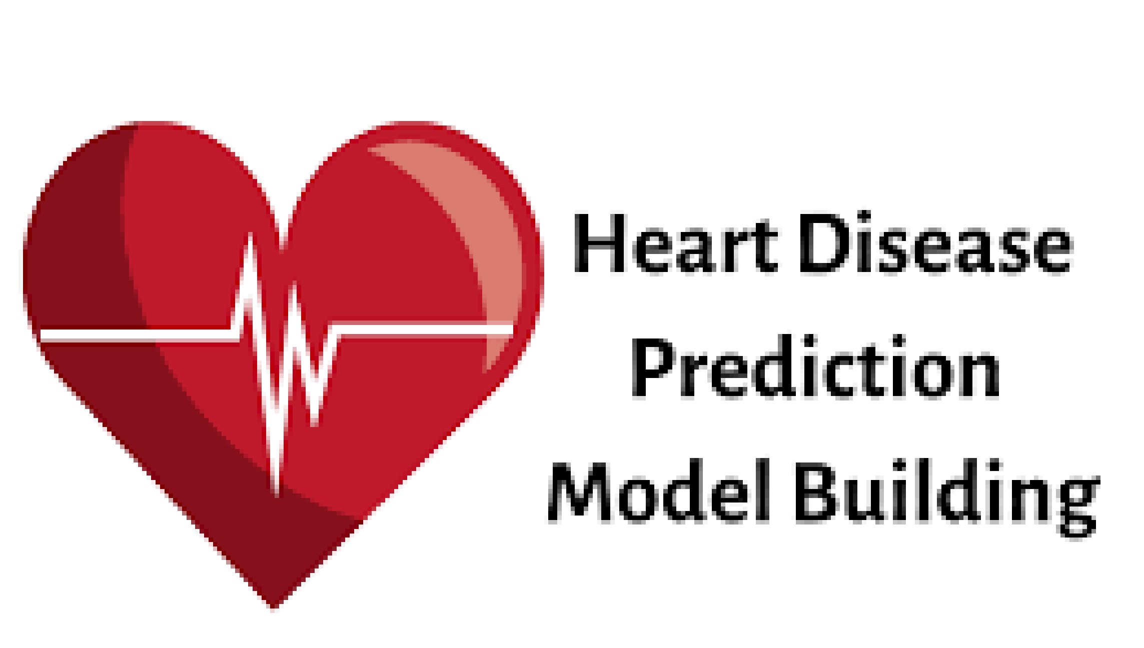
MACHINE LEARNING PROJECT PHASE 2

Heart disease Prediction



Name	*Roll Number*
PremKumar	***AM.EN.U4CSE20340***
Hareendra	***AM.EN.U4CSE20330***
Panshul sai	***AM.EN.U4CSE20324***
Ananya	***AM.EN.U4CSE20308***
Ridhika	***AM.EN.U4CSE20356***

1. Problem Definition

Heart disease can be effectively controlled with a combination of lifestyle modifications, medications, and, in rare circumstances, surgery. Heart disease symptoms can be decreased, and heart function improved with the correct therapy. A cardiologist measures vitals & hands you this data to perform Data Analysis and predict whether certain patients have heart disease. We would like to make a Machine Learning algorithm where we can train our AI to learn & improve from experience. Thus, we would want to classify patients as either positive or negative for heart disease.

2. Datasets

- Datset1 – ***heart.csv***

  Age: age of the patient [years]
  Sex: sex of the patient [M: Male, F: Female]
  ChestPainType: chest pain type [TA: Typical Angina, ATA: Atypical Angina, NAP: Non-Anginal Pain, ASY: Asymptomatic]
  RestingBP: resting blood pressure [mm Hg]
  Cholesterol: serum cholesterol [mm/dl]
  FastingBS: fasting blood sugar [1: if FastingBS > 120 mg/dl, 0: otherwise]
  RestingECG: resting electrocardiogram results [Normal: Normal, ST: having ST-T wave abnormality (T wave inversions and/or ST elevation or depression of > 0.05 mV), LVH: showing probable or definite left ventricular hypertrophy by Estes' criteria]
  MaxHR: maximum heart rate achieved [Numeric value between 60 and 202]
  ExerciseAngina: exercise-induced angina [Y: Yes, N: No]
  Oldpeak: oldpeak = ST [Numeric value measured in depression]
  ST_Slope: the slope of the peak exercise ST segment [Up: upsloping, Flat: flat, Down: downsloping]
  HeartDisease: output class [1: heart disease, 0: Normal]

  EDA

  Explore the dataset and perform the wrangling as required....

```
heart_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 918 entries, 0 to 917
Data columns (total 12 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   Age             918 non-null    int64  
 1   Sex             918 non-null    object 
 2   ChestPainType   918 non-null    object 
 3   RestingBP       918 non-null    int64  
 4   Cholesterol     918 non-null    int64  
 5   FastingBS       918 non-null    int64  
 6   RestingECG      918 non-null    object 
 7   MaxHR           918 non-null    int64  
 8   ExerciseAngina  918 non-null    object 
 9   Oldpeak         918 non-null    float64
 10  ST_Slope        918 non-null    object 
 11  HeartDisease    918 non-null    int64  
dtypes: float64(1), int64(6), object(5)
memory usage: 86.2+ KB
```

- Datset2 – ***heart_dataset.csv***

Age: age in years
Sex: sex of the patient [1: Male, 0: Female]
cp: chest pain type [TA: Typical Angina, ATA: Atypical Angina, NAP: Non-Anginal Pain, ASY: Asymptomatic]
trestbps: resting blood pressure [mm Hg]
chol: serum cholesterol [mm/dl]
fbs: fasting blood sugar [1: if FastingBS > 120 mg/dl, 0: otherwise]
restecg: resting electrocardiogram results [Normal: Normal, ST: having ST-T wave abnormality (T wave inversions and/or ST elevation or depression of > 0.05 mV), LVH: showing probable or definite left ventricular hypertrophy by Estes' criteria]
thalach: maximum heart rate achieved
exang: exercise-induced angina [1: Yes, 0: No]
Oldpeak: oldpeak = ST depression induced by exercise relative to rest
slope: the slope of the peak exercise ST segment [Up: upsloping, Flat: flat, Down: downsloping]
ca: number of major vessels (0-3) colored by flourosopy
thal: 3 = normal; 6 = fixed defect; 7 = reversable defect
target: output class [1: heart disease, 0: Normal]

```
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
```

- Datset3 – ***Heart_Disease_Prediction.csv***

Age: age of the patient [years]
Sex: sex of the patient [1: Male, 0: Female]
Chest Pain Type: chest pain type [TA: Typical Angina, ATA: Atypical Angina, NAP: Non-Anginal Pain, ASY: Asymptomatic]
BP: resting blood pressure [mm Hg]
Cholesterol: serum cholesterol [mm/dl]
FBS over 120: fasting blood sugar [1: if FastingBS > 120 mg/dl, 0: otherwise]
EKG results: displays resting electrocardiographic results 0 = normal;1 = having ST-T wave abnormality; 2 = left ventricular hyperthrophy
MaxHR: maximum heart rate achieved [Numeric value between 60 and 202]
Exercise Angina: exercise-induced angina [1: Yes, 0: No]
ST depression: induced by exercise relative to rest: displays the value which is an integer or float.

Slope of ST: the slope of the peak exercise ST segment [Up: upsloping, Flat: flat, Down: downsloping]
Number of vessels fluro: (0–3) colored by flourosopy displays the value as integer or float.
Thallium: 3 = normal; 6=fixed defect; 7 = reversable defect
HeartDisease: output class [1: Presence, 0: Absence]

```
heart_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 270 entries, 0 to 269
Data columns (total 14 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   Age                      270 non-null    int64
 1   Sex                      270 non-null    int64
 2   Chest pain type          270 non-null    int64
 3   BP                       270 non-null    int64
 4   Cholesterol              270 non-null    int64
 5   FBS over 120             270 non-null    int64
 6   EKG results              270 non-null    int64
 7   Max HR                   270 non-null    int64
 8   Exercise angina          270 non-null    int64
 9   ST depression            270 non-null    float64
 10  Slope of ST              270 non-null    int64
 11  Number of vessels fluro  270 non-null    int64
 12  Thallium                 270 non-null    int64
 13  Heart Disease            270 non-null    object
dtypes: float64(1), int64(12), object(1)
memory usage: 29.7+ KB
```

3. Prepare Data

- Pre-processing:

➔ Dataset-1

```
heart_df.isnull().sum()
```

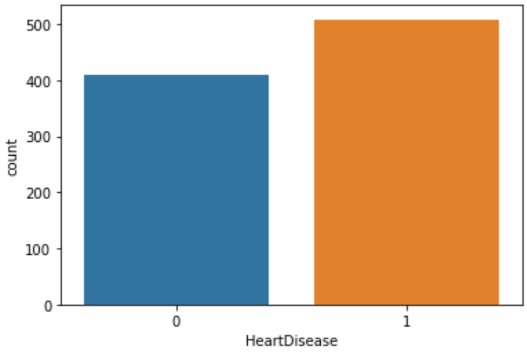
```
Age               0
Sex               0
ChestPainType     0
RestingBP         0
Cholesterol       0
FastingBS         0
RestingECG        0
MaxHR             0
ExerciseAngina    0
Oldpeak           0
ST_Slope          0
HeartDisease      0
dtype: int64
```

Looking at the above, there are no missing values for any of the attributes in the dataset.

```python
## Checking for the class imbalance of the Target Variable

sb.countplot(heart_df.HeartDisease)
heart_df.HeartDisease.value_counts()
```

```
1    508
0    410
Name: HeartDisease, dtype: int64
```




```python
## Label encode the Str attributes

from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()

heart_df['Sex'] = le.fit_transform(heart_df['Sex'])
heart_df['ChestPainType'] = le.fit_transform(heart_df['ChestPainType'])
heart_df['RestingECG'] = le.fit_transform(heart_df['RestingECG'])
heart_df['ExerciseAngina'] =  le.fit_transform(heart_df['ExerciseAngina'])
heart_df['ST_Slope'] = le.fit_transform(heart_df['ST_Slope'])

heart_df.head(10)
```

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR	ExerciseA
0	40	1	1	140	289	0	1	172	
1	49	0	2	160	180	0	1	156	
2	37	1	1	130	283	0	2	98	
3	48	0	0	138	214	0	1	108	
4	54	1	2	150	195	0	1	122	
5	39	1	2	120	339	0	1	170	
6	45	0	1	130	237	0	1	170	
7	54	1	1	110	208	0	1	142	
8	37	1	0	140	207	0	1	130	
9	48	0	1	120	284	0	1	120	

Normalization

```python
# Normalizing the data

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

sc_x_train = scaler.fit_transform(x_train)
sc_x_test = scaler.transform(x_test)

sc_x_train, x_train, sc_x_test
```

```
(array([[ 0.29076674,  0.51843486, -0.81406539, ...,  1.22793017,
          1.34191799, -0.64106918],
        [-0.55774347, -1.92888264,  0.24819067, ..., -0.81437856,
         -0.91585924,  1.03879373],
        [ 0.18470296,  0.51843486, -0.81406539, ...,  1.22793017,
          1.34191799, -0.64106918],
        ...,
        [-1.08806235,  0.51843486, -0.81406539, ...,  1.22793017,
          1.34191799, -0.64106918],
        [ 1.13927695, -1.92888264, -0.81406539, ...,  1.22793017,
          0.21302937, -0.64106918],
        [ 0.71502185,  0.51843486, -0.81406539, ..., -0.81437856,
         -0.57719266,  1.03879373]]),
 array([[56. ,  1. ,  0. , ...,  1. ,  2. ,  1. ],
        [48. ,  0. ,  1. , ...,  0. ,  0. ,  2. ],
        [55. ,  1. ,  0. , ...,  1. ,  2. ,  1. ],
        ...,
        [43. ,  1. ,  0. , ...,  1. ,  2. ,  1. ],
        [64. ,  0. ,  0. , ...,  1. ,  1. ,  1. ],
        [60. ,  1. ,  0. , ...,  0. ,  0.3,  2. ]]),
 array([[-1.83050879,  0.51843486, -0.81406539, ...,  1.22793017,
          0.21302937, -0.64106918],
        [-1.19412613,  0.51843486,  1.31044672, ..., -0.81437856,
         -0.91585924,  1.03879373],
```

Discretization

```python
from sklearn.preprocessing import KBinsDiscretizer
kbins = KBinsDiscretizer(n_bins=10, encode='ordinal', strategy='uniform')
heartdf = kbins.fit_transform(heart_df)
```

➔ Dataset-2

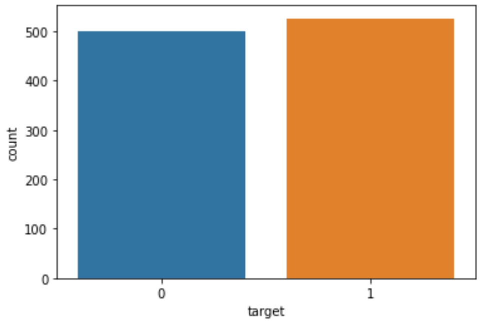
```
heart_df.isnull().sum()
```

```
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
```

```
## Checking for the class imbalance of the Target Variable

sb.countplot(heart_df.target)
heart_df.target.value_counts()
```

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



Note – This dataset has all float or int values so no need for standardscaler to be performed

```
# Normalizing the data

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

sc_x_train = scaler.fit_transform(x_train)
sc_x_test = scaler.transform(x_test)

sc_x_train, x_train, sc_x_test
```

```
(array([[-0.13713965, -1.48354065, -0.92465595, ..., -0.63762277,
         -0.82791335, -0.52211577],
        [ 1.73265357,  0.67406309, -0.92465595, ..., -2.26458194,
         -0.82791335,  1.12703403],
        [ 0.41279954,  0.67406309, -0.92465595, ..., -0.63762277,
          0.72401834,  1.12703403],
        ...,
        [-1.34700585,  0.67406309,  1.00822544, ..., -2.26458194,
         -0.82791335,  1.12703403],
        [ 1.18271439, -1.48354065,  1.00822544, ...,  0.9893364 ,
          0.72401834, -0.52211577],
        [-0.35711532,  0.67406309, -0.92465595, ...,  0.9893364 ,
         -0.82791335,  1.12703403]]),
 array([[53.,  0.,  0., ...,  1.,  0.,  2.],
        [70.,  1.,  0., ...,  0.,  0.,  3.],
        [58.,  1.,  0., ...,  1.,  1.,  3.],
        ...,
        [42.,  1.,  2., ...,  0.,  0.,  3.],
        [65.,  0.,  2., ...,  2.,  1.,  2.],
        [51.,  1.,  0., ...,  2.,  0.,  3.]]),
 array([[-1.12703017,  0.67406309,  0.04178474, ...,  0.9893364 ,
```

```
#discretization
from sklearn.preprocessing import KBinsDiscretizer
kbins = KBinsDiscretizer(n_bins=10, encode='ordinal', strategy='uniform')
heartdf = kbins.fit_transform(heart_df)
```

➔ Dataset-3

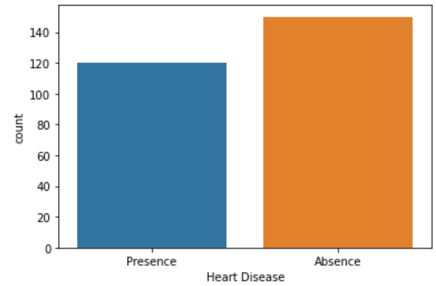
```
heart_df.isnull().sum()
```

```
Age                        0
Sex                        0
Chest pain type            0
BP                         0
Cholesterol                0
FBS over 120               0
EKG results                0
Max HR                     0
Exercise angina            0
ST depression              0
Slope of ST                0
Number of vessels fluro    0
Thallium                   0
Heart Disease              0
dtype: int64
```

```
## Checking for the class imbalance of the Target Variable

sb.countplot(heart_df['Heart Disease'])
heart_df['Heart Disease'].value_counts()
```

```
Absence     150
Presence    120
Name: Heart Disease, dtype: int64
```




```
## Label encode the Str attributes

from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()

heart_df['Heart Disease'] = le.fit_transform(heart_df['Heart Disease'])

heart_df.head(10)
```

Note – This dataset has Heart disease values as object so standardscaler to be performed

Normalization

```
# Normalizing the data

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

sc_x_train = scaler.fit_transform(x_train)
sc_x_test = scaler.transform(x_test)

sc_x_train, x_train, sc_x_test
```

```
(array([[-1.05019666, -1.39754249, -1.38751395, ...,  0.59160798,
         -0.78492777, -0.8788816 ],
        [-0.26875111,  0.71554175, -1.96134357, ...,  0.59160798,
         -0.78492777,  0.67353543],
        [-0.38038619,  0.71554175, -0.23985471, ...,  0.59160798,
         -0.78492777, -0.8788816 ],
        ...,
        [-0.15711604,  0.71554175,  0.90780453, ...,  2.18894952,
         -0.78492777,  1.19100778],
        [ 0.73596459, -1.39754249,  0.90780453, ...,  0.59160798,
         -0.78492777,  1.19100778],
        [ 1.07086982, -1.39754249, -0.23985471, ..., -1.00573356,
         -0.78492777,  1.19100778]]),
 array([[45. ,  0. ,  2. , ...,  2. ,  0. ,  3. ],
        [52. ,  1. ,  1.5, ...,  2. ,  0. ,  6. ],
        [51. ,  1. ,  3. , ...,  2. ,  0. ,  3. ],
        ...,
        [53. ,  1. ,  4. , ...,  3. ,  0. ,  7. ],
        [61. ,  0. ,  4. , ...,  2. ,  0. ,  7. ],
        [64. ,  0. ,  3. , ...,  1. ,  0. ,  7. ]]),
 array([[ 0.28942428,  0.71554175, -0.23985471, ...,  0.59160798,
          0.8188706 ,  1.19100778],
        [-0.26875111,  0.71554175, -1.38751395, ..., -1.00573356,
         -0.78492777, -0.8788816 ],
        [-0.60365635,  0.71554175, -0.23985471, ...,  0.59160798,
          1.62076978,  1.19100778],
        ...,
        [ 2.52212583,  0.71554175,  0.90780453, ..., -1.00573356,
          1.62076978, -0.8788816 ],
```

Discretization

```python
from sklearn.preprocessing import KBinsDiscretizer
kbins = KBinsDiscretizer(n_bins=10, encode='ordinal', strategy='uniform')
heartdf = kbins.fit_transform(heart_df)
```

- Summarization:

➔ Dataset1

```
In [5]: heart_df.shape
Out[5]: (918, 12)
```

```
heart_df.describe()
```

	Age	RestingBP	Cholesterol	FastingBS	MaxHR	Oldpeak	HeartDisease
count	918.000000	918.000000	918.000000	918.000000	918.000000	918.000000	918.000000
mean	53.510893	132.396514	198.799564	0.233115	136.809368	0.887364	0.553377
std	9.432617	18.514154	109.384145	0.423046	25.460334	1.066570	0.497414
min	28.000000	0.000000	0.000000	0.000000	60.000000	-2.600000	0.000000
25%	47.000000	120.000000	173.250000	0.000000	120.000000	0.000000	0.000000
50%	54.000000	130.000000	223.000000	0.000000	138.000000	0.600000	1.000000
75%	60.000000	140.000000	267.000000	0.000000	156.000000	1.500000	1.000000
max	77.000000	200.000000	603.000000	1.000000	202.000000	6.200000	1.000000

➔ Dataset2

```
In [52]: heart_df.shape
Out[52]: (1025, 14)
```

```
heart_df.describe()
Out[3]:
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope
count	1025.000000	1025.000000	1025.000000	1025.000000	1025.00000	1025.000000	1025.000000	1025.000000	1025.000000	1025.000000	1025.000000
mean	54.434146	0.695610	0.942439	131.611707	246.00000	0.149268	0.529756	149.114146	0.336585	1.071512	1.385366
std	9.072290	0.460373	1.029641	17.516718	51.59251	0.356527	0.527878	23.005724	0.472772	1.175053	0.617755
min	29.000000	0.000000	0.000000	94.000000	126.00000	0.000000	0.000000	71.000000	0.000000	0.000000	0.000000
25%	48.000000	0.000000	0.000000	120.000000	211.00000	0.000000	0.000000	132.000000	0.000000	0.000000	1.000000
50%	56.000000	1.000000	1.000000	130.000000	240.00000	0.000000	1.000000	152.000000	0.000000	0.800000	1.000000
75%	61.000000	1.000000	2.000000	140.000000	275.00000	0.000000	1.000000	166.000000	1.000000	1.800000	2.000000
max	77.000000	1.000000	3.000000	200.000000	564.00000	1.000000	2.000000	202.000000	1.000000	6.200000	2.000000

➔ Dataset3

```
In [3]: heart_df.shape
Out[3]: (270, 14)
```

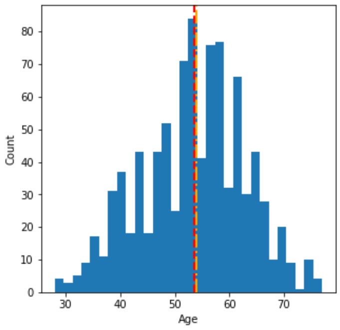
```
heart_df.describe()
```

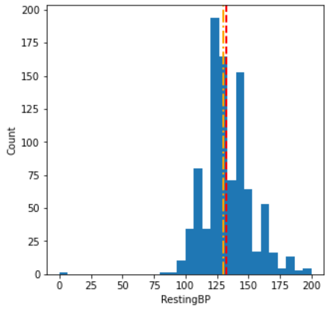
Out[59]:

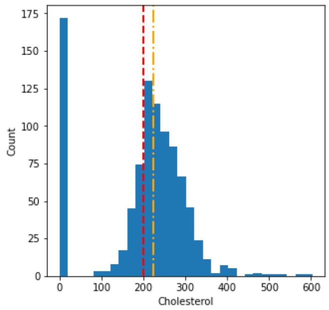
	Age	Sex	Chest pain type	BP	Cholesterol	FBS over 120	EKG results	Max HR	Exercise angina	ST depression	Slope of ST	Number of vessels fluro
count	270.000000	270.000000	270.000000	270.000000	270.000000	270.000000	270.000000	270.000000	270.000000	270.00000	270.000000	270.000000
mean	54.433333	0.677778	3.174074	131.344444	249.659259	0.148148	1.022222	149.677778	0.329630	1.05000	1.585185	0.670370
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
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
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
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
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
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

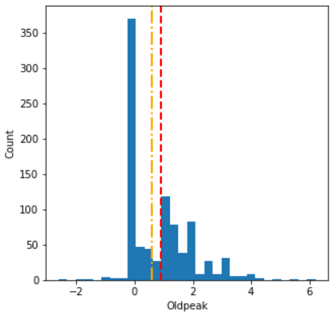
- Data Visualization

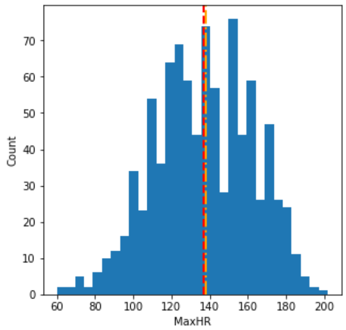
  ➔ Dataset-1



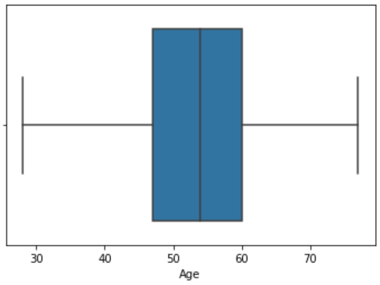


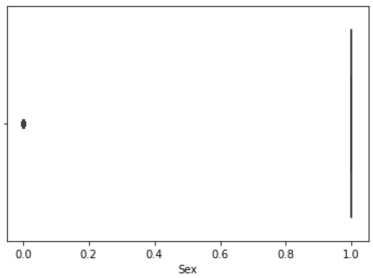


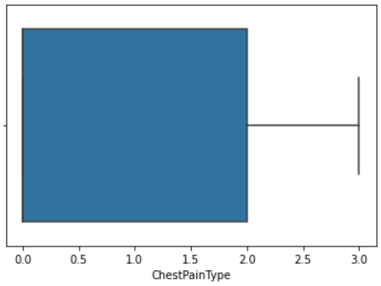


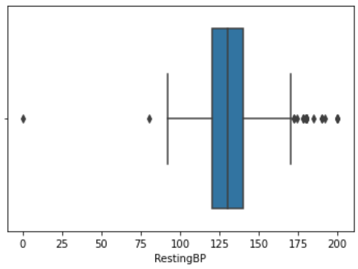


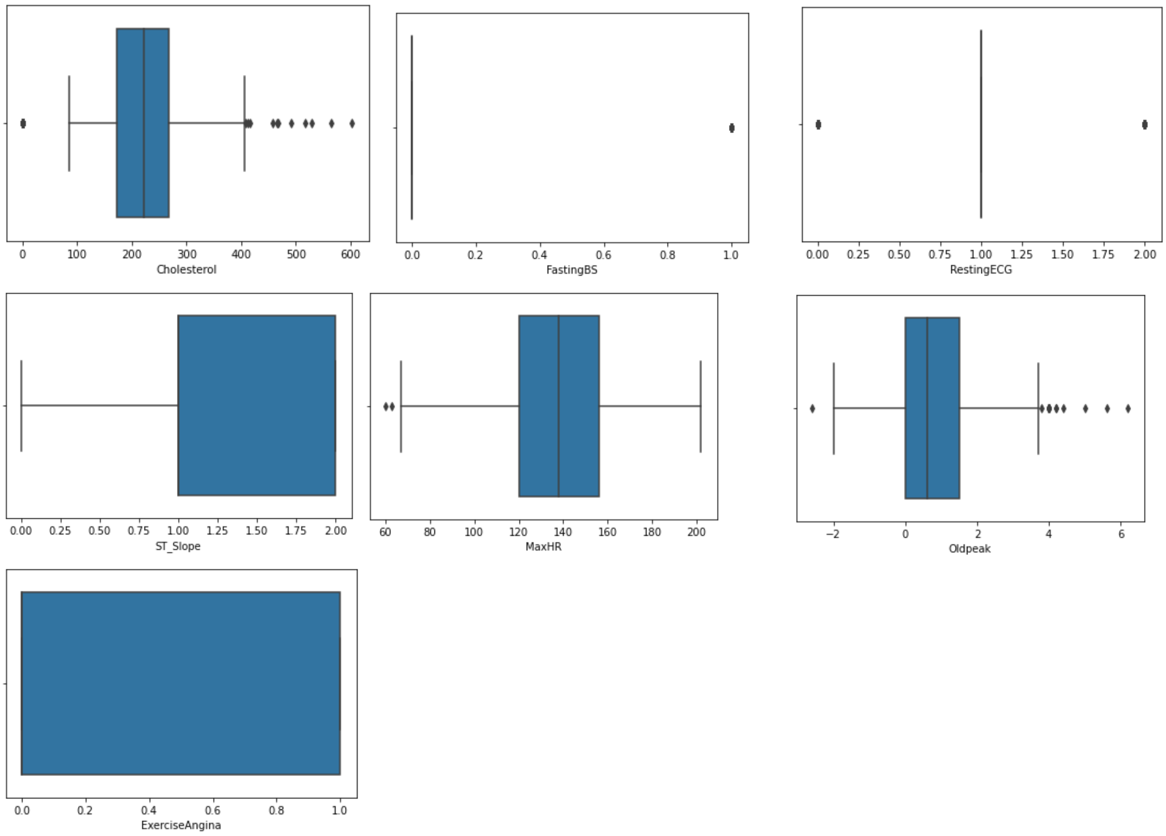
There's not much skewness among the distributions as seen above so, we are not going for any numerical transformations.



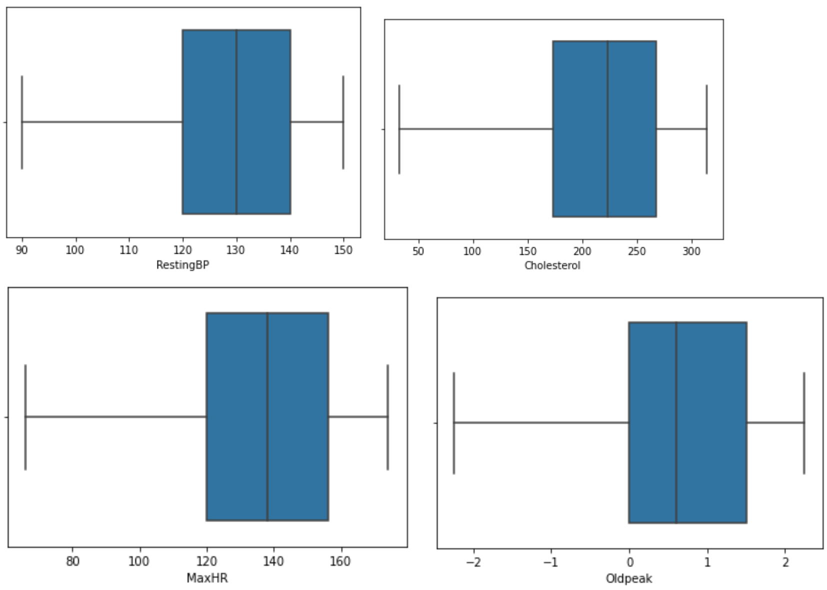




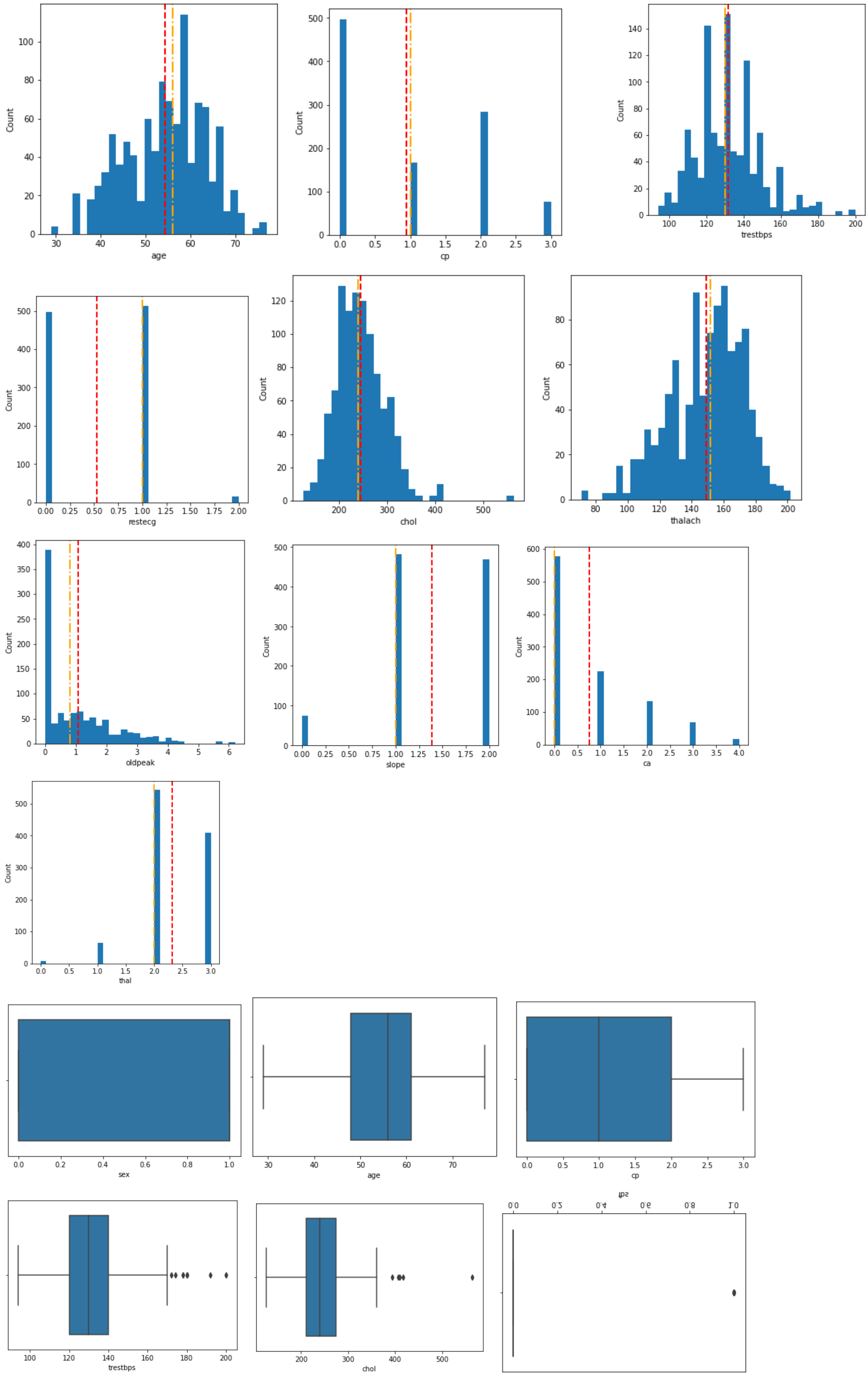




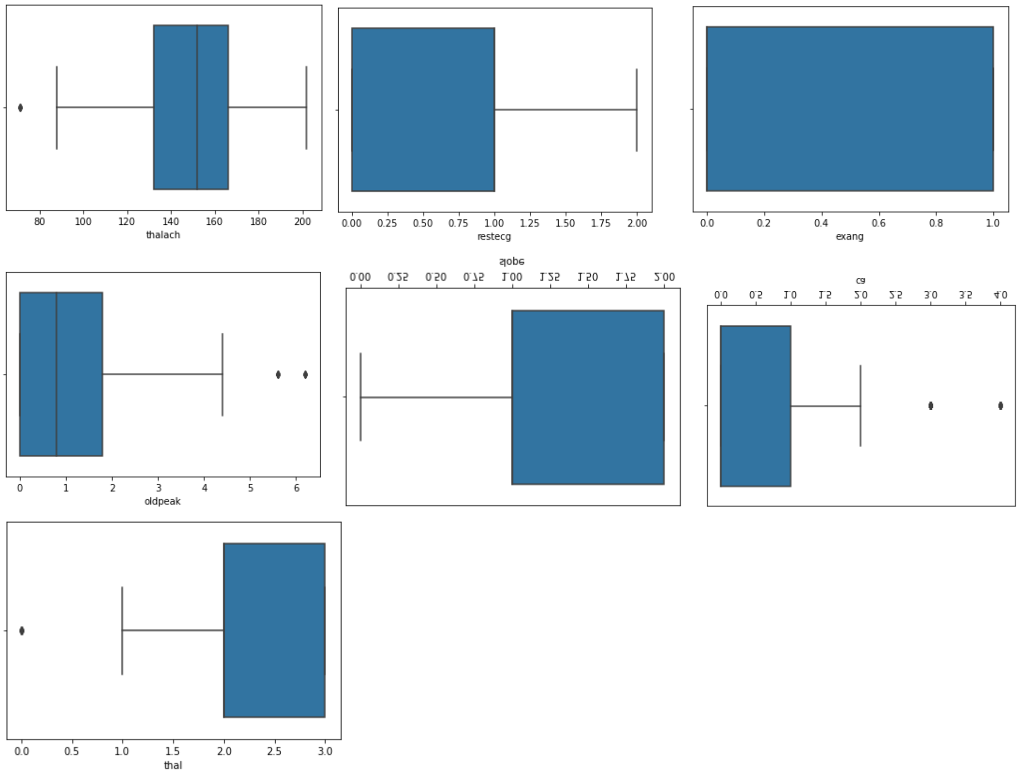
Looking at the above plots, it is clear that the attributes RestingBP, Cholesterol, MaxHR and Oldpeak contain outliers, so we would do best to cap them (we are not removing them since we are taking into account all the attributes may be a good contributor to the target).



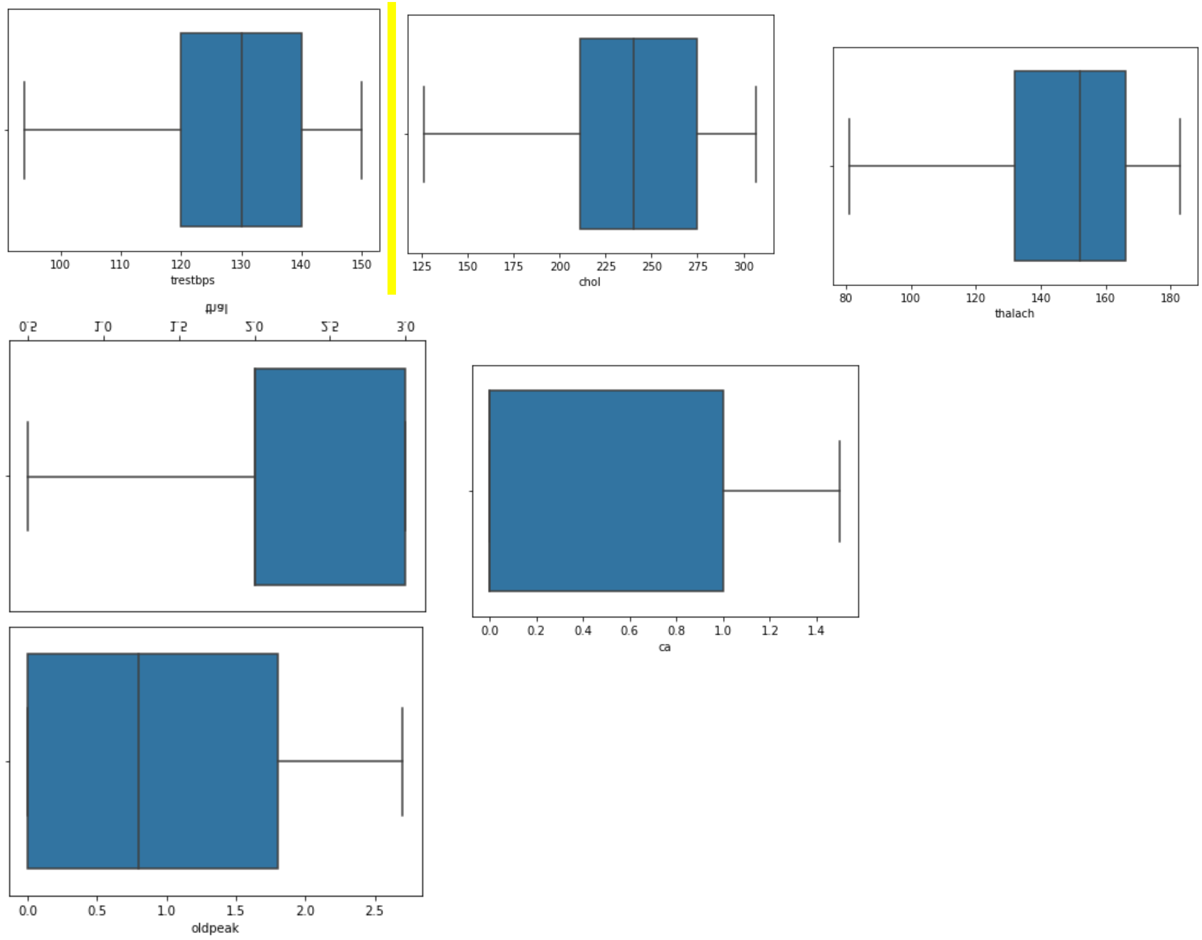
➔ Dataset-2



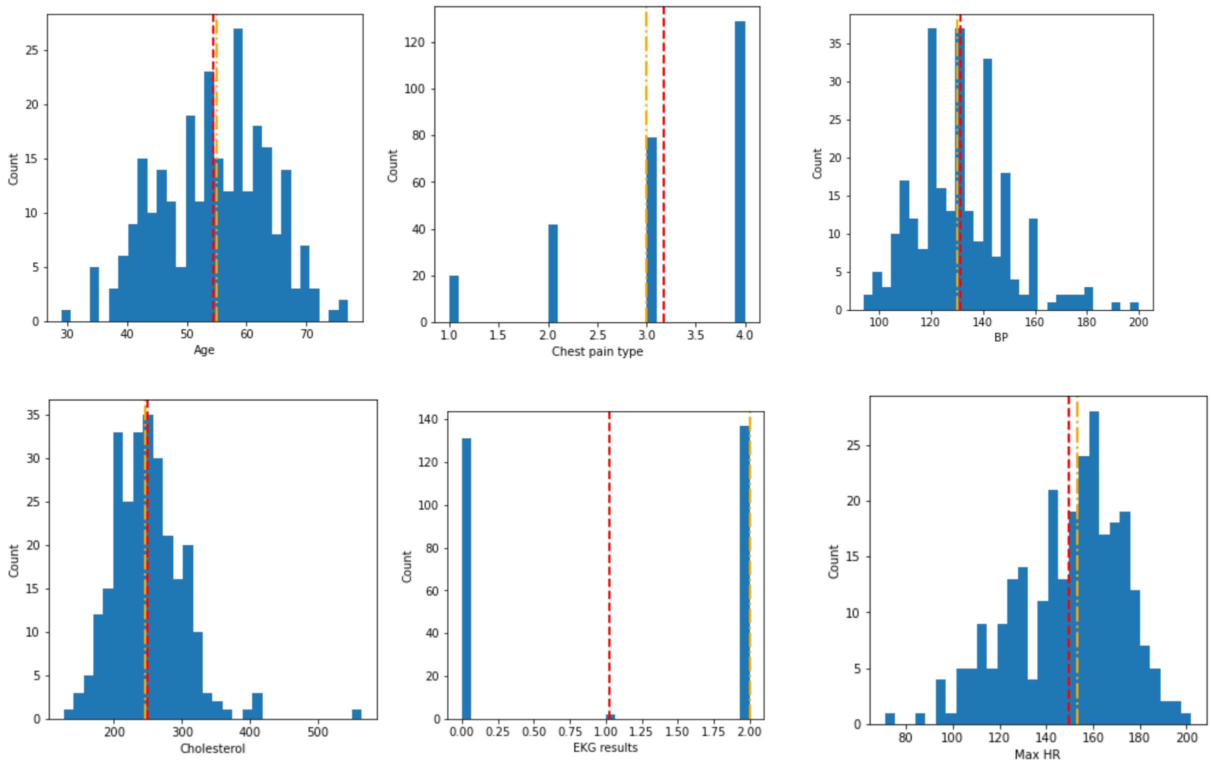
Count
age
cp
trestbps
restecg
chol
thalach
oldpeak
slope
ca
thal
sex
age
cp
trestbps
chol

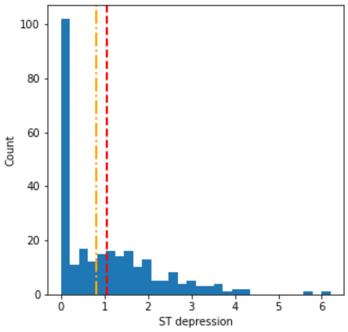
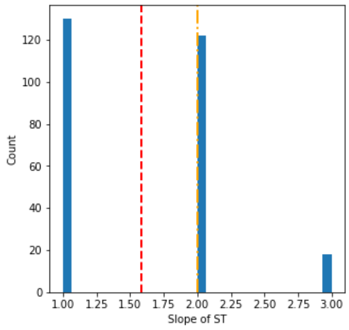
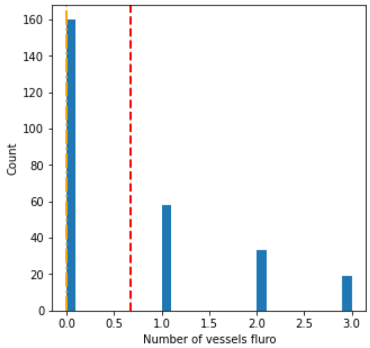
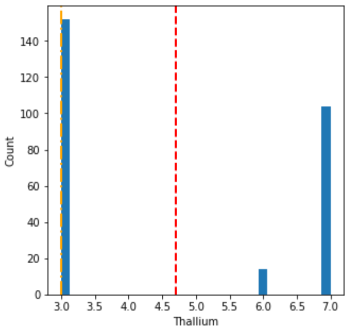
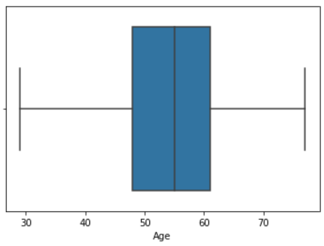
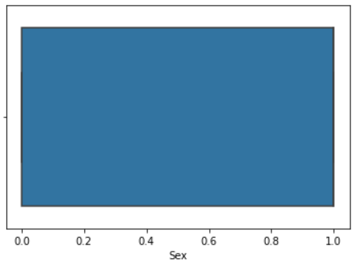
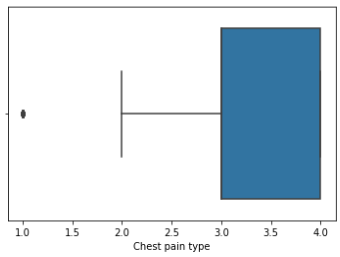
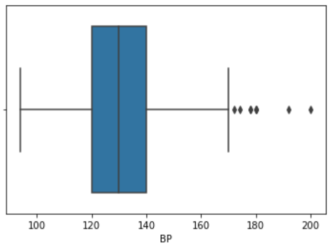
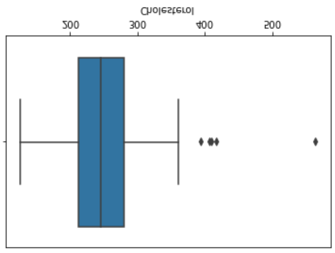
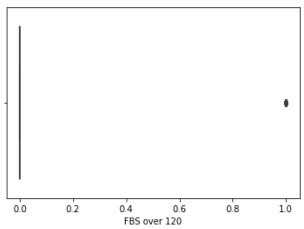
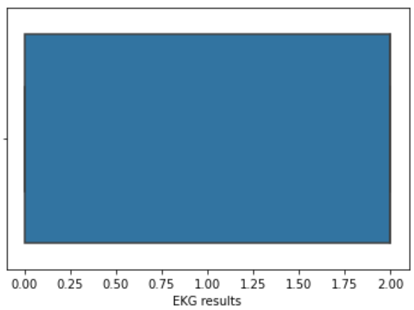
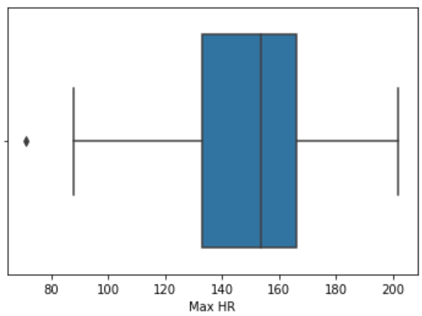


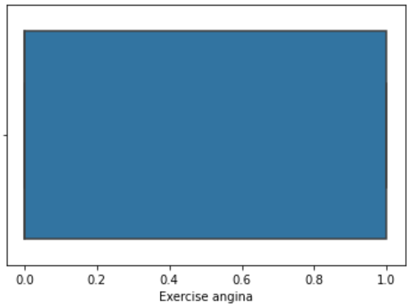
Note - Looking at the above plots, it is clear that the attributes trestbps, chol, thalach, ca, thal and oldpeak contain outliers, so we would do best to cap them (we are not removing them since we are taking into account all the attributes may be a good contributor to the target).

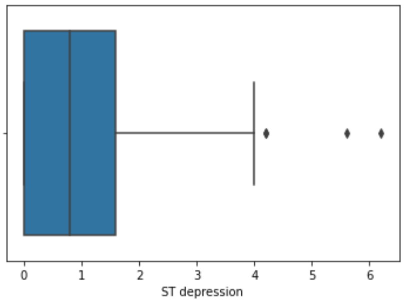

trestbps
chol
thalach
thal
ca
oldpeak

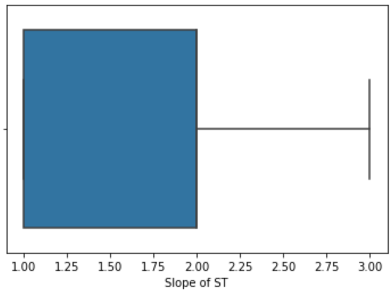
→ Dataset-3

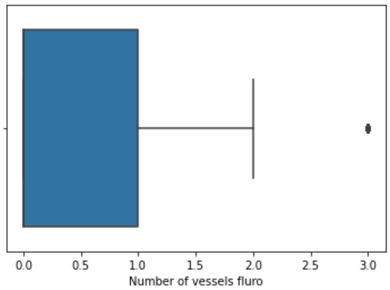

Count
Age
Chest pain type
BP
Cholesterol
EKG results
Max HR

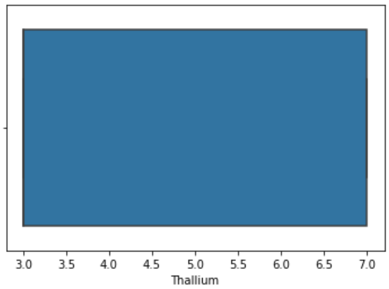

Count
ST depression

Count
Slope of ST

Count
Number of vessels fluro

Count
Thallium

Age

Sex

Chest pain type

BP

Cholesterol

FBS over 120

EKG results

Max HR



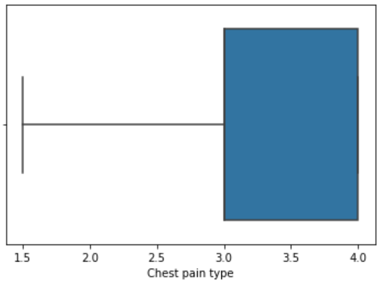


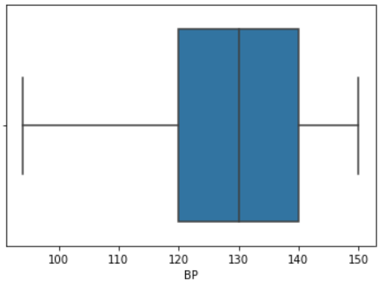


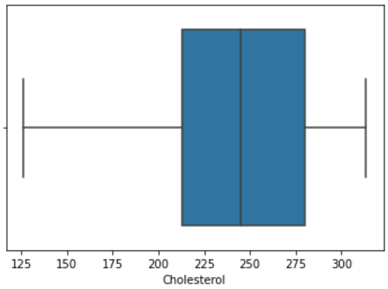


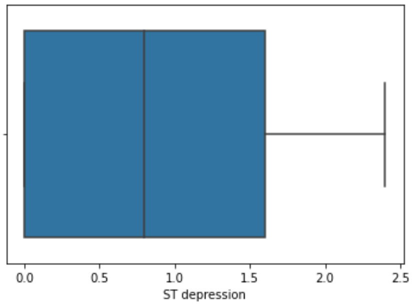


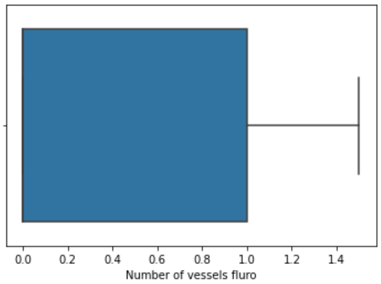
Looking at the above plots, it is clear that the attributes Chest pain type, BP, Cholestrol,ST depression,Number of vessels fluro and Max HR contain outliers, so we would do best to cap them (we are not removing them since we are taking into account all the attributes may be a good contributor to the target).

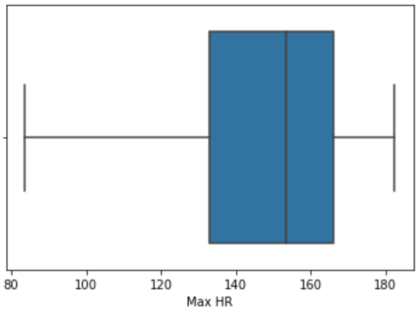












4. Python packages

```
import warnings
warnings.filter warnings('ignore')
import os
import numpy as np
import pandas as pd
import plotly.express as px
import matplotlib.pyplot as plt
import seaborn as sb
%matplotlib inline
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import MinMaxScaler          - Normalization
from sklearn.model_selection import cross_val_score, cross_val_predict
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split      -  Splitting
from sklearn.preprocessing import StandardScaler          - For transforming columns
from sklearn.linear_model import LogisticRegression       - Logistic Regression
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import recall_score, f1_score, classification_report
from sklearn.tree import DecisionTreeClassifier            -    Decision Tree
from sklearn.ensemble import RandomForestClassifier   -    Random forest
from sklearn.svm import SVC                                -    Support vector
```

5. Learning Algorithms

we'll Train various **Classification** Models on the Training set & see which yields the highest accuracy. We will *compare* the accuracy of *Logistic Regression, Decision Trees, Random Forest, and SVM (Support Vector Machine)*.
Note: these are all **supervised learning models**.

Our goal is to predict discrete values, e.g. {1,0}, {True, False}. *We try to* model relationships and dependencies between the target prediction output and the input features such that we can predict the output values for new data based on those relationships which it learned from the previous data sets

Logistic Regression - The LR is the supervised ML binary classification algorithm widely used in most application. It works on categorical dependent variable the result can be discrete or binary categorical variable 0 or 1.
Decision Trees - Decision trees are extremely useful for data analytics and machine learning because they break down complex data into more manageable parts. The main advantage of the decision tree classifier is its ability to using different feature subsets and decision rules at different stages of classification.
Random Forest - The decision tree model is built using two phases, the training phase, and the prediction phase. At the training phase, the decision tree is built using the training instances with calculation of statistical measure such as entropy and information gain of the attributes in the data set. In the prediction phase, the target class for the given test data is predicted using the test instance.

SVM (Support Vector Machine) - An SVM classifier is a linear classifier where the separating hyper plane is chosen to minimize the expected classification error of the unseen test patterns. SVM classify both linear and non-linear data. The main aim of the SVM classifier is to find the hyper plane in an n-dimensional space.

- **Dataset1**
  - **Splitting**

In [74]:

```
## Splitting the data into Train & Test

from sklearn.model_selection import train_test_split

x_train, x_test, Y_train, Y_test = train_test_split(heart_df.iloc[:, :-1].values, heart_df.iloc[:, -1].values,
                                                    test_size = 0.3, random_state = 123)

x_train.shape, x_test.shape, Y_train.shape, Y_test.shape
```

  - **Models**

Logistic Regression

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV
lr_model = LogisticRegression()
```

```
param_grid = [
    {'penalty' : ['l1', 'l2', 'elasticnet', 'none'],
    'C' : np.logspace(-4, 4, 20),
    'solver' : ['lbfgs','newton-cg','liblinear','sag','saga'],
    'max_iter' : [100, 1000,2500, 5000]
    }
]
```

```
clf = GridSearchCV(lr_model, param_grid = param_grid, cv = 3, verbose=True, n_jobs=-1)
best_clf = clf.fit(sc_x_train,Y_train)
```

```
Fitting 3 folds for each of 1600 candidates, totalling 4800 fits
```

```
best_clf.best_estimator_
```

```
LogisticRegression(C=0.08858667904100823, penalty='l1', solver='saga')
```

```
from sklearn.metrics import recall_score, f1_score, classification_report

lr_model = LogisticRegression(solver = 'saga', penalty = 'l1', C = 0.08858667904100823, random_state = 123)

Y_pred = lr_model.fit(sc_x_train, Y_train).predict(sc_x_test)
lr_acc_score = accuracy_score(Y_test, Y_pred)
print('The Recall score for the LogisticRegression model: ', recall_score(Y_test, Y_pred))
print('The corresponding F1-score: ', f1_score(Y_test, Y_pred))
print('\n\n')
print('The corresponding Classification Report: \n', classification_report(Y_test, Y_pred))
accuracy_train.append(lr_model.score(sc_x_train,Y_train))
accuracy_test.append(lr_model.score(sc_x_test,Y_test))
recall_model.append(recall_score(Y_test, Y_pred))
```

```
The Recall score for the LogisticRegression model:  0.8910256410256411
The corresponding F1-score:  0.8660436137071651



The corresponding Classification Report: 
               precision    recall  f1-score   support

           0       0.85      0.78      0.81       120
           1       0.84      0.89      0.87       156

    accuracy                           0.84       276
   macro avg       0.84      0.84      0.84       276
weighted avg       0.84      0.84      0.84       276
```

Decision tree

```
from sklearn.tree import DecisionTreeClassifier
dt_model = DecisionTreeClassifier()
```

```
parameters = {'max_depth' : (3,5,7,9,10,15,20,25)
              , 'criterion' : ('gini', 'entropy')
              , 'max_features' : ('auto', 'sqrt', 'log2')
              , 'min_samples_split' : (2,4,6)
              , 'min_samples_leaf' : (3,5,7,9,10,15,20)
              , 'min_samples_split' : (2,3,4)
             }
DT_grid  = GridSearchCV(DecisionTreeClassifier(), param_grid = parameters, cv = 5, verbose = 2,n_jobs=-1)
DT_grid.fit(sc_x_train,Y_train)
```

```
Fitting 5 folds for each of 1008 candidates, totalling 5040 fits
GridSearchCV(cv=5, estimator=DecisionTreeClassifier(), n_jobs=-1,
             param_grid={'criterion': ('gini', 'entropy'),
                         'max_depth': (3, 5, 7, 9, 10, 15, 20, 25),
                         'max_features': ('auto', 'sqrt', 'log2'),
                         'min_samples_leaf': (3, 5, 7, 9, 10, 15, 20),
                         'min_samples_split': (2, 3, 4)},
             verbose=2)
```

```
DT_grid.best_estimator_
```

```
DecisionTreeClassifier(criterion='entropy', max_depth=20, max_features='sqrt',
                       min_samples_leaf=9, min_samples_split=4)
```

```
dt_model = DecisionTreeClassifier(criterion = 'entropy', max_depth = 9, min_samples_leaf = 9,
                                  min_samples_split = 4, random_state = 123)

Y_pred = dt_model.fit(sc_x_train, Y_train).predict(sc_x_test)
dt_acc_score = accuracy_score(Y_test, Y_pred)
print('The Recall score for the DecisionTree model: ', recall_score(Y_test, Y_pred))
print('The corresponding F1-score: ', f1_score(Y_test, Y_pred))
print('\n\n')
print('The corresponding Classification Report: \n', classification_report(Y_test, Y_pred))
accuracy_train.append(dt_model.score(sc_x_train,Y_train))
accuracy_test.append(dt_model.score(sc_x_test,Y_test))
recall_model.append(recall_score(Y_test, Y_pred))
```

```
The Recall score for the DecisionTree model:  0.8782051282051282
The corresponding F1-score:  0.8589341692789969



The corresponding Classification Report: 
               precision    recall  f1-score   support

           0       0.83      0.78      0.81       120
           1       0.84      0.88      0.86       156

    accuracy                           0.84       276
   macro avg       0.84      0.83      0.83       276
weighted avg       0.84      0.84      0.84       276
```

Random Forest

```
from sklearn.ensemble import RandomForestClassifier
rf_model = RandomForestClassifier()
```

```
n_estimators=[830,833,840]
min_samples_split=[30,35]
min_samples_leaf=[30,35,40]
max_features=["auto"]
max_depth=[150,160,170]
criterion=["entropy"]
space_grid={"n_estimators":n_estimators,
           "min_samples_split":min_samples_split,
           "min_samples_leaf":min_samples_leaf,
           "max_features":max_features,
           "max_depth":max_depth,
           "criterion":criterion}
```

```
grid=GridSearchCV(estimator=rf_model,param_grid=space_grid,cv=5,verbose=2,n_jobs=-1)
grid.fit(sc_x_train,Y_train)
```

```
Fitting 5 folds for each of 54 candidates, totalling 270 fits
GridSearchCV(cv=5, estimator=RandomForestClassifier(), n_jobs=-1,
             param_grid={'criterion': ['entropy'], 'max_depth': [150, 160, 170],
                         'max_features': ['auto'],
                         'min_samples_leaf': [30, 35, 40],
                         'min_samples_split': [30, 35],
                         'n_estimators': [830, 833, 840]},
             verbose=2)
```

```
grid.best_params_
```

```
{'criterion': 'entropy',
 'max_depth': 150,
 'max_features': 'auto',
 'min_samples_leaf': 30,
 'min_samples_split': 35,
 'n_estimators': 830}
```

```
rf_model = RandomForestClassifier(criterion = 'entropy', max_depth = 150, min_samples_leaf = 35,
                                  min_samples_split = 30, n_estimators = 830, n_jobs = 1, random_state = 123)

Y_pred = rf_model.fit(sc_x_train, Y_train).predict(sc_x_test)
rf_acc_score = accuracy_score(Y_test, Y_pred)
print('The Recall score for the RandomForest model: ', recall_score(Y_test, Y_pred))
print('The corresponding F1-score: ', f1_score(Y_test, Y_pred))
print('\n\n')
print('The corresponding Classification Report: \n', classification_report(Y_test, Y_pred))
accuracy_train.append(rf_model.score(sc_x_train,Y_train))
accuracy_test.append(rf_model.score(sc_x_test,Y_test))
recall_model.append(recall_score(Y_test, Y_pred))
```

```
The Recall score for the RandomForest model:  0.9230769230769231
The corresponding F1-score:  0.8807339449541285



The corresponding Classification Report: 
               precision    recall  f1-score   support

           0       0.89      0.78      0.83       120
           1       0.84      0.92      0.88       156

    accuracy                           0.86       276
   macro avg       0.86      0.85      0.85       276
weighted avg       0.86      0.86      0.86       276
```

Support vector Classifier

```
from sklearn.svm import SVC
svc_model = SVC()
```

```
param_grid = {'C': [1,10,100],
              'gamma': [0.1,0.01,0.001, 'scale', 'auto'],
              'kernel': ['linear', 'poly', 'sigmoid']}
grid = GridSearchCV(svc_model,param_grid)
grid.fit(sc_x_train,Y_train)
```

```
GridSearchCV(estimator=SVC(),
             param_grid={'C': [1, 10, 100],
                         'gamma': [0.1, 0.01, 0.001, 'scale', 'auto'],
                         'kernel': ['linear', 'poly', 'sigmoid']})
```

```
svc_model = SVC(C = 1, gamma = 0.1, random_state = 123)

Y_pred = svc_model.fit(sc_x_train, Y_train).predict(sc_x_test)
svc_acc_score = accuracy_score(Y_test, Y_pred)
print('The Recall score for the SVM model: ', recall_score(Y_test, Y_pred))
print('The corresponding F1-score: ', f1_score(Y_test, Y_pred))
print('\n\n')
print('The corresponding Classification Report: \n', classification_report(Y_test, Y_pred))
accuracy_train.append(svc_model.score(sc_x_train,Y_train))
accuracy_test.append(svc_model.score(sc_x_test,Y_test))
recall_model.append(recall_score(Y_test, Y_pred))
```

```
The Recall score for the SVM model:  0.9423076923076923
The corresponding F1-score:  0.8963414634146342



The corresponding Classification Report:
               precision    recall  f1-score   support

           0       0.91      0.79      0.85       120
           1       0.85      0.94      0.90       156

    accuracy                           0.88       276
   macro avg       0.88      0.87      0.87       276
weighted avg       0.88      0.88      0.88       276
```

Model Evaluation

```
Models = ['Logistic Regression','Decision Tree','Random Forest','Support Vector Machine']
total_v2 = list(zip(Models,accuracy_train,accuracy_test,recall_model))
output_v2 = pd.DataFrame(total_v2, columns = ['Models','Accuracy_train','Accuracy_test','Recall'])

s_v2 = output_v2.groupby(['Models'])['Accuracy_train','Accuracy_test','Recall'].mean().reset_index().sort_values(by='Accur
s_v2.head(10).style.background_gradient(cmap='Reds')
```

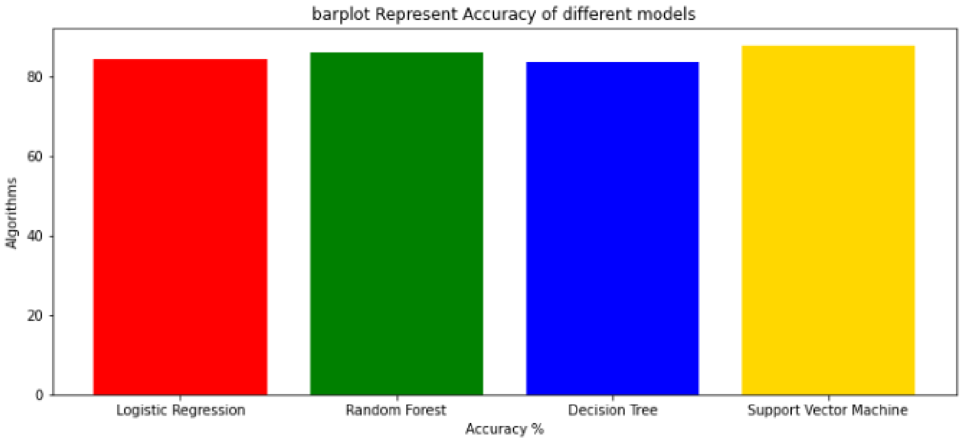
	Models	Accuracy_train	Accuracy_test	Recall
3	Support Vector Machine	0.909657	0.876812	0.942308
2	Random Forest	0.859813	0.858696	0.923077
1	Logistic Regression	0.869159	0.844203	0.891026
0	Decision Tree	0.897196	0.836957	0.878205

Comparisons

```
model_ev = pd.DataFrame({'Model': ['Logistic Regression','Random Forest','Decision Tree','Support Vector Machine'], 'Accur
print(lr_acc_score*100)
print(rf_acc_score*100)
print(dt_acc_score*100)
print(svc_acc_score*100)
```

```
84.42028985507247
85.86956521739131
83.69565217391305
87.68115942028986
```

```
colors = ['red','green','blue','gold','orange']
plt.figure(figsize=(12,5))
plt.title("barplot Represent Accuracy of different models")
plt.xlabel("Accuracy %")
plt.ylabel("Algorithms")
plt.bar(model_ev['Model'],model_ev['Accuracy'],color = colors)
plt.show()
```



Conclusion

Given the above `Recall` scores and comparions between all the models(As it is a Healthcare problem, Recall is considered to be the best metric solver in the Healthcare predictions, as we need to keep the margin of error really really small), as well as the corresponding F1 scores, we consider using `Support Vector models` here.... Exerciseangina,Chest pain is major symptoms of heart attack.

➔ **Dataset2**

- **Splitting**

```
## Splitting the data into Train & Test

from sklearn.model_selection import train_test_split

x_train, x_test, Y_train, Y_test = train_test_split(heart_df.iloc[:, :-1].values, heart_df.iloc[:, -1].values,
                                                    test_size = 0.3, random_state = 123)

x_train.shape, x_test.shape, Y_train.shape, Y_test.shape
```

```
((717, 13), (308, 13), (717,), (308,))
```

- **Models**

Logistic Regression

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV
lr_model = LogisticRegression()
```

```
param_grid = [
    {'penalty' : ['l1', 'l2', 'elasticnet', 'none'],
    'C' : np.logspace(-4, 4, 20),
    'solver' : ['lbfgs','newton-cg','liblinear','sag','saga'],
    'max_iter' : [100, 1000,2500, 5000]
    }
]
```

```
clf = GridSearchCV(lr_model, param_grid = param_grid, cv = 3, verbose=True, n_jobs=-1)
best_clf = clf.fit(sc_x_train,Y_train)
```

```
Fitting 3 folds for each of 1600 candidates, totalling 4800 fits
```

```
best_clf.best_estimator_
```

```
LogisticRegression(C=0.08858667904100823, penalty='l1', solver='liblinear')
```

```
from sklearn.metrics import recall_score, f1_score, classification_report

lr_model = LogisticRegression(solver = 'liblinear', penalty = 'l1', C = 0.08858667904100823, random_state = 123)

Y_pred = lr_model.fit(sc_x_train, Y_train).predict(sc_x_test)
lr_acc_score = accuracy_score(Y_test, Y_pred)
print('The Recall score for the LogisticRegression model: ', recall_score(Y_test, Y_pred))
print('The corresponding F1-score: ', f1_score(Y_test, Y_pred))
print('\n\n')
print('The corresponding Classification Report: \n', classification_report(Y_test, Y_pred))
accuracy_train.append(lr_model.score(sc_x_train,Y_train))
accuracy_test.append(lr_model.score(sc_x_test,Y_test))
recall_model.append(recall_score(Y_test, Y_pred))
```

```
The Recall score for the LogisticRegression model:  0.9155844155844156
The corresponding F1-score:  0.8952380952380953



The corresponding Classification Report: 
               precision    recall  f1-score   support

           0       0.91      0.87      0.89       154
           1       0.88      0.92      0.90       154

    accuracy                           0.89       308
   macro avg       0.89      0.89      0.89       308
weighted avg       0.89      0.89      0.89       308
```

Decision Tree

```
from sklearn.tree import DecisionTreeClassifier
dt_model = DecisionTreeClassifier()
```

```
parameters = {'max_depth' : (3,5,7,9,10,15,20,25)
              , 'criterion' : ('gini', 'entropy')
              , 'max_features' : ('auto', 'sqrt', 'log2')
              , 'min_samples_split' : (2,4,6)
              , 'min_samples_leaf' : (3,5,7,9,10,15,20)
              , 'min_samples_split' : (2,3,4)
             }
DT_grid  = GridSearchCV(DecisionTreeClassifier(), param_grid = parameters, cv = 5, verbose = 2,n_jobs=-1)
DT_grid.fit(sc_x_train,Y_train)
```

```
Fitting 5 folds for each of 1008 candidates, totalling 5040 fits
GridSearchCV(cv=5, estimator=DecisionTreeClassifier(), n_jobs=-1,
             param_grid={'criterion': ('gini', 'entropy'),
                         'max_depth': (3, 5, 7, 9, 10, 15, 20, 25),
                         'max_features': ('auto', 'sqrt', 'log2'),
                         'min_samples_leaf': (3, 5, 7, 9, 10, 15, 20),
                         'min_samples_split': (2, 3, 4)},
             verbose=2)
```

```
DT_grid.best_estimator_
```

```
DecisionTreeClassifier(criterion='entropy', max_depth=25, max_features='auto',
                       min_samples_leaf=3)
```

```
dt_model = DecisionTreeClassifier(criterion = 'entropy', max_depth = 15, min_samples_leaf = 3,
                                  min_samples_split = 4, random_state = 123)

Y_pred = dt_model.fit(sc_x_train, Y_train).predict(sc_x_test)
dt_acc_score = accuracy_score(Y_test, Y_pred)
print('The Recall score for the DecisionTree model: ', recall_score(Y_test, Y_pred))
print('The corresponding F1-score: ', f1_score(Y_test, Y_pred))
print('\n\n')
print('The corresponding Classification Report: \n', classification_report(Y_test, Y_pred))
accuracy_train.append(dt_model.score(sc_x_train,Y_train))
accuracy_test.append(dt_model.score(sc_x_test,Y_test))
recall_model.append(recall_score(Y_test, Y_pred))
```

```
The Recall score for the DecisionTree model:  0.922077922077922
The corresponding F1-score:  0.9342105263157895



The corresponding Classification Report:
               precision    recall  f1-score   support

           0       0.92      0.95      0.94       154
           1       0.95      0.92      0.93       154

    accuracy                           0.94       308
   macro avg       0.94      0.94      0.94       308
weighted avg       0.94      0.94      0.94       308
```

Random Forest

```
from sklearn.ensemble import RandomForestClassifier
rf_model = RandomForestClassifier()
```

```
n_estimators=[830,833,840]
min_samples_split=[30,35]
min_samples_leaf=[30,35,40]
max_features=["auto"]
max_depth=[150,160,170]
criterion=["entropy"]
space_grid={"n_estimators":n_estimators,
           "min_samples_split":min_samples_split,
           "min_samples_leaf":min_samples_leaf,
           "max_features":max_features,
           "max_depth":max_depth,
           "criterion":criterion}
```

```
grid=GridSearchCV(estimator=rf_model,param_grid=space_grid,cv=5,verbose=2,n_jobs=-1)
grid.fit(sc_x_train,Y_train)
```

```
Fitting 5 folds for each of 54 candidates, totalling 270 fits
GridSearchCV(cv=5, estimator=RandomForestClassifier(), n_jobs=-1,
             param_grid={'criterion': ['entropy'], 'max_depth': [150, 160, 170],
                         'max_features': ['auto'],
                         'min_samples_leaf': [30, 35, 40],
                         'min_samples_split': [30, 35],
                         'n_estimators': [830, 833, 840]},
             verbose=2)
```

```
grid.best_params_
```

```
{'criterion': 'entropy',
 'max_depth': 170,
 'max_features': 'auto',
 'min_samples_leaf': 30,
 'min_samples_split': 30,
 'n_estimators': 840}
```

```
rf_model = RandomForestClassifier(criterion = 'entropy', max_depth = 170, min_samples_leaf = 30,
                                  min_samples_split = 30, n_estimators = 833, n_jobs = 1, random_state = 123)

Y_pred = rf_model.fit(sc_x_train, Y_train).predict(sc_x_test)
rf_acc_score = accuracy_score(Y_test, Y_pred)
print('The Recall score for the RandomForest model: ', recall_score(Y_test, Y_pred))
print('The corresponding F1-score: ', f1_score(Y_test, Y_pred))
print('\n\n')
print('The corresponding Classification Report: \n', classification_report(Y_test, Y_pred))
accuracy_train.append(rf_model.score(sc_x_train,Y_train))
accuracy_test.append(rf_model.score(sc_x_test,Y_test))
recall_model.append(recall_score(Y_test, Y_pred))
```

```
The Recall score for the RandomForest model:  0.9090909090909091
The corresponding F1-score:  0.888888888888889



The corresponding Classification Report:
               precision    recall  f1-score   support

           0       0.90      0.86      0.88       154
           1       0.87      0.91      0.89       154

    accuracy                           0.89       308
   macro avg       0.89      0.89      0.89       308
weighted avg       0.89      0.89      0.89       308
```

Support vector classifier

```
from sklearn.svm import SVC
svc_model = SVC()
```

```
param_grid = {'C': [1,10,100],
              'gamma': [0.1,0.01,0.001, 'scale', 'auto'],
              'kernel': ['linear', 'poly', 'sigmoid']}
grid = GridSearchCV(svc_model,param_grid)
grid.fit(sc_x_train,Y_train)
```

```
GridSearchCV(estimator=SVC(),
             param_grid={'C': [1, 10, 100],
                         'gamma': [0.1, 0.01, 0.001, 'scale', 'auto'],
                         'kernel': ['linear', 'poly', 'sigmoid']})
```

```
grid.best_params_
```

```
{'C': 100, 'gamma': 0.1, 'kernel': 'poly'}
```

```
svc_model = SVC(C = 100, gamma = 0.1, random_state = 123)

Y_pred = svc_model.fit(sc_x_train, Y_train).predict(sc_x_test)
svc_acc_score = accuracy_score(Y_test, Y_pred)
print('The Recall score for the SVM model: ', recall_score(Y_test, Y_pred))
print('The corresponding F1-score: ', f1_score(Y_test, Y_pred))
print('\n\n')
print('The corresponding Classification Report: \n', classification_report(Y_test, Y_pred))
accuracy_train.append(svc_model.score(sc_x_train,Y_train))
accuracy_test.append(svc_model.score(sc_x_test,Y_test))
recall_model.append(recall_score(Y_test, Y_pred))
```

```
The Recall score for the SVM model:  0.961038961038961
The corresponding F1-score:  0.9801324503311257



The corresponding Classification Report: 
               precision    recall  f1-score   support

           0       0.96      1.00      0.98       154
           1       1.00      0.96      0.98       154

    accuracy                           0.98       308
   macro avg       0.98      0.98      0.98       308
weighted avg       0.98      0.98      0.98       308
```

Model evaluation

```
Models = ['Logistic Regression','Decision Tree','Random Forest','Support Vector Machine']
total_v2 = list(zip(Models,accuracy_train,accuracy_test,recall_model))
output_v2 = pd.DataFrame(total_v2, columns = ['Models','Accuracy_train','Accuracy_test','Recall'])

s_v2 = output_v2.groupby(['Models'])['Accuracy_train','Accuracy_test','Recall'].mean().reset_index().sort_values(by='Accuracy_test',ascendi
s_v2.head(10).style.background_gradient(cmap='Reds')
```

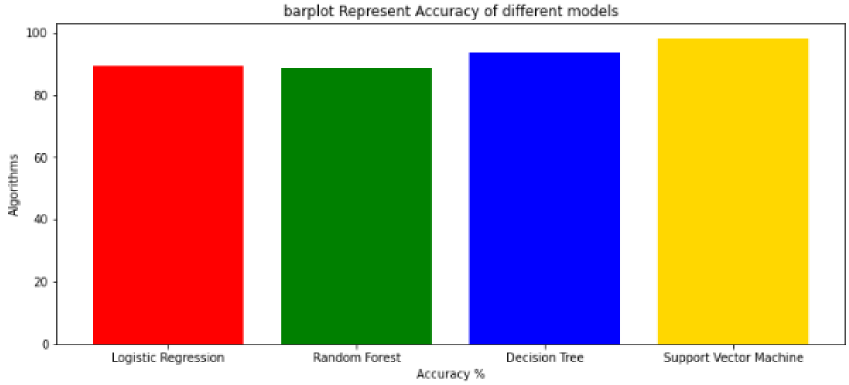
	Models	Accuracy_train	Accuracy_test	Recall
3	Support Vector Machine	1.000000	0.980519	0.961039
0	Decision Tree	0.987448	0.935065	0.922078
1	Logistic Regression	0.845188	0.892857	0.915584
2	Random Forest	0.874477	0.886364	0.909091

Comparisons

```
model_ev = pd.DataFrame({'Model': ['Logistic Regression','Random Forest','Decision Tree','Support Vector Machine'], 'Accuracy': [lr_acc_scor
print(lr_acc_score*100)
print(rf_acc_score*100)
print(dt_acc_score*100)
print(svc_acc_score*100)
```

```
89.28571428571429
88.63636363636364
93.5064935064935
98.05194805194806
```

```
colors = ['red','green','blue','gold','orange']
plt.figure(figsize=(12,5))
plt.title("barplot Represent Accuracy of different models")
plt.xlabel("Accuracy %")
plt.ylabel("Algorithms")
plt.bar(model_ev['Model'],model_ev['Accuracy'],color = colors)
plt.show()
```



Conclusion

Given the above `Recall` scores and comparions between all the models(As it is a Healthcare problem, Recall is considered to be the best metric solver in the Healthcare predictions, as we need to keep the margin of error really really small), as well as the corresponding F1 scores, we consider using `Support Vector models` here.... Exerciseangina,Chest pain is major symptoms of heart attack.

➔ **Dataset3**

- **Splitting**

```
## Splitting the data into Train & Test

from sklearn.model_selection import train_test_split

x_train, x_test, Y_train, Y_test = train_test_split(heart_df.iloc[:, :-1].values, heart_df.iloc[:, -1].values,
                                                    test_size = 0.3, random_state = 123)

x_train.shape, x_test.shape, Y_train.shape, Y_test.shape
```

```
((189, 13), (81, 13), (189,), (81,))
```

- **Models**

Logistic Regression

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV
lr_model = LogisticRegression()
```

```
param_grid = [
    {'penalty' : ['l1', 'l2', 'elasticnet', 'none'],
    'C' : np.logspace(-4, 4, 20),
    'solver' : ['lbfgs','newton-cg','liblinear','sag','saga'],
    'max_iter' : [100, 1000,2500, 5000]
    }
]
```

```
clf = GridSearchCV(lr_model, param_grid = param_grid, cv = 3, verbose=True, n_jobs=-1)
best_clf = clf.fit(sc_x_train,Y_train)
```

```
Fitting 3 folds for each of 1600 candidates, totalling 4800 fits
```

```
best_clf.best_estimator_
```

```
LogisticRegression(C=0.23357214690901212, penalty='l1', solver='saga')
```

```
from sklearn.metrics import recall_score, f1_score, classification_report

lr_model = LogisticRegression(solver = 'saga', penalty = 'l1', C = 0.23357214690901212, random_state = 123)

Y_pred = lr_model.fit(sc_x_train, Y_train).predict(sc_x_test)
lr_acc_score = accuracy_score(Y_test, Y_pred)
print('The Recall score for the LogisticRegression model: ', recall_score(Y_test, Y_pred))
print('The corresponding F1-score: ', f1_score(Y_test, Y_pred))
print('\n\n')
print('The corresponding Classification Report: \n', classification_report(Y_test, Y_pred))
accuracy_train.append(lr_model.score(sc_x_train,Y_train))
accuracy_test.append(lr_model.score(sc_x_test,Y_test))
recall_model.append(recall_score(Y_test, Y_pred))
```

```
The Recall score for the LogisticRegression model:  0.8055555555555556
The corresponding F1-score:  0.7945205479452055



The corresponding Classification Report: 
               precision    recall  f1-score   support

           0       0.84      0.82      0.83        45
           1       0.78      0.81      0.79        36

    accuracy                           0.81        81
   macro avg       0.81      0.81      0.81        81
weighted avg       0.82      0.81      0.82        81
```

Decision tree

```
from sklearn.tree import DecisionTreeClassifier
dt_model = DecisionTreeClassifier()
```

```
parameters = {'max_depth' : (3,5,7,9,10,15,20,25)
              , 'criterion' : ('gini', 'entropy')
              , 'max_features' : ('auto', 'sqrt', 'log2')
              , 'min_samples_split' : (2,4,6)
              , 'min_samples_leaf' : (3,5,7,9,10,15,20)
              , 'min_samples_split' : (2,3,4)
             }
DT_grid  = GridSearchCV(DecisionTreeClassifier(), param_grid = parameters, cv = 5, verbose = 2,n_jobs=-1)
DT_grid.fit(sc_x_train,Y_train)
```

```
Fitting 5 folds for each of 1008 candidates, totalling 5040 fits
GridSearchCV(cv=5, estimator=DecisionTreeClassifier(), n_jobs=-1,
             param_grid={'criterion': ('gini', 'entropy'),
                         'max_depth': (3, 5, 7, 9, 10, 15, 20, 25),
                         'max_features': ('auto', 'sqrt', 'log2'),
                         'min_samples_leaf': (3, 5, 7, 9, 10, 15, 20),
                         'min_samples_split': (2, 3, 4)},
             verbose=2)
```

```
DT_grid.best_estimator_
```

```
DecisionTreeClassifier(criterion='entropy', max_depth=5, max_features='auto',
                       min_samples_leaf=10)
```

```
dt_model = DecisionTreeClassifier(criterion = 'entropy', max_depth = 9, min_samples_leaf = 5,
                                  min_samples_split = 3, random_state = 123)

Y_pred = dt_model.fit(sc_x_train, Y_train).predict(sc_x_test)
dt_acc_score = accuracy_score(Y_test, Y_pred)
print('The Recall score for the DecisionTree model: ', recall_score(Y_test, Y_pred))
print('The corresponding F1-score: ', f1_score(Y_test, Y_pred))
print('\n\n')
print('The corresponding Classification Report: \n', classification_report(Y_test, Y_pred))
accuracy_train.append(dt_model.score(sc_x_train,Y_train))
accuracy_test.append(dt_model.score(sc_x_test,Y_test))
recall_model.append(recall_score(Y_test, Y_pred))
```

```
The Recall score for the DecisionTree model:  0.75
The corresponding F1-score:  0.7012987012987012



The corresponding Classification Report: 
               precision    recall  f1-score   support

           0       0.78      0.69      0.73        45
           1       0.66      0.75      0.70        36

    accuracy                           0.72        81
   macro avg       0.72      0.72      0.72        81
weighted avg       0.72      0.72      0.72        81
```

Random forest

```
from sklearn.ensemble import RandomForestClassifier
rf_model = RandomForestClassifier()
```

```
n_estimators=[830,833,840]
min_samples_split=[30,35]
min_samples_leaf=[30,35,40]
max_features=["auto"]
max_depth=[150,160,170]
criterion=["entropy"]
space_grid={"n_estimators":n_estimators,
            "min_samples_split":min_samples_split,
            "min_samples_leaf":min_samples_leaf,
            "max_features":max_features,
            "max_depth":max_depth,
            "criterion":criterion}
```

```
grid=GridSearchCV(estimator=rf_model,param_grid=space_grid,cv=5,verbose=2,n_jobs=-1)
grid.fit(sc_x_train,Y_train)
```

```
Fitting 5 folds for each of 54 candidates, totalling 270 fits
GridSearchCV(cv=5, estimator=RandomForestClassifier(), n_jobs=-1,
             param_grid={'criterion': ['entropy'], 'max_depth': [150, 160, 170],
                         'max_features': ['auto'],
                         'min_samples_leaf': [30, 35, 40],
                         'min_samples_split': [30, 35],
                         'n_estimators': [830, 833, 840]},
             verbose=2)
```

```
grid.best_params_
```

```
{'criterion': 'entropy',
 'max_depth': 150,
 'max_features': 'auto',
 'min_samples_leaf': 30,
 'min_samples_split': 30,
 'n_estimators': 830}
```

```
rf_model = RandomForestClassifier(criterion = 'entropy', max_depth = 150, min_samples_leaf = 30,
                                  min_samples_split = 30, n_estimators = 883, n_jobs = 1, random_state = 123)

Y_pred = rf_model.fit(sc_x_train, Y_train).predict(sc_x_test)
rf_acc_score = accuracy_score(Y_test, Y_pred)
print('The Recall score for the RandomForest model: ', recall_score(Y_test, Y_pred))
print('The corresponding F1-score: ', f1_score(Y_test, Y_pred))
print('\n\n')
print('The corresponding Classification Report: \n', classification_report(Y_test, Y_pred))
accuracy_train.append(rf_model.score(sc_x_train,Y_train))
accuracy_test.append(rf_model.score(sc_x_test,Y_test))
recall_model.append(recall_score(Y_test, Y_pred))
```

```
The Recall score for the RandomForest model:  0.6388888888888888
The corresponding F1-score:  0.7301587301587301



The corresponding Classification Report: 
               precision    recall  f1-score   support

           0       0.76      0.91      0.83        45
           1       0.85      0.64      0.73        36

    accuracy                           0.79        81
   macro avg       0.81      0.77      0.78        81
weighted avg       0.80      0.79      0.78        81
```

Support vctor

```
from sklearn.svm import SVC
svc_model = SVC()
```

```
param_grid = {'C': [1,10,100],
              'gamma': [0.1,0.01,0.001, 'scale', 'auto'],
              'kernel': ['linear', 'poly', 'sigmoid']}
grid = GridSearchCV(svc_model,param_grid)
grid.fit(sc_x_train,Y_train)
```

```
GridSearchCV(estimator=SVC(),
             param_grid={'C': [1, 10, 100],
                         'gamma': [0.1, 0.01, 0.001, 'scale', 'auto'],
                         'kernel': ['linear', 'poly', 'sigmoid']})
```

```
grid.best_params_
```

```
{'C': 1, 'gamma': 0.01, 'kernel': 'sigmoid'}
```

```
svc_model = SVC(C = 1, gamma = 0.1, random_state = 123)

Y_pred = svc_model.fit(sc_x_train, Y_train).predict(sc_x_test)
svc_acc_score = accuracy_score(Y_test, Y_pred)
print('The Recall score for the SVM model: ', recall_score(Y_test, Y_pred))
print('The corresponding F1-score: ', f1_score(Y_test, Y_pred))
print('\n\n')
print('The corresponding Classification Report: \n', classification_report(Y_test, Y_pred))
accuracy_train.append(svc_model.score(sc_x_train,Y_train))
accuracy_test.append(svc_model.score(sc_x_test,Y_test))
recall_model.append(recall_score(Y_test, Y_pred))
```

```
The Recall score for the SVM model:  0.8055555555555556
The corresponding F1-score:  0.7837837837837838



The corresponding Classification Report: 
              precision    recall  f1-score   support

           0       0.84      0.80      0.82        45
           1       0.76      0.81      0.78        36

    accuracy                           0.80        81
   macro avg       0.80      0.80      0.80        81
weighted avg       0.80      0.80      0.80        81
```

Model Evaluation

```
Models = ['Logistic Regression','Decision Tree','Random Forest','Support Vector Machine']
total_v2 = list(zip(Models,accuracy_train,accuracy_test,recall_model))
output_v2 = pd.DataFrame(total_v2, columns = ['Models','Accuracy_train','Accuracy_test','Recall'])

s_v2 = output_v2.groupby(['Models'])['Accuracy_train','Accuracy_test','Recall'].mean().reset_index().sort_values(by='Accur
s_v2.head(10).style.background_gradient(cmap='Reds')
```

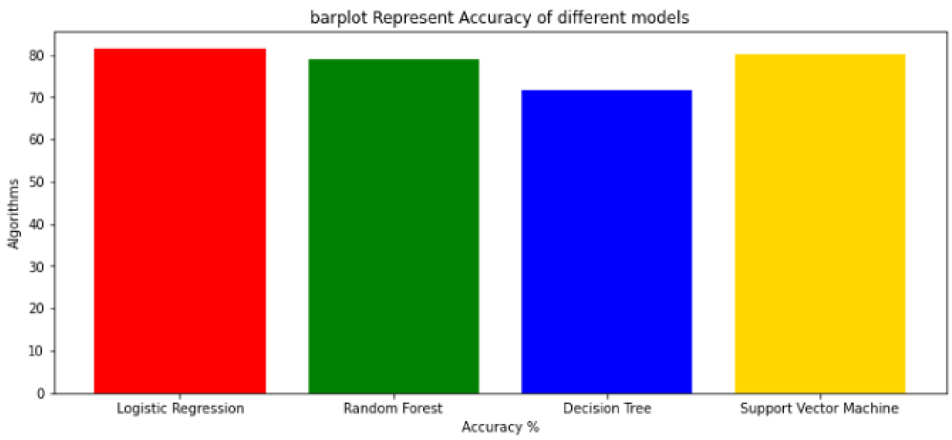
	Models	Accuracy_train	Accuracy_test	Recall
1	Logistic Regression	0.883598	0.814815	0.805556
3	Support Vector Machine	0.931217	0.802469	0.805556
2	Random Forest	0.873016	0.790123	0.638889
0	Decision Tree	0.910053	0.716049	0.750000

Comparisons

```
model_ev = pd.DataFrame({'Model': ['Logistic Regression','Random Forest','Decision Tree','Support Vector Machine'], 'Accur
print(lr_acc_score*100)
print(rf_acc_score*100)
print(dt_acc_score*100)
print(svc_acc_score*100)
```

```
81.48148148148148
79.01234567901234
71.60493827160494
80.24691358024691
```

```
colors = ['red','green','blue','gold','orange']
plt.figure(figsize=(12,5))
plt.title("barplot Represent Accuracy of different models")
plt.xlabel("Accuracy %")
plt.ylabel("Algorithms")
plt.bar(model_ev['Model'],model_ev['Accuracy'],color = colors)
plt.show()
```



Conclusion

Given the above `Recall` scores and comparions between all the models(As it is a Healthcare problem, Recall is considered to be the best metric solver in the Healthcare predictions, as we need to keep the margin of error really really small), as well as the corresponding F1 scores, we consider using `Logistic Regression` here.... Exerciseangina,Chest pain is major symptoms of heart attack.

Inference – Out of the 3 datasets, 2nd dataset having 14 features gave the best accuracy of 98.05% and recall of 96% which is the best given by SVM. In 1st dataset SVM is the best algorithm and in the 3rd dataset Logistic gave highest accuracy, but not best model compared with models of 1st and 2nd datasets.