (AM.EN.U4CSE20129)

Pranav B Nair (AM.EN.U4CSE20152**)**

Problem Definition

Machine Learning is used across many spheres around the world. The healthcare industry is no exception. Machine Learning can play an essential role in predicting presence/absence of Locomotor disorders, Heart diseases and more. Such information, if predicted well in advance, can provide important insights to doctors who can then adapt their diagnosis and treatment per patient basis.

This project is to predict whether a person is having a heart disease or not based on his health conditions like blood pressure, cholesterol etc.

Datasets:

Dataset -1:

Source: Kaggle

Link: Dataset-1

Description:

This dataset contains health information of different person to predict whether a person contains heart disease or not. It contains 13 attributes like age, sex, chest pair, blood pressure , cholesterol , FBS over 120 , EKG results , max HR , Exercise angina , ST depression , Slope of ST , Number of vessels fluro , Thallium.

This dataset was earlier used in a program to detect whether a person is in the risk of having a stroke or not.

Dataset-2:

Source: Kaggle

Link: Dataset-2

Description:

This dataset contains 76 attributes , all published experiments refer to using a subset of 14 of them. It has been used by ML researchers to this date. The “goal” field refers to the presence of heart disease in the patient. It is integer- valued from 0 to 4.

Dataset-3:

Source: Kaggle

Link: Dataset-3

Description:

This dataset is from a study of heart disease that has been open to the public for many years. The study collects various measurements on patient health and cardiovascular statistics, and of course makes patient identities anonymous.

Data Preprocessing

Dataset Used: Dataset

- **Importing Necessary Libraries and loading the dataset.**

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

```
In [3]: #Loading the dataset

        data=pd.read_csv('Heart_Disease_Prediction.csv')

        #Verifying the output by printing first5 rows of the dataset

        data.head(6)
```

Out[3]:

	Age	Sex	Chest pain type	BP	Cholesterol	FBS over 120	EKG results	Max HR	Exercise angina	ST depression	Slope of ST	Number of vessels fluro	Thallium	Heart Disease
0	70	1	4	130	322	0	2	109	0	2.4	2	3	3	Presence
1	67	0	3	115	564	0	2	160	0	1.6	2	0	7	Absence
2	57	1	2	124	261	0	0	141	0	0.3	1	0	7	Presence
3	64	1	4	128	263	0	0	105	1	0.2	2	1	7	Absence
4	74	0	2	120	269	0	2	121	1	0.2	1	1	3	Absence
5	65	1	4	120	177	0	0	140	0	0.4	1	0	7	Absence

- **Checking the size of the dataset**

```
In [4]: data.shape
Out[4]: (270, 14)
```

Inference: This dataset contains 270 rows and 14 columns including the target variable.

- **Checking the type of each feature in the dataset**

```
In [5]: data.dtypes
```

```
Out[5]: Age                       int64
        Sex                       int64
        Chest pain type           int64
        BP                        int64
        Cholesterol               int64
        FBS over 120              int64
        EKG results               int64
        Max HR                    int64
        Exercise angina           int64
        ST depression           float64
        Slope of ST               int64
        Number of vessels fluro   int64
        Thallium                  int64
        Heart Disease            object
        dtype: object
```

Inference: Out of the 14 features 12 features are of int types , 1 feature (ST Depression) is of float type and the target variable is of type string

- **Checking if the dataset contains any missing data**

```
In [8]: data.isnull()
```

Out[8]:

	Age	Sex	Chest pain type	BP	Cholesterol	FBS over 120	EKG results	Max HR	Exercise angina	ST depression	Slope of ST	Number of vessels fluro	Thallium	Heart Disease
0	False	False	False	False	False	False	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
265	False	False	False	False	False	False	False	False	False	False	False	False	False	False
266	False	False	False	False	False	False	False	False	False	False	False	False	False	False
267	False	False	False	False	False	False	False	False	False	False	False	False	False	False
268	False	False	False	False	False	False	False	False	False	False	False	False	False	False
269	False	False	False	False	False	False	False	False	False	False	False	False	False	False

270 rows × 14 columns

Inference: The dataset does not contain any missing data.

- **Converting target label to binary values using one hot encoding**

```
dummies = pd.get_dummies(data['Heart Disease'])
merged_data = pd.concat([data,dummies],axis=1)
new_data = merged_data.drop(['Heart Disease'], axis=1)
new_data = new_data.drop(['Absence'], axis=1)
new_data.rename(columns = {'Presence':'Heart Disease Predicted'}, inplace = True)
```

Output:

```
new_data.head(5)
```

	Age	Sex	Chest pain type	BP	Cholesterol	FBS over 120	EKG results	Max HR	Exercise angina	ST depression	Slope of ST	Number of vessels fluro	Thallium	Heart Disease Predicted
0	70	1	4	130	322	0	2	109	0	2.4	2	3	3	1
1	67	0	3	115	564	0	2	160	0	1.6	2	0	7	0
2	57	1	2	124	261	0	0	141	0	0.3	1	0	7	1
3	64	1	4	128	263	0	0	105	1	0.2	2	1	7	0
4	74	0	2	120	269	0	2	121	1	0.2	1	1	3	0

Type of each feature after one hot encoding:

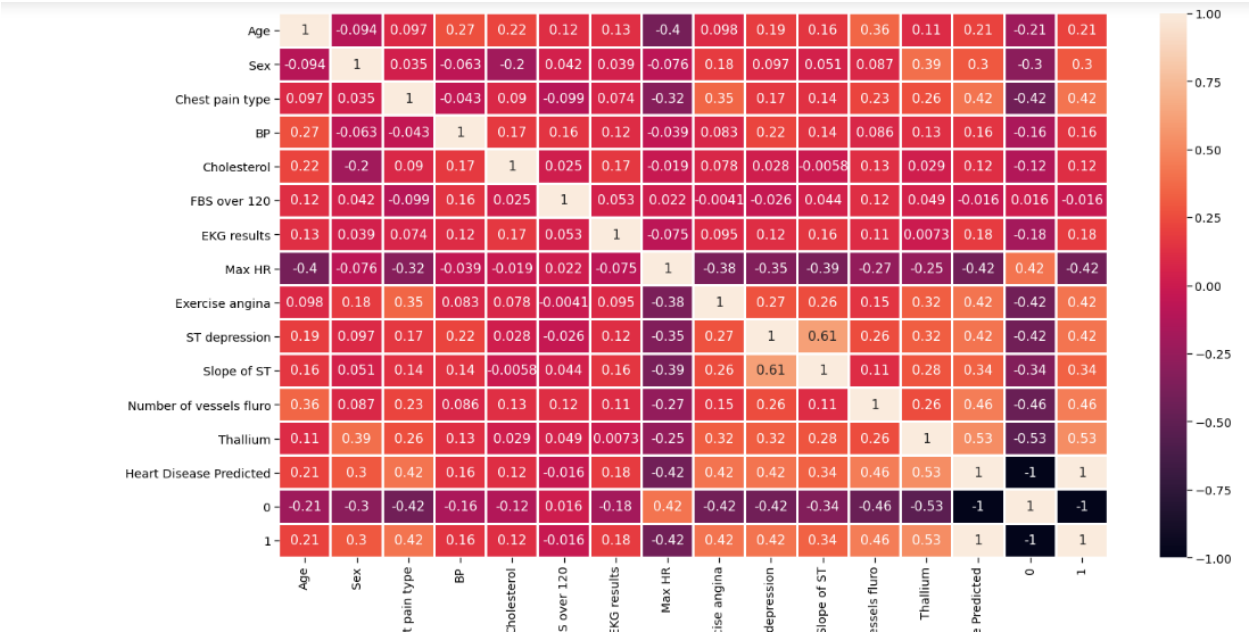
```
new_data.dtypes
```

```
Age                         int64
Sex                         int64
Chest pain type             int64
BP                          int64
Cholesterol                 int64
FBS over 120                int64
EKG results                 int64
Max HR                      int64
Exercise angina             int64
ST depression             float64
Slope of ST                 int64
Number of vessels fluro     int64
Thallium                    int64
Heart Disease Predicted     uint8
dtype: object
```

Data Summarization:

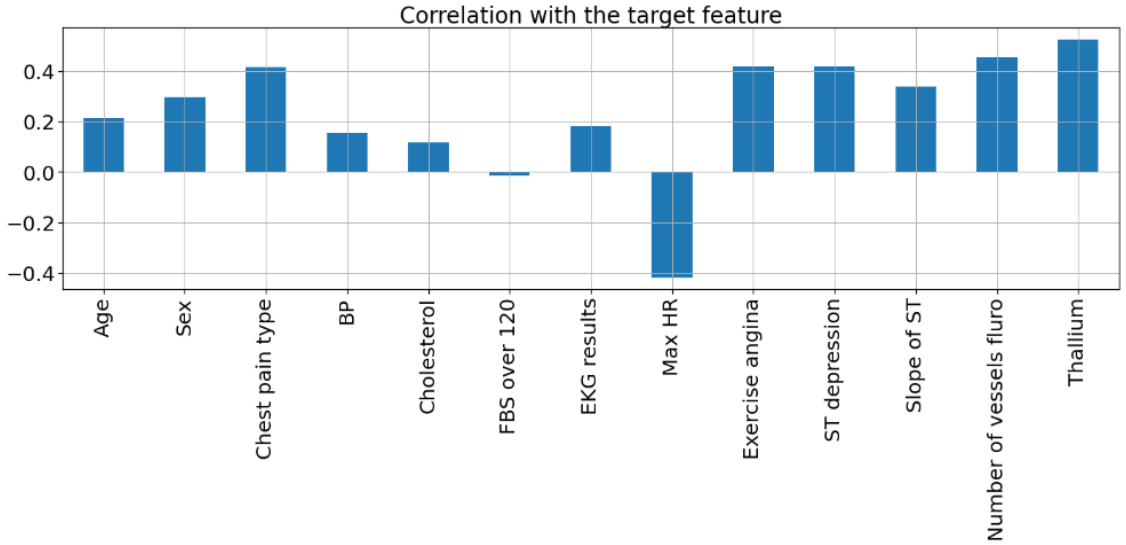
- **Checking the correlation between various features of the dataset.**

```
import seaborn as sns

plt.figure(figsize=(20,12))
sns.set_context('notebook',font_scale = 1.3)
sns.heatmap(new_data.corr(),annot=True,linewidth =2)
plt.tight_layout()
```



- **Checking the correlation of the target variable**

```
sns.set_context('notebook',font_scale = 2.3)
new_data.drop('Heart Disease Predicted', axis=1).corrwith(data['Heart Disease Predicted']).plot(kind='bar', grid=True,
                                                    figsize=(20, 10), title="Correlation with the target feature")
plt.tight_layout()
```



Inference: 1 feature (“Max Hr”) is negatively correlated with the target feature and all other features are positively correlated with the target feature.

- **Understanding the statistical details of the data**

```
data.describe()
```

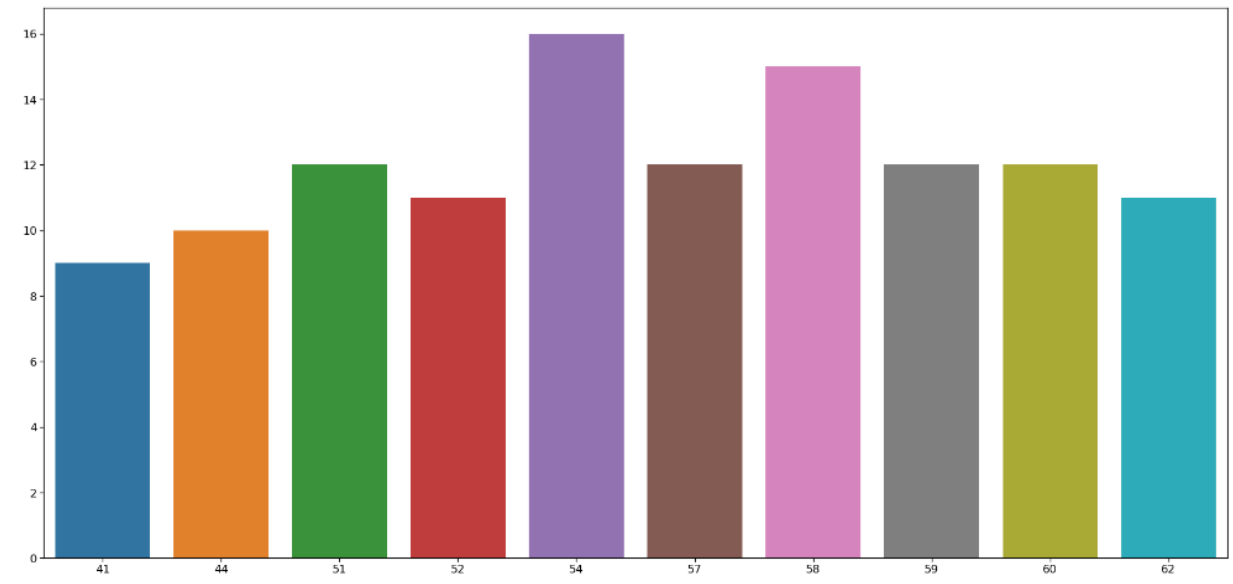
	Age	Sex	Chest pain type	BP	Cholesterol	FBS over 120	EKG results	Max HR	Exercise angina	ST depression	Slope of ST	Number of vessels fluro	Thallium	Heart Disease Predicted
count	270.000000	270.000000	270.000000	270.000000	270.000000	270.000000	270.000000	270.000000	270.000000	270.00000	270.000000	270.000000	270.000000	270.000000
mean	54.433333	0.677778	3.174074	131.344444	249.659259	0.148148	1.022222	149.677778	0.329630	1.05000	1.585185	0.670370	4.696296	0.444444
std	9.109067	0.468195	0.950090	17.861608	51.686237	0.355906	0.997891	23.165717	0.470952	1.14521	0.614390	0.943896	1.940659	0.497827
min	29.000000	0.000000	1.000000	94.000000	126.000000	0.000000	0.000000	71.000000	0.000000	0.00000	1.000000	0.000000	3.000000	0.000000
25%	48.000000	0.000000	3.000000	120.000000	213.000000	0.000000	0.000000	133.000000	0.000000	0.00000	1.000000	0.000000	3.000000	0.000000
50%	55.000000	1.000000	3.000000	130.000000	245.000000	0.000000	2.000000	153.500000	0.000000	0.80000	2.000000	0.000000	3.000000	0.000000
75%	61.000000	1.000000	4.000000	140.000000	280.000000	0.000000	2.000000	166.000000	1.000000	1.60000	2.000000	1.000000	7.000000	1.000000
max	77.000000	1.000000	4.000000	200.000000	564.000000	1.000000	2.000000	202.000000	1.000000	6.20000	3.000000	3.000000	7.000000	1.000000

Data Visualization:

➢ **Analyzing Individual features of the dataset.**

- **“Age” Feature Analysis (Checking 10 ages with their counts)**

```
plt.figure(figsize=(25,12))
sns.set_context('notebook',font_scale = 1.5)
sns.barplot(x=new_data['Age'].value_counts()[:10].index,y=new_data['Age'].value_counts()[:10].values)
plt.tight_layout()
```



Inference: 58 Age column has the highest frequency

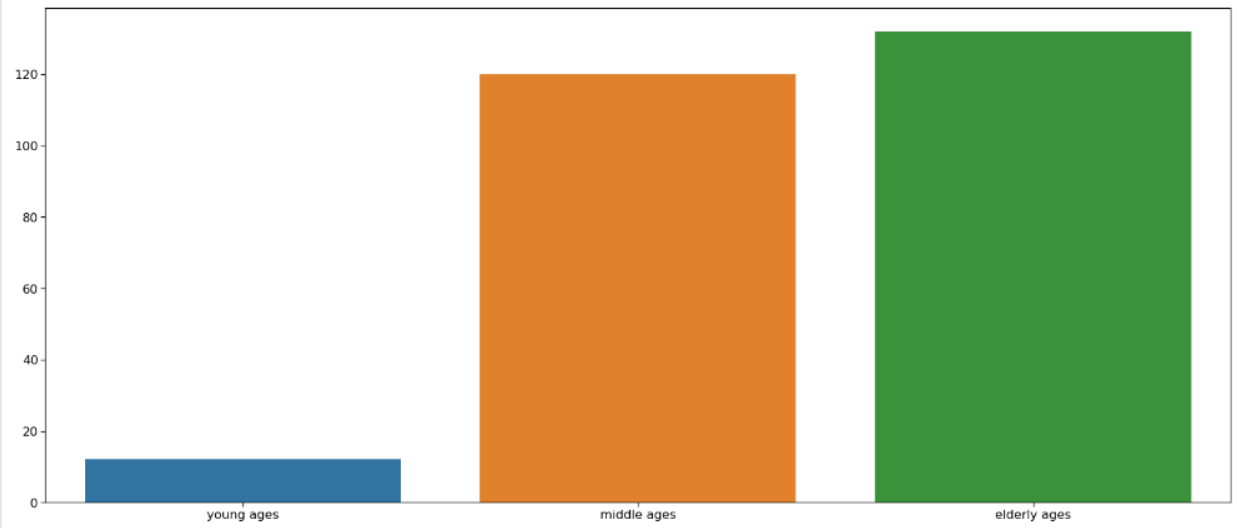
- Checking the range of ages in the dataset

```
minAge=min(new_data['Age'])
maxAge=max(new_data['Age'])
meanAge=new_data['Age'].mean()
print('Minimum Age :',minAge)
print('Maximum Age :',maxAge)
print('Mean Age :',meanAge)
```

```
Minimum Age : 29
Maximum Age : 77
Mean Age : 54.43333333333333
```

- Dividing Age feature into three different categories and plotting the graph

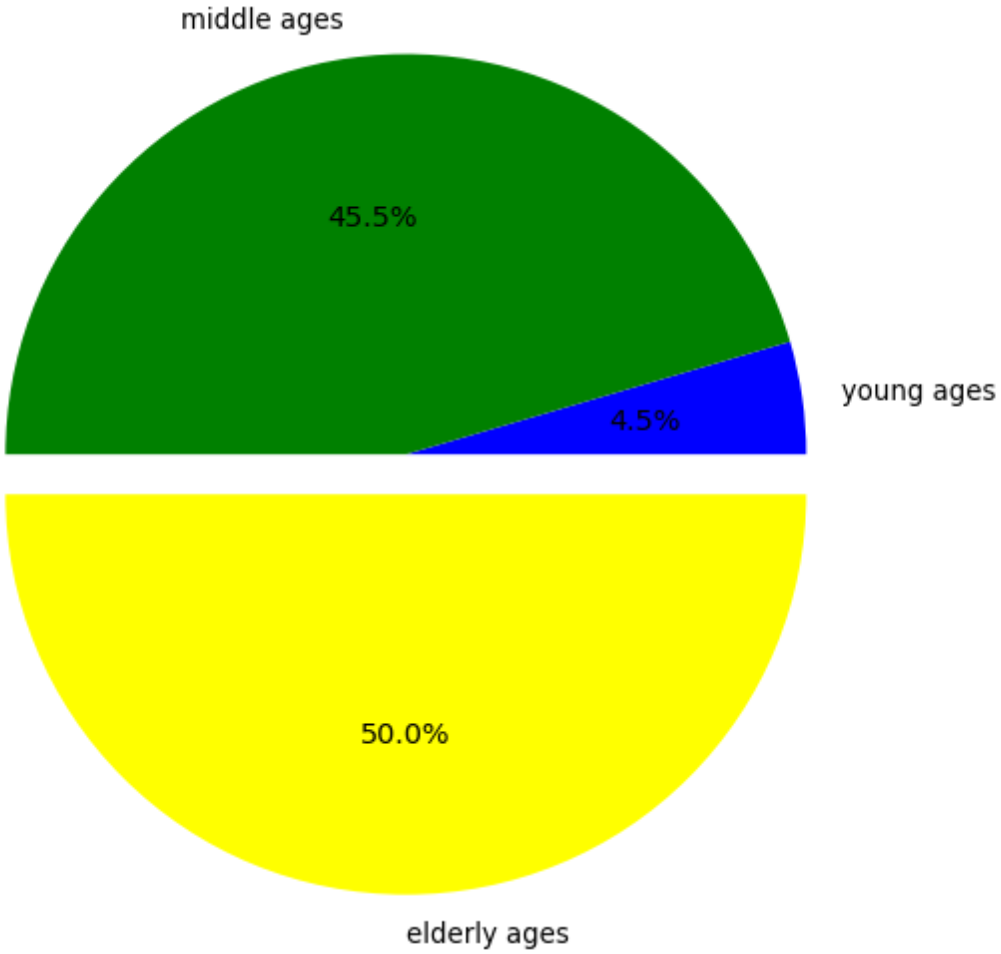
```
Young = new_data[(new_data['Age']>=29)&(new_data['Age']<40)]
Middle = new_data[(new_data['Age']>=40)&(new_data['Age']<55)]
Elder = new_data[(new_data['Age']>55)]

plt.figure(figsize=(23,10))
sns.set_context('notebook',font_scale = 1.5)
sns.barplot(x=['young ages','middle ages','elderly ages'],y=[len(Young),len(Middle),len(Elder)])
plt.tight_layout()
```



Inference: From the above graph we can infer that elder people are the most affected by the heart disease and the younger ones are the least affected,

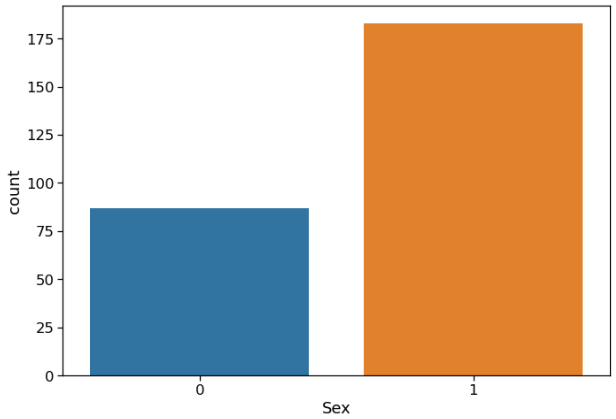
- **Plotting pie chart of people belonging to different age categories**

```
colors = ['blue','green','yellow']
explode = [0,0,0.1]
plt.figure(figsize=(10,10))
sns.set_context('notebook',font_scale = 1.2)
plt.pie([len(Young),len(Middle),len(Elder)],labels=['young ages','middle ages','elderly ages'],explode=explode,colors=colors
plt.tight_layout()
```



- **“Sex” Feature Analysis**

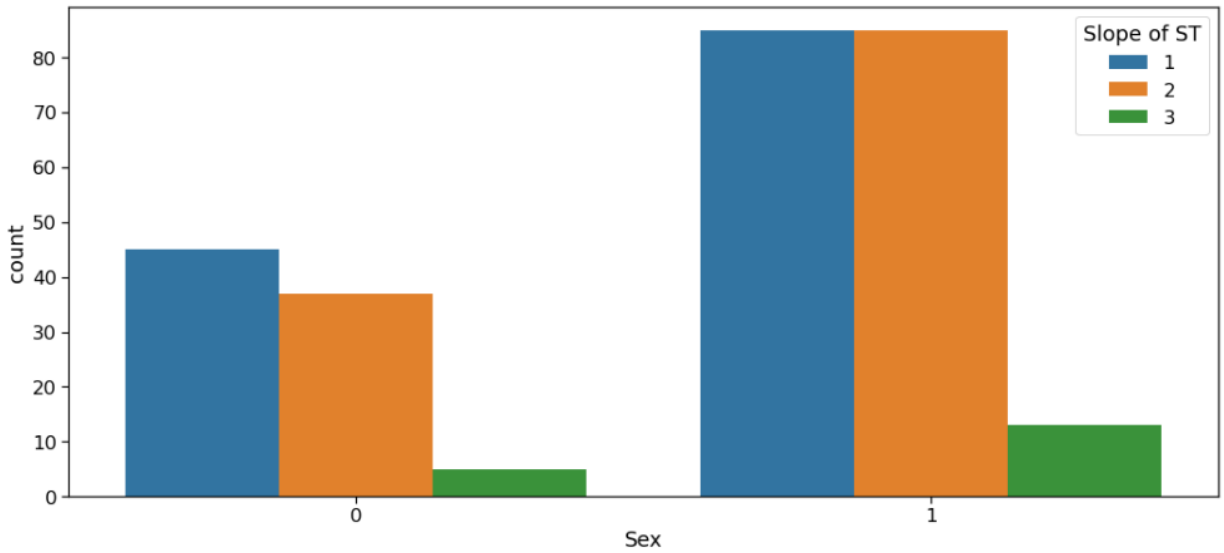
```
plt.figure(figsize=(18,9))
sns.set_context('notebook',font_scale = 1.5)
sns.countplot(new_data['Sex'])
plt.tight_layout()
```



Inference: Ratio of male to female is approximately 2:1

- **Plotting the relation between "Sex" and "Slope"**

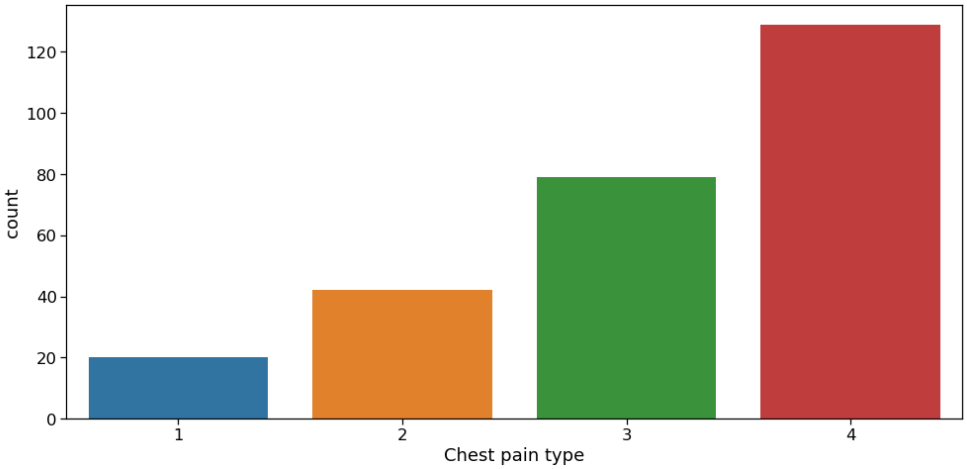
```
plt.figure(figsize=(15,7))
sns.set_context('notebook',font_scale = 1.5)
sns.countplot(data['Sex'],hue=data["Slope of ST"])
plt.tight_layout()
```



Inference: Slope value is higher as in the case of Males(1)

- "**Chest Pain Type" Feature Analysis**

```
plt.figure(figsize=(14,7))
sns.set_context('notebook',font_scale = 1.5)
sns.countplot(data['Chest pain type'])
plt.tight_layout()
```

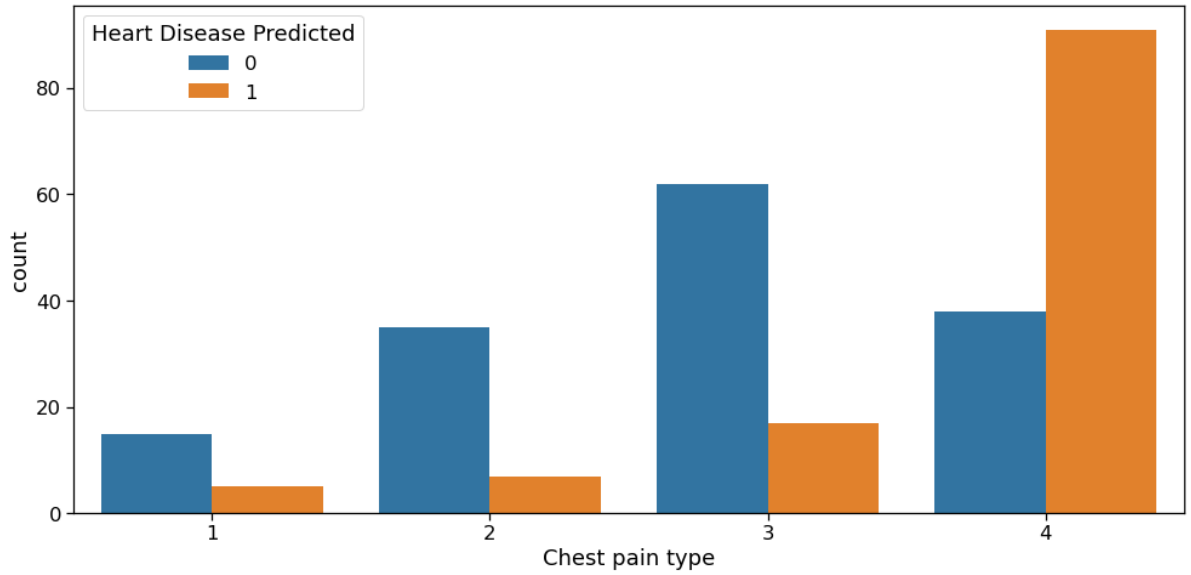


Inference: As seen above, there are 4 types of chest pain

1. Status at least
2. Condition slightly distressed
3. Condition medium problem
4. Condition too bad

- **Analyzing "Chest Pain" vs Target Label**

```
plt.figure(figsize=(14,7))
sns.set_context('notebook',font_scale = 1.5)
sns.countplot(data['Chest pain type'],hue=data["Heart Disease Predicted"])
plt.tight_layout()
```

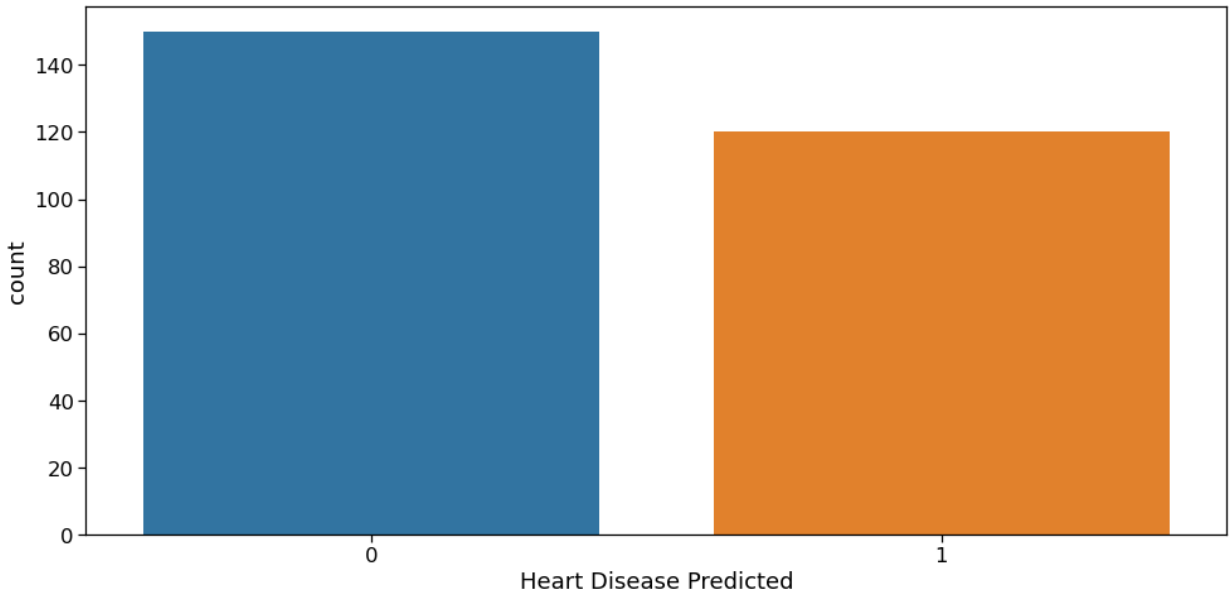


Inference:

★ People having the least chest pain are not likely to have heart disease.
★ People having severe chest pain are likely to have heart disease.
★ Elderly people are most likely to have chest pain.

- **Analyzing Target Variable**

```
plt.figure(figsize=(14,7))
sns.set_context('notebook',font_scale = 1.5)
sns.countplot(new_data['Heart Disease Predicted'])
plt.tight_layout()
```



Inference: The ratio between 1 and 0 is much less than 1.5 which indicates that the target label is not imbalanced.

Cleaned Dataset After Preprocessing: Dataset

Python Packages

1. numpy: To work with arrays
2. pandas: To work with csv files and dataframes
3. matplotlib: To create charts using pyplot
4. warnings: To ignore all warnings which might be showing up in the notebook due to past/future depreciation of a feature.
5. train_test_split: To split the dataset into training and testing data
6. StandardScaler: To scale all the features, so that the Machine Learning model better adapts to the dataset
7.

Learning Algorithms

ML Algorithms Used:

1. Decision Tree Classifier:

   Decision tree is a non-parametric supervised learning algorithm, which is utilized for both classification and regression tasks. It has a hierarchical, tree structure, which consists of a root node, branches, internal nodes and leaf nodes.Decision tree starts with a root node, which does not have any incoming branches. The outgoing branches from the root node then feed into the internal nodes, also known as decision nodes and the leaf nodes represent all the possible outcomes within the dataset

2. K Neighbors Classifier

   K-Nearest Neighbour is one of the simplest Machine Learning algorithms based on Supervised Learning technique. It assumes the similarity between the new case/data and available cases and put the new case into the category that is most similar to the available

categories.K-NN algorithm stores all the available data and classifies a new data point based on the similarity. This means when new data appears then it can be easily classified into a well suite category by using K- NN algorithm.

3. Support Vector Classifier

Support Vector Machine or SVM is one of the most popular Supervised Learning algorithms, which is used for Classification as well as Regression problems.The goal of the SVM algorithm is to create the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future. This best decision boundary is called a hyperplane.

- **Splitting Dataset into Training and Testing**

```
#Splitting the dataset into training and testing

X, y = new_data.iloc[:, :-1], new_data.iloc[:, -1]

print(X.shape)
print(y.shape)
```

```
(303, 13)
(303,)
```

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler


sc = StandardScaler()
X = sc.fit_transform(X)
```

```
X_train,X_test,y_train,y_test=train_test_split(X,y,random_state=10,test_size=0.3,shuffle=True)
```

```
print ("Training_set_x shape: " + str(X_train.shape))
print ("Training_set_y shape: " + str(y_train.shape))
print ("Testing_set_x shape: " + str(X_test.shape))
print ("Testing_set_y shape: " + str(y_test.shape))
```

```
Training_set_x shape: (212, 13)
Training_set_y shape: (212,)
Testing_set_x shape: (91, 13)
Testing_set_y shape: (91,)
```

- **Applying different ML Algorithms**

1. Decision Tree Classifier

```
from sklearn.tree import DecisionTreeClassifier
dt=DecisionTreeClassifier()
dt.fit(X_train,y_train)
```

```
DecisionTreeClassifier()
```

```
prediction=dt.predict(X_test)
accuracy_dt=accuracy_score(y_test,prediction)*100
```

```
scores_dict['DecisionTreeClassifier'] = accuracy_dt
print("Accuracy with Descion Tree Classifier: " +str(accuracy_dt))
```

```
Accuracy with Descion Tree Classifier: 75.82417582417582
```

```
print("Accuracy on training set: {:.3f}".format(dt.score(X_train, y_train)))
print("Accuracy on test set: {:.3f}".format(dt.score(X_test, y_test)))
```

```
Accuracy on training set: 1.000
Accuracy on test set: 0.758
```

Accuracy Obtained in testing dataset: 75.8%

Predicting the result of a new point

```
# Predicting with new point
X_DT=np.array([[63 ,1, 3,145,233,1,0,150,0,2.3,0,0,1]])
X_DT_prediction=dt.predict(X_DT)
print(X_DT_prediction[0])
print(Catagory[int(X_DT_prediction[0])])
```

```
0
No
```

2. K Neighbors Classifier

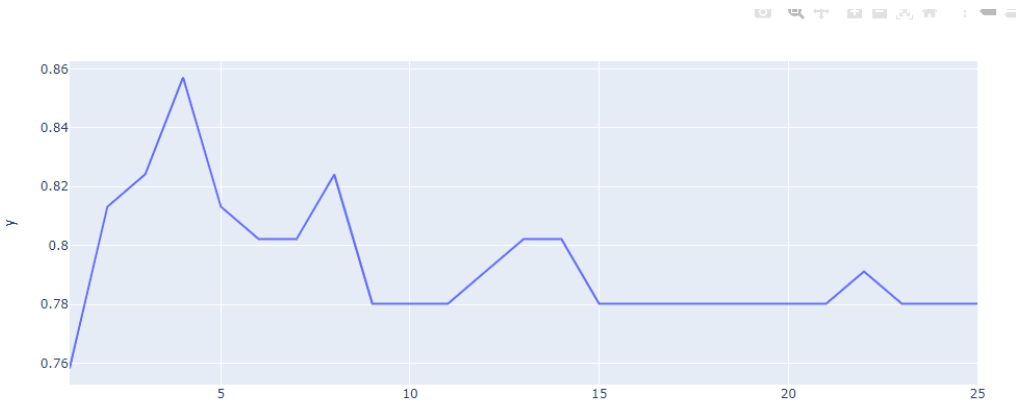
Finding the best value of k

```
from sklearn.neighbors import KNeighborsClassifier

k_range=range(1,26)

scores={}
h_score = 0       # to find the best score
best_k=0          # to find the best k
scores_list=[]

for k in k_range:

    knn=KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train,y_train)
    prediction_knn=knn.predict(X_test)
    scores[k]=accuracy_score(y_test,prediction_knn)

    if scores[k]>h_score:
        h_score = scores[k]
        best_k = k

    scores_list.append(accuracy_score(y_test,prediction_knn))

print('The best value of k is {} with score : {}'.format(best_k,h_score))
```

```
The best value of k is 4 with score : 0.8571428571428571
```

Plotting accuracies for different values of k

```
#Plotting accuracy for different values of k


px.line(x=k_range,y=scores_list)
```




```
knn=KNeighborsClassifier(n_neighbors=best_k)
knn.fit(X_train,y_train)


prediction_knn=knn.predict(X_test)
accuracy_knn=accuracy_score(y_test,prediction_knn)*100
print('accuracy_score :',accuracy_score(y_test,prediction_knn)*100,'%')
print('mean_squared_error :',mean_squared_error(y_test,prediction_knn)*100,'%')
```

```
accuracy_score : 85.71428571428571 %
mean_squared_error : 14.285714285714285 %
```

```
scores_dict['KNeighborsClassifier'] = accuracy_knn
print("Accuracy with KKN: " +str(accuracy_knn))


```

```
Accuracy with KKN: 85.71428571428571
```

Accuracy Obtained in KNN: 85.7%

Predicting the result of new point

```
#Predicting with new point
X_knn=np.array([[63 ,1, 3,145,233,1,0,150,0,2.3,0,0,1]])
X_knn=sc.transform(X_knn)
X_knn_prediction=knn.predict(X_knn)
print(X_knn)
```

```
[[ 0.9521966   0.68100522  1.97312292  0.76395577 -0.25633371  2.394438
  -1.00583187  0.01544279 -0.69663055  1.08733806 -2.27457861 -0.71442887
  -2.14887271]]
```

```
print(X_knn_prediction[0])
print(Catagory[int(X_knn_prediction[0])])
```

```
0
No
```

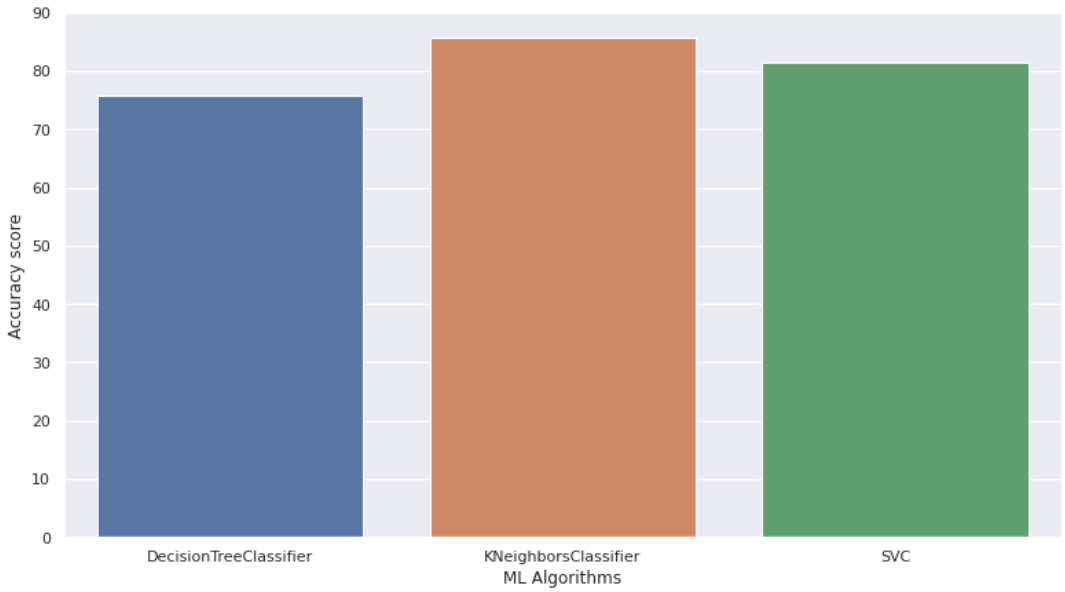
3. Support Vector Classifier

```
from sklearn.svm import SVC

model = SVC(C=2.0,kernel='rbf',gamma='auto').fit(X_train,y_train)
Y_predict = model.predict(X_test)
print('Accuracy score : {}%'.format(accuracy_score(y_test,Y_predict)*100))
scores_dict['SVC'] = accuracy_score(y_test,Y_predict)*100
```

```
Accuracy score : 81.31868131868131%
```

Accuracy Obtained with SVC: 81.3%

Plotting the accuracies of Different Algorithms



Inference: Among the 3 Algorithms, KNN has the highest accuracy.

- **Performing K Fold Cross Validation And applying It To Each Algorithm**

1. Decision Tree Classifier:

```
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold


acc_scores={}
cv = KFold(n_splits=3,shuffle=True)
cv_score_lr = cross_val_score(DecisionTreeClassifier(), X, y, cv=cv)
print(cv_score_lr)

mean_accuracy_lr = sum(cv_score_lr)/len(cv_score_lr)
mean_accuracy_lr = mean_accuracy_lr*100
mean_accuracy_lr = round(mean_accuracy_lr, 2)
print("Mean Accuracy In Desicion Tree Classifier: "+str(mean_accuracy_lr)+"%")
acc_scores['Desicion Tree Classifer']=mean_accuracy_lr
```

```
[0.74257426 0.86138614 0.73267327]
Mean Accuracy In Desicion Tree Classifier: 77.89%
```

Accuracy Obtained: 77.89%

2. K Nearest Neighbors

```
from sklearn.model_selection import cross_val_score
cv = KFold(n_splits=3, shuffle=True)
cv_score_lr = cross_val_score(KNeighborsClassifier(), X, y, cv=cv)
print(cv_score_lr)

mean_accuracy_lr = sum(cv_score_lr)/len(cv_score_lr)
mean_accuracy_lr = mean_accuracy_lr*100
mean_accuracy_lr = round(mean_accuracy_lr, 2)
print("Mean Accuracy In KNN: "+str(mean_accuracy_lr)+"%")
acc_scores['K Neighbors Classifier']=mean_accuracy_lr
```

```
[0.81188119 0.78217822 0.84158416]
Mean Accuracy In KNN: 81.19%
```

Accuracy Obtained: 81.19%

3. Support Vector Classifier

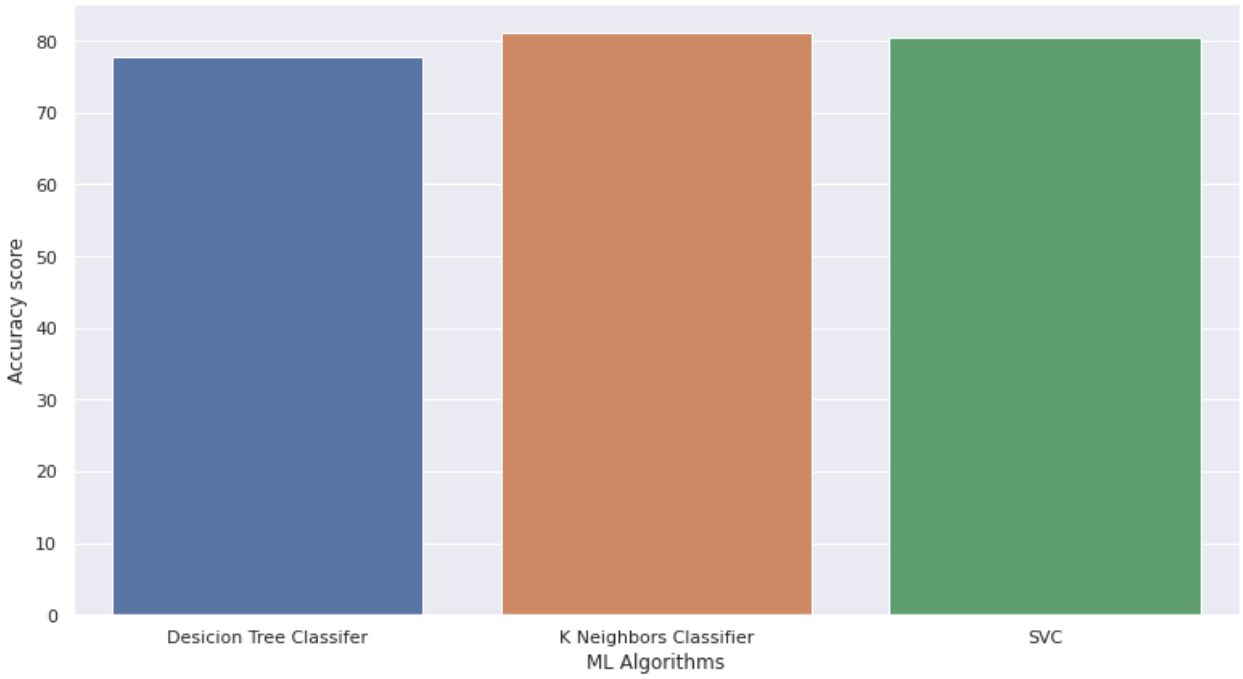
```
from sklearn.model_selection import cross_val_score
cv = KFold(n_splits=3,  shuffle=True)
cv_score_lr = cross_val_score(SVC(), X, y, cv=cv)
print(cv_score_lr)
mean_accuracy_lr = sum(cv_score_lr)/len(cv_score_lr)
mean_accuracy_lr = mean_accuracy_lr*100
mean_accuracy_lr = round(mean_accuracy_lr, 2)
print("Mean Accuracy In SVC: "+str(mean_accuracy_lr)+"%")
acc_scores['SVC']=mean_accuracy_lr
```

```
[0.79207921 0.78217822 0.84158416]
Mean Accuracy In SVC: 80.53%
```

Accuracy Obtained: 80.53%

Comparing the Three Models

```
with sns.color_palette('muted'):
  algo_name = list(acc_scores.keys())
  scores = list(acc_scores.values())

  sns.set(rc={'figure.figsize':(13,7)})
  plt.xlabel("ML Algorithms")
  plt.ylabel("Accuracy score")

  sns.barplot(algo_name,scores)
```



Conclusion: Among the three algorithms, KNN has the highest accuracy.

Pynb File Link: [Heart Disease Prediction]