Description

The dataset contains several medical predictor (Independent) variables and one target variable, (Outcome). Predictor variables include:

1. age
2. sex
3. chest pain type (4 values)
4. resting blood pressure
5. serum cholestoral in mg/dl
6. fasting blood sugar > 120 mg/dl
7. resting electrocardiographic results (values 0,1,2)
8. maximum heart rate achieved
9. exercise induced angina
10. oldpeak = ST depression induced by exercise relative to rest
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

Dataset url: https://www.kaggle.com/datasets/johnsmith88/heart-disease-dataset

Step 1: Importing the Libraries

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
```

Step 2: Load the dataset

```
# loading the csv data to a Pandas DataFrame
heart_data = pd.read_csv('/content/heart.csv')
```

Step 3: Exploratory Data Analysis

Exploratory Data Analysis (EDA), also known as Data Exploration, is a step in the Data Analysis Process, where a number of techniques are used to better understand the dataset being used.

3.1) Understanding Your Variables

```
3.1.1) Head of the dataset
3.1.2) The shape of the dataset
3.1.3) List types of columns
```

3.1.4) Info of the dataset
3.1.5) Summary of the dataset

3.1.1) Head of the Dataset

```
# Display first five records
heart_data.head()

# Display last five records
heart_data.tail()
```

3.1.2)The Shape of Dataset

```
# number of rows and columns in the dataset
heart_data.shape
```

3.1.3)List types of columns

```
heart_data.dtypes
```

3.1.4)Info of Dataset

```
# getting some info about the data
heart_data.info()

# checking for missing values
heart_data.isnull().sum()

# Statistical Summary
heart_data.describe()

# checking the distribution of Target Variable
heart_data['target'].value_counts()
```

1 --> Defective Heart

0 --> Healthy Heart

Step 4: Split the data frame in X & Y

```
X = heart_data.drop(columns='target', axis=1)
Y = heart_data['target']

X.head()

Y.head()
```

Step 5: Applying Feature Scaling

Various Data Scaling Techniques:

1. Normalizer

2. MinMax Scaler
3. Binarizer
4. Standard Scaler

```
# Apply Standard Scaler
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
SSX = scaler.fit_transform(X)
```

Step 6: Splitting the Data into Training data & Test Data

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.3,
random_state=0)

print(X.shape, X_train.shape, X_test.shape)
```

Step 7: Building Classification Algorithm

7.1) Logistic Regression

```
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression(solver='liblinear',multi_class='ovr')
lr.fit(X_train,Y_train)
```

7.2) K-Nearest Neighbors Classifier(KNN)

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier()
knn.fit(X_train,Y_train)
```

7.3) Naive-Bayes Classifier

```
from sklearn.naive_bayes import GaussianNB
nb = GaussianNB()
nb.fit(X_train, Y_train)
```

7.4) Support Vector Machine (SVM)

```
from sklearn.svm import SVC
sv = SVC(kernel='linear')
sv.fit(X_train,Y_train)
```

7.5) Decision Tree

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

7.6) Random Forest

```
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(n_estimators=20, random_state=12,max_depth=6)
rf.fit(X_train,Y_train)
```

Step 8: Making Prediction

8.1) Making Prediction using Logistic Regression

```
print(f'Initial shape: {X_test.shape}')
lr_pred = lr.predict(X_test)
print(f'{lr_pred.shape}')
```

8.2) Making Prediction using KNN

```
knn_pred = knn.predict(X_test)
knn_pred.shape
```

8.3) Making Prediction using Naive Bayes

```
nb_pred = nb.predict(X_test)
nb_pred.shape
```

8.4) Making Prediction using SVM

```
sv_pred = sv.predict(X_test)
sv_pred.shape
```

8.5) Making Prediction using Decision Tree

```
dt_pred = dt.predict(X_test)
```

8.6) Making Prediciton using Random Forest

```
rf_pred = rf.predict(X_test)
```

Step 9: Model Evaluation

```
from sklearn.metrics import accuracy_score

# Train & Test Scores of Logistic Regression
print("Accuracy (Train) score of Logistic Regression
",lr.score(X_train,Y_train)*100)
print("Accuracy (Test) score of Logistic Regression ",
lr.score(X_test,Y_test)*100)
print("Accuracy score of Logistic Regression ",
accuracy_score(Y_test,lr_pred)*100)

# Train & Test Scores of KNN
print("Accuracy (Train) score of KNN ",knn.score(X_train,Y_train)*100)
print("Accuracy (Test) score of KNN ", knn.score(X_test,Y_test)*100)
print("Accuracy score of KNN ", accuracy_score(Y_test,knn_pred)*100)

# Train & Test Scores of Naive-Bayes
print("Accuracy (Train) score of Naive Bayes ",nb.score(X_train,Y_train)*100)
print("Accuracy (Test) score of Naive Bayes ", nb.score(X_test,Y_test)*100)
print("Accuracy score of Naive Bayes ", accuracy_score(Y_test,nb_pred)*100)

# Train & Test Scores of SVM
print("Accuracy (Train) score of SVM ",sv.score(X_train,Y_train)*100)
```

```
print("Accuracy (Test) score of SVM ", sv.score(X_test,Y_test)*100)
print("Accuracy score of SVM ", accuracy_score(Y_test,sv_pred)*100)

# Train & Test Scores of Decision Tree
print("Accuracy (Train) score of Decision Tree
",dt.score(X_train,Y_train)*100)
print("Accuracy (Test) score of Decision Tree ", dt.score(X_test,Y_test)*100)
print("Accuracy score of Decision Tree ", accuracy_score(Y_test,dt_pred)*100)

# Train & Test Scores of Random Forest
print("Accuracy (Train) score of Random Forest
",rf.score(X_train,Y_train)*100)
print("Accuracy (Test) score of Random Forest ", rf.score(X_test,Y_test)*100)
print("Accuracy score of Random Forest ", accuracy_score(Y_test,rf_pred)*100)
```

Step 10: Building a Predictive System

```
input_data = (63,1,3,145,233,1,0,150,0,2.3,0,0,1)

# change the input data to a numpy array
input_data_as_numpy_array= np.asarray(input_data)

# reshape the numpy array as we are predicting for only on instance
input_data_reshaped = input_data_as_numpy_array.reshape(1,-1)

prediction = dt.predict(input_data_reshaped)
print(prediction)

if (prediction[0]== 0):
  print('The Person does not have a Heart Disease')
else:
  print('The Person has Heart Disease')
```

Step 11 : Saving the trained model

```
import pickle
import pickle
filename = 'heart_disease_model.sav'
pickle.dump(lr,open(filename,'wb'))
```