Section 1: Project overview

The main goal of the project is to analyze a medical patient's dataset which contain the various health related data and predict any upcoming heart diseases using ML (Machine Learning) and classification technique. After collecting the data, we preprocessed the data and performing an exploratory data analysis. After that we identified the most effective data that can closely related to various heart diseases. After graphical analysis and finding the correlating, we train various machine learning model such as naive bayes, KNN, Decision tree, Logistic Regression, Support Vector Machine. After training the model and test the result. At last accuracy and performance are analyzed.

Section 2: Dataset Overview

This is a classification model creating using real life dataset from kaggle.

Data Source: https://www.kaggle.com/datasets/johnsmith88/heart-disease-dataset

Description of the dataset:

Context

This data set dates from 1988 and consists of four databases: Cleveland, Hungary, Switzerland, and Long Beach V. It contains 76 attributes, including the predicted attribute, but all published experiments refer to using a subset of 14 of them. The "target" field refers to the presence of heart disease in the patient. It is integer valued 0 = no disease and 1 = disease. There are 1025 data entities in the data table.

Content

Attribute Information:

```
1.age(age in years)
2.sex(1 = male; 0 = female)
3.chest pain type (4 values 0=25%,1=50%,2=75%,3=100%, min=0,max=100)
4.resting blood pressure in mm Hg on admission to the hospital
(Min=94,25%=120,50%=130,75%=140,Max=200)
5.serum cholestoral in mg/dl (126=Min,211=25%,240=50%,275=75%,564=Max)
6.fasting blood sugar > 120 mg/dl
7.resting electrocardiographic results (values 0,1,2)
8.maximum heart rate achieved (71=Min,132=25%,152=50%,166=75%,202=Max)
9.exercise induced angina
10.oldpeak = ST depression induced by exercise relative to rest (1 =
yes; 0 = no)
11.the slope of the peak exercise ST segment
12.number of major vessels (0-3) colored by flourosopy
13.thal: 0 = normal; 1 = fixed defect; 2 = reversable defect
```

The names and social security numbers of the patients were recently removed from the database, replaced with dummy values.

Imports and Settings

Here numpy, pandas libraries are used for database manupulation, matplotlib,seaborn libraries are used for data visualization and, scikit-learn library is used for machine leaning model creation,evaluation.

```
import numpy as np
import pandas as pd
# Visualization Libraries
import matplotlib.pyplot as plt
plt.style.use('seaborn-whitegrid')
import seaborn as sns
from sklearn import datasets # for using built-in datasets
from sklearn import metrics # for checking the model accuracy
#To plot the graph embedded in the notebook
%matplotlib inline
```

loading dataset named "heart.csv"

```
heart_df=pd.read_csv('heart.csv')
heart_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]
```

```
heart_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
```

Section 3: Data Preprocessing and Exploratory data

```
heart_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
```

Since there is no missing value and the 0 value is used in various reason in the dataset which is essential and there is no abnormal data inside the dataframe so we did not need to delete any column from the dataset. For better understanding different class is labled instead of values.

```
#heart_df["target"]=heart_df["target"].replace([0,1],["no
disease","disease"])
#heart_df["sex"]=heart_df["sex"].replace([0,1],["female","male"])
#heart_df["cp"]=heart_df["cp"].replace([0,1,2,3],["no pain","mild
pain","moderate pain","severe pain"])
#heart_df["thal"]=heart_df["thal"].replace([0,1,2],["normal","fixed
defect","reversable defect"])
```

Exploratory Data Analysis

```
heart_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
```

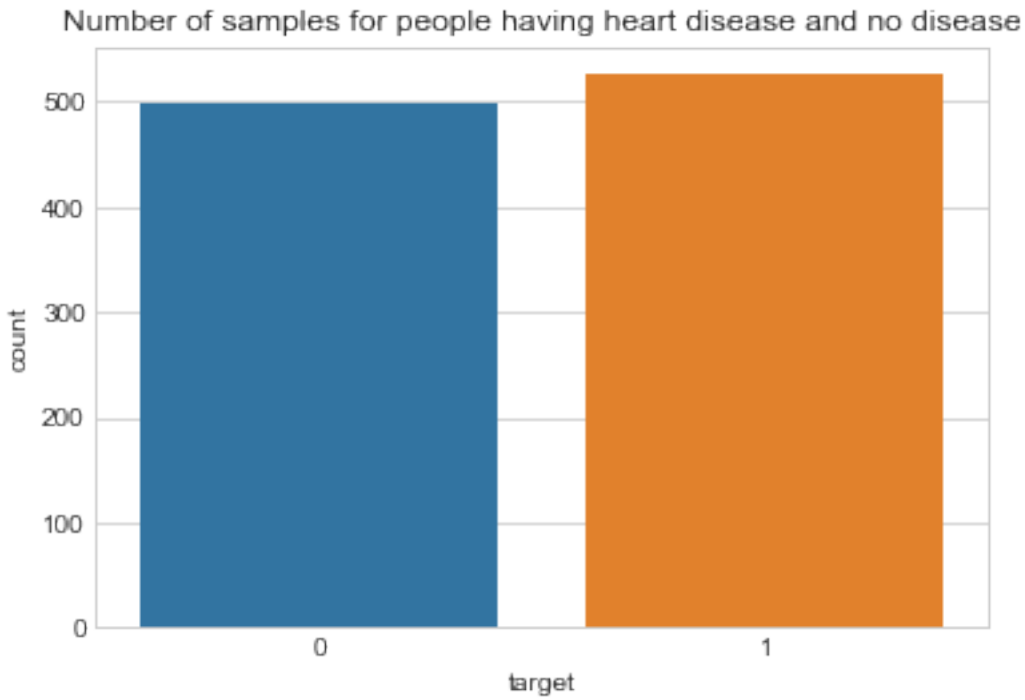
Checking for biasness

```
heart_df.groupby("target").size()
```

```
target
0    499
1    526
dtype: int64
```

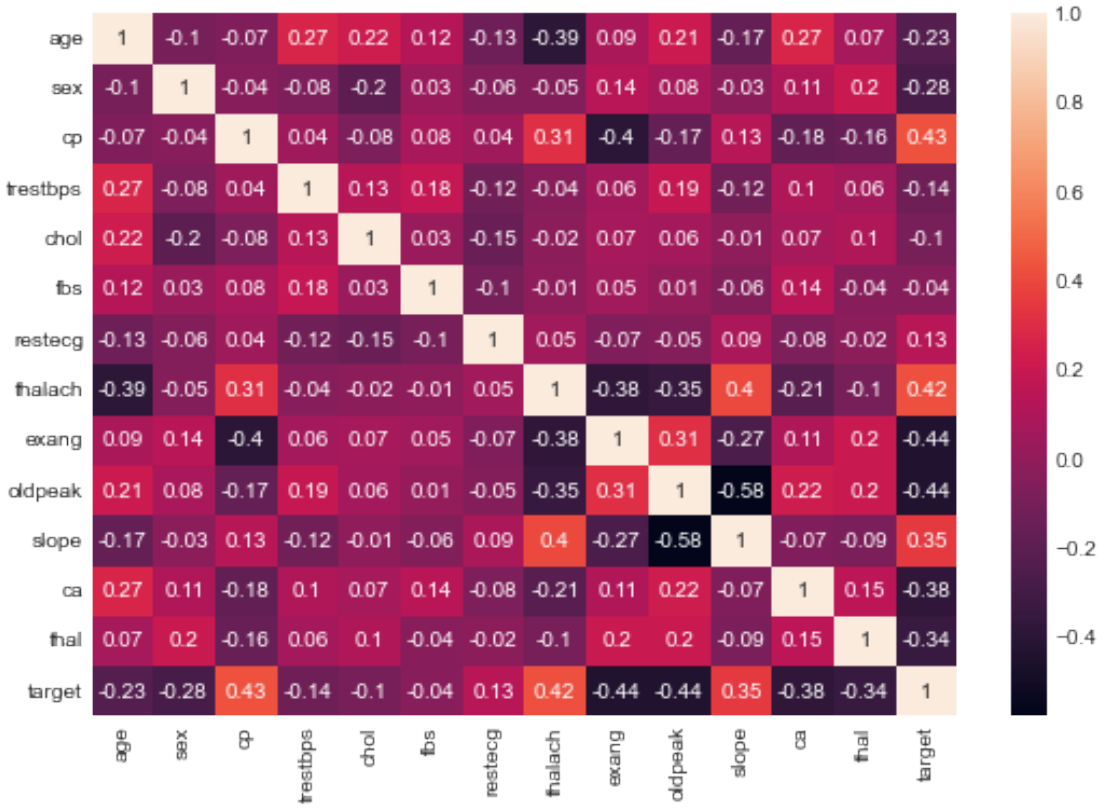
Since the number of no disease and disease are not equal but the difference is very small.There might be very negligible bias toward having diabetes.

```
# let's visualise the number of samples for each class with count plot
sns.countplot(x='target', data=heart_df)
plt.title("Number of samples for people having heart disease and no
disease");
```



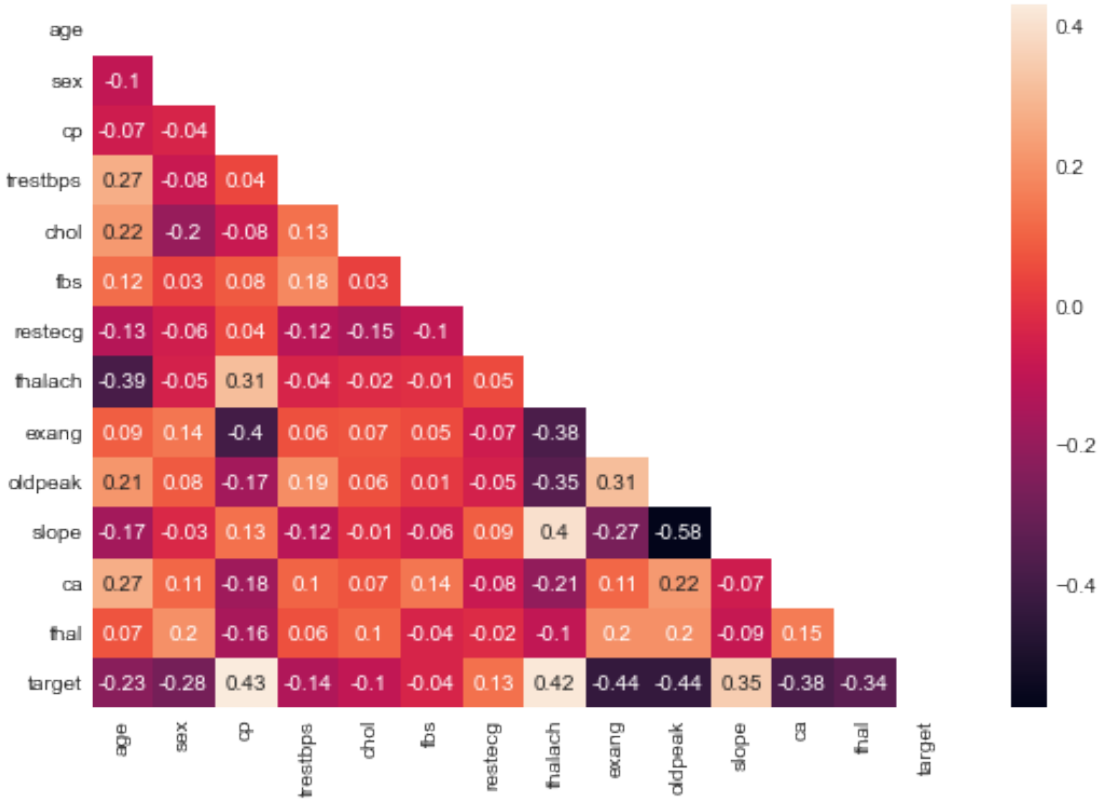
- Next, let's make a correlation matrix to quantitatively examine the relationship between variables.
- If there are features and many of the features are highly correlated, then training an algorithm with all the featues will reduce the accuracy. Thus features selection should be done carefully. This dataset has less featues but still we will see the correlation.
- The correlation matrix can be formed by using the corr function from the pandas library.
- The correlation coefficient ranges from -1 to 1 . If the value is close to 1 , it means that there is a strong positive correlation between the two variables. When it is close to -1 , the variables have a strong negative correlation.
- Then, we will use the heatmap function from the seaborn library to plot the correlation matrix.

```
correlation_matrix = heart_df.corr().round(2)
# changing the figure size
plt.figure(figsize = (9, 6))
# "annot = True" to print the values inside the square
sns.heatmap(data=correlation_matrix, annot=True);
```



Reducing the diagonal values since they indicate the relationship between ownself.Again upper/lower triangle can be reduced since they means the same relationships

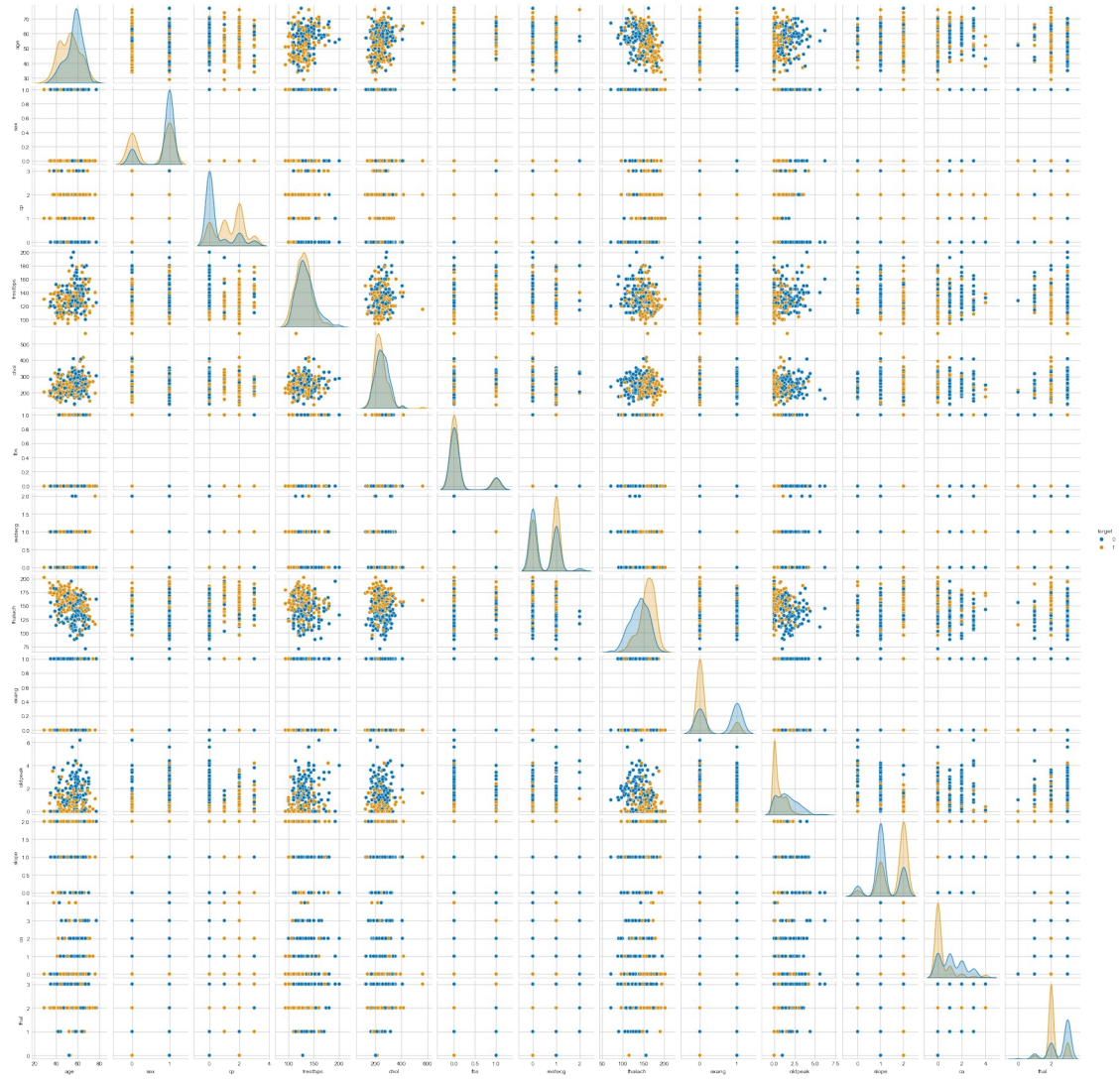
```
# Steps to remove redundant values
# Return a array filled with zeros
mask = np.zeros_like(correlation_matrix)
# Return the indices for the upper-triangle of array
mask[np.triu_indices_from(mask)] = True
# changing the figure size
plt.figure(figsize = (9, 6))
# "annot = True" to print the values inside the square
sns.heatmap(data=correlation_matrix, annot=True, mask=mask);
```



Here, Tends to 0(eg:-0.4) is weakly related

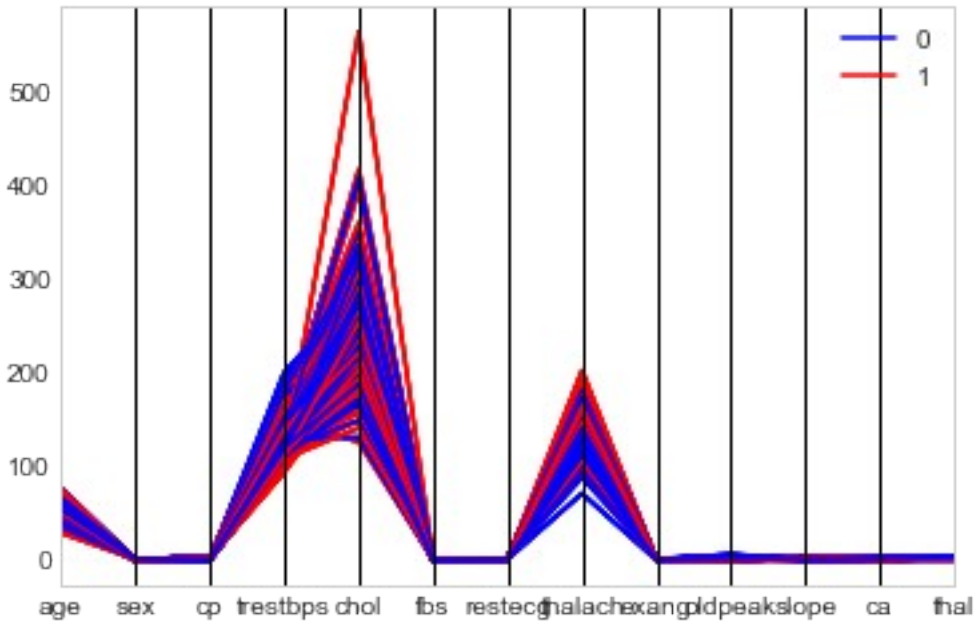
and tends to +/-1(eg: 0.7) means weakly related.

```
# let's create pairplot to visualise the data for each pair of attributes
sns.pairplot(heart_df, hue="target",height=2, palette = 'colorblind');
```



By using pair plot the co-relaiton between the feature matrix are more clear.

```
from pandas.plotting import parallel_coordinates
parallel_coordinates(heart_df, "target", color = ['blue', 'red']);
```



Ceateing Features Matrix & Target Variable

```
X = heart_df[['sex', 'oldpeak', 'slope', 'thalach','restecg']]
X
```

```
      sex  oldpeak  slope  thalach  restecg
0       1      1.0      2      168        1
1       1      3.1      0      155        0
2       1      2.6      0      125        1
3       1      0.0      2      161        1
4       0      1.9      1      106        1
...   ...      ...    ...      ...      ...
1020    1      0.0      2      164        1
1021    1      2.8      1      141        0
1022    1      1.0      1      118        0
1023    0      0.0      2      159        0
1024    1      1.4      1      113        1

[1025 rows x 5 columns]
```

```
y = heart_df['target']
y
```

```
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
```

Section 4: Model Development

Split the dataset

Here the total dataset is divided into 70% for training and 30% testing.

```
#Create Model: Support Vector Machine (SVM)
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size =
0.3, random_state = 16)
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:  (717, 5)
X_test shape:  (308, 5)
y_train shape:  (717,)
y_test shape:  (308,)
```

Create Model: Support Vector Machine (SVM)

```
 # importing the necessary package to use the classification algorithm
from sklearn import svm #for Support Vector Machine (SVM) Algorithm
model_svm = svm.SVC() #select the algorithm
model_svm.fit(X_train, y_train) #train the model with the training
dataset
y_prediction_svm = model_svm.predict(X_test) # pass the testing data
to the trained model
# checking the accuracy of the algorithm.
# by comparing predicted output by the model and the actual output
score_svm = metrics.accuracy_score(y_prediction_svm, y_test).round(4)
print("-----------------------------------")
print('The accuracy of the SVM is: {}'.format(score_svm))
print("-----------------------------------")
# save the accuracy score
score = set()
score.add(('SVM', score_svm))
```

```
-----------------------------------
The accuracy of the SVM is: 0.6883
-----------------------------------
```

Create Model: Decision Tree

```
 # importing the necessary package to use the classification algorithm
from sklearn.tree import DecisionTreeClassifier #for using Decision
Tree Algoithm
model_dt = DecisionTreeClassifier(random_state=4)
```

```
model_dt.fit(X_train, y_train) #train the model with the training
dataset
y_prediction_dt = model_dt.predict(X_test) #pass the testing data to
the trained model
# checking the accuracy of the algorithm.
# by comparing predicted output by the model and the actual output
score_dt = metrics.accuracy_score(y_prediction_dt, y_test).round(4)
print("---------------------------------")
print('The accuracy of the DT is: {}'.format(score_dt))
print("---------------------------------")
# save the accuracy score
score.add(('DT', score_dt))
```

```
---------------------------------
The accuracy of the DT is: 0.9805
---------------------------------
```

Create Model: K Nearest Neighbours (KNN)

```
# importing the necessary package to use the classification algorithm
from sklearn.neighbors import KNeighborsClassifier # for K nearest
neighbours
#from sklearn.linear_model import LogisticRegression # for Logistic
Regression algorithm
model_knn = KNeighborsClassifier(n_neighbors=3) # 3 neighbours for
putting the new data into a class
model_knn.fit(X_train, y_train) #train the model with the training
dataset
y_prediction_knn = model_knn.predict(X_test) #pass the testing data to
the trained model
# checking the accuracy of the algorithm.
# by comparing predicted output by the model and the actual output
score_knn = metrics.accuracy_score(y_prediction_knn, y_test).round(4)
print("---------------------------------")
print('The accuracy of the KNN is: {}'.format(score_knn))
print("---------------------------------")
# save the accuracy score
score.add(('KNN', score_knn))
```

```
---------------------------------
The accuracy of the KNN is: 0.8701
---------------------------------
```

Create Model: Logistic Regression

```
 # importing the necessary package to use the classification algorithm
from sklearn.linear_model import LogisticRegression # for Logistic
Regression algorithm
model_lr = LogisticRegression()
model_lr.fit(X_train, y_train) #train the model with the training
dataset
y_prediction_lr = model_lr.predict(X_test) #pass the testing data to
the trained model
```

```
# checking the accuracy of the algorithm.
# by comparing predicted output by the model and the actual output
score_lr = metrics.accuracy_score(y_prediction_lr, y_test).round(4)
print("---------------------------------")
print('The accuracy of the LR is: {}'.format(score_lr))
print("---------------------------------")
# save the accuracy score
score.add(('LR', score_lr))
```

```
---------------------------------
The accuracy of the LR is: 0.7597
---------------------------------
```

Create Model: Naive Bayes

```
 # importing the necessary package to use the classification algorithm
from sklearn.naive_bayes import GaussianNB
model_nb = GaussianNB()
model_nb.fit(X_train, y_train) #train the model with the training
dataset
y_prediction_nb = model_nb.predict(X_test) #pass the testing data to
the trained model
# checking the accuracy of the algorithm.
# by comparing predicted output by the model and the actual output
score_nb = metrics.accuracy_score(y_prediction_nb, y_test).round(4)
print("---------------------------------")
print('The accuracy of the NB is: {}'.format(score_nb))
print("---------------------------------")
# save the accuracy score
score.add(('NB', score_nb))
```

```
---------------------------------
The accuracy of the NB is: 0.75
---------------------------------
```

Compare Accuracy Score of Different Models

```
print("The accuracy scores of different Models:")
print("----------------------------------------")
for s in score:
 print(s)
```

```
The accuracy scores of different Models:
----------------------------------------
('NB', 0.75)
('KNN', 0.8701)
('LR', 0.7597)
('SVM', 0.6883)
('DT', 0.9805)
```

Section 5: discussion and conclusion

Comperison of model

In this section, we analyze five models: Naive Bias, KNN, Logistics Regression, Supervised Model, and Decision Tree. One of them Supervised Models provide the least accuracy (68%) Nive Bias and Logistic Regression both provide moderate accuracy with little variation between them (75% and 75.9 percent, respectively). KNN and Decision Tree, however, are fairly accurate. KNN comes in second place in terms of accuracy with 87 percent, while Decision Tree provides the most accurate outcome with 98 percent. Decision trees outperform linear regression for independent variables that fall into a categorical category. Compared to LR, decision trees handle collinearity better. Both perform well when there is a low number of features and little training data. KNN is slow in real time because it must maintain track of all training data and locate neighbor nodes.

Conclusion

One of the key elements of supervised learning is classification. Machine learning applications for supervised classification include speech recognition, handwriting recognition, face recognition, and document classification. Successful object recognition and classification are tasks for machine learning algorithms to succeed. In this section, we explored various machine learning classification methods and some of their practical applications for predicting heart disease. This helps in early detection of heart failure by medicine. Since 98% of heart disease cases can be detected with the best predictions, Machine learning models can help stop patients before they reach critical stages.